Universidad de la República

Proyecto de final de carrera

EdRo - Un enjambre de robots

Autores:
Ylan Archimowicz
Santiago Martínez

Tutores:
Pablo Monzón
Rafael Canetti

19 de octubre de 2011

Resumen

En el presente documento se hará énfasis en los fundamentos y conceptos básicos de la colaboración en comunidades robóticas. En particular, se estudiará un subconjunto de éstas, los enjambres, los cuales se destacan del resto debido a su interacción entre agentes y agente-entorno.
Se presenta una técnica formal para la toma de decisiones por parte de robots, mediante la optimización de una función de costos. Para esto, serán estudiados los conceptos fundamentales de optimización y se presentará un algoritmo muy utilizado en el área, llamado "Particle Swarm Optimization". El mismo simula un enjambre de partículas para conseguir el óptimo de una función de costos.
Se diseñará e implementará un simulador orientado a la prueba y depuración de conductas colectivas del tipo anteriormente mencionado. El mismo es coherente a un modelo abstracto de un robot propuesto, inspirado en el estudio de un robot particular.
Se resolverá el problema de formación en línea recta para diferentes niveles de información del agente, problema que entendemos es "representativo".

Índice general

1 Introducción

1.1 Objetivos del proyecto

El presente documento constituye la documentación final del Proyecto de Fin de Carrera titulado "Enjambre de Robots" realizado para el Instituto de Ingeniería Eléctrica de la Facultad de Ingeniería, Universidad de la República. Los integrantes del grupo de trabajo son Santiago Martínez y Pablo Archimowicz, estudiantes de Ingeniería, opción Electrónica y Telecomunicaciones respectivamente. El proyecto en cuestión se llevo a cabo en el período comprendido entre Marzo de 2010 y junio de 2011, bajo la tutoría de los ingenieros Pablo Monzón y Rafael Canetti. El objetivo del proyecto fue el análisis de resolución de problemas mediante enjambre de robots y aplicaciones.

1.2 Motivación

Un robot es una entidad virtual o mecánica artificial. En la práctica, generalmente es un sistema electromecánico que, por su forma o movimientos, parece tener un propósito propio. La palabra robot puede utilizarse tanto para mecanismos físicos como para sistemas virtuales de software, aunque generalmente suele llamarse a estos últimos con el término de bots.
No hay un acuerdo sobre qué máquinas se pueden considerar robots, en cambio sí existe un consenso general entre el público y los expertos de qué suelen hacer los robots. Algunas de estas tareas pueden ser moverse, hacer funcionar un brazo mecánico, sentir y manipular su entorno y mostrar un comportamiento inteligente, especialmente si ese comportamiento imita al de los humanos o de otros animales.
Actualmente podría considerarse que un robot es una computadora con la capacidad y el propósito de movimiento, y que en general es capaz de desarrollar múltiples tareas de manera flexible según su programación. Así es que podría diferenciarse de algún electrodoméstico específico.
Aunque las historias sobre ayudantes y acompañantes artificiales, así como los intentos de crearlos, tienen una larga historia, las máquinas totalmente autónomas no aparecieron hasta el siglo XX. El primer robot programable y dirigido de forma digital, el Unimate, fue instalado en 1961 por General Motors para levantar piezas calientes de metal de una máquina de tinte y colocarlas [1].
Por lo general, la gente reacciona de forma positiva ante los robots con los que se encuentra. Los robots domésticos para la limpieza y mantenimiento del hogar son cada vez más comunes. No obstante, existe una cierta ansiedad sobre el impacto económico de la automatización y la amenaza del armamento robótico, una ansiedad que

se ve reflejada en el retrato a menudo perverso y malvado de robots presentes en obras de la cultura popular. Comparados con sus colegas de ficción, los robots reales siguen siendo limitados.
El crecimiento actual de la robótica da una oportunidad a desarrollar diferentes robots y algoritmos para varias aplicaciones. Continuo con dicho crecimiento nace la robótica colectiva, que se basa en varios agentes colaborando para lograr algún objetivo común. Como un subconjunto de este tipo de comportamiento, nace la robótica de enjambre que se basa en comportamientos vistos en la naturaleza. A rasgos generales basa su comportamiento en un algoritmo sencillo e igual para todos los agentes, los cuales podrán resolver una tarea únicamente colaborando entre ellos. La robótica de enjambre es un área en investigación actualmente en el mundo, su éxito se basa en tareas como el rescate, es decir, lugares que sean extremadamente peligrosos donde la utilización de un único agente que realice todo solo podría ser un problema. Esto se debe a que si este se dañara no se podría continuar con las tareas, sin embargo al poseer cien o más agentes colaborando, si uno de ellos o algunos se dañaran se podría continuar con las labores. También se podría utilizar la robótica de enjambre en lineas de producción, dado que, frenar la producción por un desperfecto de un robot, puede significar grandes costos, sin embargo, por lo comentado anteriormente, en caso de ser varios agentes, no sería necesario frenar la producción en el caso que uno de ellos se dañe.

1.3 Colaboración en robótica

La cooperación es el trabajo en equipo realizado por parte de un grupo de personas o entidades mayores con un mismo objetivo. Por lo general utilizan métodos comunes, en vez de trabajar de forma separada en competición.
Los escenarios de aplicación de la robótica han evolucionado en las últimas décadas, desde entornos muy simples y controlados a entornos muy dinámicos en exteriores. Al mismo tiempo, para afrontar ciertas aplicaciones, la cooperación en grupos de varios robots se ha convertido en una necesidad. Una tendencia en la actualidad es la investigación en sistemas que consideren la colaboración entre robots y sensores heterogéneos presentes en el entorno para multitud de aplicaciones, como robótica de servicio en entornos urbanos, o monitorización de desastres. La razón fundamental es que estas aplicaciones involucran entornos dinámicos, con condiciones cambiantes para la percepción, entre otros. En la mayoría de las ocasiones, un único agente (por ejemplo un robot o una cámara) no permite conseguir la robustez y eficacia necesarias. En estos casos, la cooperación de diferentes agentes (robots, sensores en el entorno) puede ser muy relevante[2].

1.3.1 Enjambres en robótica

Los enjambres de robots son un tipo de aproximación para la coordinación de sistemas de agentes autónomos que interactúan con ellos mismos y con su entorno. Una característica distintiva de esta aproximación es que los modelos poseen un elevado número de robots y que los mismos tienen un comportamiento relativamente simple. Este hecho es atractivo ya que implica costos menores de fabricación y consumo.

La meta de esta aproximación consiste en la determinación, tanto a nivel físico de los agentes como de sus conductas, de modo que mediante las interacciones entre agentes y agente-entorno emerjan patrones de comportamiento colectivo predeterminados de forma autoorganizada y distribuida. Esto significa que la organización del conjunto, a partir del comportamiento individual de los agentes y sin ningún otro elemento externo involucrado, incrementa su complejidad obteniendo así nuevas propiedades, las cuales no pueden ser reducibles a la suma de los comportamientos individuales. De este modo, la resolución de un problema requiere de un comportamiento individual relativamente simple, en comparación con el comportamiento necesario por parte de un solo agente para resolver la misma tarea (incluso más simple que el de un conjunto de agentes que operen sin autoorganización).
Procesos autoorganizativos se manifiestan comúnmente en la naturaleza. Un ejemplo claro son los patrones adoptados por las hojas en la tallos de las plantas (filotaxis) con el fin de aprovechar mejor la incidencia de los rayos solares [3] [4] [5].
Teorías sociales definen diversos tipos de orden social como procesos derivados de la autoorganización de las personas, denominado orden espontáneo. A esto se atribuye la evolución de la vida sobre la tierra, el lenguaje humano, la economía de mercado y el sistema de precios, entre otros [6].
Comportamientos relacionados con el concepto de emergencia se observan en muy diversos ámbitos, desde el surgimiento de las adhocracias, el software colaborativo, las redes sociales, la formación de la conciencia mediante la interacción neuronal, los tornados y los comportamientos eusociales de ciertos tipos de animales. Ejemplos de este tipo de comportamiento son las termitas en la creación de "catedrales" donde habitan, las hormigas en la búsqueda de alimento, las abejas y los pájaros en la migración y los peces por supervivencia. A este tipo de conductas adoptadas por un conjunto de animales se la denomina *inteligencia de enjambre*.
El estudio del comportamiento de agentes robóticos para alcanzar el estado de emergencia es una rama de la inteligencia artificial, denominada también *inteligencia de enjambre* la cual surge en forma biomimética al comportamiento observado en la naturaleza. Las aplicaciones para la robótica de enjambres consisten, por ejemplo, en la miniaturización extrema (nanorobótica), siendo aplicables a redes de sensores y actuadores en micromaquinaria o el cuerpo humano y en los diseños ultra económicos, como son los sistemas de sensores agrícolas, minería, etc.
Actualmente, existen diversos grupos de investigación en robótica de enjambre, enfocados en muy diversas áreas. El Laboratorio de "Bio-Ingeniería e Inteligencia en Sistemas Autónomos" de la Universidad de Bristol, en Inglaterra [7], realiza un estudio enfocado en la inteligencia de un enjambre de robots. Los objetivos del estudio consisten en la comprensión de la forma en que ha evolucionado la cultura en los seres "culturales" presentes en la naturaleza, a partir de comportamientos simples e imitación [8].
El proyecto "Swarm-Bots" [9] realizado en colaboración con grupos de investigación de Suiza, Bélgica e Italia, coordinados por la Comisión Europea CORDIS [10] se centra en el estudio de técnicas de autoorganización y autoensamblado de agentes robóticos.

1.4 Formulación de problemas como optimización de un costo

La acción tomada en forma individual, puede ser realizada mediante la optimización de una función de costos dependiente del estado del agente y su condición para con el alcance del objetivo. Esta función, el agente la trata de minimizar conforme evolucionan las iteraciones. Si el proceso es metaheurístico, las siguientes posiciones resultarán mejores que la actual respecto al costo asociado al problema. Este tipo de estrategia de acción, permiten al robot desempeñarse en forma automática y robusta al caso, es decir, independiente a cada caso particular a resolver. La evolución de sistemas a partir de la minimización de un costo es un efecto visto con normalidad en la naturaleza. En la filotaxis anteriormente descrita las plantas toman acción en la distribución de sus hojas con el fin de maximizar la incidencia solar. Las abejas también manifiestan este tipo de estrategia al diseñar las celdas de sus panales en forma hexagonal con la finalidad de minimizar la cantidad de cera necesaria para construirlo.
Lo anteriormente descrito puede modelarse como un problema de setup según la figura 1.1, donde existe una función de costos dependiente de la posición del los robots, una estrategia de acción, la cual implementa un algoritmo de optimización en función de la posición actual de los robots y el costo percibido en esta condición, determinando la nueva velocidad de cada agente. La nueva posición de los robots es determinada por la dinámica de los mismos, en función de la velocidad obtenida. La dinámica del agente depende únicamente de los robots con los cuales se desea resolver el problema. La Función de Costos es diseñada para ser mínima (nula, en general) en la condición del sistema (posición y velocidad, generalmente) de los robots en la cual se alcanza el objetivo. El valor de setup, $f_{deseado}$, es elegido de forma coherente al valor mínimo elegido antes. El Algoritmo de Optimización es determinado en función de las características geométricas de la función de costos y la performance que generalmente éstos poseen.
En general, el estudio de la performance se realiza observando el rendimiento del

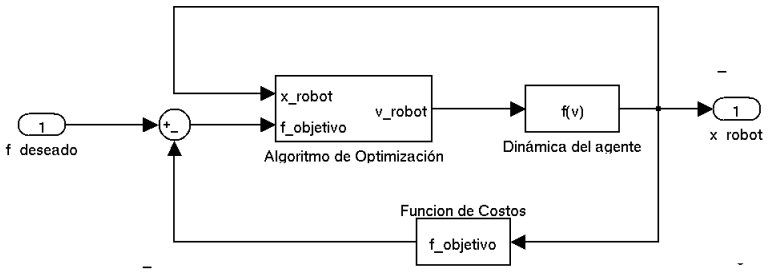


Figura 1.1: Diagrama de bloques del modelo de optimización

algoritmo con funciones de test estándar como la función de Rastrigin, Rosenbrock, Sphere, etc. Existen algoritmos de optimización diseñados en forma biomimética utilizando inteligencia de enjambre. Ejemplos claros son *"Ant Colony Optimization"* [11] y *"Particle Swarm Optimization"* [12]. El algoritmo *Ant Colony Optimization* (ACO) es un algoritmo de optimización de funciones multidimensionales, no lineales ni convexas, basado en el comportamiento de una colonia de hormigas cuando realizan la búsqueda de comida. éste fue propuesto por Marco Dorigo en 1992 para su

tesis de doctorado. *Particle Swarm Optimization* (PSO) surgió como un algoritmo de optimización de funciones multidimensionales, no lineales, no convexas y continuas propuesto por James Kennedy y Russell Eberhart en 1995. Posteriormente se presentó la versión para funciones discretas [13]. El método fue desarrollado simulando un comportamiento social simple, inspirado en bandadas de pájaros. éste, simula un enjambre de partículas con un número relativamente alto de agentes que mediante un conjunto de pocas reglas de comportamiento simple y comunicación entre ellas, logran navegar en el espacio y alcanzar el mínimo global en el dominio determinado de la función a optimizar, evitando mínimos locales de la misma. De este modo, las partículas aprenden de su propia experiencia y de la del resto del enjambre, pudiendo así alcanzar el óptimo en el espacio de búsqueda.
El algoritmo PSO determina la nueva velocidad de las partículas en función de su experiencia previa, la experiencia previa del enjambre y su inercia. De este modo, cada partícula del enjambre se ve influenciada, a nivel particular, por su experiencia previa y a nivel colectivo por la experiencia previa del resto del enjambre. Así, si al menos un elemento del enjambre consigue alcanzar el óptimo, el resto de las partículas del enjambre se verán influenciadas por ésta, haciendo tender a los agentes del enjambre a ese lugar. En caso de no ser el óptimo global el alcanzado, sino uno local, tanto la influencia particular de cada agente como su inercia provocarán que el enjambre, en conjunto, esquive esta zona y forzará, en cierto modo, a las partículas atrapadas en este mínimo local a escapar del lugar.

1.5 Organización del documento

El presente documento se divide en tres partes. Primero fueron presentados los objetivos del proyecto, la motivación que llevó a su creación, los conceptos básicos de colaboración en robótica (capítulo 1) y los conceptos principales sobre optimización (capítulo 2), basado esencialmente en el libro "Programación Lineal y no Lineal" de D.E. Luenberger [14]. También se estudia un algoritmo optimización destacado, llamado Particle Swarm Optimization (de ahora en mas PSO) y un robot en particular, el *Khepera III* (capítulos 3 y 4, respectivamente).
A continuación se pasa a una segunda parte donde se expondrá una breve descripción del simulador diseñado, conjuntamente con un modelo del robot en diferentes niveles de información (capítulo 5).
Finalmente se realiza la resolución de un problema basado en robótica de enjambre (capítulo 6) y se presentan las conclusiones del proyecto (capítulo 8).
El aporte fundamental de este proyecto se ve reflejado en las últimas dos partes del documento, en particular, en el diseño del simulador utilizado y en las conclusiones a las cuales se arribó a partir de la experiencia obtenida.

2 Optimización

2.1 Introducción

El presente capítulo está basado en el libro "Programación Lineal y no Lineal" de D.E. Luenberger [14].
En estos tiempos ya está bien fijado el concepto de optimización como uno de los principios básicos de análisis de muchos problemas complejos de decisión o ubicación. Al utilizar esta filosofía de la optimización se enfoca un problema complejo de decisión, que incluye la selección de valores para cierto número de variables interrelacionadas, centrando la atención en un solo objetivo diseñado para cuantificar el rendimiento y medir la calidad de la decisión. Este único objetivo se maximiza según las restricciones que pueden limitar la selección de los valores de las variables de decisión.
Como es lógico, es rara la ocasión en que se puede representar en su totalidad toda la complejidad de las interacciones de variables, restricciones y objetivos apropiados, cuando se trata de un problema complejo de decisión. Al igual que sucede con todas las técnicas cuantitativas de análisis, una formulación determinada de optimización deberá considerarse como una aproximación. Así pues, la optimización deberá considerarse como un instrumento de conceptualización y análisis, más que como un principio que produzca la solución filosóficamente correcta.

2.1.1 Tipos de problemas

1 *Programación lineal*

La programación lineal es, sin duda, el mecanismo más natural para formular una gran cantidad de problemas con el mínimo esfuerzo. Un problema de programación lineal se caracteriza, como su nombre indica, por funciones lineales de las incógnitas. En principio se podría pensar que las formulaciones de programación lineal son populares porque las matemáticas son mas agradables, la teoría más rica, y los cálculos, más simples para los problemas lineales que para los que no lo son. Pero éstas no son las razones fundamentales. Desde el punto de vista de las propiedades matemáticas y de los cálculos, hay clases de problemas de optimización mucho más generales que los problemas de programación lineal y que tienen teorías elegantes y eficaces, y para las cuales se dispone de algoritmos efectivos. Parece que la popularidad de la programación lineal estriba principalmente en la fase de formulación del análisis, más que en la fase de solución y por una buena razón; por un lado, gran número de restricciones y objetivos que surgen en la práctica son lineales, por otro lado es que suelen ser los mas fáciles de definir.

Así pues, aunque una función objetivo no sea lineal pura a causa de su definición intrínseca, a menudo es mucho más fácil definirla como lineal que elegir cualquier otra forma funcional.

2 *Problemas sin restricciones*

A diferencia de lo que puede parecer en un principio, puede argumentarse que los problemas sin restricciones son realmente útiles dado que si se amplía el ámbito de un problema para tener en cuenta todas las variables de decisión relevantes, entonces no puede haber restricciones o, dicho de otra forma, las restricciones representan delimitaciones artificiales del ámbito de aplicación, y cuando se amplía el ámbito, las restricciones desaparecen. Otro argumento es que a veces es fácil convertir problemas con restricciones en problemas sin ellas. Por ejemplo, el único efecto de las restricciones de igualdad no es otro que limitar los grados de libertad, sobre todo al hacer que ciertas variables sean funciones de otras.
Además de representar un tipo importante de problemas prácticos, el estudio de problemas sin restricciones proporciona un componente clave para un caso más general de problemas, como son los problemas con restricciones.

3 *Problemas con Restricciones*

Muchos problemas prácticos se formulan como problemas con restricciones. Esto se debe a que en la mayoría de los casos un problema complejo, como la planificación de una dependencia gubernamental grande, no puede tratarse directamente teniendo en cuenta todas las elecciones posibles, sino que debe descomponerse en subproblemas separados; cada subproblema tiene restricciones que han sido impuestas para limitar su campo de aplicación.
El enunciado del problema general de la programación matemática puede ser:

$$\begin{aligned} minimizar \quad & f(x) \\ sujeto\ a: \quad & h_i(x) = 0,\ \forall i = 1, \cdots, m \\ & g_i \leq 0\ ,\ \forall j = 1, \cdots, r \\ & x \in S \end{aligned}$$

En esta formulación, x es un vector de incógnitas n-dimensional, f, $h_i,\ i = 1, 2...m$ y $g_j,\ j = 1, 2, ..., r$ son funciones con valores reales de las variables $x_1, x_2, ..., x_n$. El conjunto S es un subconjunto de un espacio n-dimensional. La función f es la función objetivo del problema y las ecuaciones, las desigualdades y conjunto S son restricciones.
En general se introducen restricciones adicionales para simplificar el problema en cierto sentido, Por ejemplo, generalmente es necesario que las funciones del problema sean continuas, o quizá que tengan derivadas continuas. Además, no se permite que el conjunto S sea arbitrario, sino que en general ha de ser una región conexa del espacio n-dimensional. De esta forma se asegura la posibilidad de hacer cambios pequeños en x.

4 *Tamaño de los problemas*

Una medida evidente de la complejidad de un problema de programación es su tamaño, medido en función del número de variables incógnitas o del número de restricciones. Los problemas se pueden clasificar en tres clases: problemas a pequeña escala, que tienen como máximo 5 incógnitas y restricciones, problemas a escala intermedia que poseen entre cinco y cien variables y problemas a gran escala con más de cien e incluso miles de variables y restricciones. Esta clasificación no es completamente rígida, sino que refleja, al menos aproximadamente, tanto el tamaño como las diferencias básicas de enfoque correspondientes a los problemas de distintos tamaños. Por regla general los problemas a pequeña escala se pueden resolver a mano. Los problemas a escala intermedia se pueden resolver con un computador y un lenguaje de programación matemática de uso general. Los problemas a gran escala necesitan lenguajes sofisticados que se sirven de una estructura especial y suelen requerir computadores con gran poder de cálculo.

5 *Algoritmos iterativos y convergencia*

Como norma, al buscar un vector que resuelva el problema de programación, se elige un vector inicial x_0 y el algoritmo genera un vector mejorado x_1. El proceso se repite y encuentra una solución mejor x_2. Continuando así, se halla una sucesión de puntos cada vez más apropiados, que tiende a un punto solución x^*. En los problemas de programación lineal la sucesión generada es de longitud finita, obteniendo el punto solución exactamente después de un número finito (aunque al principio no esté especificado) de pasos. En problemas de programación no lineal, generalmente, la sucesión no alcanzará nunca el punto solución, sino que converge hacia él. A los efectos prácticos el proceso termina cuando se obtiene un punto suficientemente próximo al punto solución.
La teoría de los algoritmos iterativos puede dividirse en tres aspectos. El primero se refiere a la creación de los algoritmos propiamente dichos. El segundo aspecto es la comprobación de que un algoritmo dado generará, de hecho, una sucesión convergente hace un punto solución. El tercer aspecto es el análisis de la convergencia local, y trata de la forma en que la sucesión generada de puntos converge hace la solución. Un problema no se puede considerar resuelto sólo porque se conoce un algoritmo que convergerá a la solución, pues tal vez se necesite mucho tiempo para reducir el error hasta una tolerancia aceptable. Por lo que al sugerir un algoritmo es esencial disponer de alguna estimación del tiempo necesario para llegar a la solución. Es la tasa de convergencia de la teoría lo que permite la evaluación y comparación cuantitativas de algoritmos distintos.

2.1.2 Propiedades básicas de los programas lineales

Un problema de programación lineal es un problema de optimización matemática en el cual la función objetivo es lineal en las incógnitas y las restricciones constan de igualdades y desigualdades lineales. Más allá de las diferencias entre la forma de las restricciones y de la función objetivo encontradas en situaciones concretas, los problemas de programación lineal pueden convertirse a la siguiente forma estándar:

$$
\begin{aligned}
\textit{minimizar} \quad & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\
\textit{sujeto a}: \quad & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\
& a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\
& a_{31}x_1 + a_{32}x_2 + \cdots + a_{3n}x_n = b_3 \\
& \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
& a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \\
& x_1 \geq 0;\ x_2 \geq 0; \cdots;\ x_m \geq 0;
\end{aligned}
$$

donde b_i, c_i y a_i son constantes reales fijas con b_i mayor a 0 y las x_i también son reales siendo estas las variables a determinar.
Al modelar problemas reales como un problema de optimización matemática, muchos de ellos, naturalmente, toman una forma de programa lineal. Para abordarlos con el mismo conjunto de estrategias, es necesario transformarlos a la forma estándar.
Forma 1, desigualdades en las restricciones
Téngase el problema de programación lineal:

$$
\begin{aligned}
\textit{minimizar} \quad & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\
\textit{sujeto a}: \quad & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1 \\
& a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leq b_2 \\
& a_{31}x_1 + a_{32}x_2 + \cdots + a_{3n}x_n \leq b_3 \\
& \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
& a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m \\
& x_1 \geq 0;\ x_2 \geq 0; \cdots;\ x_m \geq 0;
\end{aligned}
$$

este problema puede transformarse a un problema de la forma estándar introduciendo variables de holgura ($z_1 ... z_m$), de modo que el problema pueda expresarse del siguiente modo:

$$
\begin{aligned}
\textit{minimizar} \quad & c_1x_1 + c_2x_2 + \cdots + c_nx_n + 0.z_1 + 0.z_2 + ... + 0.z_m \\
\textit{sujeto a}: \quad & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + z_1 = b_1 \\
& a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n + z_2 = b_2 \\
& a_{31}x_1 + a_{32}x_2 + \cdots + a_{3n}x_n + z_3 = b_3 \\
& \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
& a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n + z_m = b_m \\
& x_1 \geq 0;\ x_2 \geq 0; \cdots;\ x_m \geq 0; \\
& z_1 \geq 0;\ z_2 \geq 0; \cdots;\ z_m \geq 0;
\end{aligned}
$$

el cual es un programa lineal de la forma estándar. El mismo posee m variables más respecto al problema original.
También las desigualdades pueden darse en el otro sentido, problema el cual puede convertirse a uno de la forma estándar introduciendo m variables excedentes, de modo que una restricción del tipo:

$$a_{ij}x_1 + a_{ij+1}x_2 + \cdots + a_{in}x_n \geq b_i;$$

puede convertirse a la forma:

$$\begin{aligned} a_{ij}x_1 + a_{ij+1}x_2 + \cdots + a_{in}x_n - z_i &= b_i; \\ z_i &\geq 0; \end{aligned}$$

siendo ahora una restricción de un problema de la forma estándar.
Forma 2, presencia de variables libres
Téngase un problema de programación lineal, el cual difiere de la forma estándar debido a la ausencia de una restricción de no negatividad de una de las variables presentes. Supóngase que la variable x_i no presenta restricciones de no negatividad, para transformarlo a la forma estándar se agregan las variables u_i y v_i de modo que:$x_i = u_i - v_i;\ u_i \geq 0;\ v_i \geq 0$; de este modo, se sustituye x_i por su representación en las variables u_i y v_i preservando la linealidad del problema y transformándolo a un problema de la forma estándar con $n+1$ variables. Esta técnica abre la posibilidad de infinitos resultados ya que la representación de un x_i dado no es única, sino infinita. Otra forma de abordar este problema es eliminar la variable x_i, Esto se consigue despejándola de una de las restricciones en la cual esté presente esta variable. Por ejemplo en la i-ésima restricción:

$$a_{ij}x_1 + a_{ij+1}x_2 + ... + a_{in}x_n = b_i;$$

donde si a_{ij} es no nulo se obtiene:

$$x_1 = -\frac{a_{ij+1}}{a_{ij}}x_2 - ... - \frac{a_{in}}{a_{ij}}x_n + \frac{b_i}{a_{ij}};$$

de este modo puede sustituirse x_i en las $m-1$ restantes restricciones y en el funcional a minimizar, dando un problema de $m-1$ restricciones (debido a que, en este caso, la i-ésima restricción desaparece) y $n-1$ incógnitas. Debido a que cualquier combinación lineal de las variables resulta en un x_i factible debido a la no negatividad de esta variable, no se agregan restricciones y por lo tanto el problema se presenta en forma estándar.

La programación lineal es de utilidad para modelar gran cantidad de problemas reales, algunos de ellos pueden encontrarse en [14].

6 *Soluciones Básicas*

Considérese el conjunto de ecuaciones que representan en forma matricial el conjunto de restricciones:

$$A_{m\times n}x_{n\times 1} = b_{m\times 1};$$

entonces, si el rango de $A_{m\times n}$ es m, se puede elegir un conjunto de m columnas linealmente independientes de $A_{m\times n}$ (supóngase las m primeras) y llámesele $B_{m\times m}$ la matriz que éstas conforman. Por construcción, $B_{m\times m}$ es no singular y por lo tanto el sistema de ecuaciones:

$$B_{m\times m}x_B = b_{m\times 1};$$

tiene solución única x_B.
Ahora se considera

$$x_{n\times 1} = \begin{bmatrix} x_B \\ 0 \end{bmatrix}_{n\times 1}$$

entendiéndose entonces que las primeras m componentes son las componentes del vector x_B y las restantes $n-m$ son 0.
Por lo tanto, el vector $x_{n\times 1}$ es solución del sistema de ecuaciones $Ax_{n\times 1} = b_{m\times 1}$ lo que plantea la definición de solución básica del sistema de ecuaciones lineal.

Definición **1** *Dado un sistema de m ecuaciones lineales y n incógnitas $Ax_{n\times 1} = b_{m\times 1}$, sea $B_{m\times m}$ una submatriz de $A_{m\times n}$ de tamaño $m \times n$ no singular cualquiera formada por algunas columnas de $A_{m\times n}$. Entonces, si las $n-m$ componentes de $x_{n\times 1}$ no asociadas a las columnas de $B_{m\times m}$ son 0, la solución del problema resultante se llama solución básica del problema $Ax_{n\times 1} = b_{m\times 1}$ respecto a la base $B_{m\times m}$.*

Para que tenga sentido la definición anterior, deben de tomarse una serie de hipótesis respecto a las características de la matriz $A_{m\times n}$. éstas son, que $n > m$ y que las m filas de $A_{m\times n}$ sean linealmente independientes (para que el problema $B_{m\times m}x_B = b_{m\times 1}$ tenga, al menos una solución).
Dichas hipótesis son asumidas a lo largo de este desarrollo y ambas se denominan hipótesis del rango completo.
Además, cuando un vector $x_{n\times 1}$ cumple también las restricciones de no negatividad se dice que es factible para las restricciones. Si también es una solución básica, se lo denomina solución factible básica.

7 *Teorema fundamental de la programación lineal*

Considérese el programa lineal de la forma estándar:

$$\begin{aligned} \textit{minimizar} \quad & c^t_{1\times n}x_{n\times 1} \\ \textit{sujeto a} \quad & A_{m\times n}x_{n\times 1} = b_{m\times 1}; \\ & x_{n\times 1} \geq 0; \end{aligned}$$

una solución factible para estas restricciones que alcance el valor mínimo de la función objetivo se la denomina solución factible optimal, si además es básica, se la denomina solución factible básica optimal.

Teorema **1** *(Teorema fundamental de la programación lineal): Dado un programa lineal de la forma estándar (como el programa anterior), donde $A_{m\times n}$ cumple la hipótesis del rango completo, entonces:*

- *Si hay una solución factible, hay una solución factible básica;*
- *Si hay una solución factible optimal, hay una solución factible básica optimal.*

Gracias a este teorema, el problema de programación lineal se reduce a la búsqueda entre soluciones factibles básicas. Por lo tanto, para un problema de la forma estándar con n variables y m restricciones existen, a lo sumo, $C_m^n = \frac{n!}{m!(n-m)!}$ soluciones básicas, existiendo un conjunto finito de posibilidades.

8 *Relaciones con la convexidad*

Los resultados comentados anteriormente tienen interpretación geométrica en la teoría de conjuntos convexos que vale la pena mostrar debido a su claridad.

Definición **2** *Un punto x de un conjunto C se denomina punto extremo de C si no hay dos puntos distintos x_1 y x_2 en C tales que $x = \alpha x_1 + (1-\alpha)x_2$ para algún $\alpha, 0 \leq \alpha \leq 1$.*

Es decir, que x no pertenece estrictamente al interior de ningún segmento formado por dos elementos cualesquiera del conjunto. Un ejemplo son los vértices de un triángulo.
Para el siguiente teorema, es necesario introducir el concepto de *polítopo*.

Definición **3** *Los polítopos son una generalización a cualquier dimensión de un polígono bidimensional, o un poliedro tridimensional.*

Teorema **2** *(Equivalencia de puntos extremos y soluciones básicas): Sea $A_{m\times n}$ de rango m y $b_{m\times 1}$ un m-vector. Sea K el polítopo convexo formado por todos los x_{n1} que satisfacen*

$$A_{m\times n}x_{n\times 1} = b_{m\times 1};$$
$$x_{n\times 1} \geq 0;$$

Entonces, un vector $x_{n\times 1}$ es un punto extremo de K si, y solo si, $x_{n\times 1}$ es una solución factible básica de las restricciones anteriores.

La correspondencia anterior entre puntos extremos y soluciones factibles permite demostrar cierta propiedades geométricas del polítopo convexo K definido por el conjunto de restricción anterior.

Corolario **1** *Si el conjunto convexo K anteriormente mencionado es no vacío, entonces tiene al menos un punto extremo.*

COROLARIO **2** *Si hay una solución optimal finita, es decir, de módulo acotado, a un problema de programación lineal, hay una solución optimal finita que es un punto extremo del conjunto de restricción.*

COROLARIO **3** *El conjunto de restricción K anteriormente descrito tiene a lo sumo un número finito de puntos extremos.*

COROLARIO **4** *Si el polítopo convexo K es acotado, entonces K es un poliedro convexo.*

Para ejemplificar los resultado anteriores, téngase el siguiente conjunto de restricciones de E^3:

$$\begin{aligned} &x_1 + x_2 + x_3 = 1; \\ &2x_1 + 3x_2 = 1; \\ &x_1 \geq 0;\ x_2 \geq 0;\ x_3 \geq 0; \end{aligned}$$

que puede representarse en forma matricial de la siguiente forma:

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} x_1 \geq 0; x_2 \geq 0; x_3 \geq 0;$$

Dada esta matriz de restricciones, se puede acotar la cantidad de soluciones básicas (como se observó anteriormente) a un valor de $C_2^3 = \frac{3!}{2!(3-2)!} = 3$ que corresponde a las diferentes formas de seleccionar dos columnas de las tres presentes. Las posibles soluciones básicas se determinan con las diferentes bases B_i generadas con las columnas de la matriz anterior resolviendo el sistema de ecuaciones $B_i x_{B_i} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. Ahora, determinando las diferentes bases de E^2 se obtienen las posibles soluciones básicas:

$$B_1 = \begin{bmatrix} 1 & 1 \\ 2 & 3 \end{bmatrix}; B_2 = \begin{bmatrix} 1 & 1 \\ 3 & 0 \end{bmatrix}; B_3 = \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix};$$

para B_1:

$$x_{B_1} = \begin{bmatrix} 2 \\ -1 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{B_1} = \begin{bmatrix} 2 \\ -1 \\ 0 \end{bmatrix}$$

para B_2:

$$x_{B_2} = \begin{bmatrix} 1/3 \\ 2/3 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{B_2} = \begin{bmatrix} 0 \\ 1/3 \\ 2/3 \end{bmatrix}$$

para B_3:

$$x_{B_3} = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{B_3} = \begin{bmatrix} 1/2 \\ 0 \\ 1/2 \end{bmatrix}$$

Sólo los dos últimos resultados son efectivamente soluciones factibles básicas ya que la primera no cumple la condición de no negatividad.

2.1.3 Dualidad

Asociado a todo problema de programación lineal, hay un problema correspondiente de programación lineal dual. Ambos se construyen a partir de los mismos coeficientes de costo y restricciones, pero si uno de ellos es de minimización, el otro es de maximización, y los valores optimales de las funciones objetivo correspondientes, si son finitos, son iguales. Las variables del problema dual se pueden interpretar como precios asociados a las restricciones del problema original, y mediante esta asociación se puede proporcionar una caracterización económicamente significativa al dual cuando haya tal caracterización para el primal.

9 *Programas lineales duales*

Se parte de la estrategia usual de considerar problemas en forma estándar, dado que la relación de dualidad es más simétrica para aquellos problemas que se expresan tan sólo en función de desigualdades.
Primal

$$\begin{aligned} minimizar \quad & c^T x \\ sujeto\ a \quad & Ax \geq b;\ x \geq 0 \end{aligned}$$

Dual

$$\begin{aligned} maximizar \quad & \lambda^T b \\ sujeto\ a \quad & \lambda^T A \leq c^T;\ \lambda \geq 0 \end{aligned}$$

Si A es una matriz $m \times n$, entonces x es un vector columna n-dimensional, b es un vector columna m-dimensional, c^T es un vector fila n-dimensional y λ es un vector fila m-dimensional. El vector x es la variable del problema primal y λ es la variable del problema dual.
Los dos problemas se denominan forma simétrica de la dualidad y se pueden utilizar para definir el dual de cualquier programa lineal. Es importante señalar que las funciones de los problemas primal y dual se pueden invertir, Así, al estudiar en detalle el proceso de obtención del dual a partir del primal, el intercambio de los vectores de costo y restricción, la trasposición de los coeficientes de la matriz, la inversión de las desigualdades de restricción y el cambio de minimización a maximización, se observa que aplicando este mismo proceso al dual produce el primal.

Se puede hallar el dual de cualquier problema de programación lineal convirtiendo el problema a la forma del primal expuesto antes. Por ejemplo, dado un problema lineal de la forma estándar

$$\begin{aligned} minimizar \quad & c^T x \\ sujeto\ a \quad & Ax = b;\ x \geq 0 \end{aligned}$$

se lo puede expresar como:

$$\begin{aligned} minimizar \quad & c^T x \\ sujeto\ a \quad & Ax \geq b;\ -Ax \geq -b;\ x \geq 0 \end{aligned}$$

y a este se le aplica el mismo procedimiento explicado anteriormente.

LEMA **1** *(Lema de dualidad débil)*

$$Primal \tag{2.1}$$

$$\begin{aligned} minimizar \quad & c^T x \\ sujeto\ a \quad & Ax = b;\ x \geq 0 \end{aligned}$$

$$Dual \tag{2.2}$$

$$\begin{aligned} maximizar \quad & \lambda^T b \\ sujeto\ a \quad & \lambda^T A \leq c^T;\ \lambda \geq 0 \end{aligned}$$

si x, λ son factibles para 2.1 y 2.2, respectivamente, entonces $c^T x \geq \lambda^T b$

COROLARIO **5** *Si x_0, λ_0 son factibles para 2.1 y 2.2, respectivamente y si $c^T x_0 = \lambda_0^T b$, entonces x_0 y λ_0 son optimales para sus respectivos problemas.*

10 *Teorema de dualidad de la programación lineal*

Si uno de los problemas 2.1 o 2.2 tiene una solución optimal finita, también la tiene el otro, y los valores correspondientes de las funciones objetivo son iguales. Si un problema tiene un objetivo no acotado, el otro no tiene una solución finita.

TEOREMA **3** *Sea el problema de programación lineal 2.1 con solución factible básica optimal correspondiente a la base B. Entonces, el vector λ que satisfaga $\lambda^T = c_B^T B^{-1}$ es una solución optimal para el problema dual 2.2. Los valores optimales de ambos problemas son iguales.*

11 *Interpretación geométrica*

Las relaciones de dualidad pueden considerarse en función de las interpretaciones duales de las restricciones lineales. Considérese un problema lineal en forma estándar. Por motivos de concreción téngase el problema:

$$\begin{aligned} \textit{minimizar} \quad & 18x_1 + 12x_2 + 2x_3 + 6x_4 \\ & 3x_1 + x_2 - 2x_3 + x_4 = 2 \\ & x_1 + 3x_2 - x_4 = 2x_1 \geq 0;\ x_2 \geq 0;\ x_3 \geq 0;\ x_4 \leq 0; \end{aligned}$$

donde las columnas de las restricciones se representan en el espacio de requerimientos de la figura 2.1.
El problema dual es:

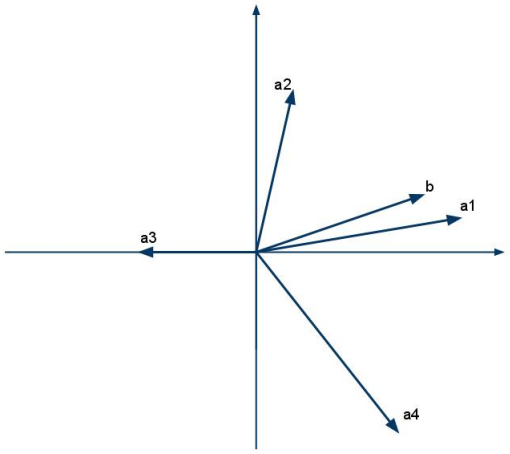


Figura 2.1: Espacio de requerimientos primales

$$\begin{aligned} \textit{maximizar} \quad & 2\lambda_1 + 2\lambda_2 \\ \textit{sujeto a} \quad & 3\lambda_1 + \lambda_2 \leq 18\lambda_1 + 3\lambda_2 \leq 12 \\ & -2\lambda_1 \leq 2 \\ & \lambda_1 - \lambda_2 \leq 6 \end{aligned}$$

El problema dual se ilustra geométricamente en la figura 2.2. Cada columna a_i del primal define una restricción del dual como un semiespacio cuya cota es ortogonal con respecto a ese vector columna y está situada en un punto determinado por c_i. El objetivo dual se maximiza en un punto extremo de la región factible dual. En ese punto, hay exactamente dos restricciones duales activas. Estas restricciones activas corresponden a una base optimal del primal. De hecho, el vector que define el objetivo dual es una combinación lineal positiva de los vectores. En el ejemplo, b es una combinación positiva de a_1 y a_s. Las ponderaciones de esta combinación son las x_i de la solución del primal.

12 *Sensibilidad*

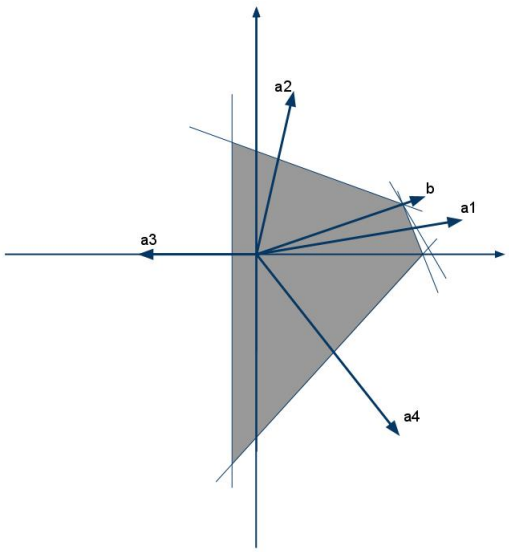


Figura 2.2: Espacio de requerimientos duales

Sea el problema de programación lineal 2.1, la base optimal es B, con la solución correspondiente $(x_B, 0)$, donde $x_B = B^{-1}b$. Una solución del problema dual correspondiente $\lambda^T = c_B^T B^{-1}$.
Ahora, suponiendo no degeneración, los cambios pequeños en el vector b no harán que cambie la base optimal, Así, para $b+\Delta b$ la solución optimal es $x = (x_B+\Delta x_B, 0)$. Donde $\Delta x_B = B^{-1}\Delta b$. Por tanto, el incremento correspondiente de la función de costo es $\Delta z = c_B^T \Delta x_B = \lambda^T \Delta b$
Esta ecuación demuestra que λ produce la sensibilidad del costo optimal según los cambios pequeños del vector b. Si se resolviera un nuevo problema con b sustituida por $b + \Delta b$, el cambio en el valor optimal de la función objetivo sería $\lambda^T \Delta b$.

13 *Holgura complementaria*

Teorema **4** *(Holgura complementaria-forma asimétrica). Sea x y λ soluciones factibles para los problemas primal y dual, respectivamente. Una condición necesaria y suficiente para que los dos sean soluciones optimales es que para toda i*

- $x_i > 0 \Rightarrow \lambda^T a_i = c_i$
- $x_i = 0 \Leftarrow \lambda^T a_i < c_i$

Teorema **5** *(Holgura complementaria-forma simétrica). Sea x y λ soluciones factibles para los problemas primal y dual, respectivamente. Una condición necesaria y suficiente para que los dos sean soluciones optimales es que para toda i y j:*

- $x_i > 0 \Rightarrow \lambda^T a_i = c_i$
- $x_i = 0 \Leftarrow \lambda^T a_i < c_i$
- $\lambda_j > 0 \Rightarrow a^j x = b_j$
- $\lambda_j = 0 \Leftarrow a^j x > b_j$

donde a^j es la j-ésima fila de A.

2.1.4 Propiedades básicas de la solución y algoritmos

A continuación se estudian los problemas de optimización de la forma:

$$\begin{aligned} &minimizar && f(x) \\ &sujeto\ a && x \in \Omega \end{aligned}$$

(2.3)

donde f es una función real, no necesariamente lineal, y Ω, el conjunto factible, es un subconjunto de E^n.

14 *Condiciones necesarias de primer orden*

La primer cuestión en el estudio de un problema de optimización es saber si existe una solución. Una herramienta a utilizar para abordar este problema es el teorema de Weierstras, que establece que si f es continua y Ω compacto existe una solución. De todos modos, el centro del problema consiste en caracterizar los puntos solución y diseñar métodos para hallarlos. Al analizar el problema general se distinguen dos tipos de puntos solución: puntos *mínimos locales* y puntos *mínimos globales.*

Definición **4** *Un punto $x^* \in \Omega$ es un punto mínimo relativo o un punto mínimo local de f en Ω, si existe un $\epsilon > 0$ tal que $f(x) \geq f(x^*) \forall x \in \Omega$ a una distancia de x^* menor que ϵ (es decir, $x \in \Omega$ y $|x - x^*| \leq \epsilon$). Si $f(x) > f(x^*) \forall x \in \Omega$, $x \neq x^*$ a una distancia de x^* menor que ϵ, entonces se dice que x^* es un punto mínimo relativo estricto de f en Ω.*

Definición **5** *Un punto $x^* \in \Omega$ es un punto mínimo global de f en Ω, si $f(x) \geq f(x^*) \forall x \in \Omega$. Si $f(x) > f(x^*) \forall x \in \Omega$, $x \neq x^*$, entonces se dice que x^* es un punto mínimo global estricto de f en Ω.*

Por definición del problema, siempre se busca un punto mínimo global de f en el conjunto Ω. Sin embargo, desde el punto de vista práctico, muchos procedimientos sólo permiten comparar valores entre puntos cercanos, por lo que la atención se centra en los puntos mínimos relativos.
Por regla general, para hallar condiciones y soluciones globales a un problema, el mismo debe cumplir ciertas propiedades de convexidad que garanticen que todo mínimo relativo sea mínimo global. Por este motivo, la atención se centra en la búsqueda de mínimos relativos.

2.1.5 Direcciones factibles

Para deducir condiciones necesarias satisfechas por un punto mínimo relativo x^*, la idea básica consiste en considerar un movimiento que se aleje del punto en una dirección dada. En cualquier dirección, la función objetivo puede considerarse de una variable. Así, dada $x \in \Omega$, se dice que d es una dirección factible en x si existe algún $\bar{\alpha} > 0$ tal que $x + \alpha d \in \Omega \forall \alpha, 0 \leq \alpha \leq \bar{\alpha}$.

PROPOSICION **1** *(condiciones necesarias de primer orden). Sea Ω un subconjunto de E^n y $f \in C^1$, una función en Ω. Si x^* es un punto mínimo relativo estricto de f en Ω, entonces para cualquier $d \in E^n$ que sea una dirección factible en x^*, resulta $\nabla f(x^*)d \geq 0$.*

COROLARIO **6** *(caso sin restricciones). Sea Ω un subconjunto de E^n y $f \in C^1$, una función en Ω. Si x^* es un punto mínimo relativo estricto de f en Ω y x^* es un punto interior de Ω, entonces $\nabla f(x^*) = 0$.*

Las condiciones necesarias en el caso sin restricciones producen n ecuaciones (cada componente de ∇f) de n incógnitas (cada componente de x^*), pudiendo resolverse para determinarse la solución.

2.2 Ejemplos de problemas sin restricciones

15 *Aproximación*

Un uso común de optimización es para la aproximación de funciones, suponga el caso que durante un experimento se observa el valor de una función g en m puntos, $x_1,\ x_2, \ldots, x_m$, con lo que se conocen $g(x_1),\ g(x_2), \ldots,\ g(x_3)$ y se desea aproximar la función por un polinomio de grado n (o menor), donde $m > n$:

$$h(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_0$$

En relación con cualquier elección de polinomio, existirá un conjunto de errores $\varepsilon_k = g(x_k) - h(x_k)$. Se define la mejor aproximación como el polinomio que minimiza la suma de los cuadrados de estos errores, es decir, se minimiza

$$\sum_{k=1}^{m} (\varepsilon_k)^2$$

Lo cual, a su vez, significa que se minimiza

$$f(a) = \sum_{k=1}^{m} [g(x_k) - (a_n x_k^n + a_{n-1}x_k^{n-1} + ... + a_0)]^2$$

respecto a $a = (a_0,\ a_1, \ldots,\ a_n)$ para hallar los mejores coeficientes. Ésta es una expresión cuadrática en los coeficientes a. Para hallar una representación compacta de este objetivo de define $q_{ij} = \sum_{k=1}^{m} (x_k)^{i+j},\ b_j = \sum_{k=1}^{m} g(x_k)(x_k)^j,\ c = \sum_{k=1}^{m} g(x_k)^2$. Entonces, recurriendo al álgebra se puede mostrar que

$$f(a) = a^T Q a - 2b^T a + c$$

donde:

$$Q = [q_{ij}],\ b = (b_1,\ b_2, ...,\ b_{n+1})$$

Las condiciones necesarias de primer orden establecen que se debe anular el gradiente de f, entonces

$$Qa = b$$

que es un sistema de ecuaciones de $n + 1$ ecuaciones de donde puede determinarse a.

16 *Control*

Los problemas dinámicos, donde las variables corresponden a acciones tomadas en una sucesión de momentos de tiempo, generalmente pueden formularse como problemas de optimización sin restricciones.
Supóngase que se controla la posición de un objeto por una serie de fuerzas de control correctoras. El error de posición (distancia a la posición deseada) está controlado por la ecuación:

$$x_{k+1} = x_k + u_k$$

donde x_k es el error en el tiempo k y u_k es la fuerza efectiva aplicada al tiempo k. El valor de x_0 está dado. La sucesión $u_0,\ u_1, \ldots,\ u_n$ deberá seleccionarse para minimizar el objetivo

$$J = \sum_{k=0}^{n} [x_k^2 + u_k^2]$$

esto representa un compromiso entre el deseo de igualar x_k a cero y el reconocimiento de que la acción de control u_k es costosa. El problema se puede convertir a un problema sin restricciones eliminando las variables $x_k,\ k = 1,\ 2, \ldots,\ n$ del objetivo. Se observa que

$$x_k = x_0 + u_0 + u_1 + ... + u_{k-1}$$

Así pues, el objetivo se puede expresar como

$$J = \sum_{k=0}^{n} [(x_0 + u_0 + ... + u_{k-1})^2 + u_k^2]$$

que es una función cuadrática en las incógnitas u_k y estructuralmente posee la misma forma que el ejemplo anterior.

2.2.1 Condiciones de segundo orden

La proposición anterior se basa en una aproximación de primer orden en las proximidades del punto mínimo relativo. Se pueden conseguir condiciones adicionales teniendo en cuenta aproximaciones de orden superior.

Proposicion **2** *(Condiciones necesarias de segundo orden). Sea Ω un subconjunto de E^n y $f \in C^2$, una función en Ω. Si x^* es un punto mínimo relativo de f en Ω, entonces para cualquier $d \in E^n$ que sea una dirección factible en x^*, resulta*

- $\nabla f(x^*)d \geq 0$.
- *si* $\nabla f(x^*)d = 0,$ *entonces* $d^T \nabla^2 f(x^*)d \geq 0$.

Nuevamente es de interés el caso en que el punto de minimización es interior a Ω como en el caso sin restricciones obteniéndose así el siguiente resultado:

Proposicion **3** *(Condiciones necesarias de segundo orden: caso sin restricciones). Sea x^* un punto interior del conjunto Ω y supóngase que x^* es un punto mínimo relativo de f en Ω, $f \in C^2$.*
Entonces:

$$\nabla f(x^*) = 0 \tag{2.4}$$

$$\text{para toda } d,\ d^T \nabla^2 f(x^*)d \geq 0 \tag{2.5}$$

La condición 2.5 equivale afirmar que la matriz $\nabla^2 f$ es semidefinida positiva. La estructura de esta matriz es la determinante primaria de la tasa de convergencia de algoritmos diseñados para minimizar la función f.

2.2.2 Condiciones suficientes para un mínimo relativo

Las siguientes condiciones son sólo aplicables a problemas sin restricciones o cuando el punto mínimo es interior a la región factible.

PROPOSICION **4** *(condiciones suficientes de segundo orden: caso sin restricciones). Sea $f \in C^2$ una función definida en una región, en la que el punto x^* es un punto interior. Además, supóngase que*

- $\nabla f(x^*) = 0$
- $\nabla^2 f$ *es semidefinida positiva*

Entonces, x^* es un punto mínimo relativo estricto de f.

2.2.3 Funciones convexas y cóncavas

DEFINICIÓN **6** *Una función f definida en un conjunto convexo Ω es convexa si, para toda $x_1,\ x_2 \in \Omega$ y toda $\alpha,\ 0 \leq \alpha \leq 1$, se cumple $f(\alpha x_1 + (1-\alpha)x_2) \leq \alpha f(x_1) + (1-\alpha)f(x_2)$ Si, para toda $\alpha,\ 1 > \alpha > 0$ y $x_1 \neq x_2$, se cumple $\alpha f(x_1) + (1-\alpha)f(x_2) > f(\alpha x_1 + (1-\alpha)x_2)$ entonces se dice que f es estrictamente convexa.*

DEFINICIÓN **7** *Una función g definida en un conjunto convexo Ω se dice que es cóncava si $-g$ es convexa. La función g es estrictamente cóncava si $-g$ es estrictamente convexa.*

2.2.4 Combinación de funciones convexas

PROPOSICION **5** *Sean f_1 y f_2 funciones convexas en el conjunto convexo Ω. Entonces, la función $f_1 + f_2$ es convexa en Ω.*

PROPOSICION **6** *Sea f una función convexa en el conjunto convexo Ω. Entonces, la función af es convexa para cualquier $a \geq 0$.*

PROPOSICION **7** *Sea f una función convexa en un conjunto convexo Ω. El conjunto $\Gamma_c = [x :\ x \in \Omega,\ f(x) \leq c]$ es convexo para todo numero real c.*

2.2.5 Propiedades de las funciones convexas diferenciables

PROPOSICION **8** *Sea $f \in C^1$, entonces, f es convexa en un conjunto convexo Ω si, y sólo si $f(y) \geq f(x) + \nabla f(x)(y-x) \quad \forall x,\ y \in \Omega$.*

PROPOSICION **9** *Sea $f \in C^2$, entonces, f es convexa en un conjunto convexo Ω que contenga un punto interior si, y solo si, la matriz hessiana F de f es semidefinida positiva en todo Ω.*

2.2.6 Minimización y maximización de funciones convexas

TEOREMA **6** *Sea f una función convexa definida en el conjunto convexo Ω. Entonces, el conjunto Γ donde f alcanza su mínimo es convexo y cualquier mínimo relativo de f es un mínimo global.*

El siguiente teorema establece que si f es continuamente diferenciable y convexa, entonces las condiciones necesarias de primer orden son necesarias y suficientes para un punto mínimo global.

TEOREMA **7** *Sea $f \in C^1$, convexa en el conjunto convexo Ω. Si hay un punto $x^* \in \Omega$ tal que, $\forall y \in \Omega,\ \nabla f(x^*)(y - x^*) \geq 0$, entonces x^* es un punto mínimo global de f en Ω.*

2.2.7 Convergencia global de algoritmos de descenso

17 *Algoritmos*

Un algoritmo se considera como una transformación que convierte puntos de un espacio en otros puntos del mismo espacio. Dado un punto x en un espacio X, la salida de un algoritmo aplicado a x es un punto del espacio X. Operando de manera iterativa se aplica el algoritmo a los nuevos puntos en forma sucesiva, produciendo una sucesión de puntos.

DEFINICIÓN **8** *Un algoritmo A es una transformación definida en un espacio X que asigna a todo punto $x \in X$ un subconjunto de X.*

El aspecto más importante de esta definición es que es una transformación punto conjunto de X. Un algoritmo A genera una sucesión de puntos como sigue: dado $x_k \in X$, el algoritmo produce $A(x_k)$ que es un subconjunto de X. De este subconjunto se selecciona un elemento arbitrario x_{k+1}. Debido a esto, en general no es posible predecir la sucesión que generará el algoritmo con tan sólo el punto inicial x_0. La finalidad de esta incertidumbre es reflejar el grado de incertidumbre que se puede obtener en la aplicación práctica de un algoritmo particular, por ejemplo, por las diferencias entre el poder de cálculo de computadoras utilizadas.

18 *Descenso*

La idea básica de una función descendente, definida a continuación, es que para puntos que están fuera del conjunto solución Γ, un solo paso del algoritmo produce un decremento en el valor de la función descendente.

DEFINICIÓN **9** *Sea $\Gamma \subset X$ un conjunto solución de 2.3 dado y sea A un algoritmo en X. Una función continua Z con valores reales en X es una función descendente para Γ y A si satisface:*

- *si $x \notin \Gamma$ e $y \in A(x)$, entonces $Z(x) < Z(y)$*
- *si $x \in \Gamma$ e $y \in A(x)$, entonces $Z(x) \leq Z(y)$*

Dado el problema

$$\begin{aligned} \textit{minimizar} \quad & f(x) \\ \textit{sujeto a} \quad & x \in \Omega \end{aligned}$$

un planteamiento natural es hacer que Γ sea el conjunto de los puntos de minimización y definir un algoritmo A en Ω de forma que f decrezca en cada paso y sirva de función descendente. éste es el planteamiento en la mayoría de los casos.

2.2.8 Transformaciones Cerradas

La propiedad de que los algoritmos sean cerrados es una generalización de las transformaciones punto a conjunto de continuidad para transformaciones punto a punto.

Definición **10** *Una transformación punto a conjunto A de X a Y se dice que es cerrada en $x \in X$ si las hipótesis*

- $x_k \to x,\ x_k \in X$
- $y_k \to y,\ y_k \in A(x_k)$

implican

- $y \in A(x)$

La transformación punto a conjunto A se dice que es cerrada en X si es cerrada en cada punto de X.

2.2.9 Teorema de la convergencia global

Supóngase la siguiente situación. Se posee un conjunto solución Γ y puntos generados por el algoritmo $x_{k+1} \in A(x_k)$ donde cada punto nuevo siempre hace decrecer estrictamente la función descendente Z, al menos que se alcance el conjunto solución Γ. Entonces, en condiciones apropiadas, el resultado es que la sucesión converge al conjunto solución. El teorema de la convergencia global establece las condiciones técnicas para las cuales se garantiza la convergencia.

Teorema **8** *(Teorema de la convergencia global) Sea A un algoritmo en X y supóngase que dada x_0, se genera la sucesión $\{x_k\}_{k=0}^{\infty}$ que cumple $x_{k+1} \in A(x_k)$ dado un conjunto solución $\Gamma \subset X$, supóngase que:*

- *todos los puntos x_k están contenidos en un conjunto compacto $S \subset X$*
- *existe una función continua Z en X tal que:*
  - *si $x \notin \Gamma$, entonces $Z(x) > Z(y) \forall y \in A(x)$*
  - *si $x \in \Gamma$, entonces $Z(x) \geq Z(y) \forall y \in A(x)$*
- *la transformación A es cerrada en puntos que están fuera de Γ*

Entonces, el límite de cualquier subsucesión convergente de $\{x_k\}$ es una solución.

Corolario **7** *Si por las condiciones del teorema de la convergencia global, Γ está formado por un solo punto $\bar{x}$, entonces la sucesión $\{x_k\}$ converge a $\bar{x}$.*

2.2.10 Rapidez de convergencia

La rapidez de convergencia es un tema importante; existe una rica y simple teoría para predecir la efectividad relativa de una amplia clase de algoritmos.

19 *Orden de convergencia*

Sea la sucesión $\{r_k\}$ convergente al límite r^*. Se define el orden de convergencia de $\{r_k\}$ como el máximo de los números no negativos p que satisfagan:

$$0 \leq \lim_{k\to\infty} \frac{|r_{k+1} - r^*|}{|r_k - r^*|^p} < \infty$$

en cierto sentido, los valores grandes del orden p implican una convergencia más rápida, pues se reduce la distancia al límite r^*, al menos en la cola, con la p-ésima potencia en un sólo paso. Entonces si existe:

$$\beta = \lim_{k\to\infty} \frac{|r_{k+1} - r^*|}{|r_k - r^*|^p}$$

asintóticamente resulta:

$$\beta\,|r_k - r^*|^p = |r_{k+1} - r^*|$$

Como ejemplo la sucesión $r_k = a^k$, con $0 < a < 1$ converge a cero con orden unitario, pues $\frac{r_{k+1}}{r_k} = a$.

20 *Convergencia lineal*

Definición **11** *Si el limite anterior converge a un β, $\beta < 1$ se dice que la sucesión converge linealmente a r^* con tasa de convergencia β.*

Se puede decir que una sucesión linealmente convergente, con tasa de convergencia β, tiene una cola que converge, al menos tan rápido como la sucesión geométrica $c\beta^k$ para alguna constante c.

2.3 Condiciones para la minimización con restricciones

Los problemas que se tratan son los del tipo general de programación no lineal del la forma

$$\begin{aligned} \textit{minimizar} \quad & f(x) \\ \textit{sujeto a} \quad & h_1(x) = 0;\ g_1(x) \leq 0 \\ & h_2(x) = 0;\ g_2(x) \leq 0 \\ & \dots \\ & h_m(x) = 0;\ g_p(x) \leq 0 \\ & x \in \Omega \subset E^n \end{aligned}$$

donde $m \leq n$ y las funciones $f,\ h_i\ i = 1, 2, ..., m$ y $g_j\ j = 1, 2, ..., p$ son continuas y en general, se supone que tienen segundas derivadas parciales continuas. Por simplicidad, se introducen las funciones con valores vectoriales $h = (h_1, h_2, ..., h_m)$ y $g = (g_1, g_2, ..., g_p)$ y se vuelve a escribir como

$$\begin{aligned} \textit{minimizar} \quad & f(x) \\ \textit{sujeto a} \quad & h(x) = 0,\ g(x) \leq 0 \\ & x \in \Omega \end{aligned}$$

Las restricciones $h(x) = 0$ y $g(x) \leq 0$ se denominan restricciones funcionales, en tanto que la restricción $x \in \Omega$ es una restricción de conjunto. Un punto x que cumple todas las restricciones funcionales se denomina factible.
Una restricción de desigualdad $g_i \leq 0$ se dice que es activa en un punto factible x si $g_i(x) = 0$; en caso contrario se denomina inactiva.
Las restricciones activas en un punto factible x restringen el dominio de factibilidad en las proximidades del mismo, mientras que las otras restricciones, las inactivas, no ejercen influencia en las proximidades de x.

2.3.1 El plano tangente

Un conjunto de restricciones de igualdad en E^n

$$\begin{aligned} & h_1(x) = 0 \\ & h_2(x) = 0 \\ & \dots \\ & h_n(x) = 0 \end{aligned}$$

define un subconjunto de E^n, que se considera mejor tratándolo como una hipersuperficie. Si las restricciones son regulares en todas partes, es decir, que los vectores gradientes a las restricciones son linealmente independientes $\forall x$, esta hipersuperficie tiene dimensión $n - m$.
Si las funciones h_i pertenecen a C^1 la superficie definida por ellas se denomina uniforme.
Asociado a un punto de una superficie uniforme está el plano tangente en ese punto.
Una curva en una superficie S es una familia de puntos $x(t) \in S$ continuamente parametrizados por t para $a \leq t \leq b$. La curva es diferenciable si existe x' y es doblemente diferenciable si existe x''.
Considerense todas las curvas diferenciables en S que pasan por uno punto x^*. El plano tangente en x^* se define como el conjunto de las derivadas en x^* de todas estas curvas diferenciables. El plano tangente es un subespacio de E^n.
Un punto x^* que satisfaga la restricción $h(x^*) = 0$ se denomina punto regular de la restricción si los vectores gradientes $\nabla h_1(x^*),\ \nabla h_2(x^*), \dots,\ \nabla h_m(x^*)$ son linealmente independientes.

TEOREMA **9** *En un punto regular x^* de la superficie S definida por $h(x) = 0$, el plano tangente es igual a $M = \{y : \nabla h(x^*)y = 0\}$*

2.3.2 Condiciones necesarias de primer orden

LEMA **2** *Sea x^* un punto regular de las restricciones $h(x) = 0$ y un punto extremo local (mínimo o máximo) de f sujeto a estas restricciones. Entonces todo los $y \in E^n$ que cumplan $\nabla h(x^*)y = 0$ debe cumplir también $\nabla f(x^*)y = 0$*

El lema anterior expresa que $\nabla f(x^*)$ es ortogonal al plano tangente. A continuación, se concluye que esto implica que $\nabla f(x^*)$ es una combinación lineal de los gradientes de h en x^*, una relación que da lugar a la introducción de los multiplicadores de Lagrange.

TEOREMA **10** *Sea x^* un punto extremo local de f sujeto a las restricciones $h(x) = 0$. Supóngase, además, que x^* es un punto regular de estas restricciones. Entonces, existe un $\lambda \in E^n$ tal que $\nabla f(x^*) + \lambda^T \nabla h(x^*) = 0$*

2.3.3 Condiciones de segundo orden

Sea x^* un mínimo local de f, sujeto a $h(x) = 0$, un punto regular de estas restricciones. Entonces existe un $\lambda \in E^n$ tal que $\nabla f(x^*) + \lambda^T \nabla h(x^*) = 0$. Si se representa por M el planto tangente $M = \{y : \nabla h(x^*)y = 0\}$, entonces la matriz $L(x^*) = F(x^*) + \lambda^T H(x^*)$ es semidefinida positiva en M, esto es, $y^T L(x*)y \geq 0 \forall y \in M$

21 *Condiciones de suficiencia de segundo orden*

Supóngase que hay un punto x^* que satisface $h(x^*) = 0$ y un $\lambda \in E^n$ tal que $\nabla f(x^*) + \lambda^T \nabla h(x^*) = 0$
Supóngase también que la matriz $L(x^*) = F(x^*) + \lambda^T H(x^*)$ es definida positiva en $M = \{y : \nabla h(x^*)y = 0\}$. Entonces, x^* es un mínimo local estricto de f sujeto a $h(x) = 0$.

2.3.4 Valores propios en el subespacio tangente

Dado cualquier vector $y \in M$, el vector Ly esta en E^n, pero no necesariamente en M. Se proyecta Ly ortogonalmente de vuelta a M, y se dice que el resultado es la restricción de L a M operando en y. Así, se obtiene una transformación lineal de M a M. Sin embargo, la transformación se determina de manera algo implícita, pues no se dispone de una representación matricial explícita. Un vector $y \in M$ es un vector propio de L_M si existe un número real λ tal que $L_M y = \lambda y$; el λ correspondiente es un valor propio de L_M. Esto coincide con la definición estándar. En función de L, se observa que y es un vector propio de L_M si Ly se puede expresar como la suma de λy y un vector ortogonal a M. Para obtener una representación matricial para L_M es necesario introducir una base en el subespacio M. Por sencillez, es mejor introducir una base ortonormal, por ejemplo, $e_1,\ e_2, ...,\ e_{n-m}$. Definase la matriz E como la matriz de $n \times\ (n-m)$ cuyas columnas constan de los vectores e_i. Entonces, cualquier vector y en M se puede expresar como $y = Ez$ para algún $z \in E^{n-m}$ y por supuesto, LEz representa la acción de L sobre dicho vector. Para proyectar

este resultado de vuelta a M y expresarlo en función de la base $e_1,\ e_2, ...,\ e_{n-m}$, no hay más que multiplicar por E^T. Así, $E^T LEz$ es el vector cuyas componentes dan la representación en función de la base; y, en correspondencia, la matriz de $(n-m) \times\ (n-m)$ $E^T LE$ es la representación matricial de L restringida a M.
Los valores propios de L restringidos a M se pueden hallar determinando los valores propios de $E^T LE$. Estos valores propios son independientes de la base ortonormal determinada E.

2.3.5 Sensibilidad

Sea el problema

$$\begin{aligned} minimizar \quad & f(x) \\ sujeto\ a \quad & h(x) = 0 \end{aligned}$$

que tiene una solución en el punto x^*, que es un punto regular de las restricciones. Sea el correspondiente vector multiplicador de Lagrange. Ahora, considérese la familia de problemas

$$\begin{aligned} minimizar \quad & f(x) \\ sujeto\ a \quad & h(x) = c \end{aligned}$$

donde $c \in E^m$. Para un margen suficientemente pequeño de c cerca del vector cero, el problema tendrá un punto solución $x(c)$ cerca de $x(c) \equiv x^*$. Para cada una de estas soluciones hay un valor correspondiente $f(x(c))$ que se puede considerar como una función de c, el lado derecho de las restricciones. Las componentes del gradiente de esta función se pueden interpretar como la tasa incremental de cambio en valor por unidad de cambio en los requerimientos de restricción. Así, como los precios incrementales de los requerimientos de restricción medidos en unidades del objetivo.

Teorema **11** *(Teorema de sensibilidad). Sea $f,\ h \in C^2$ y considérese la familia de problemas*

$$\begin{aligned} minimizar \quad & f(x) \\ sujeto\ a \quad & h(x) = c \end{aligned}$$

Supóngase que para $c = 0$ hay una solución local x^* que es un punto regular y que, junto con su vector multiplicador de Lagrange asociado, satisface las condiciones de suficiencia de segundo orden para un mínimo local estricto. Entonces, para toda $c \in E^m$ en una región que contenga 0 existe una $x(c)$, que depende continuamente de c, tal que $x(0) = x^*$, y tal que $x(c)$ es un mínimo local. Además $\nabla_c f(x(c))|_{c=0} = -\lambda^T$

2.3.6 Restricciones de desigualdad

22 *Condiciones necesarias de primer orden*

DEFINICIÓN **12** *Sea x^* un punto que satisfaga las restricciones*

$$h(x*) = 0$$
$$g(x*) \leq 0$$

$$(2.6)$$

y sea J el conjunto de índices para el cual $g_j(x^*) = 0$. Entonces, se dice que x^* es un punto regular de las restricciones 2.6 si los vectores gradientes $\nabla h_i(x^*),\ \nabla g_j(x^*),\ 1 \leq i \leq m,\ j \in J$ son linealmente independientes.

DEFINICIÓN **13** *(Condiciones de Kuhn-Tucker). Sea x^* un punto mínimo relativo para el problema*

$$\begin{aligned} \textit{minimizar} \quad & f(x) \\ \textit{sujeto a} \quad & h(x) = 0,\ g(x) \leq 0 \end{aligned}$$

y supóngase que x^* es un punto regular para las restricciones. Entonces existen un vector $\lambda \in E^m$ y un vector $\mu \in E^p,\ \mu \geq 0$ tal que

$$\nabla f(x^*) + \lambda^T \nabla h(x^*) + \mu^T \nabla g(x^*) = 0 \tag{2.7}$$
$$\mu^T g(x^*) = 0 \tag{2.8}$$

23 *Condiciones de segundo orden*

Supóngase las funciones $f,\ g,\ h \in C^2$ y que x^* es un punto regular de las restricciones 2.6. Si x^* es un punto mínimo relativo para el problema, entonces existe $\lambda \in E^m$, $\mu \in E^p$ tales que se cumplen 2.7 y 2.8 y tales que $L(x^*) = F(x^*) + \lambda^T H(x^*) + \mu^T G(x^*)$ es semidifinida positiva en el subespacio tangente de las restricciones activas en x^*.

24 *Condiciones de suficiencia de segundo orden*

Sea $f,\ g,\ h \in C^2$. Las restricciones suficientes para que un punto x^* que satisfaga 2.6 sea un punto mínimo relativo estricto del problema es que exista $\lambda \in E^m$, $\mu \in E^p$, tal que $\nabla f(x^*) + \lambda^T \nabla h(x^*) + \mu^T \nabla g(x^*) = 0$ y la matriz hessiana $L(x^*) = F(x^*) + \lambda^T H(x^*) + \mu^T G(x^*)$ sea definida positiva en el subespacio $M' = \{y : \nabla h(x^*)y = 0,\ \nabla g_j(x^*)y = 0,\ \vee j \in J\}$ donde $J = \{j :\ \ g_j(x^*) = 0,\ \ j > 0\}$

25 *Sensibilidad*

TEOREMA **12** *Sean $f,\ g,\ h \in C^2$ y considérese la familia de problemas*

$$\begin{aligned} \textit{minimizar} \quad & f(x) \\ \textit{sujeto a} \quad & h(x) = c \\ & g(x) \leq d \end{aligned}$$

$$(2.9)$$

Supóngase que para $c = 0,\ d = 0$*, hay una solución local* x^**, que es un punto regular y que, junto con los multiplicadores de Lagrange asociados,* $\lambda,\ \mu \geq 0$*, satisface las condiciones de suficiencia de segundo orden para un mínimo local estricto. Además, supóngase que ninguna restricción de desigualdad activa es degenerada. Entonces, para toda* $(c, d) \in E^{m+p}$ *en una región que contiene a* $(0, 0)$*, hay una solución* $x(c, d)$ *que depende continuamente de* (c, d)*, tal que* $x(0, 0) = x^*$*, y tal que* $x(c, d)$ *es un punto mínimo relativo de 2.9. Además,*

$$\nabla_c f(x(c,d))|_{0,0} = -\lambda^T$$
$$\nabla_d f(x(c,d))|_{0,0} = -\mu^T$$

2.4 Métodos básicos de descenso

Se describen técnicas básicas para la resolución iterativa de problemas sin restricciones. Existe una estructura fundamental para muchos algoritmos descendentes. Se comienza en un punto inicial, se determina de acuerdo a una regla fija una dirección de movimiento y después se sigue esa dirección hacia un mínimo de la función objetivo de esa recta. En el nuevo punto se determina la nueva dirección y se repite el proceso. La diferencia entre cada algoritmo radica en la regla de elección de la siguiente dirección de búsqueda.
El proceso de determinar el mínimo en una recta dada se denomina búsqueda lineal que no es otra cosa que un procedimiento de minimización unidimensional. Estos algoritmos son la base de los métodos de minimización en programación no lineal.
Se describen métodos para la resolución del problema de búsqueda lineal

2.4.1 Búsqueda de fibonacci

El método determina el valor mínimo de una función en un intervalo cerrado. La función debe cumplir como requisito ser unimodal, o sea, que posea en el intervalo un solo mínimo relativo. El punto mínimo se estimará midiendo el valor de la función en cierto numero de puntos. Ahora el problema se centra en escoger una cantidad dada de puntos de medición sucesivos que, sin conocer explícitamente la función, se pueda determinar la región más pequeña posible de incertidumbre en la cual deba estar el mínimo.
En este método, los anchos de incertidumbre varían como sigue:

- $d_1 = c_2 - c_1$, anchura inicial de incertidumbre
- d_k, anchura de incertidumbre después de k mediciones.

entonces, después de N mediciones resulta:

$$d_k = \left(\frac{F_{N-k+1}}{F_N}\right) d_1$$

donde los enteros Fk son elementos de la sucesión de Fibonacci generada por la relación de recurrencia

$$F_N = F_{N-1} + F_{N-2},\ F_0 = F_1 = 1$$

obteniéndose como resultado $1, 1, 2, 3, 5, 8, 13 \ldots$ El procedimiento para reducir el ancho es el siguiente:
Las dos primeras mediciones se hacen simétricas y a una distancia $\left(\frac{F_{N-1}}{F_N}\right) d_1$ de lo extremos de los intervalos iniciales; según cuál de ellos tenga menor valor se determinará un intervalo de incertidumbre de ancho $d_2 = \left(\frac{F_{N-1}}{F_N}\right) d_1$. El tercer punto de medición se coloca en este intervalo y simétrico respecto a la medición hecha en el intervalo. Esto proporciona un nuevo intervalo de incertidumbre $d_3 = \left(\frac{F_{N-2}}{F_N}\right) d_1$. Si se permitiese realizar infinitas medidas, se puede observar que la sucesión de Fibonacci converge linealmente al mínimo general de la función con razón de convergencia 0.618.

2.4.2 Búsqueda lineal mediante ajuste de curvas

En la mayoría de los problemas se puede suponer que, además de ser unimodal, la función posee cierta suavidad. Por lo tanto se podrían diseñar técnicas más eficientes que se aprovechen de esta suavidad, como son las técnicas por ajuste de curvas. En las mismas se determina una curva suave que pasa por los puntos medidos para determinar una estimación del mínimo. Se pueden diseñar diversas técnicas dependiendo si se pueden medir o no las derivadas de la función y su valor.

26 *Método de Newton*

Supóngase que se va a minimizar una función de una variable y supóngase que en un punto donde se hace una medición, se pueden calcular el valor funcional, su derivada primera y segunda. Entonces se puede construir una función cuadrática que concuerde con estos datos en el punto de medida. Entonces se podría calcular el siguiente punto de medida como el mínimo de esta función cuadrática, dando como resultado:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

Sea g una función con segunda derivada continua y x^* que satisface $g(x^*) = 0$, $g'(x^*) \neq 0$. Entonces, si x_0 está suficientemente cerca de x^*, la sucesión $\{x_k\}_{k=0}^{\infty}$, generada por el método de Newton, converge a x^* con un orden de convergencia de al menos 2.
Pueden generarse variantes de este método utilizando menos información del punto pero utilizando mas puntos de medida, por ejemplo, información del valor funcional y la derivada en un punto, y el valor de la derivada en un punto de medida anterior. También pueden realizarse ajustes cúbicos con datos de un punto de medida o más.

27 *Método de mayor pendiente*

El método de descenso de mayor pendiente está definido por el algoritmo iterativo

$$x_{k+1} = x_k - \alpha_k g_k$$

donde α_k es un escalar no negativo que minimiza $f(x_k - \alpha g_k)$ y $g_k = \nabla f(x_k)^T$. En palabras, a partir de un punto se busca en la dirección del gradiente negativo hasta un punto mínimo de esa recta. Ese punto mínimo se toma como el siguiente punto de la sucesión.

Si se define el conjunto solución como los puntos x, donde $\nabla f(x) = 0$. Entonces, $Z(x) = f(x)$ es una función descendente para A, pues para $\nabla f(x) \neq 0$

$$\min_{0 \leq \alpha < \infty} f(x - \alpha g(x)) < f(x)$$

Así, por el teorema de la convergencia global, si la sucesión $\{x_k\}$ está acotada, tendrá puntos límite y cada uno de ellos es una solución.

28 *Caso Cuadrático*

TEOREMA **13** *(Descenso de mayor pendiente, caso cuadrático, $f(x) = \frac{1}{2}x^T Qx - x^T b$ con Q definida positiva). Para cualquier $x_0 \in E^n$, el método de mayor pendiente converge a un único punto mínimo x^* de f si. Es más, con $E(x) = \frac{1}{2}(x - x^*)^T Q(x - x^*)$, en todo punto k se cumple*

$$E(x_{k+1}) \leq \left(\frac{A - a}{A + a}\right)^2 E(x_k)$$

siendo A y a los valores propios mayor y menor respectivamente de Q.

29 *Caso no cuadrático*

En este caso puede realizarse utilizando el resultado anterior, obteniéndose como resultado que la razón de convergencia utilizando el método de descenso de mayor pendiente no es mayor que $\left(\frac{A-a}{A+a}\right)^2$ siendo A y a los valores propios mayor y menor respectivamente de la matriz hessiana de f.
Habiéndose adentrado en las cuestiones y conceptos básicos de optimización, a continuación se presenta un algoritmo de optimización que opera sobre funciones de costo no lineales en espacios sin restricciones. El mismo es uno de los más populares de los últimos tiempos debido a su gran facilidad de implementación y fiabilidad de sus resultados, "Particle Swarm Optimization" (PSO).

3 Particle Swarm Optimization (PSO)

3.1 Introducción

PSO surge como un algoritmo de optimización de funciones multidimensionales, no lineales, no convexas y continuas derivado de conceptos naturales simples. éste fue propuesto por James Kennedy y Russell Eberhart en 1995 [12]. PSO es muy simple en su implementación, introduciendo pocos parámetros y requiriendo sólo operadores matemáticos simples, por lo tanto resulta muy atractivo desde el punto de vista computacional ya que requiere de poca memoria y posee gran velocidad. Posteriormente fue presentada la versión para funciones discretas [13].

3.2 Motivación

El método fue desarrollado simulando un comportamiento social simple, inspirado en el comportamiento visto en bandadas de pájaros cuando migran en forma sincrónica y en escuelas de peces que responden ante eventos externos por interacción entre peces vecinos.
El algoritmo simula un enjambre de partículas con un número relativamente alto de agentes que mediante un conjunto de pocas reglas de comportamiento simple y comunicación entre ellas, logran navegar en el espacio y alcanzar el mínimo global en el dominio determinado de la función a optimizar, evitando mínimos locales de la misma. De este modo, las partículas aprenden de su propia experiencia y de la del resto del enjambre, pudiendo así alcanzar el óptimo en el espacio de búsqueda.

3.3 Descripción

El algoritmo implementa un enjambre de partículas las cuales se inicializan en forma aleatoria en el espacio de búsqueda y se rigen mediante un comportamiento simple, que depende de su experiencia y la del enjambre.
La experiencia previa juega un rol importante en la decisión de la partícula respecto a dónde dirigirse, en particular, las zonas donde se obtuvo gran performance resultan atractivas, ya que cabe la posibilidad de encontrar buenos resultados en sus proximidades. Debido a esto, una partícula del enjambre puede aprender de su experiencia previa con el fin de encontrar mejores resultados.
Formalmente, la posibilidad de encontrar mejores resultados en las proximidades de una región donde se obtuvo un buen desempeño está justificada por la continuidad

del espacio y probable continuidad del costo en esa región. La continuidad provoca que en lugares muy cercanos se obtengan costos similares pudiendo encontrarse lugares del espacio con costos menores a los alguna vez alcanzados.
Matemáticamente, la velocidad tomada por alguna partícula del enjambre que se corresponde con un comportamiento del tipo mencionado puede ser de la forma:

$$\overrightarrow{v_i^{k+1}} = w\ \overrightarrow{v_i^k} + c\left(\overrightarrow{x_{p_i}^k} - \overrightarrow{x_i^k}\right)$$

donde $\overrightarrow{x_{p_i}^k}$ es la posición de la i-ésima partícula en la cual la misma percibió el menor costo, es decir, la mejor posición alguna vez alcanzada por la partícula hasta el k-ésimo paso. El parámetro w es denominado inercia.
Un inconveniente con partículas que se rigen por un comportamiento de este tipo es que describen trayectorias rectilíneas, es decir, nunca cambian su dirección aunque si su sentido. Si una partícula se encuentra en "descenso" la mejor posición alcanzada por la misma es siempre la actual, por lo que $\overrightarrow{v_i^{k+1}} = w\ \overrightarrow{v_i^k}$. En cambio, si en determinado momento el costo percibido aumenta, la velocidad tendrá una componente en la dirección de la mejor posición alcanzada

$$c\left(\overrightarrow{x_{p_i}^k} - \overrightarrow{x_i^k}\right) = -c\left|\left|\overrightarrow{x_{p_i}^k} - \overrightarrow{x_i^k}\right|\right| \frac{\overrightarrow{v_i^k}}{\left|\left|\overrightarrow{v_i^k}\right|\right|}$$

la cual es opuesta a la que poseía anteriormente (ver figura 3.1). Es fácil observar que

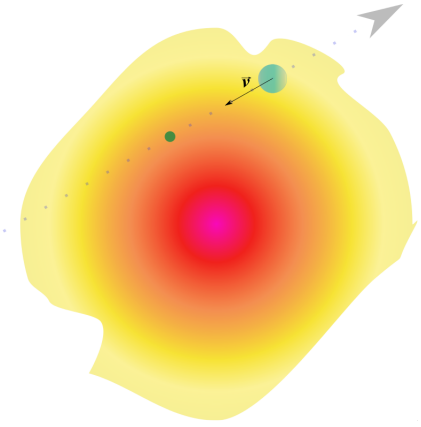
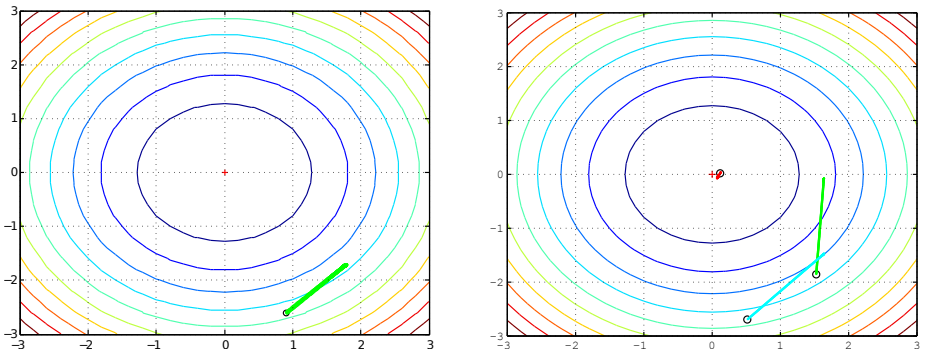


Figura 3.1: Partícula del enjambre (celeste) navegando en el espacio de búsqueda. La partícula describe una trayectoria rectilínea en una dirección (en gris y punteado) a medida que encuentra valores cada vez más pequeños de la función de costos (el gradiente de colores cada vez más cálidos representa zonas del espacio cada vez más óptimas). Cuando el costo percibido aumenta, la partícula cambia la dirección de su movimiento y se estabiliza en el mínimo de la función restringida a la trayectoria descrita (verde). éste, es el punto de tangencia de la trayectoria con la curva de nivel más baja atravesada.

debido a los movimientos rectilíneos, cada partícula se detendrá tangente a la curva de nivel más baja que logre alcanzar. Por lo tanto ninguna partícula podrá llegar

al mínimo global de una función de costos a no ser que el mismo se encuentre en la trayectoria que describe alguna de ellas, sin importar la cantidad de partículas. En la figura 3.2 se muestran las trayectorias de dos enjambres, uno de una partícula (3.2.a) y otro de tres partículas (3.2.b) aplicados a un paraboloide. En dichas figuras se observa claramente los puntos del espacio donde se detienen las partículas y sus trayectorias rectilíneas. En la figura 3.2.b se observa que una de las partículas



(a) (b)

Figura 3.2: Minimización de un paraboloide de la forma $z = x^2 + y^2$ mediante un enjambre de partículas con el algoritmo de comportamiento individual presentado, utilizando una sola partícula (a) y tres partículas (b). Las trayectorias descritas por las partículas en ambos casos se corresponde con un movimiento rectilíneo en el cual inicialmente las partículas se mueven en una misma dirección, ya que desciende la función de costos en cada paso. Luego de llegar a la curva de nivel de más bajo valor, la dirección trata de cambiar hasta conseguirlo para luego oscilar entorno al punto más bajo encontrado.
El comportamiento muestra la independencia con el número de partículas, por lo cual no se genera ningún beneficio al incrementar la cantidad de partículas. Puede intuirse que el conocimiento de información colectiva además de la individual, en principio, favorecería a la minimización. Como se observa en el caso (b), el conocimiento de información colectiva podría haber cambiado las trayectorias de las partículas lejanas al óptimo debido a la presencia de una partícula en las proximidades del mismo.

describe una trayectoria muy próxima al óptimo. Si el resto de las partículas pudieran reconocer la proximidad de ésta al óptimo y aprender de ella podrían conseguir mejores posiciones así como aprenden de su experiencia propia.
En forma análoga al comportamiento individual, una formulación que contemple la experiencia previa del enjambre puede ser:

$$\overrightarrow{v_i^{k+1}} = w\ \overrightarrow{v_i^k} + c_1 \left(\overrightarrow{x_{p_i}^k} - \overrightarrow{x_i^k} \right) + c_2 \left(\overrightarrow{x_g^k} - \overrightarrow{x_i^k} \right)$$

donde el nuevo término $\overrightarrow{x_g^k}$ es la posición en la cual una partícula percibió el menor costo, es decir, la mejor posición alguna vez alcanzada por alguna partícula del enjambre hasta el k-ésimo paso.
Si una partícula del enjambre se encuentra en "descenso" entonces su mejor posición será la actual ($\overrightarrow{x_{p_i}^k} = \overrightarrow{x_i^k}$) y si además es la más próxima al óptimo, la mejor posición global será la mejor posición alcanzada por ésta ($\overrightarrow{x_g^k} = \overrightarrow{x_i^k}$), por lo tanto su velocidad será de la forma $\overrightarrow{v_i^{k+1}} = w\ \overrightarrow{v_i^k}$ describiendo una trayectoria rectilínea. Mientras la mejor posición alcanzada por el enjambre sea la de esta partícula, la misma se comportará como el algoritmo individual anteriormente comentado, dependiendo

únicamente de su inercia y su experiencia previa. Si en cambio, una partícula se encuentra en "descenso" pero no determina actualmente la mejor posición alcanzada a nivel global, entonces el comportamiento de la misma será, salvo la influencia de su inercia, preponderantemente colectivo.
Para obtener un conjunto de velocidades más diversa para las partículas del enjambre, puede agregarse cierta "erraticidad", por ejemplo introduciendo pesos aleatorios (u_1 y u_2 que varíen entre 0 y 1) que afecten a los términos individual y colectivo. De este modo la expresión resultante es de la forma:

$$\overrightarrow{v_i^{k+1}} = w\ \overrightarrow{v_i^k} + c_1 u_1 \left(\overrightarrow{x_{p_i}^k} - \overrightarrow{x_i^k} \right) + c_2 u_2 \left(\overrightarrow{x_g^k} - \overrightarrow{x_i^k} \right) \tag{3.1}$$

De este modo, una partícula además de aprender de su experiencia previa lo hace de la experiencia previa del resto del enjambre, pudiendo así alcanzar mejores resultados. La figura 3.3 muestra un enjambre de tres partículas con este comportamiento aplicado al problema del paraboloide. En la misma se observa que si una partícula alcanzó un muy buen resultado el resto serán atraídas a esa zona pudiendo así, en conjunto, alcanzar el óptimo de la función de costos.
De este modo, cada partícula del enjambre se ve influenciada, a nivel particular,

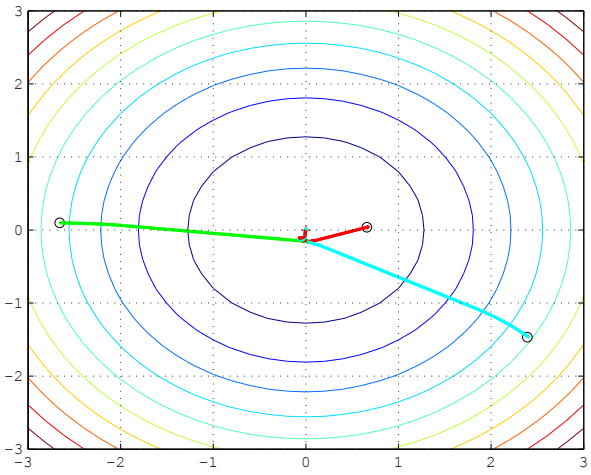
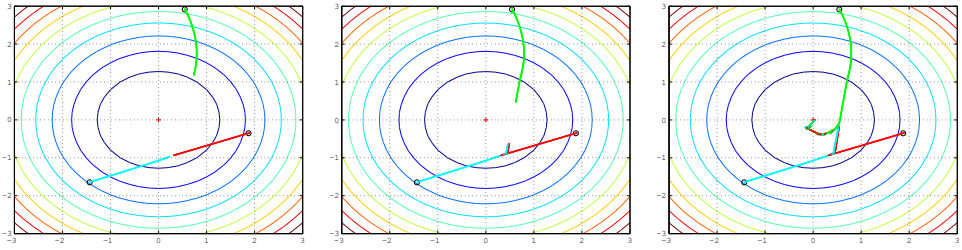


Figura 3.3: Enjambre de tres partículas implementando un comportamiento que depende de la experiencia particular y colectiva. La partícula roja parte muy próxima al óptimo e inicialmente se detiene en el punto de su trayectoria con costo más bajo, tal y como lo describe su comportamiento individual. Luego, comienza a acercarse al óptimo mediante la interacción con el resto de las partículas (verde y celeste), las cuales alcanzaron la posición de ésta gracias a su comportamiento colectivo.

por su experiencia previa y a nivel colectivo por la experiencia previa del resto del enjambre. Así, si al menos un elemento del enjambre consigue alcanzar el óptimo, el resto de las partículas del enjambre se verán influenciadas por ésta, haciendo tender a los agentes del enjambre a ese lugar con un peso dependiente de la experiencia de cada partícula y su inercia.
En la figura 3.4 se observa una secuencia donde claramente se nota la dependencia de la experiencia colectiva en el comportamiento individual, ayudando esto a alcanzar el



(a) (b) (c)

Figura 3.4: Secuencia del algoritmo de optimización con tres partículas. Inicialmente (a), las partículas verde y celeste "siguen" a la partícula roja, ya que es la que se encuentra más próxima al óptimo y con su avance continúa acercándose a éste (por lo que describe una trayectoria rectilínea). Luego (b), las partículas siguen a la partícula verde ya que, gracias a la interacción entre ellas, desvió su trayectoria en (a) encontrándose ahora más cerca del óptimo. En (c) se encuentran las tres partículas y por interacción entre ellas alcanzan el mínimo de la función de costos.
Por el beneficio obtenido gracias a las posiciones y trayectorias diversas (como en (b)) para alcanzar el óptimo, es notorio que cuanto más partículas y más diversas sus posiciones, la búsqueda resulta más eficaz

mínimo de la función de costos. Un inconveniente en los problemas de optimización es la presencia de mínimos locales, es decir, zonas donde el valor funcional resulta bastante pequeño respecto a sus proximidades pero no es el menor valor que puede alcanzarse. Este tipo de zonas suelen "atrapar" a los algoritmos de optimización, resultando en valores subóptimos. En el contexto presente, en el que la acción de un robot puede decidirse en función del valor óptimo obtenido, un valor subóptimo puede traducirse en una decisión equivocada del agente robótico, produciendo que el mismo pueda no cumplir con su objetivo.
Un ejemplo esquemático de este echo es el caso de un robot que intenta llegar a una posición determinada, la que se denomina objetivo, en un entorno que posee un solo obstáculo (ver figura 3.5). La estrategia adoptada por el robot será basada en optimización matemática.
Para decidir su acción el robot determinará el mínimo de una función de costos me-

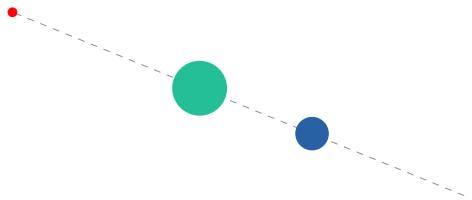


Figura 3.5: Robot (azul) recorriendo el entorno tratando de alcanzar el objetivo (rojo) intentando además evitar colisiones con el obstáculo presente (verde). Para esto, la estrategia que adoptará el mismo será determinada por la optimización de una función de costos dependiente de los objetivos planteados y la condición del robot respecto a éstos.

diante algún algoritmo de optimización. Para esto es necesario formular una función de costos dependiente de la posición del robot que sea mínima en el objetivo y, para evitar colisiones, que el costo aumente en las proximidades del obstáculo. Para esto se considera una función de costos del siguiente modo:

$$f(\overrightarrow{x_r}) = k_T||\overrightarrow{x_r} - \overrightarrow{x_T}||^2 + k_O||\overrightarrow{x_r} - \overrightarrow{x_O}||^{-2}$$

donde $\overrightarrow{x_r}$, $\overrightarrow{x_T}$ y $\overrightarrow{x_O}$ son las posiciones del robot, el objetivo y el obstáculo respectivamente. En el supuesto que el obstáculo no se encuentre sobre el objetivo ($\overrightarrow{x_T} \neq \overrightarrow{x_O}$) una estrategia por parte del robot que minimice el costo involucrará dirigirse hacia el objetivo (ya que el peso del primer término será nulo) e intentar estár lo más lejos posible del obstáculo (ya que el peso del segundo término decrece parabólicamente con la distancia al mismo y en particular es infinito sobre éste). Las constantes k_T y k_O determinan el compromiso entre alcanzar al objetivo y evitar el obstáculo. Altos valores de la primera relativos a la segunda provocarán que el robot trate de aproximarse muy rápidamente al objetivo sin importar las colisiones con el obstáculo. En cambio, altos valores de la segunda respecto a la primera producirán que el robot se aleje mucho del obstáculo dejando de lado el llegar al objetivo.
En el caso de encontrarse fijo el objetivo y el obstáculo, el problema bidimensional es de la forma (con $k_T = k_O = 1$):

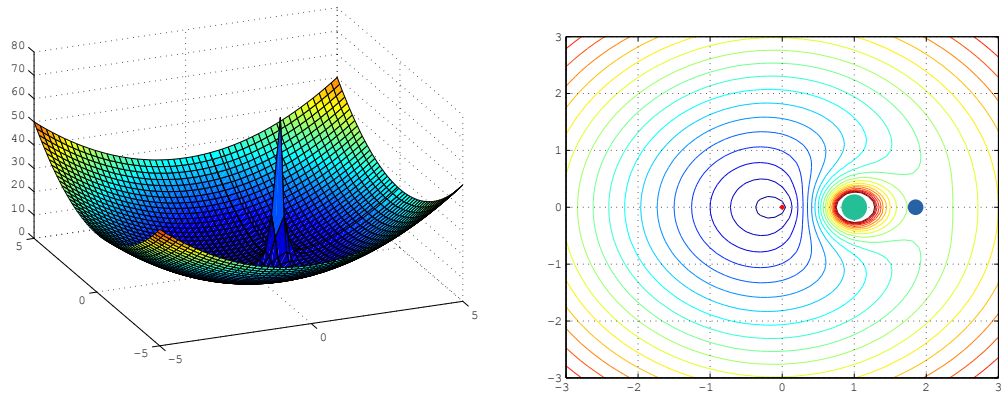
$$f(x,y) = (x - x_T)^2 + (y - y_T)^2 + \frac{1}{(x - x_O)^2 + (y - y_O)^2}$$

que por simplicidad, tomando $(x_T, y_T) = (0,0)$ y $(x_O, y_O) = (1,0)$, resulta:

$$f(x,y) = x^2 + y^2 + \frac{1}{(x-1)^2 + y^2}$$

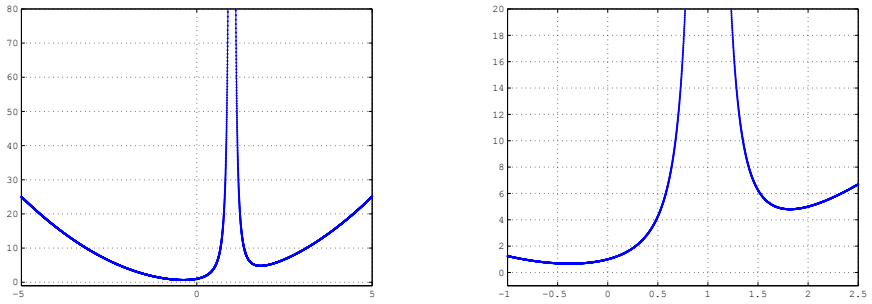
La función de costos así formulada puede observarse en la figura 3.6.a. Claramente, la misma no es convexa.
El robot, utilizando como comportamiento para optimizar la función de costos un algoritmo similar al algoritmo PSO propuesto, podría mantener su velocidad en caso de encontrarse con mejores posiciones cada vez, o ser atraído a la mejor posición previa en caso de que lo anterior no ocurra.
Suponiendo que el robot inicialmente posee velocidad en dirección al objetivo y el obstáculo simultáneamente (dirección como en la figura 3.5) éste mantendrá su velocidad ya que percibe valores cada vez menores de la función de costos, como se ilustra en la imagen 3.6.b. Esto sucederá hasta encontrarse muy próximo al obstáculo ya que el costo comenzará a crecer (ver figura 3.7) y por lo tanto el robot adquirirá velocidad opuesta a la que poseía, alejándose del obstáculo pero también del objetivo. El robot terminará atrapado en un mínimo local de la función de costos ya que alejarse del obstáculo lo perjudica, por alejarse también del objetivo, pero acercarse al objetivo implica aproximarse al obstáculo, resultando también una mala posición.

La función de costos utilizada, el paraboloide, es una función convexa, por lo que encontrar un mínimo local de la misma es encontrar su mínimo absoluto.
En caso de estar en presencia de una función no convexa puede no ser el óptimo global el alcanzado, sino uno local. Debido a esto, la influencia particular de cada agente, su inercia y la erraticidad en la velocidad juegan un papel fundamental en el comportamiento particular, ya que provocarán que el enjambre, en conjunto, esquive esta zona y forzará, en cierto modo, a las partículas atrapadas en este mínimo local a escapar del lugar.



(a) (b)

Figura 3.6: Función de costos para el caso en que el objetivo y el obstáculo se encuentren detenidos. De (a) puede observarse claramente el carácter no convexo de esta función de cotos

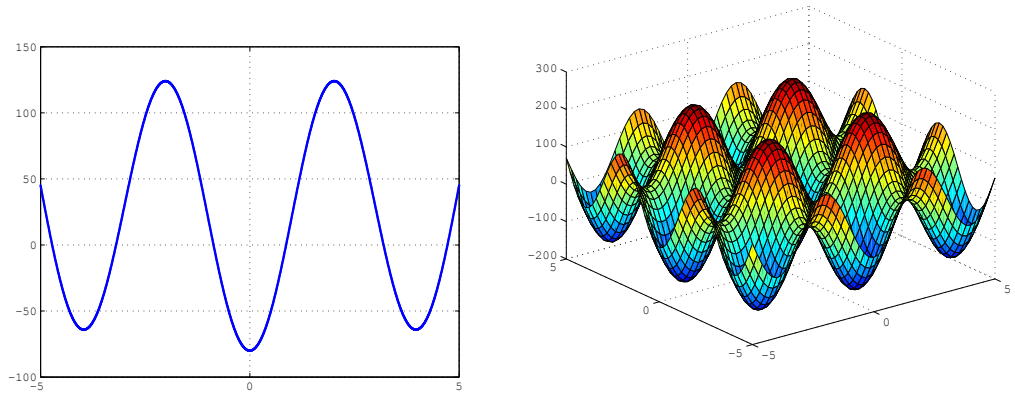


(a) (b)

Figura 3.7: Función de costos en la dirección adoptada por el robot (a) y su magnificación entorno al origen (b). El mínimo global de la misma se encuentra en $x \approx -0.3803$ que se corresponde con la posición final del robot, la misma resulta próxima al objetivo. Debido a la forma de la función de costos formulada, un incremento en el peso correspondiente a la proximidad al objetivo (k_T) ubicará al mínimo global de la función más próximo al origen. Idealmente con $k_T = +\infty$, el mínimo resultaría 0 ya que el peso de la proximidad al obstáculo resultaría despreciable.
La función de costos también presenta un mínimo local entorno al valor 1.8192 en el cual, según los pesos asignados a cada término, es un punto donde se encuentra un buen compromiso entre la proximidad al objetivo y la lejanía al obstáculo, aunque no es el mejor.

Un claro ejemplo de lo expresado anteriormente consiste en la minimización mediante un enjambre de partículas de una función de costos de Rastigin bidimensional. La misma posee una expresión funcional de la forma $z = k_1 + x^2 + y^2 - k_2(cos(\frac{\pi}{2}x) + cos(\frac{\pi}{2}y))$ que es representada gráficamente en la figura 3.8. Esta función es utilizada comúnmente como función de test de algoritmos de optimización matemática, claramente es no convexa y su mínimo global se da en $(0, 0)$.
La figura 3.9 muestra las trayectorias de un enjambre compuesto por 3 partículas en la minimización de la función de Rastrigin. Las mismas ejecutan como regla de comportamiento el algoritmo número 3.1 con $c_1 = 0$. Por este motivo, las partículas poseen un comportamiento preponderantemente global, guiándose por su inercia y respondiendo al desempeño del enjambre por sobre el desempeño particular. En dicha figura puede observarse que una de las partículas se encuentra próxima a un

mínimo local (celeste) e intenta posicionarse en el punto más cercano al mismo, las otras se ven atraídas a la posición alcanzada por la primera, ya que posee el valor funcional más bajo alcanzado por el enjambre, pese a la cercanía de alguna (verde) al mínimo global de la función. El algoritmo así implementado converge a un mínimo local.



(a) (b)

Figura 3.8: Función de Rastrigin unidimensional (a) y bidimensional (b) con los parámetros $(k_1, k_2) = (20, 100)$. La misma claramente no es convexa y posee mínimos locales en forma periódica en el espacio, con período 4 en ambas coordenadas. Su mínimo global se encuentra en $(0, 0)$.

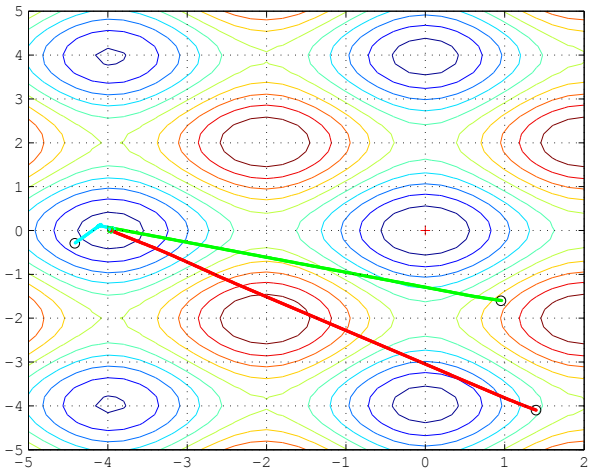


Figura 3.9: Algoritmo sólo con inercia y conducta colectiva aplicado a la función de Rastrigin. Las partículas del enjambre son atraídas a la mejor posición global alcanzada, sin importar su pasado. Debido a esto, las partículas son atrapadas en un mínimo local a causa de que una de las partículas (celeste) se encontraba próxima a este punto, ignorando el resto su cercanía al mínimo (como la partícula verde).

Dándole peso al desempeño individual ($c_1 \neq 0$), cada partícula aprende de sí misma y del resto pudiendo así evitar soluciones subóptimas.
La figura 3.10 muestra una secuencia del algoritmo así implementado. Se observa que la cercanía de una partícula al óptimo influye en el comportamiento de la misma y en el resto, evitando así los mínimos locales y alcanzando el mínimo global.

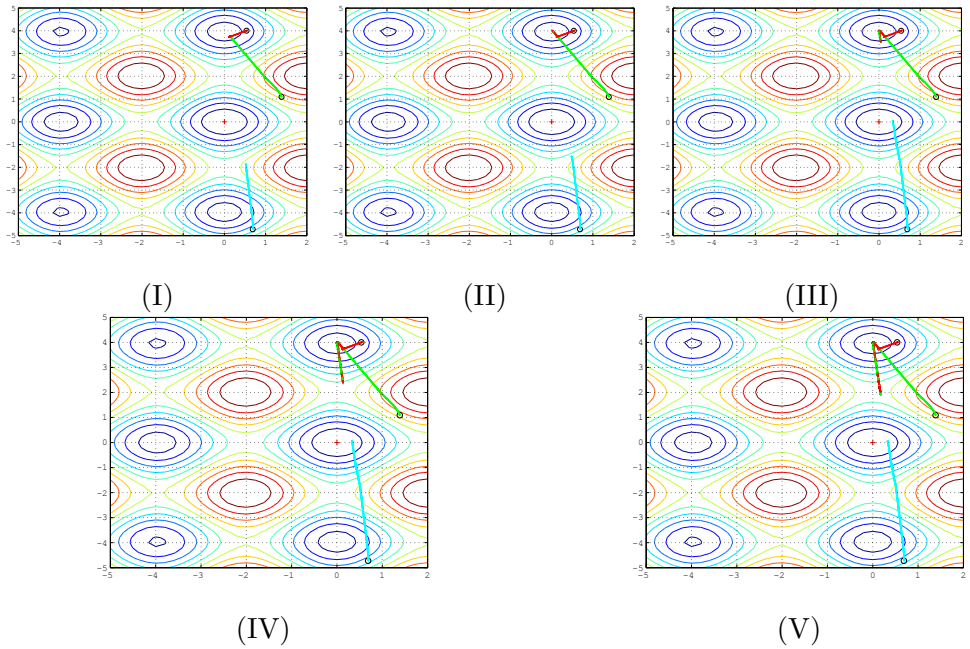


Figura 3.10: Evolución de las trayectorias de las partículas de un enjambre con 3 agentes ejecutando el comportamiento propuesto. El efecto es similar al observado en la figura 3.4 con la diferencia que en (I), (II) y (III) dos de las partículas son atraídas a un mínimo local próximo a ellas en vez de simplemente seguir trayectorias rectilíneas

3.4 Performance

La elección de la terna de parámetros (w, c_1, c_2) determina el comportamiento del enjambre, el cual puede ser muy diferente entre ternas. Una gran inercia w provocará que las partículas produzcan trayectorias de poca curvatura(sumamente condicionadas por su velocidad inicial e insensibles a su experiencia previa y a la del resto del enjambre).
Las constantes c_1 y c_2 son parámetros sociales y cognitivos que determinan la influencia particular y colectiva del comportamiento. Un alto valor del primero en comparación con el segundo provocará que las partículas determinen su comportamiento en forma cuasi-individual, mientras que en el caso contrario, el comportamiento será sumamente colectivo.

3.5 Aplicaciones

En [15] se presenta un análisis de los algoritmos heurísticos más importantes y su influencia en la investigación en "Robot Motion Planning" (RMP) desde el año 1983 hasta 2007. En ese intervalo de tiempo se ha utilizado PSO en el 2.04 % de las investigaciones en RMP (considerar que, como fue dicho anteriormente, el algoritmo PSO se creó en el año 1995).
Debido a las características abstractas pero a su vez biomiméticas a un enjambre real del algoritmo PSO, el mismo ha sido utilizado tanto para optimización matemática como objeto de inspiración de conductas propias de agentes reales.
En [16] se plantea un problema de "path planning" en un entorno desconocido el cual presenta obstáculos, tanto estáticos como dinámicos. En [23] se pretende encontrar

un objetivo en un entorno con obstáculos.
En ambos problemas ([16] y [23]) se utiliza para su resolución un grupo de robots, concibiendo a cada uno como una partícula del enjambre del algoritmo PSO.
En cambio, en [17] se pretende que un conjunto de agentes particulares adopten una conducta colectiva idéntica a la de un proceso de flocado. Para conseguir el objetivo se propone una acción de control distribuida la cual, por un lado, puede acelerar la convergencia del conjunto magnificando su amplitud y según una relación determinada y por otro lado se pretende que sea de energía mínima. Para determinar la magnitud de la acción de control óptima, que magnifica la velocidad de convergencia y minimiza la energía, se utiliza el algoritmo PSO con un alto número de partículas.
Debe quedar claro ahora que un camino de gran utilidad y ventajoso para la resolución de problemas con robots es el de la optimización de un costo dependiente de los objetivos a conseguir por parte de los mismos.
Enfatizando el beneficio de encontrarse en un estado, en cierto sentido, próximo al deseado y el perjuicio de encontrarse en un estado lejano al deseado o casi prohibitivo (por ejemplo, debido a situaciones que afecten la integridad del robot, como estar cerca de un borde, o críticas para el funcionamiento adecuado, como el alto consumo) podrían conseguirse los objetivos deseados. Debido a esto, es natural que las funciones de costo no sean lineales (como cuando se utilizan funciones de barrera) y en ciertas ocasiones tampoco convexas. Para la optimización de este tipo de costos se requieren algoritmos potentes que "naveguen" de forma inteligente en el espacio de búsqueda intentado encontrar el mínimo global de la función, evitando los mínimos locales de la misma, ya que resultan en estados subóptimos. Para esta finalidad puede utilizarse, por ejemplo, *PSO* el cual es simple de implementar y fiable en sus resultados.
Un componente más en la resolución de un problema con robots y en muchísimos otros casos es la prueba mediante simulación. Un ambiente de simulación para probar los comportamientos diseñados es una herramienta fundamental en casi cualquier tipo de proceso de diseño, por no decir en todos. Mediante la simulación es posible testear el funcionamiento del enjambre de robots en un ambiente controlado y libre de riesgo pudiendo así realizar las primeras y no tan primeras pruebas de comportamiento.
Cuanto más "real" sea el simulador menor será el error respecto a la prueba real, siendo la simulación, en ciertos casos, una muy fiable medida del éxito o fracaso de un algoritmo.
A continuación se presenta un simulador diseñado para probar los comportamientos implementados con el fin de determinar su eficacia.

4 Robot Khepera III

4.1 Introducción

La integración del robot *Khepera III* se basa en diez años de experiencia en robótica de miniatura. Las características disponibles en esta plataforma pueden igualar a las prestaciones de robots mucho más grandes. Posee una potencia de cálculo actualizable integrada en el sistema *KoreBot*, múltiples sensores de largo y corto alcance para la detección de objetos, batería intercambiable para obtener mayor autonomía, y una unidad diferencial para cálculos de odometría.
El *Khepera III* es capaz de moverse sobre una mesa, en el piso del laboratorio y en superficies rugosas como alfombras.
El bus del Khepera es compatible con el sistema *KoreBot*, por lo que cualquier ampliación *KoreBot* puede ser conectada en la parte superior del robot. Las especificaciones del bus están totalmente abiertas y disponibles para permitir desarrollos a medida.

4.2 Características

La arquitectura del *Khepera III* proporciona modularidad, utiliza un sistema de extensión por medio de un bus que le proporciona una cantidad casi ilimitada de configuraciones.
La base del robot se puede usar con o sin la tarjeta *KoreBot II*. Esta tarjeta, cuenta con un sistema operativo Linux estándar embebido para el desarrollo de aplicaciones autónomas. Sin la tarjeta *KoreBot II*, el robot puede ser operado remotamente, con una interfaz sencilla con cualquier computadora. Los programas para la operación remota, pueden ser escritos en MatLab®, LabView®, o con cualquier lenguaje de programación que soporte la comunicación por puerto serie.
La base del robot incluye nueve sensores infrarrojos para la detección de obstáculos y cinco sensores de ultrasonidos para la detección de objetos a largo alcance. También posee opcionalmente un par de sensores infrarrojos inferiores para el seguimiento de línea y/o la detección del borde de una mesa.
A través del *KoreBot II*, el robot también soporta tarjetas de extensión Compact Flash, Wi-Fi, Bluetooth, espacio de almacenamiento adicional, entre otros.
Con la extensión *KoreBot II*, el *Khepera III* es capaz de integrar un completo estándar del sistema operativo Linux. Proporciona un entorno conocido de programación en C/C++ para el desarrollo de aplicaciones. Las bibliotecas existentes pueden ser trasladadas al robot, lo que permite el desarrollo de algoritmos y aplica-

ciones integradas y portátiles.
El compilador cruzado GNU C/C++ para el *KoreBot II* proporciona una herramienta para la compilación de código. También es compatible con todas las funciones de la biblioteca estándar de C y casi todas las otras bibliotecas POSIX.
El *Khepera III* una herramienta para los experimentos y demostraciones de robótica tales como:

- Navegación
- Inteligencia Artificial
- Sistemas Multi-Agentes
- Comportamiento colectivo
- Programación en tiempo real

Sus módulos son:

- Una pinza para la manipulación y el transporte de objetos.
- Una batería.
- El *KoreBot* II.
- Una tarjeta de ampliación para el *KoreBot* de entrada y salida de datos, denominada KoreIO.
- Una cámara USB denominada *KoreUSBCam*.
- Una cámara Wireless denominada *KoreWirelessCam*.

Accesorios

- Un cargador externo

Software

- El compilador de C
- El simulador 3D, V-REP.

4.3 Comunicación con el KoreBot

Para lograr la comunicación con el *KoreBot* desde un pc linux se deben seguir los siguientes pasos:

- Configurar el bluetooth desde el applet de la barra de tareas con key 0000.
- Usar el comando `hcitool scan` para obtener la MAC del robot.
- Usar el comando `sdptool browse` para obtener información de la configuración serial en el robot, en particular el canal.

- Levantar una conexión serial mediante `rfcomm connect 0 00:07:80:85:9D:F2 1`, los dos últimos parámetros son la MAC y el canal, pueden variar según el resultado de los pasos 2 y 3 (realizar esta acción como súper usuario).
- Levantar la minicom para abrir una conexión ssh con el robot (realizar esta acción como super usuario) `minicom /dev/rfcomm0`.
- Configurar minicom en el puerto `/dev/rfcomm0`, 115200 Baudios, 8 bits de datos, 1 de start, 1 de stop, sin paridad y sin control de hardware.
- Resetear el robot.
- Loguearse en el *KoreBot* como usuario: *root*, sin clave.

En este momento se estará trabajando como si se encontrarán en la consola del robot.

4.4 Programación

Se debe escribir el código en C y luego compilarlo de la siguiente manera: Para generar el binario se debe modificar al archivo "MyMakefile" en `development_k2_v1.0/lib{\it KoreBot}/src/test` Donde dice

```
#Test Programs
TARGETS = kb_config_test kmot_pantilt koala_test \
  soundtest koreio_test koreio_auto kmot_monitor gpio_test \
  kb_lrftest kmot_ipserver kmotLE_monitor koreioLE_test korebase_test \
  koreioLE_auto koreio_gripper koreio_debug k3_monitor

#Test Programs which require pthread
PTHREAD_TARGETS = koala_demo_client koala_demo kmot_test kmotLE_test khepera3_test
```

Se debe dejar solo

```
#Test Programs
TARGETS = nuestroArchivo

#Test Programs which require pthread
```

Luego se debe ejecutar en una terminal

```
> cd /home/USUARIO/development_k2_v1.0/libKoreBot-1.11-kb1
> source ../env.sh
> make clean
> make
```

(Se debe cambiar `USUARIO` por su nombre de usuario) Y por último se pasará por medio de `zterm` de la minicom el archivo generado.

5 Simulador-EdRo

5.1 Descripción

El simulador EdRo es una plataforma de simulación para generar, testear y depurar conductas de comportamiento colectivo para sistemas de modelos basados en agentes. En particular, el simulador permite experimentar conductas asociadas a la robótica de enjambre para la familia de robots *Khepera* III [25].
El simulador permite seleccionar las características del entorno donde se ejecutará la simulación, como ser la dimensión del espacio, cantidad de obstáculos (circulares) y dimensión de los mismos, así como la posibilidad de fijar objetivos físicos. Es posible también determinar la cantidad de agentes robóticos a utilizar, las dimensiones de los mismos, su ubicación en el espacio, sus características cinemáticas y características respecto al sensado del entorno y percepción del mismo, como ser umbrales de recepción de datos de sensores infrarrojos, adquisición y tratado de imágenes, etc.
El simulador EdRo es muy adecuado para proyectos de investigación y educación relacionados con la robótica de enjambre, donde se podrán probar diferentes algoritmos para robótica de enjambre en entornos dinámicos.
Para lograr esto se utiliza un modelo abstracto y cinemático del robot. El mismo oficia de interfaz entre el entorno y la estrategia que se desea implementar. Desde este punto de vista (función de control del robot), el robot recibe un conjunto de magnitudes físicas (velocidades) y devuelve un conjunto de medidas sensoriales del entorno (ver figura 5.1). De este modo, el algoritmo recibe como datos las medidas sensoriales del agente robótico y devuelve el conjunto de velocidades a adoptar por el mismo como se observa en la figura 5.2.
Desde el punto de vista del entorno, el robot es un actor que recibe medidas sen-

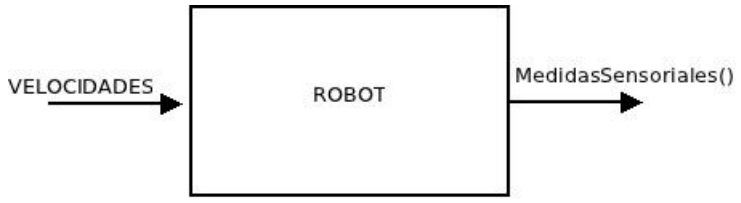


Figura 5.1: modelo abstracto del robot, desde el punto de vista de la acción de control.

soriales y toma acción en función de éstas, básicamente adquiriendo cierto conjunto de velocidades (ver figura 5.3).
Con esta disociación del robot en dos modelos desde diferentes puntos de vista se realiza el diseño del simulador. El modelo del robot desde el punto de vista del entorno, es el utilizado para el diseño del entorno de simulación y sus normas de

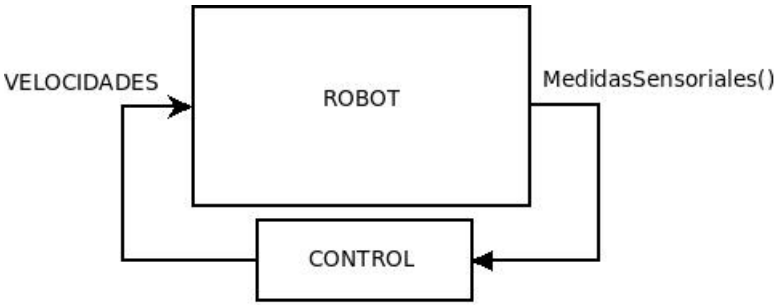


Figura 5.2: modelo abstracto del robot manejado mediante una acción de control.

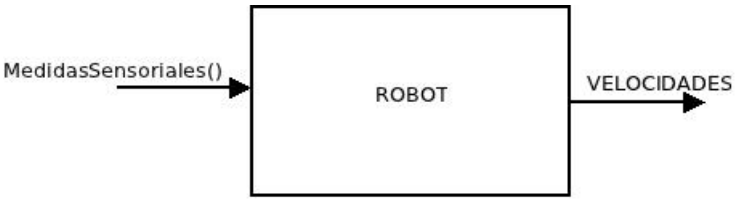


Figura 5.3: modelo abstracto del robot, desde el punto de vista del entorno.

conducta, ya que este modelo "ve" al robot como un agente presente en el entorno que recaba datos de sus proximidades y toma acción en función de estos, moviéndose con cierta velocidad.
El modelo desde el punto de vista de la función de control es el utilizado para la programación del comportamiento del agente ya que el robot provee al algoritmo de datos del entorno y éste determina su velocidad.
Formalmente se puede decir, desde el punto del entorno, que el robot realiza un mapeo entre las magnitudes sensadas y la velocidad que él adquiere:

$$\overrightarrow{v} = \mathcal{F}(magnitudesSensadas()) \tag{5.1}$$

y desde el punto de vista del algoritmo, el mismo realiza otro mapeo entre las magnitudes sensadas, provistas por el robot, y el conjunto de velocidades a adoptar por el agente:

$$\overrightarrow{v} = \mathcal{G}(magnitudesSensadas()) \tag{5.2}$$

además, el agente es representado mediante modelos cinemáticos y por este motivo se regirá según un modelo de la forma:

$$\overrightarrow{p_{k+1}} = \overrightarrow{v_k} + \overrightarrow{p_k} \tag{5.3}$$

siendo $\overrightarrow{p_k}$ y $\overrightarrow{v_k}$ las posiciones y velocidades del agente respectivamente, en el k-ésimo paso.

5.2 Modelado del robot en diferentes niveles de información

Desde el punto de vista del análisis y resolución de problemas, un elemento decisivo en la elección de la solución es el nivel de "sensación" del entorno que posee

el agente robótico. A partir de los datos adquiridos del entorno y eventualmente de su experiencia, el robot realiza la toma de decisiones pertinentes, reaccionando ante los datos recabados en el espacio, generalmente dinámico. Al proceso de interpretar los datos recibidos en función de ellos y de la experiencia del agente robótico se lo conoce como "percepción".
Este hecho provoca que las decisiones tomadas serán mejores, cuanto más información se recabe del entorno, es decir, cuanto más nivel de sensación del entorno posea el agente. En forma dual, puede decirse que para la ejecución de una tarea dada por parte del agente robótico, un incremento en el nivel de información recibida del entorno provocará una disminución en la dificultad de la toma de decisiones, traduciéndose esto en una disminución en la complejidad del algoritmo que implemente el robot.
Para el análisis y resolución de los problemas se modela al robot en diferentes niveles de información, a modo de resolver problemas desde un nivel de conocimiento del entorno alto, hasta un nivel muy bajo.
Quitando hipótesis en el funcionamiento del robot se obtiene un modelo del mismo cada vez más realista y menos ideal, agregándose, en consecuencia, complejidad a los algoritmos en forma incremental conforme se quitan suposiciones. De este modo el algoritmo logra adaptándose de mejor modo al problema enfrentado en la realidad. En el siguiente análisis se distinguen cinco modelos a partir del mismo modelo cinemático (figura 5.1), cada uno aproximándose más a la realidad que el anterior.

5.2.1 Nivel 1 (Robot ideal, con conocimiento total del entorno en un sistema de referencias absoluto)

El modelo del robot más ideal no posee ninguna cota en velocidad y tiene completo conocimiento del entorno, es decir, conoce las posiciones de todos los elementos del mismo.
La hipótesis de pleno conocimiento del entorno es muy fuerte y puede llegar a condicionar al algoritmo en forma importante como fue comentado anteriormente. Esta hipótesis se ve justificada ya que en cierto tipo de contextos, como el fútbol de robots, se posee una cámara que observa el terreno, reconociendo las posiciones de los robots propios y ajenos para informarsela a cada uno bajo un mismo sistema de referencias.

5.2.2 Nivel 2 (Robot con conocimiento total del entorno en un sistema de referencias propio con sensado omnidireccional de rango infinito)

El cambio se produce en que el robot efectivamente conoce todo el terreno, pero en forma relativa a él mismo. Esto se puede observar como el efecto que produciría un sistema de sensores omnidireccional de rango infinito. Esto, puede pensarse como un sistema de sensores, por ejemplo IR, alrededor del robot, con un alto número de ellos y que las distancias a los objetos a medir por los mismos son menores que el umbral de visión de ellos.

5.2.3 Nivel 3 (Robot con conocimiento parcial del entorno en un sistema de referencia propio con sensado omnidireccional de rango finito y comunicación inalámbrica peer to peer)

Este modelo conserva las características del modelo anterior y conserva el sistema de sensado omnidireccional pero ahora éste posee rango de sensado finito. Como contrapartida, se agrega la hipótesis de la comunicación punto a punto entre agentes pudiendo realizarse la misma en la realidad por wifi, bluetooth o algún otro stack de comunicación inalámbrica.

5.2.4 Nivel 4 (Robot con conocimiento parcial del entorno en un sistema de referencia propio con sensado omnidireccional de rango finito y comunicación inalámbrica por broadcast)

Este grado de información modifica la forma de comunicación inalámbrica, que posibilitaba la identificación de cada agente, por el envío de mensajes por broadcast con las mismas tecnologías con la que puede implementarse la comunicación del modelo anterior.

5.2.5 Nivel 5 (Robot con conocimiento parcial del entorno en un sistema de referencia propio con sensado discreto de rango finito y comunicación inalámbrica por broadcast)

éste es el modelo más real del robot y es el nivel alcanzado por el simulador diseñado. El mismo posee 9 sensores distribuidos en forma simétrica alrededor del agente, pero no equiespaciadamente (ver figura 5.4) los cuales poseen un rango de sensado acotado. El modelo conserva el mismo tipo de comunicación que el anterior y agrega características físicas al modelo. Desde el punto de vista dinámico, el agente posee radio y es rígido. Cinemáticamente, el agente tiene una cota en velocidad, tanto lineal como angular.

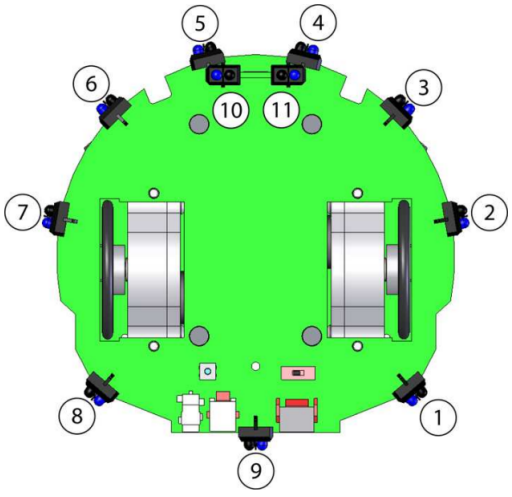


Figura 5.4: Disposición de los sensores IR del robot khepera III visto desde abajo. El mismo posee 9 sensores alrededor de sí mismo y otros 2 debajo. Estos últimos son para implementar, por ejemplo, un seguidor de línea.

6 Formación

6.1 Definición del problema

Partiendo los robots desde una posición inicial aleatoria y en un espacio finito, el problema consiste en, evitando colisiones, realizar una formación entre los robots. La misma debe avanzar en el espacio en una dirección dada, tratando de mantener la disposición geométrica lo más que sea posible, durante todo el tiempo.
En particular, se pretende que los agentes realicen una formación en línea recta, manteniéndose equiespaciados a una distancia dada.

6.2 Resolución del nivel 1

Gracias a las grandes capacidades del agente y prestaciones del modelo, es posible resolver en forma simple el problema de formación. Para otros niveles de información, ciertas estrategias peritirán compensar la falta de datos, como será comentado posteriormente.
Conociendo la posición absoluta de cada robot, cualquier agente puede procesar estos datos y generar la mejor posición para cada uno de los otros y la de sí mismo. Interpretando las posiciones de los robot como puntos del plano, el cálculo puede realizarse utilizando, por ejemplo, mínimos cuadrados (LMS). El problema así formulado consiste en determinar la recta que mejor ajusta a esos puntos. Una vez obtenidos los parámetros de la recta, se puede determinar la posición correspondiente a cada agente en la misma. Como ejemplo, podría asignársele al agente más lejano su proyección ortogonal sobre la recta, con el fin de minimizar su desplazamiento. Dada esta asignación, se calculan las posiciones correspondientes para el resto de los robots, para que cumplan con las restricciones de distancia entre ellos.

6.3 Resolución mediante un problema de optimización

Otra forma de abordar el problema de formación es la reformulación del mismo como un problema de optimización matemática. Con este enfoque, se pretende determinar una disposición espacial de los robots que resuelve el problema y es, en cierto sentido, óptima. Este tipo de formulación resulta muy beneficioso ya que además de determinar una configuración que resuelve el problema, minimiza un costo determinado simultáneamente.
Una posible reformulación consiste en optimizar alguna cualidad deseada del sistema (tiempo, consumo, etc.) y como restricciones imponer las características geométricas de las configuraciones que son solución. De este modo, el conjunto de disposiciones

que son factibles al problema de minimización, son soluciones del problema inicial (formación). A partir de este conjunto, se toma la disposición que resulte óptima, respecto al costo elegido.
En otras palabras, del espacio de configuraciones realizables, las restricciones del problema de optimización limitan las configuraciones a las eficaces (que resuelven el problema) y el costo toma de estas últimas a las más eficientes.
Una formulación como la anteriormente descrita para el problema planteado, consistiría en determinar la velocidad de cada agente $\overrightarrow{v_i^k}$, para que las siguientes posiciones de cada uno $\overrightarrow{r_i^{k+1}}$, produzcan que los agentes se encuentren alineados y a una distancia dada, el uno del otro.
Un posible conjunto de restricciones para conseguir lo anterior sería el siguiente:

(I)

$$d\left(\overrightarrow{r_{i+1}^{k+1}},\ \overrightarrow{r_i^{k+1}}\right) = D,\ \forall i = 1..n-1 \qquad (a)$$

$$\left(\overrightarrow{r_{i+1}^{k+1}} - \overrightarrow{r_i^{k+1}} \wedge \overrightarrow{r_{i+2}^{k+1}} - \overrightarrow{r_{i+1}^{k+1}}\right) = \overrightarrow{0},\ \forall i = 1..n-2 \qquad (b)$$

el cual determina que el conjunto de configuraciones factibles, son las que poseen una distancia D entre agentes dos a dos y presentan a los mismos alineados tres a tres.
Una formulación más cómoda de (I) sería la siguiente:

(I)

$$\left\langle \overrightarrow{r_{i+1}^{k+1}} - \overrightarrow{r_i^{k+1}},\ \overrightarrow{r_{i+1}^{k+1}} - \overrightarrow{r_i^{k+1}} \right\rangle = D^2,\ \forall i = 1..n-1 \qquad (a)$$

$$\left\langle \left(\overrightarrow{r_{i+1}^{k+1}} - \overrightarrow{r_i^{k+1}} \wedge \overrightarrow{r_{i+2}^{k+1}} - \overrightarrow{r_{i+1}^{k+1}}\right), \left(\overrightarrow{r_{i+1}^{k+1}} - \overrightarrow{r_i^{k+1}} \wedge \overrightarrow{r_{i+2}^{k+1}} - \overrightarrow{r_{i+1}^{k+1}}\right) \right\rangle = 0,\ \forall i = 1..n-2 \qquad (b)$$

de este modo, el conjunto de configuraciones factibles posee un número de restricciones igual a $2n-3$ ($n-1$ de (a) y $n-2$ de (b)), siendo n la cantidad de agentes.
Otro planteamiento en forma de restricciones de las configuraciones solución, podría ser sólo en función de las distancias entre agentes. Quitando las restricciones (b) de la forma (I) es fácil observar que el conjunto de realizaciones factibles es mayor, incluyendo, por ejemplo, polígonos regulares los cuales poseen como vértices a los agentes y las aristas poseen longitud igual a la distancia impuesta. Existen más configuraciones factibles a este problema, las cuales determinan figuras abiertas, ya que no existe ningún vínculo entre el primer agente y el último (ver figura 6.1).
Es posible limitar nuevamente el conjunto factible a las configuraciones solución al problema de formación, agregando la restricción de que las distancias entre agentes no contiguos, sea un múltiplo de la distancias que deben tomar los que sí lo son, dependiendo este valor de la lejanía entre ellos. Esto significa que dado un agente, el agente contiguo a éste deberá encontrarse a una distancia D, el siguiente a una distancia igual a $2D$, el siguiente a $3D$, etc.
Este conjunto de restricciones sería:

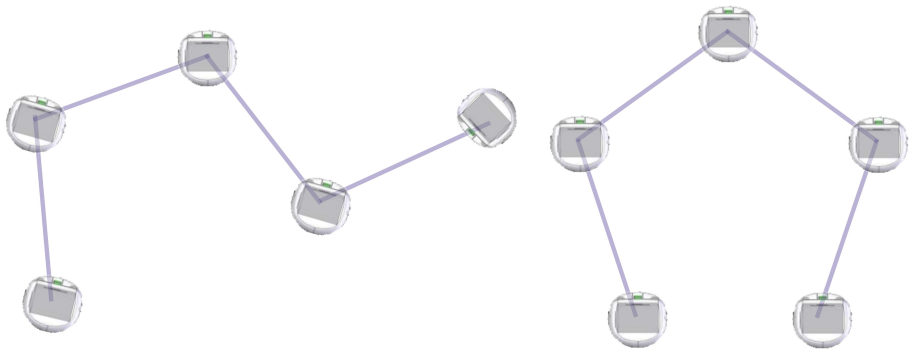


Figura 6.1: Dos configuraciones factibles diferentes para las restricciones planteadas. A la derecha un polígono regular y a la izquierda uno no regular.

(II)

$$\left\langle \overrightarrow{r_{i+1}^{k+1}} - \overrightarrow{r_i^{k+1}},\ \overrightarrow{r_{i+1}^{k+1}} - \overrightarrow{r_i^{k+1}} \right\rangle = D^2,\ \forall i = 1..n-1 \qquad (a)$$

$$\left\langle \overrightarrow{r_{w+j}^{k+1}} - \overrightarrow{r_w^{k+1}},\ \overrightarrow{r_{w+j}^{k+1}} - \overrightarrow{r_w^{k+1}} \right\rangle = (jD)^2,\ \forall w = 1..n-1,\ \forall j = 2..n-w-1 \qquad (b)$$

las cuales llegan a un número de $\frac{(n-1)n}{2}$ ($n-1$ de (a) y $n-2+n-3...1$ de (b)), siendo n nuevamente el número de agentes.
La figura 6.2 muestra la evolución del número de restricciones de ambas formulaciones. De ésta, se observa que para un número de agentes superior a 3, el número de restricciones de la formulación I es menor que el de II, incrementándose la diferencia en forma cuadrática, conforme se incrementa la cantidad de agentes.
Debido a esta observación, es claro que el conjunto de restricciones II posee mayor

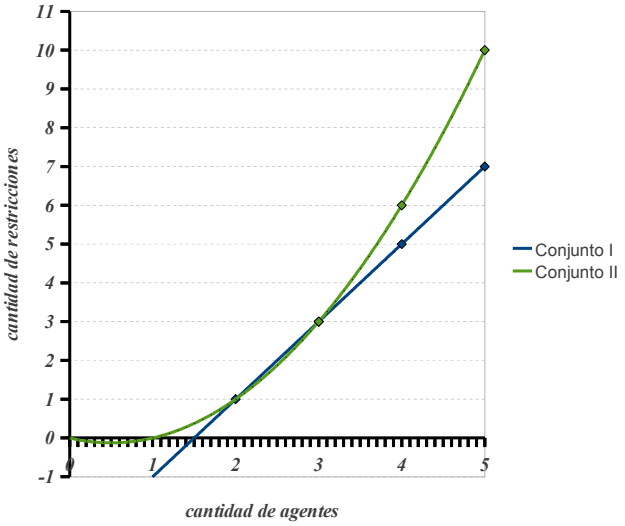


Figura 6.2: Evolución de la cantidad de restricciones para las formulaciones planteadas, en función de la cantidad de agentes presentes. El conjunto I posee un comportamiento afín, mientras que el conjunto II uno parabólico. Esto produce una diferencia notable en la cantidad de restricciones para un número de robots relativamente grande. (Notar que el conjunto de restricciones I tiene sentido para una cantidad de agentes mayor o igual a 3, mientras que el II para una cantidad mayor o igual a 2)

redundancia que I, la cual se ve incrementada a medida que aumentan el número

de robots. Esta redundancia puede ser útil cuando existe incertidumbre en los datos que se manejan. Cuanto más redundancia se tenga en las restricciones, menor será la sensibilidad a las perturbaciones que los datos posean. Un ejemplo simple de este hecho es en el que se posee una configuración inicial como la de la figura 6.3. En éste, todos los agentes se encuentran alineados, a distancia D unos de otros, menos los agentes "1" y "2", que se encuentran a distancia $D1 \neq D$. Claramente, esta configuración no es factible para las restricciones I, ni para las restricciones II (observar que para I las restricciones de alineación están satisfechas).
Debido a una perturbación en los datos, ya sea por el cálculo de la distancia, por la

Figura 6.3: Configuración inicial con tres agentes alineados y a distancias D y $D1$. La misma claramente no es factible para ninguno de los dos conjuntos de restricciones planteados. Un error puede provocar un mal cálculo en la distancia entre los agentes que distan $D1$ y puede ser tomada esta configuración como factible, si se evalúa la misma con el conjunto de restricciones I, debido a su falta de redundancia. En cambio, la evaluación de esta disposición de los agentes con el conjunto de restricciones II, determina que la misma no es factible, ya que sólo este error no es suficiente para alterar la robustez del conjunto (se necesitaría que se produzca un error en el cálculo de la distancia entre agentes que distan $D1 + D$ obteniéndose $2D$ lo cual propone una situación poco probable).

transmisión de los mismos o por algún otro factor, resulta corrupta la distancia entre los agentes "1" y "2", determinándose que la misma es D. Ergo, la configuración pertenece al conjunto de configuraciones factibles de I. En el caso de II, se determina que la configuración no es factible, ya que dista $D + D1 \neq 2D$ del agente 3.
La influencia de la redundancia en los datos y su utilidad es un área no menor en el tratamiento de señales, para más información de cómo insensibiliza ésta a los datos ruidosos, ver [24].
Por otro lado, el conjunto de restricciones I posee gran interés, debido a la utilización de información local. Es decir, que las restricciones involucran sólo a agentes cercanos entre sí. Este hecho no es menor, ya que en la práctica, las medidas locales son mucho más simples de obtener que las que no lo son (inclusive pueden ser imposibles).
Como fue comentado anteriormente, la reformulación como problema de optimización tiene la ventaja de "mejorar" cierta característica del sistema, simultáneamente con resolver el problema. Para el caso de la formación, un costo razonable puede ser la distancia recorrida por cada agente. Esto implica un menor costo energético y un tiempo menor en establecer la configuración deseada. En cierto modo, esto es coherente con la solución implementada con LMS's, al determinar la recta que minimice las distancias cuadráticas a los agentes y asignar al agente más lejano la proyección ortogonal de él sobre la recta.
De este modo, para la reformulación como problema de optimización matemática, se utiliza como costo la distancia recorrida por los agentes y como restricciones, se utiliza el conjunto I, debido a su menor tamaño y utilización de datos locales.
Para la reformulación se obtiene:

$$
\begin{aligned}
\textit{minimizar} \quad & \left\{ \sum_{i=1}^{n} \left\langle \overrightarrow{v_i^k},\ \overrightarrow{v_i^k} \right\rangle \right\} \\
\textit{sujeto a}: \quad & \left\langle \overrightarrow{r_{i+1}^{k+1}} - \overrightarrow{r_i^{k+1}},\ \overrightarrow{r_{i+1}^{k+1}} - \overrightarrow{r_i^{k+1}} \right\rangle = \ D^2 \\
& \forall i = 1..n-1 \\
& \left\langle \left(\overrightarrow{r_{i+1}^{k+1}} - \overrightarrow{r_i^{k+1}} \wedge \overrightarrow{r_{i+2}^{k+1}} - \overrightarrow{r_{i+1}^{k+1}} \right), \left(\overrightarrow{r_{i+1}^{k+1}} - \overrightarrow{r_i^{k+1}} \wedge \overrightarrow{r_{i+2}^{k+1}} - \overrightarrow{r_{i+1}^{k+1}} \right) \right\rangle = 0 \\
& \forall i = 1..n-2.
\end{aligned}
$$

Como las próximas posiciones de los agentes (en el $k+1$-ésimo paso) se determinan como:

$$\overrightarrow{r_i^{k+1}} = \overrightarrow{r_i^k} + \overrightarrow{v_i^k}$$

el problema de optimización posee como variables las velocidades de los agentes $\overrightarrow{v_i^k}$, resultando:

$$
\begin{aligned}
\textit{minimizar} \quad & \left\{ \sum_{i=1}^{n} \left\langle \overrightarrow{v_i^k},\ \overrightarrow{v_i^k} \right\rangle \right\} \\
\textit{sujeto a}: \quad & \left\langle \overrightarrow{r_{i+1}^k} + \overrightarrow{v_{i+1}^k} - \overrightarrow{r_i^k} - \overrightarrow{v_i^k},\ \overrightarrow{r_{i+1}^k} + \overrightarrow{v_{i+1}^k} - \overrightarrow{r_i^k} - \overrightarrow{v_i^k} \right\rangle = \ D^2 \\
& \forall i = 1..n-1 \\
& \left\langle \left(\left(\overrightarrow{r_{i+1}^k} + \overrightarrow{v_{i+1}^k} - \overrightarrow{r_i^k} - \overrightarrow{v_i^k} \right) \wedge \left(\overrightarrow{r_{i+2}^k} + \overrightarrow{v_{i+2}^k} - \overrightarrow{r_{i+1}^k} - \overrightarrow{v_{i+1}^k} \right) \right), \right. \\
& \qquad \left. \left(\left(\overrightarrow{r_{i+1}^k} + \overrightarrow{v_{i+1}^k} - \overrightarrow{r_i^k} - \overrightarrow{v_i^k} \right) \wedge \left(\overrightarrow{r_{i+2}^k} + \overrightarrow{v_{i+2}^k} - \overrightarrow{r_{i+1}^k} - \overrightarrow{v_{i+1}^k} \right) \right) \right\rangle = 0 \\
& \forall i = 1..n-2
\end{aligned}
$$

El problema así formulado posee n incógnitas vectoriales (las velocidades de los agentes, $\overrightarrow{v_i^k},\ i = 1..n$) que resultan en $2n$ incógnitas escalares ($(v_{i_x}^k,\ v_{i_y}^k),\ i = 1..n$), ya que el problema es bidimensional. Como ya fue visto, esta formulación posee $2n-3$ restricciones. Si el problema fuera un problema de programación lineal, se estaría en condiciones de decir que el programa posee 3 variables libres, lo cual enriquecería el conjunto de soluciones factibles. El hecho es que el problema no es un programa lineal, debido a dos factores. Por un lado, las restricciones de distancia entre partículas y por otro lado, a la función de costos utilizada (que depende de la norma de las variables vectoriales). El efecto de la no linealidad se ve, en primera instancia, en que no es cierto que se posean 3 grados de libertad en el problema. Sin embargo, la cantidad de grados de libertad del programa lineal, con la misma cantidad de restricciones y variables, es una cota superior en los grados de libertad del programa no lineal.
Eso se puede observar en el ejemplo siguiente.
Suponiendo que se poseen 3 agentes, el problema anterior poseería 6 variables y 3 restricciones. Imponiendo la posición de los 3 agentes, es decir, dejándolos fijos (ver

figura 6.4), se eliminan todas las variables del problema. Esto provoca que no exista ninguna solución factible no trivial (la única configuración factible corresponde al conjunto de agentes ya formados).

Imponiendo las posiciones de 2 agentes y dejando uno libre se agrega holgura al

Figura 6.4: Configuración inicial del problema de formación con tres agentes, los cuales no poseen ningún grado de libertad. Evidentemente este problema no posee solución, ya que no se posee factibilidad para ninguna disposición posible, a no ser, claramente, una configuración con los agentes ya formados a la distancia deseada y alineados. Desde el punto de vista matemático, el universo de configuraciones se reduce a una, la inicial, ya que el problema no presenta variables y por lo tanto, en caso de ser verificadas las restricciones por esta disposición, la misma será la única, por lo que también será óptima para cualquier tipo de costo formulado.

problema, teniendo el mismo 2 variables.

Nuevamente el problema tiene un conjunto factible vacío, salvo las configuraciones con los agentes fijos, distando D o $2D$ entre sí.

Un problema con 3 variables, correspondería, por ejemplo, a un problema con un agente fijo, otro libre y uno con un grado de libertad en su movimiento. Este grado de libertad en el movimiento del agente, le permite tratar de acercarse o alejarse hasta D del agente fijo o de tratar de ubicarse a $2D$ de éste (en cualquiera de los casos, el agente libre podría ubicarse de forma tal de realizar la formación). De todos modos, dada la partícula fija, existe sólo una región del plano en la cual la partícula con un grado de libertad puede encontrarse, para que exista alguna solución al problema (ver figura 6.5).

Puede observarse que para 4 variables cualesquiera, el problema sí posee configuraciones factibles, para cualquier condición inicial de los agentes.

Si bien el principal elemento del problema de optimización son las restricciones (que determinan las configuraciones que se encuentran formadas), éstas pueden eliminarse, o mejor dicho, es posible formular un problema de optimización equivalente al primero (que posea las mismas soluciones) pero que sea libre de restricciones.

Otro argumento para la supresión de las restricciones es que existen algoritmos de optimización muy potentes para problemas libres de restricciones, como lo son "Ant Colony Optimization" o "Particle Swarm Optimization", este último descrito en el capítulo **??**.

En este problema equivalente, la función de costos toma un papel fundamental, siendo esta la que debe poseer la información que en el problema original tenían las restricciones. Básicamente, esto se consigue agregando al costo original términos, uno por cada restricción, los cuales se anulen cuando se cumple la restricción correspondiente y sean positivos cuando no. Así, el problema de optimización posee una función de costos compuesta por una suma de términos positivos o nulos. De este modo, un conjunto de variables que lo minimice, minimizará cada término por separado. Cualquier configuración inicial es factible y la eficacia del problema anterior es representada en la eficiencia de este otro.

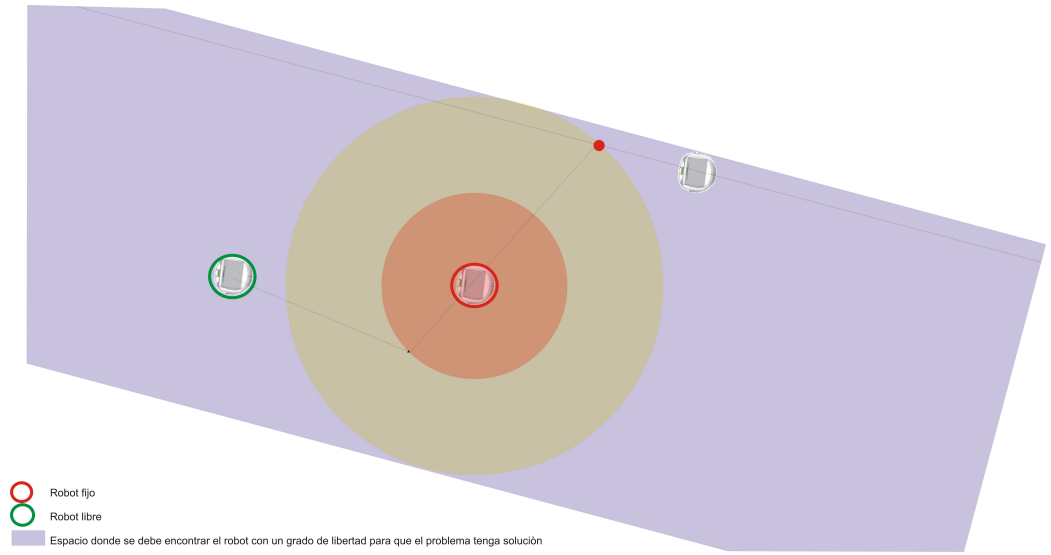


Figura 6.5: Formulación del problema de formación para tres agentes, con uno de ellos libre (verde, con dos grados de libertad), otro fijo (rojo, sin grados de libertad) y otro con un sólo grado de libertad (lo que se traduce en la posibilidad de un movimiento rectilíneo por parte del mismo). El problema posee tres grados de libertad. A diferencia de un problema de programación lineal, el cual asegura solución, ya que se posee la misma cantidad de variables que de restricciones, existe sólo una región del plano en la cual la partícula con un grado de libertad puede encontrarse inicialmente, para que exista alguna solución al problema (región gris). La misma es delimitada por las rectas tangentes a la circunferencia que posee radio $2D$ y como centro la posición del agente fijo. Estas rectas cumplen que son paralelas a la dirección "libre" del agente. Si el agente en cuestión pertenece a esta banda, el mismo aspirará a encontrarse a distancia D del agente fijo, si su trayectoria es secante a la circunferencia de radio D y mismo centro. En caso contrario, las soluciones involucrarán que el agente se encuentre a distancia $2D$ del agente fijo (notar que el agente libre puede ubicarse donde sea necesario). El problema así formulado puede poseer 4, 3, 2, 1 o ninguna solución, en función de las posiciones iniciales del agente fijo y la del agente con un solo grado de libertad.

La reformulación podría ser la siguiente:

$$
\begin{aligned}
min \quad & \left\{ \sum_{i=1}^{n} \left\langle \overrightarrow{v_i^k},\ \overrightarrow{v_i^k} \right\rangle + \sum_{i=1}^{n-1} \left[\left\langle \overrightarrow{r_{i+1}^k} + \overrightarrow{v_{i+1}^k} - \overrightarrow{r_i^k} - \overrightarrow{v_i^k},\ \overrightarrow{r_{i+1}^k} + \overrightarrow{v_{i+1}^k} - \overrightarrow{r_i^k} - \overrightarrow{v_i^k} \right\rangle - D^2 \right]^2 \right. \\
& + \sum_{i=1}^{n-2} \left\langle \left(\left(\overrightarrow{r_{i+1}^k} + \overrightarrow{v_{i+1}^k} - \overrightarrow{r_i^k} - \overrightarrow{v_i^k} \right) \wedge \left(\overrightarrow{r_{i+2}^k} + \overrightarrow{v_{i+2}^k} - \overrightarrow{r_{i+1}^k} - \overrightarrow{v_{i+1}^k} \right) \right), \right. \\
& \left. \left. \left(\left(\overrightarrow{r_{i+1}^k} + \overrightarrow{v_{i+1}^k} - \overrightarrow{r_i^k} - \overrightarrow{v_i^k} \right) \wedge \left(\overrightarrow{r_{i+2}^k} + \overrightarrow{v_{i+2}^k} - \overrightarrow{r_{i+1}^k} - \overrightarrow{v_{i+1}^k} \right) \right) \right\rangle = 0
\end{aligned}
\tag{6.2}
$$

Esta representación del problema, asigna igual importancia a la configuración final de los agentes, como a la eficiencia propiamente dicha. Esto significa que una configuración que no cumpla cierta restricción pero sea muy eficiente respecto al costo del primer problema, se encuentra en las mismas condiciones que una que las cumpla pero no eficientemente. Para que posea prioridad la configuración, puede asignársele pesos a cada término de la función de costos, siendo más altos los correspondientes a las restricciones, a modo de distinguir casos como el presentado.

De este modo, la función de costos asociada al problema de optimización resulta:

$$min \quad \left\{ k_v \sum_{i=1}^{n} \left\langle \overrightarrow{v_i^k},\ \overrightarrow{v_i^k} \right\rangle + k_d \sum_{i=1}^{n-1} \left[\left\langle \overrightarrow{r_{i+1}^k} + \overrightarrow{v_{i+1}^k} - \overrightarrow{r_i^k} - \overrightarrow{v_i^k},\ \overrightarrow{r_{i+1}^k} + \overrightarrow{v_{i+1}^k} - \overrightarrow{r_i^k} - \overrightarrow{v_i^k} \right\rangle - D^2 \right]^2 \right.$$
$$+k_a \sum_{i=1}^{n-2} \left\langle \left(\left(\overrightarrow{r_{i+1}^k} + \overrightarrow{v_{i+1}^k} - \overrightarrow{r_i^k} - \overrightarrow{v_i^k} \right) \wedge \left(\overrightarrow{r_{i+2}^k} + \overrightarrow{v_{i+2}^k} - \overrightarrow{r_{i+1}^k} - \overrightarrow{v_{i+1}^k} \right) \right), \right.$$
$$\left. \left. \left(\left(\overrightarrow{r_{i+1}^k} + \overrightarrow{v_{i+1}^k} - \overrightarrow{r_i^k} - \overrightarrow{v_i^k} \right) \wedge \left(\overrightarrow{r_{i+2}^k} + \overrightarrow{v_{i+2}^k} - \overrightarrow{r_{i+1}^k} - \overrightarrow{v_{i+1}^k} \right) \right) \right\rangle = 0 \right.$$

(6.4)

6.4 Resolución del nivel 2

Como fue advertido anteriormente, para este nivel de información es posible utilizar la resolución del problema anterior para resolverlo.
Una forma de abordar el problema consiste en que un agente procese los datos y calcule las velocidades del resto y la de sí mismo.
En un análisis previo se determinó que es posible resolver el problema manteniendo un agente fijo, lo cual favorece a la resolución y simplifica el análisis.
Entonces, un agente fijo oficiará de origen de coordenadas y a partir de medidas del entorno obtendrá la posición de cada uno de los agentes referida a sí mismo. Por lo tanto, la posición actual y la velocidad del i-ésimo agente, en el paso k, será:

$$\overrightarrow{r_i^k} \sim r_i^k{}_{\langle (\psi_r)_i^k}$$
$$\overrightarrow{v_i^k} \sim v_i^k{}_{\langle (\psi_v)_i^k}$$

Una vez recibidos los datos, el agente los deberá procesar para determinar la velocidad que debe adoptar cada uno de los otros, según el algoritmo anterior.

6.5 Resolución del nivel 3

En este nivel, el impedimento para aplicar la estrategia anterior es la falta de visibilidad de todos los agentes robóticos presentes. Bajo el supuesto de que el conjunto de robots conforma un grafo conexo, puede utilizarse el algoritmo anterior. Que un grafo sea conexo significa que no existen grupos separados de robots que no se logren ver entre ellos y ningún robot se encuentra aislado.
Así un robot determinará las velocidades de los agentes mediante sus posiciones actuales, las cuales pueden determinarse mediante la visibilidad dada por el grafo y no por el sensado directo.
Supóngase el caso de la figura 6.6, en el cual el agente *1* se encargará del procesamiento (o será un nodo "padre" en el grafo). El agente *1* logra sensar al agente *2* pero no al *3*. El agente *2* sí logra ver al *3*.
Sean: $\overrightarrow{r_{12}} = r_{12\langle\psi_{12}}$, $\overrightarrow{r_{21}} = r_{21\langle\psi_{21}}$ y $\overrightarrow{r_{23}} = r_{23\langle\psi_{23}}$ las coordenadas con las que *1*

representa a *2*, *2* a *1* y *2* a *3* respectivamente.
Sean Δ_{x23} y Δ_{y23} las proyecciones ortogonales del vector $\overrightarrow{r_{23}}$ sobre las direcciones colineal y perpendicular a la dirección $\overrightarrow{12}$ con origen en el agente *2*. Entonces, a partir de los datos sensados por el agente *2*, pueden determinarse estos valores del siguiente modo:

$$\begin{aligned} \Delta_{x_{23}} &= r_{23} cos(\psi_{21} - \psi_{23} - \pi) \\ \Delta_{y_{23}} &= r_{23} sin(\psi_{21} - \psi_{23} - \pi) \end{aligned}$$

Con estos datos, recibidos mediante comunicación inalámbrica, el agente *1* puede determinar la posición del agente *3*, $\overrightarrow{r_{13}} = r_{13\langle\psi_{13}}$, a partir de los valores determinados por el agente *2*, según las expresiones:

$$\begin{aligned} r_{13} &= \sqrt{(r_{12} + \Delta_{x_{23}})^2 + \Delta_{y_{23}}^2} \\ \psi_{13} &= \psi_{12} - arctg\left(\frac{\Delta_{y_{23}}}{\Delta_{x_{23}} + r_{12}}\right) \end{aligned}$$

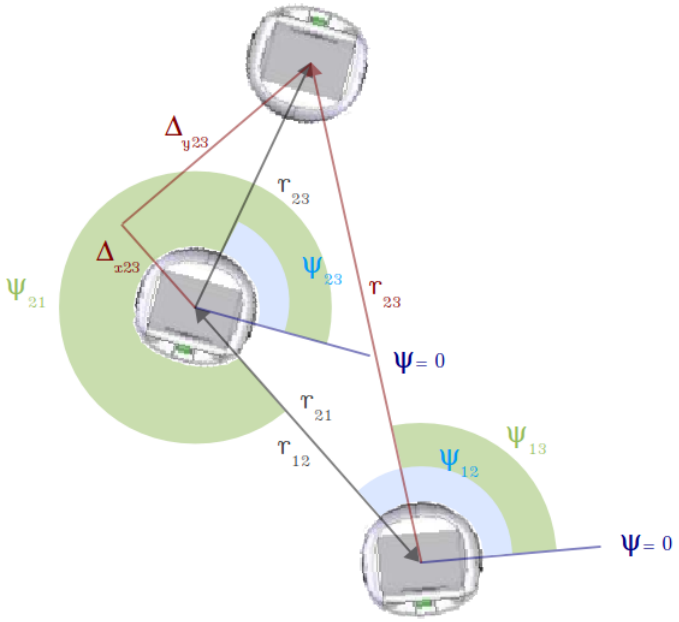


Figura 6.6: Determinación de la posición del agente 3 relativa al agente 1. El agente 1 observa al agente 2 pero no al agente 3. Sin embargo, el agente 2 si logra ver al 3. Mediante este "grafo" conexo el agente 1 logra determinar la posición del agente 3, solicitando información al agente 2. De este modo, se puede generalizar la determinación de la posición de un agente no visible por otro en forma recursiva, determinando la posición del siguiente agente en el grafo (siempre en la suposición de la conexión del mismo).

Un aspecto a destacar es que el cálculo por parte del agente *2* se realiza en función de información local, basado en diferencias entre medidas (básicamente en los

ángulos) al igual que en *1*. Esto es bueno ya que el cálculo no se basa en ningún sistema de referencia o dirección común, por lo que el algoritmo presenta robustez a las direcciones o sistemas de referencia de cada agente.
Del mismo modo en que el agente *1* determina la posición relativa del agente *3* a partir de la visibilidad obtenida por *2*, un agente *0* que ve a *1* pero no ve a *2* ni a *3*, podría determinar la posición relativa de *3* descendiendo por el grafo, gracias a la visibilidad obtenida por *1*.

6.6 Resolución de los niveles 4 y 5

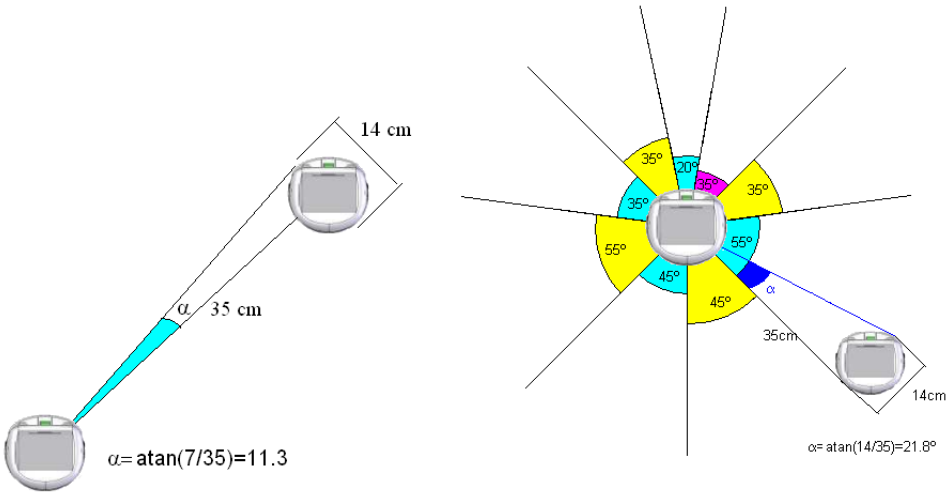
La gran diferencia entre los 2 últimos niveles de información planteados, es el cambio del sistema de sensado omnidireccional a uno discreto, en particular, con 9 sensores. El sensado discreto puede compensarse utilizando una estrategia de barrido, por la cual un agente simula un sensado omnidireccional. La diferencia entre un sensado omnidireccional y un barrido es que el segundo puede verse como un incremento en la cantidad de sensores, aumentando notablemente la resolución del sensado pero sin poder llegar a la resolución de un sistema continuo de medida como el primero. En diversos casos, puede disminuirse la resolución requerida mediante suposiciones respecto a las dimensiones de los elementos presentes en el entorno, pudiendo determinar una resolución mínima para poder detectar y determinar la cantidad de elementos presentes. Básicamente, el cálculo es realizado a partir de las dimensiones de los elementos que se encuentran en el entorno y bajo la suposición de que existe continuidad entre muestras consecutivas (esta idea es ejemplificada en subsección *formarse*). En estos casos, la resolución requerida es tal, que un barrido no produce diferencias con respecto a un sensado omnidireccional.
Otro aspecto del barrido, es que las medidas no son simultáneas, por lo que medidas obtenidas en ángulos cercanos pueden poseer una diferencia temporal. Además, la adquisición de todas las medidas produce un incremento en el tiempo de adquisición proporcional a la resolución que se desee obtener.
En resumen, bajo la hipótesis que las velocidades de los elementos del entorno visible por el agente son suficientemente bajas en comparación con la velocidad de adquisición de las medidas, el barrido produce una aproximación importante del nivel 5 al nivel 4.

6.6.1 Descripción

El algoritmo planteado en este caso se basa en 3 estados y en detectar cuándo termina cada uno de ellos. El primero se denomina *formarse*, consiste en culminar formados en linea recta, sin importar el sentido hacia el que apunten los agentes. El segundo de ellos, denominado *direccionarse*, tiene como objetivo dejar a todos los agentes apuntando hacia la misma dirección. Por último, el tercero se denomina *avance* y posee como meta principal barrer el espacio manteniendo la formación. Es sumamente importante aclarar que este algoritmo funciona únicamente si todos los agentes comienzan formando un grafo conexo, aunque no necesariamente el grafo debe ser cerrado.

6.6.2 Formarse

El comportamiento de los agentes durante el estado *formarse*, consiste primeramente en sortear un líder aleatorio mediante comunicación *WI-FI*. Para esto, todos los agentes poseen un número de robot, asignado de antemano, y sortean un número aleatorio entre *0* y *1*. Todos comunicarán su resultado y quien posea el número inferior será el líder. En lo sucesivo el líder será quien le dará permiso a los agentes para moverse. Empezará moviéndose él y a continuación quien posea el siguiente número de robot y así sucesivamente.
Cuando un robot recibe el permiso de moverse, en primera instancia sensa el entorno. Como fue comentado anteriormente, con frecuencia sucede que varios agentes quedan en zonas ciegas, por más que estén dentro del rango de visión. Para resolver este inconveniente, se optó por un mecanismo de barrido, tal y como fue descrito anteriormente. Se determinó un barrido consistente en 5 giros de 9° cada uno, completando así un giro de 45°. La elección de los 9° se basa en el radio de un robot y la visión máxima (ver figura 6.7.a). La elección de los 45° está determinada para que se logre recorrer toda la zona de visión, sin dejar a ningún agente en zonas ciegas, en función de la separación de los sensores, el radio de un robot y la visión máxima de los mismos (ver figura 6.7.b).
Luego de sensar el entorno, el agente debe decidir su movimiento en función de la



(a) (b)

Figura 6.7: Determinación de la resolución mínima y ángulo de barrido mínimo para la percepción de agentes en el entorno. En (a) se observa el cálculo para obtener, al menos, un valor de un robot que se encuentre en el entorno visible, es decir, el ángulo máximo de giro entre sensados del entorno para asegurarse la determinación de todos los agentes presentes. Suponiendo a los agentes de forma cilíndrica y un radio dado por la circunferencia de radio mínimo que circunscribe al agente ($7cm$), puede determinarse dicho ángulo mediante el umbral de visión de los sensores infrarrojos ($35cm$). Siendo esto de un valor de $\arctan \frac{7cm}{35cm} \approx 11.3°$.
En (b) se observa el mínimo ángulo de giro del robot para sensar a todos lo agentes presentes, sin necesidad de realizar un giro completo. El mismo es determinado por el ángulo ciego más grande que posea el robot, debido a la distribución de sus sensores (55°) y suponiendo la posición del robot más desfavorable, es decir, tangente a uno de los sensores e interior al ángulo formado a una distancia igual al umbral de visión de los sensores, el mismo resulta en $55° - \arctan \frac{14cm}{35cm} \approx 33.2°$.
Por estos motivos se determina un barrido de 45° en total, realizando 5 medidas las cuales serán tomadas cada 9°.

cantidad de agentes percibidos. En caso que detecte a un único robot, se posicionará a una distancia D del mismo, manteniendo la dirección que tenía anteriormente (ver figura 6.8). En caso que detecte 2 robots, optará, en primer lugar, por dirigirse al punto medio del segmento que éstos forman. Si el procesamiento de los datos percibidos le indica que él no entraría en ese lugar, entonces intentará posicionarse a una distancia D de alguno de ellos 2 y en la dirección que ellos forman, optando entre las 2 posibilidades por aquella que se encuentre más cerca de él (ver figura 6.9). En el caso que esta decisión lo lleve a colisionar y lo pueda determinar de antemano, el robot realizará un movimiento en 2 tramos, moviéndose por las tangentes al robot con quien colisionaría (ver figura 6.10). Si de todas formas no es posible realizar el movimiento, el agente se quedará en su lugar.

Decidida la próxima posición, la misma deberá cumplir con ciertos requisitos para

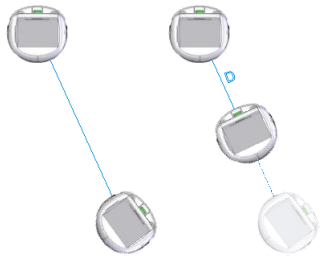


Figura 6.8: Luego de recibido el permiso para moverse y haber sensado el entorno, el agente que se encuentra en la parte inferior de la imagen solo logra observar a otro robot, por lo que su entorno se reduce a lo que se observa en la parte izquierda de la imagen. En estas circunstancias decide moverse en la misma dirección que forman ambos, hasta ubicarse a una distancia D del otro, como se observa en la parte derecha de la imagen.

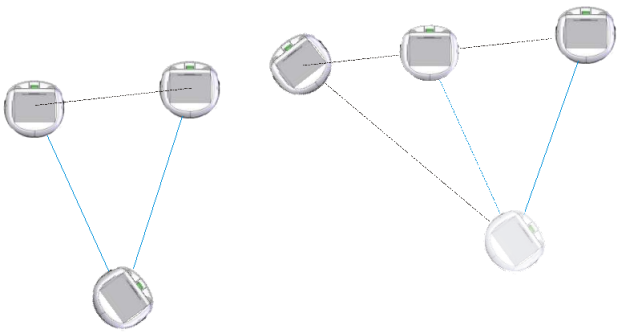


Figura 6.9: Luego de recibido el permiso para moverse y haber sensado el entorno, el agente que se encuentra en la parte inferior de la imagen logra observar a 2 robots más. Por lo tanto, su entorno se reduce a lo que se observa en la parte izquierda de la imagen. En primer lugar intentará colocarse entre medio de estos, pero determinará que no tiene suficiente espacio. Por esto, se colocará a una distancia D de alguno de los dos robots y en la dirección que estos forman, optando en este caso por el extremo izquierdo, dado que, se encuentra más cerca que el otro. Esto se puede observar en la parte derecha de la imagen.

poder alcanzarse. Uno de estos es que el resto de los robots no pierdan la comunicación con el grafo. Para lograrlo, cada robot determina, entre los agentes visibles para él, cuáles logran verse entre ellos. Los robots que no logran percibir a otros, son considerados para no abandonarlos al momento de determinar la nueva posición.
El robot que logra percibir a otros no depende del primero y por este motivo puede ser ignorado al determinar la nueva posición. Esta estrategia puede determinar gru-

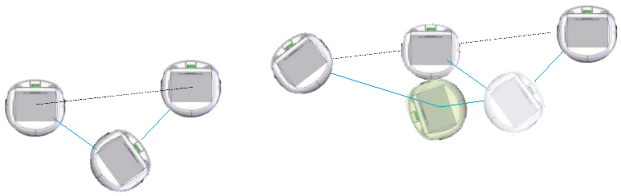


Figura 6.10: Luego de recibido el permiso para moverse, y haber sensado el entorno, el agente que se encuentra en la parte inferior de la imagen logra observar a 2 robots más, por lo que su entorno se reduce a lo que se observa en la parte izquierda de la imagen. En primer lugar intentará colocarse entre medio de éstos, pero determinará que no tiene suficiente espacio, por lo tanto se colocará a una distancia D de alguno de los dos robots en la dirección que éstos forman, optando en este caso por el extremo izquierdo, dado que, se encuentra mas cerca que el otro extremo. Sin embargo, si intenta realizar este movimiento colicionará con otro agente, y por lo tanto, decide realizar un movimiento doble, moviéndose de manera de minimizar el largo de la trayectoria, es decir, por las tangentes del robot al que colisionaría, esto se puede observar en la parte derecha de la imagen.

pos disjuntos de robots, si se procesan todos los agentes en forma simultánea, ya que los agentes próximos entre sí se ven unos con otros. Según esta estrategia, no se perdería la conexión del grafo si el robot se mueve sin considerarlos, lo cual es erróneo. Para solucionar este inconveniente, la determinación de cuántos agentes logran verse entre ellos se realiza en forma secuencial, descartando a los agentes que ya fueron considerados y supuestos como visibles por otros (ver 6.11). El robot que desea moverse, luego de determinar los robots que dependen exclusivamente de él para la conexión del grafo, no puede alejarse más que el umbral de visión (a menos una tolerancia por errores de medida) de ellos. Se define el conjunto de los puntos posibles como la intersección de las circunferencias cuyo radio es igual al umbral de visión, menos cierta tolerancia, con centro en cada uno de los robots a no abandonar (ver figura 6.12). Finalmente, el robot determinará su próxima posición, como aquella que mantenga la conexión del grafo en su entorno cercano y se encuentre lo más próxima posible a la determinada anteriormente, gracias a la condición de formación. Para un mejor funcionamiento del algoritmo, el robot evaluará primero a aquellos robots que se encuentren más alejados de la próxima posición elegida de antemano (determinada por la condición de formación). De esta manera, estos robots (los más lejanos) tienen más posibilidades de ver a otro y serán descartados (ver figura 6.12).

Otra requisito para determinar la nueva posición de un agente será no moverse mas del umbral de visión. Esta condición se debe a que un agente conoce sólo su entorno cercano y por lo tanto los cálculos realizados sólo tiene validez allí. En su implementación, esta condición simplemente modifica el módulo de la próxima posición y lo fija de tal manera de no moverse mas allá de lo que ve el agente (ver figura 6.13).

Una vez determinada y alcanzada la próxima posición, el robot le informará al líder que ha terminado su movimiento, y éste le dará permiso al siguiente para moverse. Esto se repetirá hasta que el líder entienda que se encuentran todos los agentes formados. Para esto, el líder interroga al resto, consultando si creen estar formados y a cuántos robots ven. En caso que un agente vea 3 robots o más, el mismo responderá que no está formado, en caso de ver a dos robots se verificará que el ángulo entre éstos sea de 180°, en caso contrario, no estará formado. En conclusión, un agente se encuentra formado sólo si ve a un agente o ve a dos agentes que distan entre ellos 180° aproximadamente (se deja cierta tolerancia por incertidumbre en el

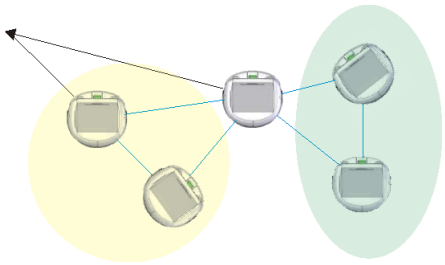


Figura 6.11: Luego de recibido el permiso para moverse, y haber sensado el entorno, el agente que se encuentra en medio de la imagen logra observar a 4 robots más, por lo que su entorno se reduce a lo que se observa en la imagen. Luego de decidir que su próxima posición debería ser el extremo superior de los agentes que se encuentran sobre la izquierda de la imagen, deberá decidir hasta donde puede moverse sin romper el grafo. Según el primer algoritmo explicado en el documento, el robot decide si un agente puede ser abandonado o no, dependiendo de si logra observar a un tercer agente. En este caso, los otros 4 robots logran observar al menos a 2 agentes, por ejemplo el de arriba a la derecha, logra observar al de abajo a la derecha, y a su vez el de abajo a la derecha logra observar al de arriba a la derecha, por lo tanto decidirá que ambos dos no dependen de él en la conexión con el grafo, lo que es erróneo, dado que si se realiza el movimiento descripto anteriormente, el grafo será dividido en dos.

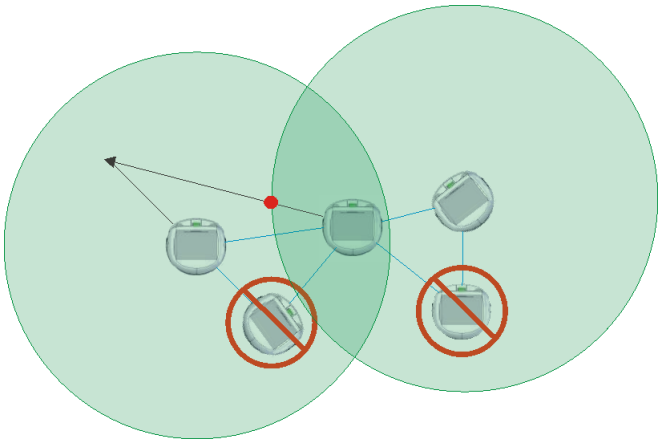


Figura 6.12: Luego de recibido el permiso para moverse, y haber sensado el entorno, el agente que se encuentra en medio de la imagen logra observar a 4 robots más, por lo que su entorno se reduce a lo que se observa en la imagen. Luego de decidir que su próxima posición debería ser el extremo superior de los agentes que se encuentran sobre la izquierda de la imagen, deberá decidir hasta donde puede moverse sin romper el grafo. Según el segundo algoritmo explicado en el documento, el robot decide si un agente puede ser abandonado o no, dependiendo de si logra observar a un tercer agente, sin embargo esto lo realiza con cierto orden, comenzando por aquellos que se encuentran mas lejos del lugar al que quiere ir. Por lo tanto, en este caso, comenzará por el robot ubicado en el extremo inferior derecho de la imagen, y decidirá que éste sí logra sensar a otro robot. Sin embargo, al continuar con el que se encuentra en el extremo superior derecho, decidirá que no observa a nadie más, dado que el otro robot que logra observar (el que se encuentra en el extremo inferior derecho) ya pasó por este algoritmo y por lo tanto, no puede ser considerado nuevamente. Es decir, se debe observar si logra sensar a un tercer agente pero a su vez, este tercer agente no puede haber sido estudiando antes por este mismo algoritmo. Por lo tanto, el robot no podrá abandonar al gente ubicando en el extremo superior derecho ni al que se encuentra en el extremo superior izquierdo, por lo tanto su movimiento será recortado como muestra la imagen y efectivamente no romperá el grafo.

sensado).
Luego que todos los agentes le responden al líder, existen 3 casos:

- **1** Al menos un agente no está formado
- **2** Todos los agentes están formados pero hay al menos 3 o más que manifiestan ver a un solo robot.
- **3** Todos los agentes están formados y sólo 2 manifiestan ver a un solo robot.

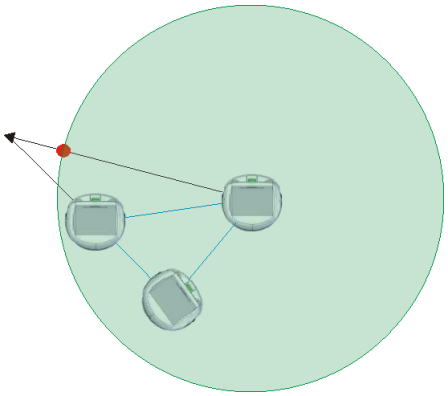


Figura 6.13: Luego de recibido el permiso para moverse, y haber sensado el entorno, el agente que se encuentra en la parte derecha de la imagen logra observar a 2 robots más, por lo que su entorno se reduce a lo que se observa en la parte izquierda de la imagen. Una vez decidido su próximo movimiento, deberá verificar que el mismo no se encuentre mas allá de su umbral de visión, en este caso, el umbral de visión es inferior, y por lo tanto modificará el módulo de su movimiento, manteniendo la dirección y el sentido, como se observa en la figura.

En el caso *1* y *2*, el líder informa que se continuará en el estado 1, *formarse* y le dará permiso a alguien para moverse. En el caso 3 informará que se cambiará al estado 2, *direccionarse*.

6.6.3 Direccionarse

Durante el segundo estado, todos los robots que ven a dos agentes, compiten (como en el estado 1) sorteando un número aleatorio entre 0 y 1. Quién posea el número inferior será quien se mueva primero. Este agente decidirá en qué dirección desplazarse sensando su entorno y dirigiéndose en forma perpendicular a sus vecinos (cualquiera de las dos direcciones posibles). Cuando termina de moverse, informará que ya ha finalizado y luego se moverán quienes al sensar vean una diferencia con su último sensado anterior y todavía no se hayan movido en este estado. Es decir, luego que un robot se mueva, lo harán sus vecinos y así sucesivamente hasta que todos se hayan movido una vez. Estos movimientos se harán en la dirección en la que percibió que se movió su vecino. Para esto realiza la diferencia entre la posición actual del vecino y la que tenía anteriormente (ver figura 6.14). Cada uno informará que ya se ha movido, y quienes todavía no lo hayan hecho volverán a sensar su entorno y así sucesivamente, propagándose el movimiento hasta que todos se muevan una vez, momento en el cual el líder informará que se pasará al próximo estado, *avance*.

6.6.4 Avance

El siguiente estado en el algoritmo es el avance en formación, en el cual se supone una disposición inicial con todos los agentes alineados y orientados hacia una misma dirección (condiciones en las cuales se dispone el enjambre debido a los estados anteriores). Aún así, es razonable implementar cierto tipo de corrección mientras evoluciona el enjambre con el fin de conseguir mayor robustez y poder mantener válida esta hipótesis, como se verá luego.
El algoritmo en este estado determina la velocidad $\overrightarrow{v^k}$ de cada agente en coordenadas

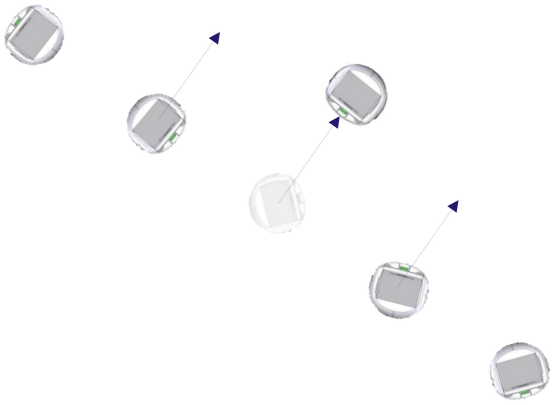


Figura 6.14: Luego de haberse formado, los agentes deben quedar posicionados hacia la misma dirección, por lo tanto realizan un sorteo entre aquellos que se encuentran en el medio de la formación (sensan a dos agentes). En este caso el ganador fue el agente del medio, y decidió moverse hacia arriba (siempre en dirección perpendicular a la formación, pero podría haberlo hecho hacia abajo). Luego de esto los agentes que se encuentran al lado, pueden copiar su movimiento, y así sucesivamente hasta que todo el enjambre se haya movido, de esta manera quedarán todos posicionados hacia la misma dirección.

cartesianas relativas a sí mismos $\left(v_x^k,\ v_y^k\right)$ como muestra la figura 6.15. La velocidad horizontal de un agente, v_x^k, es determinada de forma tal que la distancia horizontal de los agentes visibles por éste (en el siguiente paso) sea múltiplo de la distancia deseada para la formación (D), esto significa que el agente observado más cercano debería encontrarse a distancia D, el siguiente a $2D$ y así sucesivamente, en forma simétrica a ambos lados del robot (ver figura 6.16).
De este modo, siendo $\mathcal{R}_p^k$ y $\mathcal{L}_p^k$ el conjunto de los agentes vistos por el p-ésimo agente

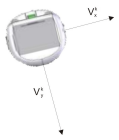


Figura 6.15: Sistema de coordenadas cartesianas relativo al agente.

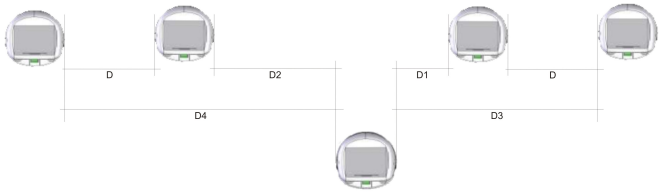


Figura 6.16: Proyección horizontal de la distancia del robot ubicado en el centro, con el resto del enjambre que éste logra sensar.

a su derecha y su izquierda respectivamente en el paso k, la velocidad horizontal de éste puede determinarse como:

$$v_{p_x}^k = \sum_{\mathcal{R}_p^k}(iD - \langle r_i^k, \hat{j_p}\rangle) + \sum_{\mathcal{L}_p^k}(\langle r_i^k, \hat{j_p}\rangle - iD) \tag{6.5}$$

Nótese que este cálculo es independiente de la posición en la formación, aún cuando un robot se encuentre en un extremo de la misma y en el caso de estar formados a

distancia deseada, ésta velocidad será nula.
El cálculo se realiza intentando pertenecer al conjunto factible a las restricciones II del problema planteado en la sección 6.3. Esto es debido a su inmunidad a errores en medidas como fue comentado también en dicha sección.
Como fue formulada, la velocidad horizontal es la encargada de posicionarse "en medio" del espacio entre 2 agentes (en caso de no estar en un extremo), esto significa que la velocidad v_x es la encargada de alcanzar la formación cuando todos los agentes se encuentran alineados. Por este motivo, para completar un algoritmo que produzca la formación deseada, es necesario agregar una componente de adelanto o atraso para poder alcanzar o esperar agentes. Para ello se define la velocidad vertical v_y en función de la cantidad de agentes percibidos tanto detrás como delante del robot. A diferencia de v_x (que toma valores reales), v_y toma valores sobre un conjunto discreto y positivo. La velocidad vertical que adopta el p-ésimo robot en el paso k depende del estado del resto de los agentes y del de sí mismo del siguiente modo:

$$v_{p_y}^k = \overline{v_y} + (F_p^k - B_p^k)\delta \tag{6.6}$$

donde $\overline{v_y}$ es la velocidad media del robot, δ es un "cuanto"de velocidad, F_p^k y B_p^k son la cantidad de agentes que el robot percibe por delante y por detrás respectivamente. Nuevamente, el cálculo de la velocidad en esta dirección es independiente de la posición del agente para con el enjambre.
Con esta forma de velocidad planteada, si el enjambre se encuentra formado, cada robot adquirirá velocidad $\overline{v_y}$. Si un robot se encuentra muy atrás aumentará drásticamente su velocidad e irá decrementándola a medida que se acerca al resto hasta llegar a velocidad $\overline{v_y}$, en cambio, si un robot se encuentra muy por delante del resto, disminuirá su velocidad e irá incrementándola a medida que el resto se le acerca.
Con las velocidades horizontal y vertical puede determinarse la velocidad que el p-ésimo agente adquirirá en el k-ésimo paso de la siguiente forma:

$$\overrightarrow{v_p^k} = \left[\sum_{\mathcal{R}_p^k}(iD - \langle r_{ik},\ \hat{j_p}\rangle) + \sum_{\mathcal{L}_p^k}(\langle r_{ik},\ \hat{j_p}\rangle - iD)\right]\hat{i_p} + \left[\overline{v_y} + (F_p^k - B_p^k)\delta\right]\hat{j_p} \tag{6.7}$$

Por lo que cada agente logrará centrarse en el espacio disponible entre otros dos y adelantarse o atrasarse convenientemente para no romper la alineación, independientemente de su posición final en la formación.
Dos aspectos importantes se encuentran relacionados con la hipótesis que todos los robots se encuentran alienados y orientados hacia la misma dirección. El primero es que la dirección de avance de la formación está determinada por la dirección y orientación inicial de la misma en este estado. El segundo es el peso que posee dicha hipótesis (que ya propuesta la forma con que se determinará la velocidad de cada agente es evidente) la cual de no cumplirse torna al algoritmo ineficaz. Existe un inconveniente que resulta fatal debido a esto y es la falta de unidireccionalidad del enjambre después de iterar, esto significa que por más que los 2 estados previos dejen a cada agente del enjambre en una misma dirección y sentido, la ejecución de este algoritmo no asegura esto mismo, produciendo un error al ejecutarlo sucesivas veces.

Para compensar este inconveniente, es necesaria una corrección del ángulo luego de la iteración para dejarlo como al principio. Para conseguirlo, el agente memoriza el defasaje producido por la velocidad adoptada y al final del paso conoce cuánto debe corregir.

$$\overrightarrow{v_p^k} = \sqrt{||v_{px}^k||^2 + ||v_{py}^k||^2}\Big\langle_{\arctan \frac{v_{px}^k}{v_{py}^k}} = {v_p}^k{}_{\langle \varphi_p^k} \Rightarrow \psi_p^k = -\varphi_p^k \tag{6.8}$$

Debido a que tanto el ángulo de la velocidad como el ángulo de la corrección poseen cierta incertidumbre asociada (ruido de representación de las velocidades, retardos en los pasos utilizados y problemas dinámicos como deslizamiento de ruedas) es natural que la corrección no sea perfecta, dependiendo del agente y del entorno su efectividad.
Para lograr mejores resultados, se puede utilizar la misma técnica con la que se determina el movimiento horizontal de un agente, es decir, sumando las distancias las cuales debería moverse según cada agente sensado pero aplicándolo a la velocidad angular del robot. De este modo un agente tratará de girar de forma tal de encontrarse perpendicular a cada uno de los agentes de su entorno. Matemáticamente, el ángulo de giro (en sentido horario respecto de $\hat{j}$) se determinaría como la suma de las contribuciones de cada agente visible incrementándola o decrementándola según el cuadrante al cual pertenezca el agente (ver figura 6.17).
Formalmente, siendo $\mathcal{C}_p^k = \mathcal{R}_p^k \bigcup \mathcal{L}_p^k$ el conjunto de todos los agentes visibles por el

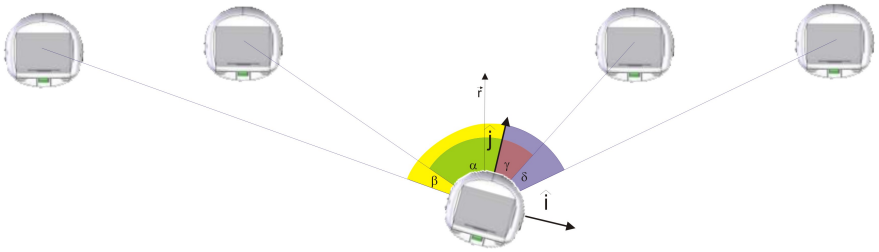


Figura 6.17: Ángulos que forma cada uno de los agentes que logra sensar, relativo a la posición del agente del medio de la figura.

p-ésimo agente en el paso k, el ángulo de corrección resulta:

$$\theta_p^k = \sum_{\mathcal{C}_p^k} \varphi_i^k + \frac{\langle \overrightarrow{r_i^k},\ \hat{i_p}\rangle \langle \overrightarrow{r_i^k},\ \hat{j_p}\rangle}{||\langle \overrightarrow{r_i^k},\ \hat{i_p}\rangle \langle \overrightarrow{r_i^k},\ \hat{j_p}\rangle||}\pi \tag{6.9}$$

por lo tanto, la velocidad angular $\widetilde{\omega_p^k}$ para la corrección del ángulo luego de culminar una iteración es la suma de las correcciones angulares (en este modelo), resultando:

$$\widetilde{\omega_p^k} = \theta_p^k + \psi_p^k = \sum_{\mathcal{C}_p^k} \varphi_i^k + \frac{\langle \overrightarrow{r_i^k},\ \hat{i_p}\rangle \langle \overrightarrow{r_i^k},\ \hat{j_p}\rangle}{||\langle \overrightarrow{r_i^k},\ \hat{i_p}\rangle \langle \overrightarrow{r_i^k},\ \hat{j_p}\rangle||}\pi - \arctan \frac{v_{p_x}^k}{v_{p_y}^k} \tag{6.10}$$

Existe un inconveniente respecto a la visibilidad entre agentes como fue comentado anteriormente y se propuso como solución al mismo un barrido del entorno, el cual es costoso desde el punto de vista temporal. Por lo tanto no es conveniente utilizar este tipo de técnica ya que, por un lado, la topología del entorno percibido cambia rápidamente en comparación con la velocidad del barrido y por otro lado son deseables las trayectorias suaves e ininterrumpidas. Para subsanar este inconveniente, se plantea dotar de memoria a los agentes. Los robots ya utilizaban memoria, por ejemplo para almacenar el defasaje producido por la velocidad, pero era solamente por cuestiones de corrección. Ahora el planteamiento es utilizar memoria para ejecutar el algoritmo que hasta ahora era algebraico.
Memorizando las posiciones relativas de los agentes observados anteriormente un agente puede estimar la posición de otro que no sea detectado actualmente, en función de dónde fue observado por última vez, la estrategia adoptada por el primero y la posible estrategia adoptada por el segundo.
La estimación de datos es una herramienta de gran aplicación en diversas áreas, por lo cual ha sido un tema de gran investigación, sobre todo desde la década del 60, cuando se propuso una técnica de estimación a partir de datos sensoriales denominado filtro de Kalman [18].
Un ejemplo es [19] donde se propone una técnica para la autodeterminación de la posición de un agente robótico, a partir de sensores propios y sensores externos. Mediante la estimación de la posición gracias a los datos recabados se logra una disminución importante en el error cometido respecto a la odometría solamente.
En [21] se utiliza la estimación por parte de un robot para relaizar "SLAM" (localización y mapeo simultáneo) en un entorno desconocido. También se ha propuesto en el "SLAM" multirobot [22].
Otro ejemplo es [20] donde se estima la cantidad de área de silicio necesaria al sintetizar un programa realizado en "C" para un FPGA.
En este caso la estimación es realizada suponiendo que el agente a ser estimado no adquiere grandes velocidades y básicamente las pérdidas son producidas por el sensado del entorno en forma discreta, que es la motivación de esta estrategia. Bajo estas suposiciones, la estimación de un agente se centra, en primera instancia, en la estimación del ángulo al cual se encuentra ya que al no adquirir grandes velocidades puede suponerse, en el corto plazo, dentro del rango de visión del agente que realiza la estimación y exigiendo aún más, próximo a la posición en la que se encontraba. Suponer fija la posición del agente a estimar, conlleva a cometer errores. Sin embargo, esta suposición simplifica notablemente los cálculos, resultando atractiva a pesar de ello.
Del modo comentado, el p-ésimo agente estima la posición del i-ésimo en el paso k, $\widehat{\overrightarrow{r_i^k}}$, en función de su medición en el paso $k-1$ del siguiente modo:

$$\widehat{\overrightarrow{r_i^k}} = \widehat{r_i^k}_{\langle \widehat{\varphi_i^k}} = r_i^{k-1}{}_{\langle \varphi_i^{k-1} - \Delta\varphi_p^k} \tag{6.11}$$

donde r_i^{k-1} es la posición del p-ésimo agente en el paso anterior, es decir, la última posición medida para el agente a estimar. El valor φ_i^{k-1} es el ángulo del i-ésimo agente respecto al agente p en el paso anterior, que es la última medida y $\Delta\varphi_p^k$ es el cambio en el ángulo del agente p entre el paso k y el anterior ($\Delta\varphi_p^k = \varphi_p^k - \varphi_p^{k-1}$).

Como puede verse en la figura 6.18, el estimador del agente p para el agente i recalcula el ángulo al cual se encuentra i suponiéndolo fijo y toma como la distancia a éste la posición anterior, coherente con la suposición realizada. Esta estimación es exacta en el caso que el agente efectivamente se encuentre detenido o gire sobre sí mismo y será razonable en el caso que el agente se traslade en forma relativamente lenta. Con esta medida se logra suavizar el cambio brusco en la topología del entorno

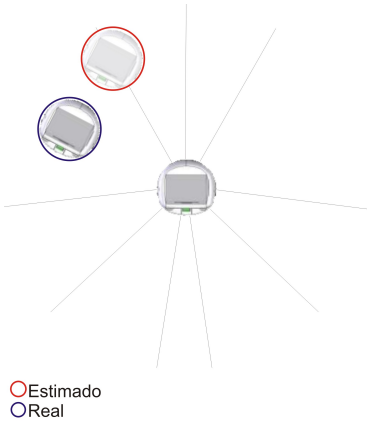


Figura 6.18: Dado que el agente en cuestión (el que se encuentra en el centro de la figura) no logra sensar al otro agente en este paso de iteración, pero si lo había logrado en el paso anterior, supondrá que éste último se encuentra en el mismo lugar donde se encontraba antes. En coordenadas relativas a si mismo, deberá sumar a la distancia y al ángulo que tenía de distancia en el paso anterior, lo que él se haya movido y girado. Es fácil notar que la posición real y la estimada serán iguales únicamente si el agente estimado se queda quieto o gira sobre sí mismo, sin embargo dado que esto no sucede a menudo, la estimación será coherente por un lapso de tiempo, pasado éste tiempo, el robot deberá ser visto nuevamente (por el siguiente sensor) o será perdido.

percibido, debido al sensado discreto y discontinuo de forma tal que los agentes sigan decidiendo sus acciones como lo hacían anteriormente.
Para generalizar la estimación puede determinarse la posición de un agente que hace más de una iteración que no es percibido, manteniendo la distancia a la cual se encuentra y acumulando los cambios angulares (siguiendo las mismas hipótesis).
De este modo la estimación del i-ésimo agente por parte del agente p en el paso k, siendo que la última medición sucedió hace m pasos, resulta:

$$\widehat{\overrightarrow{r_i^k}} = \widehat{r_i^k}_{\left\langle \widehat{\varphi_i^k} \right.} = r_i^{k-1} \left\langle \varphi_i^{k-m} - \sum_{j=1}^{m} \Delta\varphi_p^{k-m+i} \right. \tag{6.12}$$

Existen ciertos problemas alrededor de esto, unos intrínsecos a la estimación y otros propios del estimador (representación del agente). Respecto al problema de estimación, como el sentido común puede indicar, la confiabilidad del elemento estimado decae notablemente al transcurrir las iteraciones ya que el mismo no tiene ningún paso de corrección, como otras técnicas de estimación poseen [19], por este motivo deben tomarse precauciones al utilizar un estimador si el mismo posee gran influencia en el comportamiento de un agente. Podría aplicársele pesos a los datos estimados dependientes de la cantidad de iteraciones desde que los mismos fueron calculados.

Estos pesos pueden tener un comportamiento diferente según las iteraciones transcurridas aunque todos deberán ser altos en pocas iteraciones y bajos para un alto número de ellas. Diferentes estrategias determinarán diferentes comportamientos. Por ejemplo un modelo lineal, en el cual el peso de los datos comience en *1* y decaiga linealmente a *0* en n iteraciones, supondrá que el dato inicialmente es muy confiable y lo será por los siguientes n pasos, decayendo su confiabilidad a una taza constante hasta suponerse no confiable. Un modelo exponencial asignará un alto peso inicial y rápidamente restará importancia a los datos estimados. El modelo elegido para el peso de los datos dependerá del caso y qué se conoce de los elementos que se logran percibir del entorno (La figura 6.19 muestra diferentes tipos de pesos para los datos estimados).

En este caso, el peso de las estimaciones es muy grande, ya que influye en el com-

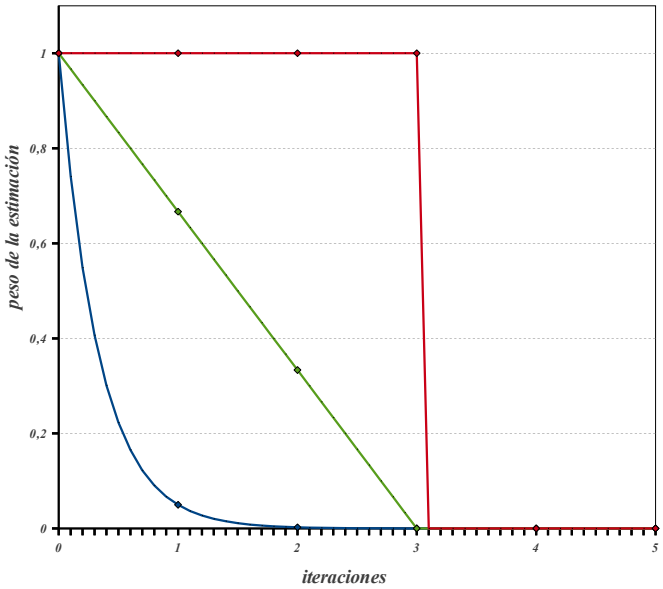


Figura 6.19: diferentes tipos de pesos en función de las iteraciones

portamiento de un agente de igual modo que un agente percibido. Por este motivo, se propone un peso del tipo escalón, el cual asigna al dato estimado un peso igual al de un agente para cierto número de iteraciones y descarta el dato para las siguientes. Al número de iteraciones umbral se lo denomina *persistencia*.

Respecto al estimador, el hecho de estimar la posición de un agente mediante la acumulación de defasajes, incrementa la incertidumbre asociada al cálculo linealmente con las iteraciones. Esto provoca una estimación cada vez menos confiable. Por lo tanto, el incremento en la dispersión del dato estimado conforme evolucionan las iteraciones, se suma a la pérdida natural de confiabilidad propia de un proceso de estimación sin corrección, dándole un mayor peso a lo comentado anteriormente. El valor óptimo de persistencia, entonces, depende íntimamente del problema como del hardware utilizado, por lo que el mismo es conveniente que sea determinado en forma empírica.

De este modo el algoritmo de avance en formación se aplica tanto a los agentes percibidos como a los agentes estimados, recalculando las estimaciones en cada it-

eración, descartando algunas y tomando nuevas. Es necesario un procesamiento más para que no existan conflictos entre los datos reales y los estimados. Puede suceder que un agente del entorno sea percibido por algún sensor y a su vez sea estimada su posición debido a una pérdida de visión reciente de otro sensor (o de ese mismo). De este modo existirán 2 representaciones del mismo agente influyendo en la conducta del robot (ver figura 6.20). Para evitar este inconveniente se realiza un preprocesado previo a la ejecución del algoritmo. El mismo consiste en que una vez realizadas las medidas y estimados los agentes, se descarten aquellas estimaciones las cuales disten menos que cierta tolerancia de una medición real, evitando así redundancia en los agentes presentes.

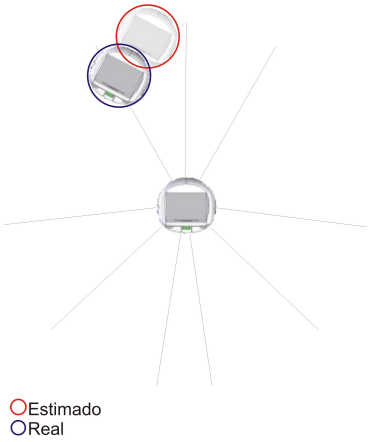


Figura 6.20: Aquí se observa uno de los problemas de esta estimación, un agente puede moverse rápidamente de un sensor a otro, y por lo tanto, ser estimado y visto al mismo tiempo, lo que implicaría un grave error a la hora de ejecutar el algorítmo, dado que, un solo robot es considerado dos veces en posiciones muy cercanas. Para solucionar esto, se supone que una estimación de un robot no puede estar demasiado cerca de un robot real, esto será suficiente dado que, la estimación se realiza por un corto periodo de tiempo y por lo tanto, el robot real no puede haberse alejado mucho de la estimación realizada. En este caso, el robot estimado no solo se encuentra demasiado cerca del real, sino que hasta estarían superpuestos si fueran reales y por lo tanto la estimación será descartada.

6.6.5 Implementación y Simulación

Se implementó el algoritmo de formación tal y como fue comentado anteriormente en el simulador diseñado, los parámetros utilizados fueron los siguientes:

- Entorno
  - Tamaño del entorno: 1500cm × 1500cm
- Robots
  - Cantidad: 5
  - Radio: 40cm
  - Velocidad Máxima: $100\frac{\text{cm}}{\text{s}}$
  - Umbral de Visión: 181cm
- Objetivos
  - Cantidad: 0
- Obstáculos
  - Cantidad: 0

La posiciones iniciales de los robots, en centímetros, fueron:

- Robot1: [777.71 757.59]
- Robot2: [652.31 644.20]
- Robot3: [661.78 904.04]
- Robot4: [971.71 632.38]
- Robot5: [1004.8 833.18]

que pueden ser observadas en la figura 6.21 (se debe tener presente en todas las imágenes del ambiente de simulación, que el origen de coordenadas se encuentra en el ángulo superior izquierdo y todas las coordenadas son positivas).

Esta formación inicial, posee la singular característica que el primer robot que debe moverse (el que está posicionado en el centro) debería hacerlo hacia el medio de los dos agentes que tiene a su izquierda, por ser los que se encuentran más cercanos a él. Sin embargo, si realiza dicha acción, dejará el grafo dividido en dos, por lo tanto, deberá moverse hacia la posición deseada sin abandonar a los agentes que se encuentra a su derecha, su posición se puede observar en la figura 6.22.a.
A continuación deberá realizar su movimiento el segundo robot (se encuentra en el extremo superior izquierdo). En primera instancia debería moverse hacia el centro de los dos robots que logra sensar (se encuentran a la derecha-abajo de este agente). Sin embargo, sus medidas no le permitieron realizar dicho movimiento. Si bien el espacio es suficiente, dada la tolerancia por errores en la medida que se proporciona, el espacio debe ser mayor que el que ocupa un agente. Por esta razón, determinó de manera satisfactoria moverse hacia un extremo de la recta formada por los dos agentes sensados por él, el extremo escogido fue el de arriba, dado que es el mas

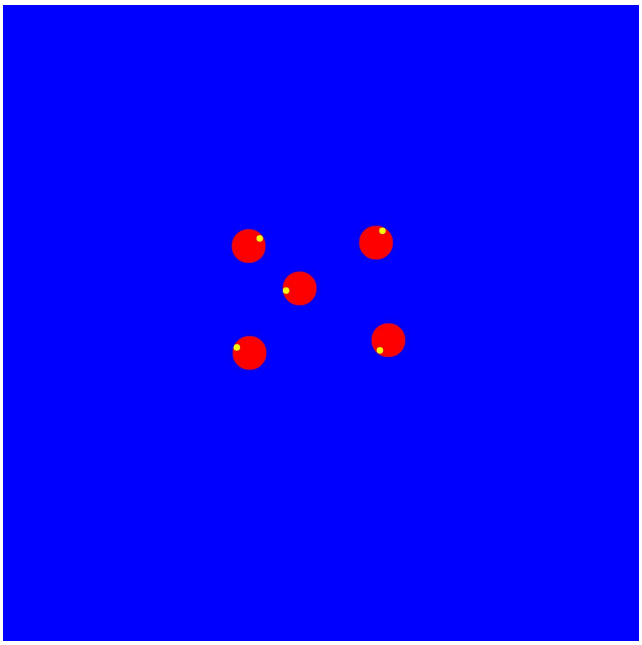


Figura 6.21: Configuración inicial del entorno durante la simulación.

cercano a su posición actual (ver figura 6.22.b).
Luego, quien debe realizar su movimiento, es el robot que se encuentra en el extremo inferior izquierdo. éste solo logra sensar a un agente y por lo tanto se desplaza en la dirección que forma con dicho agente, hasta encontrarse a una distancia D', que será mayor que D (distancia a la que deberán posicionarse para barrer el espacio) para lograr una separación mayor entre agentes en este estado y de esta manera lograr que un agente pueda colocarse entre medio de dos de manera mas sencilla (ver figura 6.22.c).
El siguiente movimiento, será el realizado por el agente que se encuentra en el extremo superior derecho. El mismo se desplazará al medio de los dos robots que logra sensar (ver figura 6.22.d).
Luego se mueve el agente ubicado en el extremo inferior derecho, alejándose del único agente que logra sensar (ver figura 6.22.e).
Luego de esto, el líder (el agente que se movió primero) consulta si todos se encuentran alineados, como su decisión es negativa, continúa el mismo algoritmo, moviéndose él en primera instancia. En este caso, logra sensar a 3 robots, decidiendo ir al extremo inferior de la recta formada por los dos robots que se encuentran arriba a la derecha, sin embargo, dicho punto esta fuera de su umbral de visión y por lo tanto, se dirigirá en dicha dirección pero culminando su movimiento antes de quedar fuera de la circunferencia de visión que poseía antes del movimiento (ver figura 6.22.f).
Luego realizan esta misma estrategia en forma repetitiva hasta quedar formados en linea recta (ver figura 6.23).
Luego de encontrarse en linea recta y que el líder lo informa, se realiza un sorteo entre los 3 robots que se encuentran en el medio, el ganador fue el robot de más abajo, quien decide una dirección para quedar mirando todos hacia el mismo lado (ver figura 6.24).

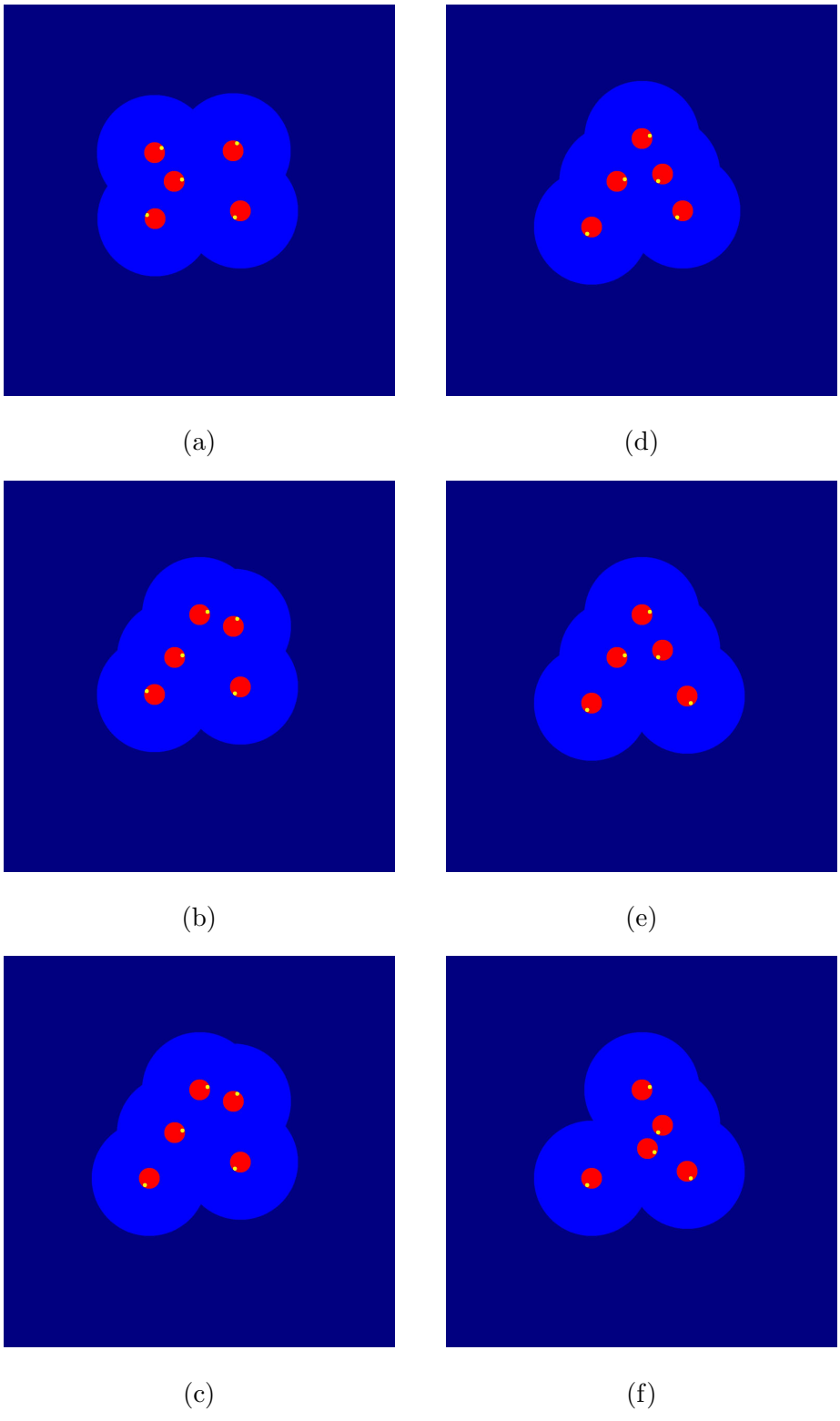


Figura 6.22: Movimientos realizados durante la parte de formación del algoritmo.

Por último, proceden a avanzar (ver figura 6.25).

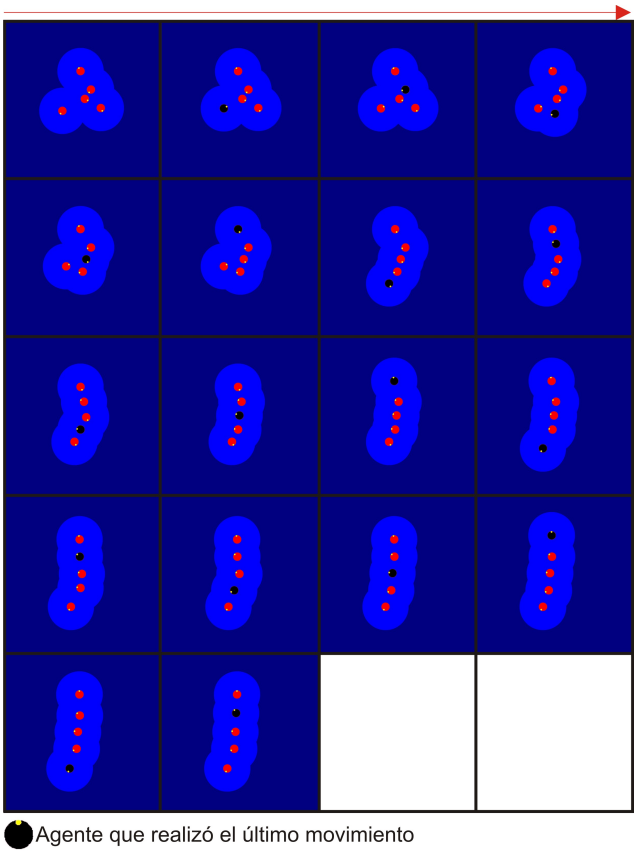


Figura 6.23: Movimientos realizados hasta lograr quedar alineados y pasar al próximo algoritmo (direccionarse).

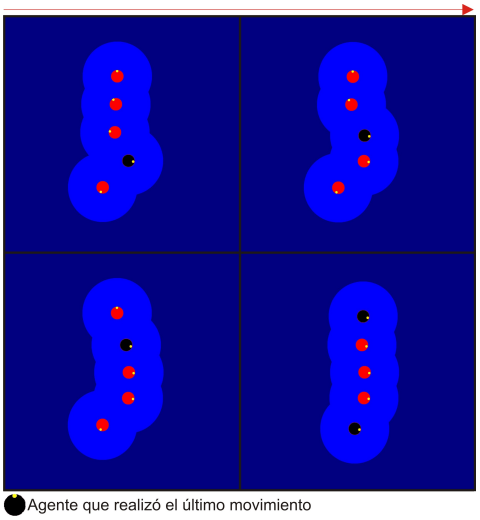


Figura 6.24: Movimientos realizados durante la etapa de direccionarse, para que todos los agentes queden posicionados en la misma dirección.

También se realizaron diversas simulaciones variando parámetros fundamentales del enjambre como son la cantidad de robots y el umbral de visión de los mismos. Se realizaron simulaciones con 3, 5 y 8 robots manteniendo el resto de los parámetros

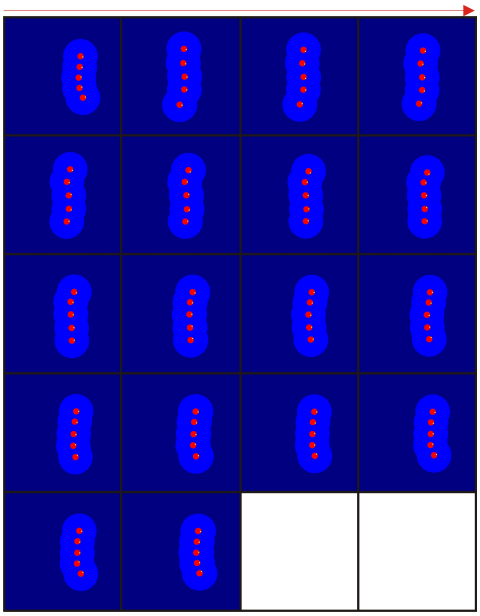


Figura 6.25: Avance de los agentes en linea recta, estando anteriormente ya posicionados en esta configuración.

fijos, con los siguientes valores:

- Entorno
  - Tamaño del entorno: 1500cm $\times$ 1500cm
- Robots
  - Radio: 40cm
  - Velocidad Máxima: $100\frac{\text{cm}}{\text{s}}$
  - Umbral de Visión: 181cm
- Objetivos
  - Cantidad: 0
- Obstáculos
  - Cantidad: 0

Se realizaron 5 simulaciones para cada caso, con posiciones iniciales aleatorias. Se midieron la cantidad de iteraciones totales necesarias para la convergencia y la cantidad de colisiones totales entre agentes. Los resultados se presentan en la tabla 6.26.

Para observar la influencia del umbral de visión, se realizaron 5 simulaciones con 5 agentes en las mismas condiciones que las simulaciones anteriores, variando el parámetro en cuestión. En la tabla 6.28 se presentan los resultados promedio obtenidos.

cantidad de robots	iteraciones promedio	iteraciones promedio por robot	colisiones promedio
3	3	1	0
5	19	4	1
8	210	26	2

Figura 6.26:

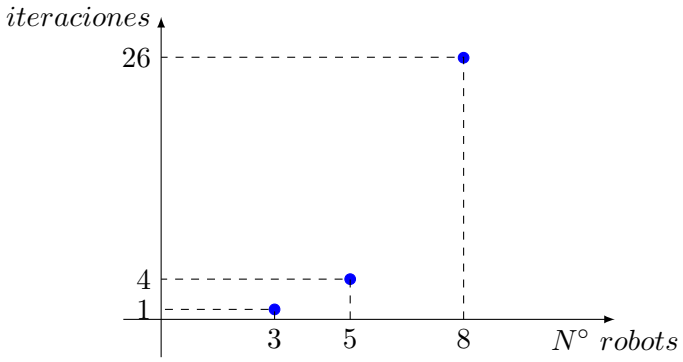


Figura 6.27:

6.7 Comentarios respecto al algoritmo y la colaboración

Cada estado del algoritmo utiliza una estructura organizativa-colaborativa diferente. El estado *1* utiliza una estructura del tipo maestro-esclavo dinámica, en la cual un robot compite con el resto para tomar el rol de maestro, otorgando privilegios a un esclavo a moverse y prohibiendo que el resto lo haga. En la tabla 6.26 puede observarse que la cantidad de iteraciones necesarias para lograr la convergencia en este estado, se incrementa notablemente con la cantidad de agentes. Este fenómeno es esperable, ya que, por un lado, los movimientos se realizan en forma unitaria, lo que deriva directamente en un incremento temporal. Por otro lado, la complejidad del problema se ve incrementada al aumentar la cantidad de agentes ya que, el sistema posee un estado inicial cada vez más desordenado y se pretende lograr el mismo resultado, provocando esto un mayor costo temporal.
Con este tipo de organización, se logra un problema de carácter *offline* en cada paso. El mismo así formulado, requiere comportamientos de gran simplicidad por parte de los esclavos en su accionar, pudiendo abordar problemas sumamente complejos. Como desventaja, este tipo de organización posee, generalmente, baja eficiencia en términos temporales.
Adicionalmente a lo comentado, en la tabla 6.28 puede apreciarse el carácter puramente local del algoritmo implementado. Esto se manifiesta en el incremento en la cantidad de iteraciones, al aumentar el umbral de visión de los agentes. Debido al aumento en la visión, generalmente, el agente logrará sensar una mayor cantidad de robots, por lo que los contemplará al momento de tomar la desición de cómo proseguir. Lo que en un principio puede pensarse como algo favorable para alcanzar el objetivo, en realidad no lo es, dado que el algoritmo opera localmente y es difícil diseñar un criterio para descartar información. Esto es similar al comportamiento observado en el cerebro humano. Cuando el mismo se encuentra concentrado en una tarea específica, descarta toda aquella información que a priori no es relevante y

umbral(cm)	iteraciones promedio	iteraciones por robot	colisiones promedio
181	24	5	1
150	15	3	0
120	15	3	1
90	17	3	1
60	26	5	1

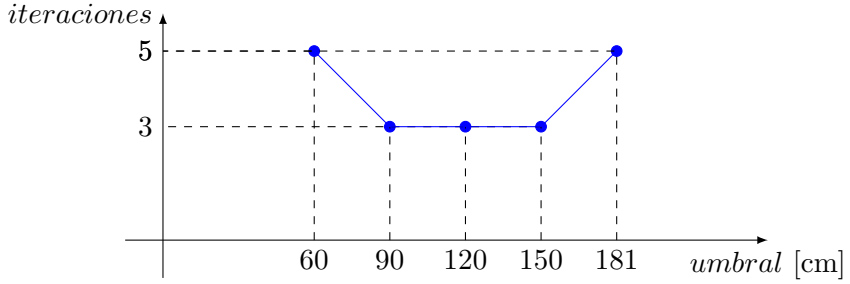


Figura 6.28:

"completa los espacios" extraídos con datos pasados.
Como la lógica indica, un gran decremento en la visión de los agentes hará que los mismos sensen muy poca información del entorno y por lo tanto sea más difícil alcanzar el objetivo.
De lo mencionado anteriormente, se desprende la necesidad de un compromiso entre la cantidad de información que maneja un agente y la que existe en el entorno de trabajo.
El estado *3* utiliza una organización radicalmente diferente al *1*. La misma es una organización del tipo descentralizada y sin líderes, en la cual todos lo agentes se mueven en forma simultánea en un sistema que, evidentemente, no es *offline*. En contraposición al sistema anterior, el problema es abordado en forma integral, logrando costos temporales sumamente menores y obteniendo gran escalabilidad, ya que el comportamiento es independiente de la cantidad de robots y los tiempos no cambian debido al movimiento simultáneo.
La independencia de los robots, unos respecto de otros logra que, ante el eventual cambio en la cantidad de robots (debido a fallas o agotamiento de baterías, por ejemplo), el sistema logre adaptarse satisfactoriamente (si se cumplen las hipótesis anteriormente comentadas).
El estado *2* tiene una organización intermedia entre el esclavismo (organización *1*) y el anarquismo (organización *3*). La misma puede determinarse como una organización con un líder. En esta organización, todos los agentes se comportan del mismo modo y sin restricciones por parte de otros (como el caso con un maestro) pero uno toma iniciativa propia en su movimiento y debido al comportamiento individual poseído, el resto de los agentes lo imitan (los agentes próximos imitan a éste y así se propaga el comportamiento para el resto).
Como puede intuirse, este comportamiento posee un costo en tiempo mayor al estado *3* pero menor que el estado *1*.

7 Gestión de Proyecto

7.1 Resumen

30 *Estudiante:*

Santiago Leandro Martínez Bentancor

- CI : 4.815.092-2
- e-mail : santiagolmb@gmail.com

31 *Estudiante:*

Pablo Ylan Archimowicz Buenavida

- CI : 4.337.802-6
- e-mail : ylanar@hotmail.com

32 *Cliente:*

Grupo de control - Instituto de Ingeniería Eléctrica.

33 *Tutores:*

Pablo Monzón y Rafael Canetti

34 *Fuente de financiación:*

Cliente.

35 *Fecha prevista de finalización:*

Marzo de 2011

36 *Total de horas a realizar previstas por el grupo de proyecto:*

1500 horas

37 *Fecha y descripción de los entregables intermedios:*

1	Documentación del objetivo especifico 1, donde se incluye el estudio realizado y los problemas propuestos con su solución.	10/08/10
2	Documentación del objetivo especifico 2, donde se incluyen las principales características del robot a utilizar y el modelo abstracto propuesto con la justificación de esta elección.	30/10/10
3	Documentación del objetivo especifico 3, donde se incluye el diseño del simulador y su manual de usuario.	30/01/11
4	Documentación del objetivo especifico 4, donde se incluye la definición de los problemas propuestos, sus reformulaciones como problemas de control, el modelado como problema de optimización y por último la elección el algoritmo de resolución junto con la simulación.	15/06/11

1.

Figura 7.1: cronograma de tareas y entregables

7.2 Descripción del proyecto

El proyecto, a rasgos generales, consiste en estudiar, resolver y simular un enjambre de robots. La robótica de enjambres es un nuevo tipo de aproximación para la coordinación de sistemas de modelos basados en agentes, constituidos por un alto número de robots relativamente simples. La meta de esta aproximación es estudiar el diseño de robots (tanto a nivel físico, como de sus conductas de comportamiento), de forma que emerjan patrones de comportamiento colectivo predeterminados mediante las interacciones entre robots y de robots con su entorno, siguiendo el ejemplo de los patrones de comportamiento emergente que se observan en los insectos sociales, llamados inteligencia de enjambre. En este proyecto se hará énfasis en la inteligencia de enjambre, lo que implica un estudio del comportamiento individual de manera que emerjan patrones de comportamiento colectivo deseados.

7.3 Objetivo General del proyecto

Analizar y resolver problemas mediante enjambre de robots y aplicaciones, con la intención de establecer las bases para que tanto los grupos de investigación en el área como el grupo de control las posean en un futuro para desarrollar tareas mas especificas en esta temática.

38 *Definición del problema:*

Encontrar la mejor estrategia de colaboración para que un conjunto de n agentes robóticos se comporte como un enjambre, con el fin de resolver una tarea dada.

39 *Antecedentes:*

Este proyecto aborda un área de estudio importante estudio en otros países, este paradigma de robótica es relativamente nuevo lo cual lo deja con cierta escasez en material bibliográfico en relación con otros pero se encuentre en pleno desarrollo.

40 *Criterios de éxito:*

El 100 % de las tareas formuladas a realizar por los robots son completadas exitosamente cumpliendo los parámetros establecidos en las mismas.

7.4 Actores, supuestos y restricciones

41 *Actores:*

- Grupos de investigación en el área de robótica colectiva.
- Grupo de control

42 *Supuestos:*

- No va a surgir un nuevo paradigma en el área de la robótica que deje a este como obsoleto en el corto plazo.
- Las tareas propuestas evalúan correctamente el desempeño del conjunto de robots.
- Se poseen robots con la tecnología suficiente para ejecutar el proyecto.

43 *Restricciones:*

- El proyecto debe tener una duración no mayor a 1 año y medio.
- El tiempo dedicado al desarrollo de este no puede superar las 1500 horas.
- Las tareas se deben poder realizar con un máximo de 5 agentes.

7.5 Especificación funcional del proyecto

Al culminar el proyecto, se obtendrá un conjunto de herramientas y estrategias las cuales permitirán comprender algunos de los comportamientos colaborativos más conocidos entorno a la robótica de enjambre. Las mismas permitirán generar la noción de cuáles problemas son plausibles de ser resueltos con las estrategias utilizadas y cómo describir dichos problemas como problemas abordables con la metodología utilizada.

7.6 Alcance del proyecto

Adentrarse en el estudio de comportamientos colaborativos haciendo énfasis en los paradigmas mas conocidos y en base a estos realizar la simulación y programación de un enjambre de robots.

N°	Riesgo	Probabilidad	Impacto
1	Un integrante del grupo abandona	baja	extremo
2	Los robots no estarán disponibles cuando se los necesite	media	alto
3	Surge un nuevo paradigma en el área de la robótica que deja a este obsoleto en el corto plazo	baja	medio
4	La estimación de tiempos asignados a cada tarea es menor que el requerido	media	medio
5	Cae el servidor con la documentación y el trabajo realizado hasta el momento	media	extremo
6	Las licencias del software caducan	media	alto
7	Los robots sufren desperfectos y no están disponibles cuando se los necesite	baja	alto

Impacto\Probabilidad	baja	media	baja
bajo			
moderado			
medio	3	4	
alto	7	2,6	
extremo	1	5	

Figura 7.2: Análisis de riesgos

7.7 Análisis de riesgos

44 *Plan de respuestas y acciones preventivas*

- **Riesgo 2**: Se coordinará con Gonzalo Tejera, el encargado del área de robótica del InCo, los horarios en los que se encontrarán disponibles los robots.
- **Riesgo 5**: Se almacenará la información en dos equipos y un pendrive de manera de mantener redundancia de información así en caso de que uno de los equipos se dañe la información estará disponible en otros lugares y no se perderá.
- **Riesgo 6**: Se utilizará, dentro de lo posible, software libre. En caso de usar software privativo se verificará que la fecha de caducidad de la licencia sea, al menos, tres meses mayor que la de finalización del proyecto.
- **Riesgo 1**: Se asume.
- **Riesgo 4**: Se verificará periódicamente ir a la par del Gantt y en caso de necesitar mas tiempo, se reservará un lapso de 30 días.
- **Riesgo 7**: Se determinarán los daños que han sufrido los robots y se evaluará la posibilidad de instanciar el modelo abstracto del robot, obtenido en el objetivo específico II, con las partes del robot que funcionen.

7.8 Cronograma detallado del proyecto

45 *WBS*

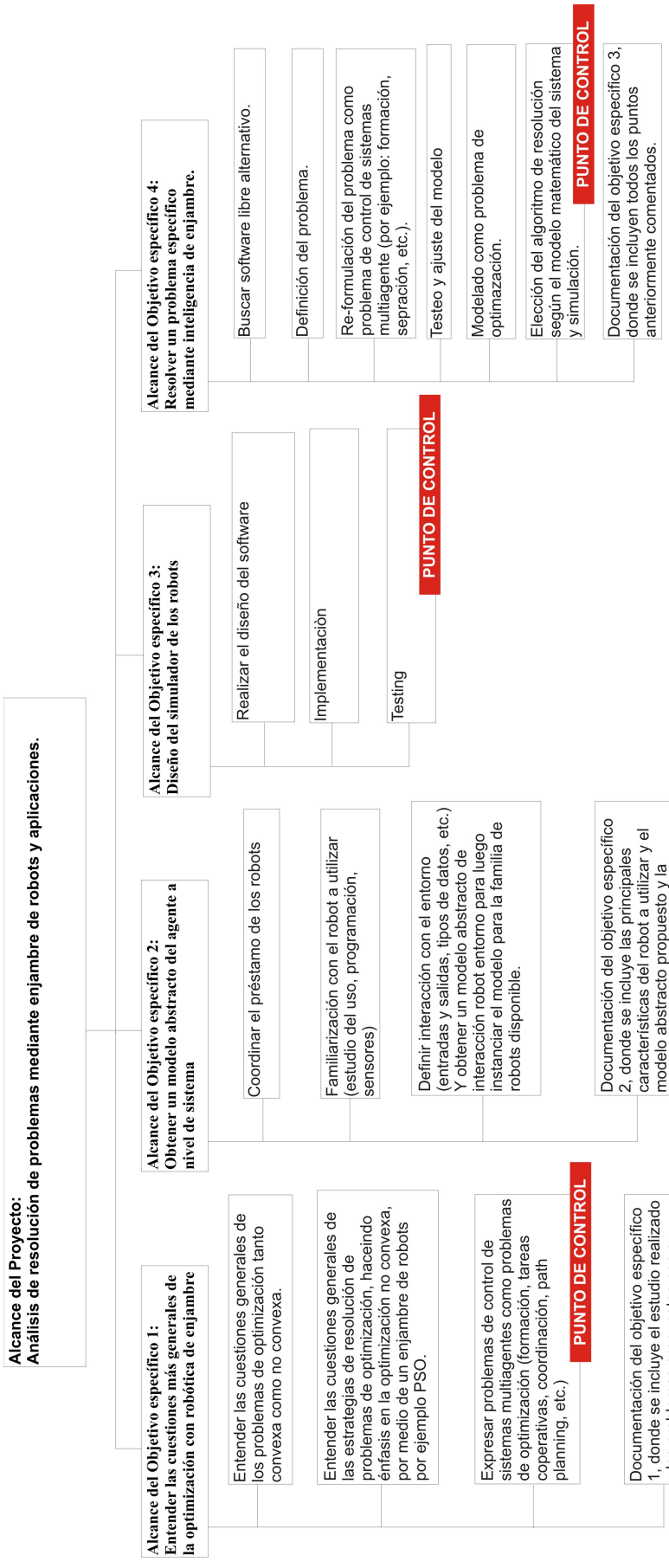
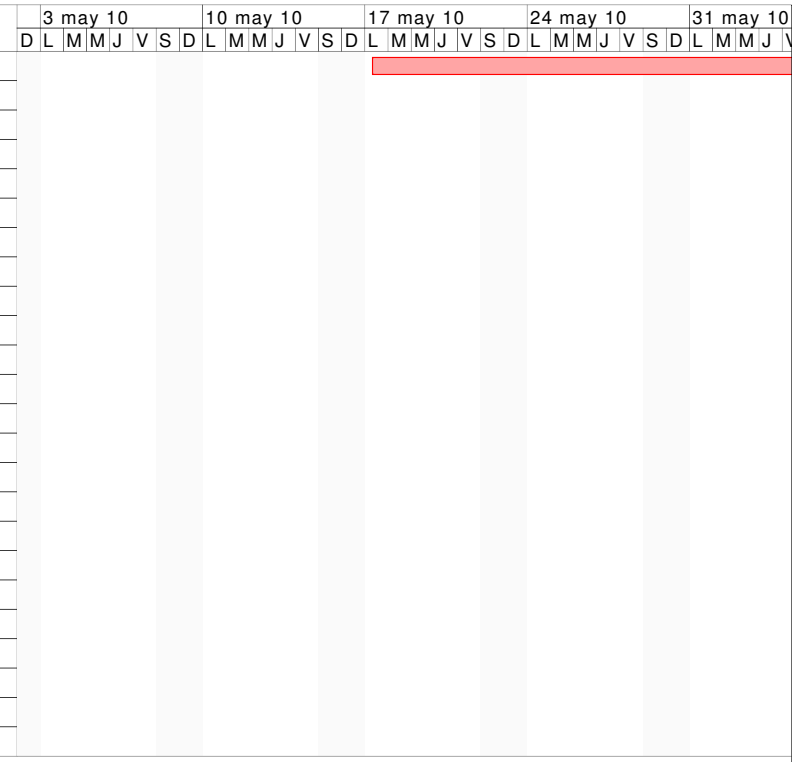


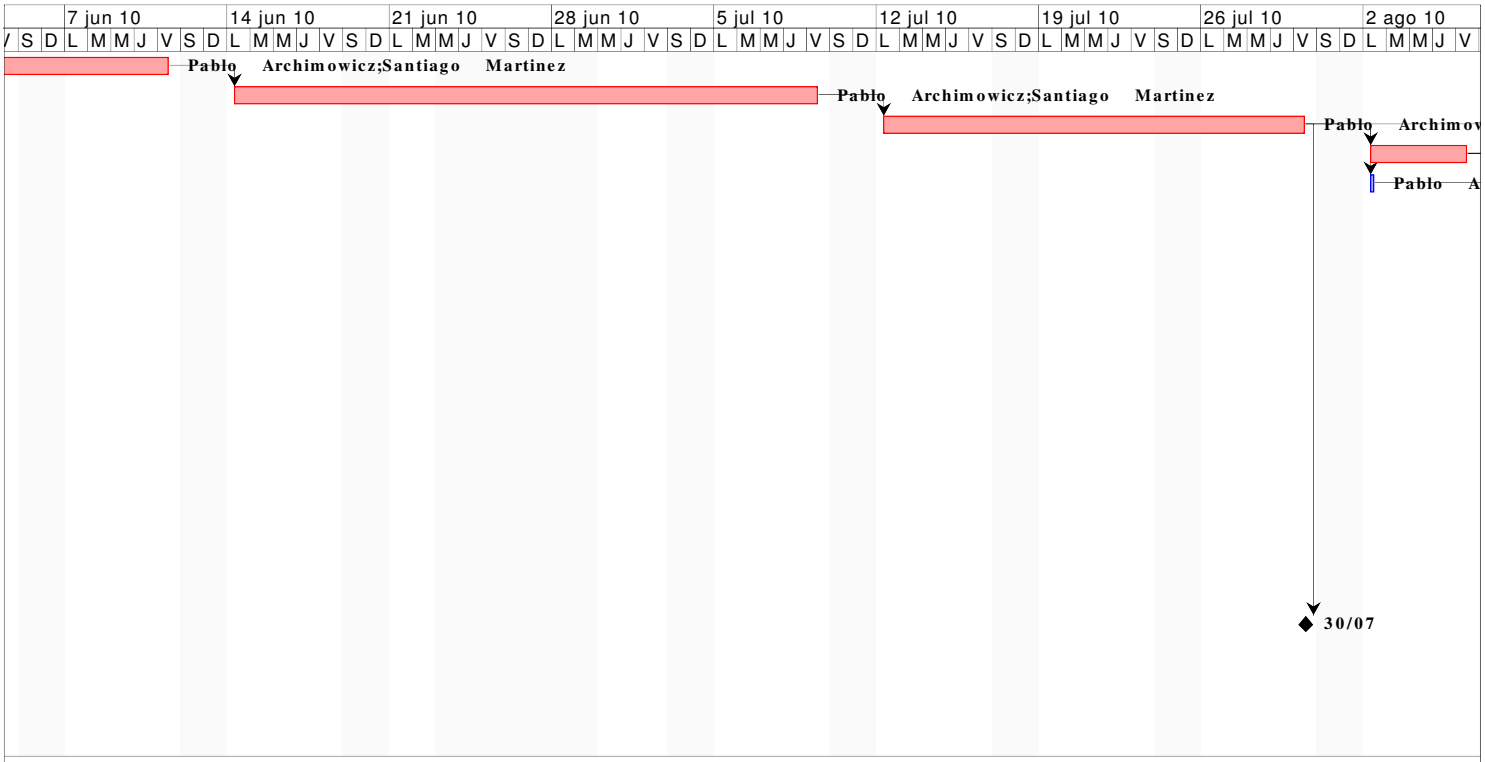
Figura 7.3: WBS

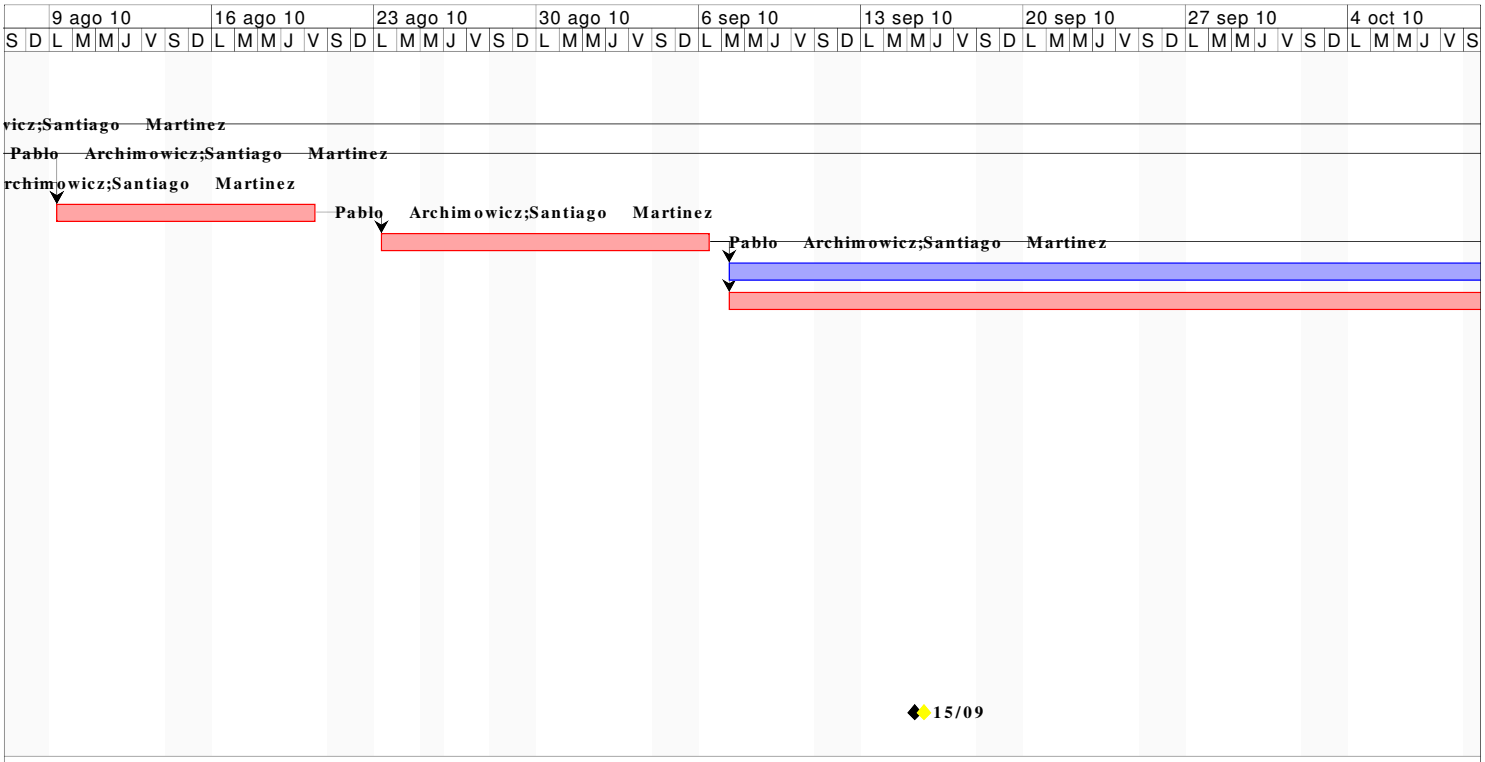
46 *Diagrama de Gantt*

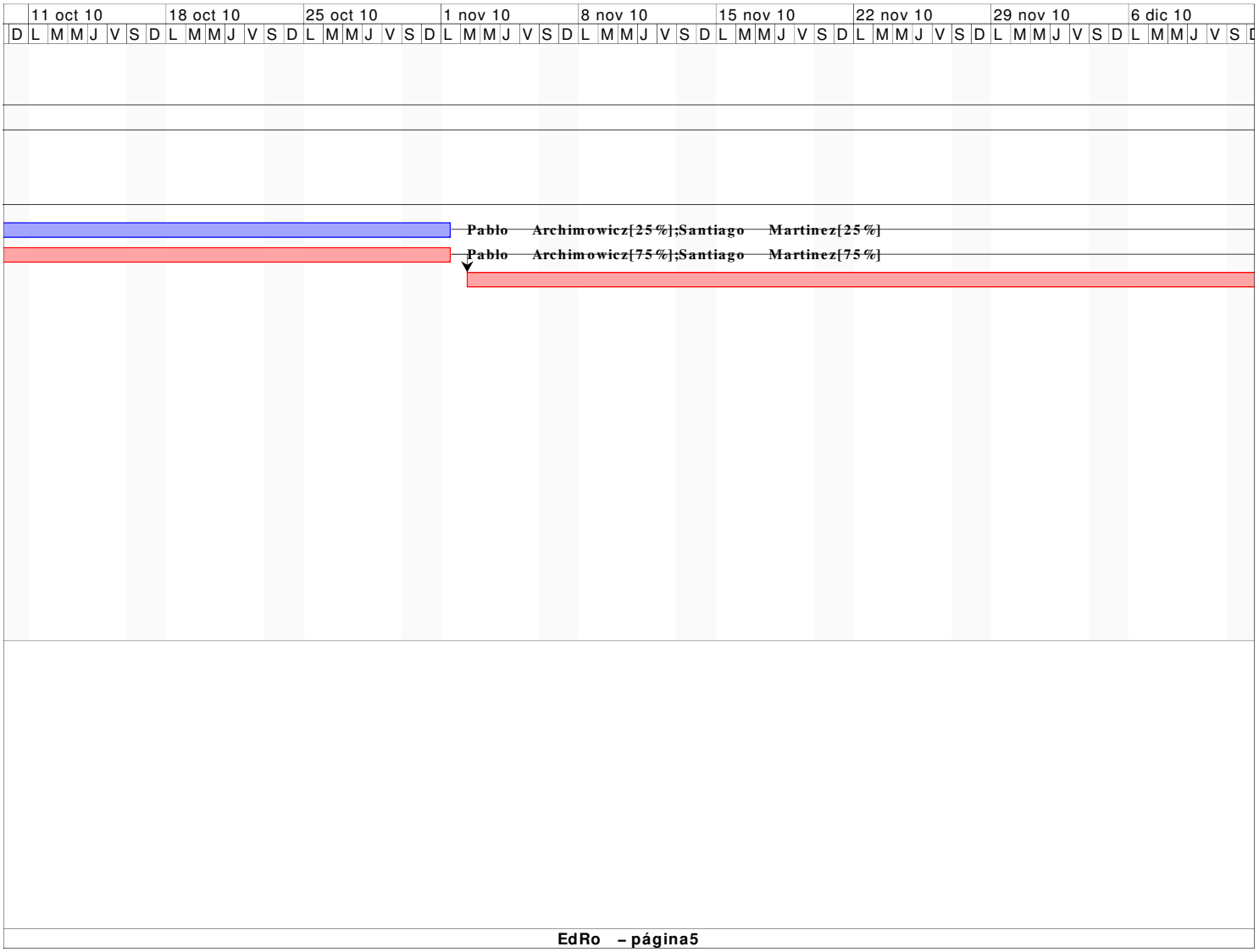
		Nombre	Duración	Inicio	Terminado
1		1.1	20 days	17/05/10 08:00 AM	11/06/10 ...
2		1.2	15 days	14/06/10 08:00 AM	09/07/10 ...
3		1.3	15 days	12/07/10 08:00 AM	30/07/10 ...
4		Documentacion Objetivo Específico 1	5 days	02/08/10 08:00 AM	06/08/10 ...
5		2.1	1 day	02/08/10 08:00 AM	02/08/10 ...
6		2.2	10 days	09/08/10 08:00 AM	20/08/10 ...
7		2.3	10 days	23/08/10 08:00 AM	06/09/10 ...
8		Documentacion Objetivo Específico 2	40 days	07/09/10 08:00 AM	01/11/10 ...
9		3.1	40 days	07/09/10 08:00 AM	01/11/10 ...
10		3.2	45 days	02/11/10 08:00 AM	03/01/11 ...
11		3.3	15 days	04/01/11 08:00 AM	13/01/11 ...
12		4.1	5 days	04/01/11 08:00 AM	06/01/11 ...
13		4.2	10 days	06/01/11 01:00 PM	13/01/11 ...
14		4.3	10 days	14/01/11 08:00 AM	21/01/11 ...
15		4.4	5 days	21/01/11 01:00 PM	26/01/11 ...
16		4.5	10 days	26/01/11 01:00 PM	02/02/11 ...
17		4.6	50 days	03/02/11 08:00 AM	14/03/11 ...
18		Documentacion Objetivo Específico 4	50 days	03/02/11 08:00 AM	14/03/11 ...
19		documentacion del proyecto	30 days	15/03/11 08:00 AM	06/04/11 ...
20		punto de control 1	0 days?	30/07/10 12:00 PM	30/07/10 ...
21		punto de control 2	0 days?	13/01/11 05:00 PM	13/01/11 ...
22		punto de control 3	0 days?	14/03/11 12:00 PM	14/03/11 ...
23		documento de avance 1	0 days?	15/09/10 08:00 AM	15/09/10 ...
24		documento de avance 2	0 days?	15/02/11 08:00 AM	15/02/11 ...

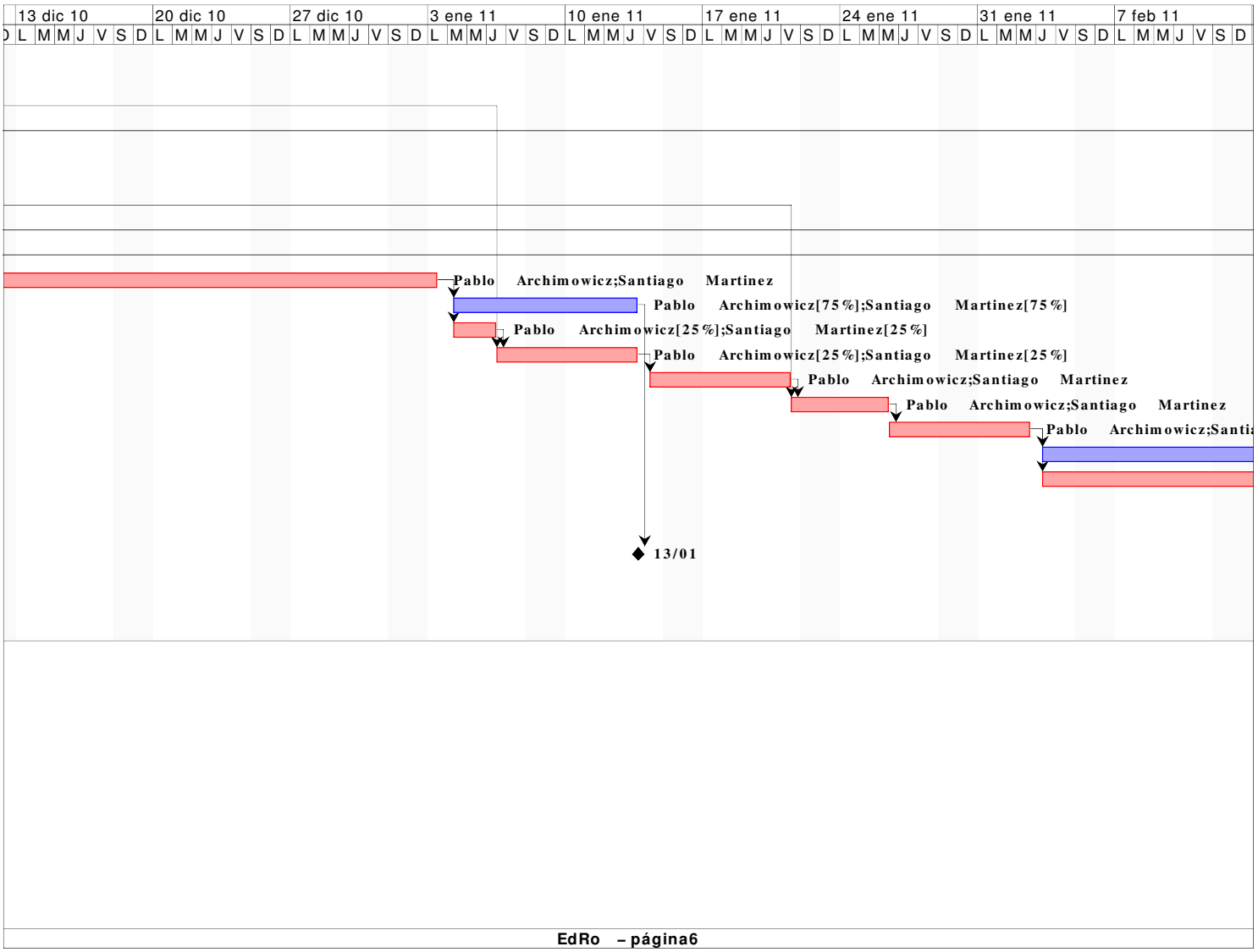
Predecesores	Nombres del Recurso
	Pablo Archimowicz;Santiago Martinez
1	Pablo Archimowicz;Santiago Martinez
2	Pablo Archimowicz;Santiago Martinez
3	Pablo Archimowicz;Santiago Martinez
3	Pablo Archimowicz;Santiago Martinez
4;5	Pablo Archimowicz;Santiago Martinez
6	Pablo Archimowicz;Santiago Martinez
7	Pablo Archimowicz[25%];Santiago Martinez[25%]
7	Pablo Archimowicz[75%];Santiago Martinez[75%]
9	Pablo Archimowicz;Santiago Martinez
10	Pablo Archimowicz[75%];Santiago Martinez[75%]
10	Pablo Archimowicz[25%];Santiago Martinez[25%]
12;3	Pablo Archimowicz[25%];Santiago Martinez[25%]
13	Pablo Archimowicz;Santiago Martinez
14;7	Pablo Archimowicz;Santiago Martinez
15	Pablo Archimowicz;Santiago Martinez
16	Pablo Archimowicz[50%];Santiago Martinez[50%]
16	Pablo Archimowicz[50%];Santiago Martinez[50%]
4;8;9;18	Pablo Archimowicz;Santiago Martinez
3	
11	
17	

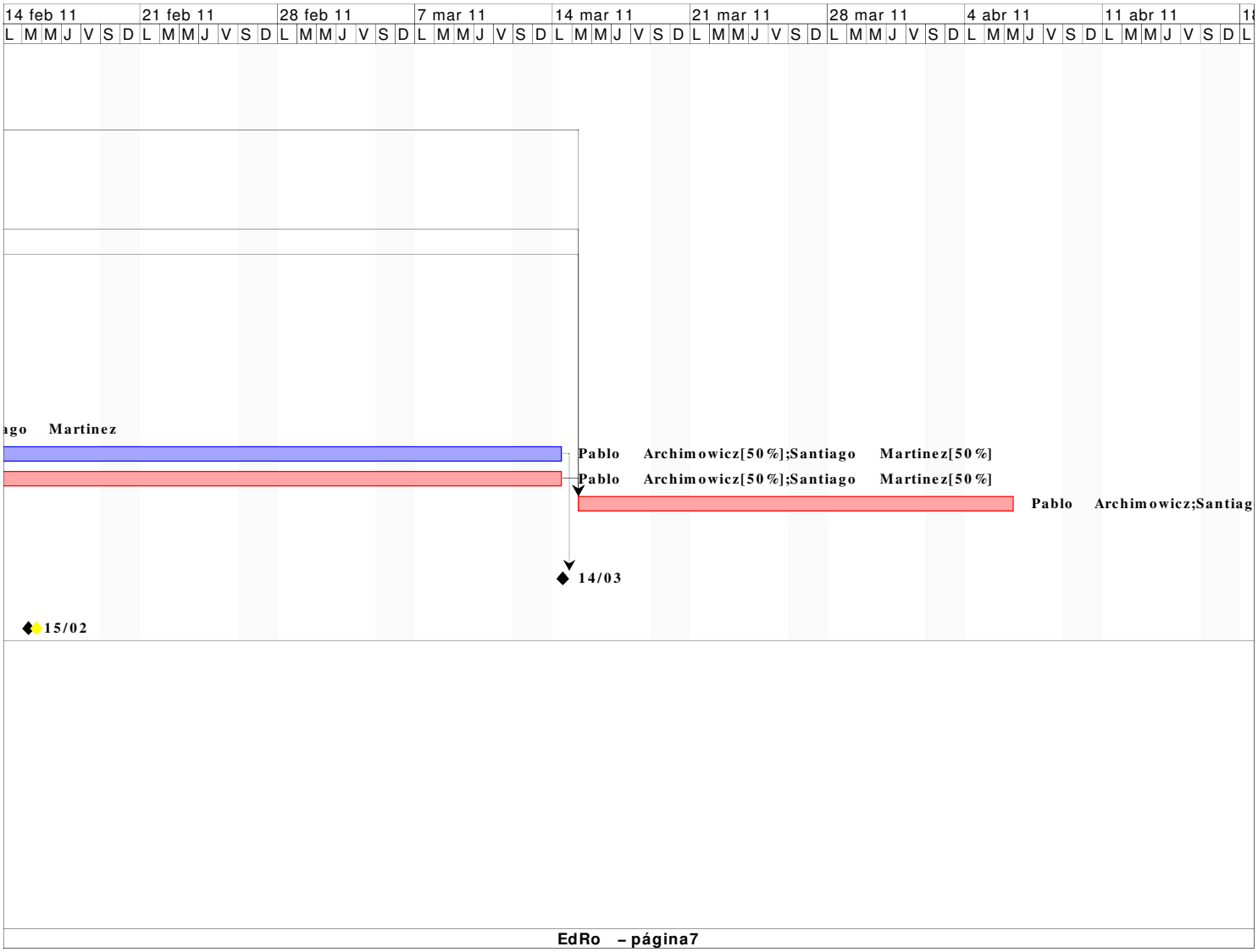



7 jun 10
14 jun 10
21 jun 10
28 jun 10
5 jul 10
12 jul 10
19 jul 10
26 jul 10
2 ago 10
Pablo Archimowicz;Santiago Martinez
Pablo Archimowicz;Santiago Martinez
Pablo Archimov
Pablo A
30/07


9 ago 10
16 ago 10
23 ago 10
30 ago 10
6 sep 10
13 sep 10
20 sep 10
27 sep 10
4 oct 10
S D L M M J V S D L M M J V S D L M M J V S D L M M J V S D L M M J V S D L M M J V S D L M M J V S D L M M J V S
vicz;Santiago Martinez
Pablo Archimowicz;Santiago Martinez
rchimowicz;Santiago Martinez
Pablo Archimowicz;Santiago Martinez
Pablo Archimowicz;Santiago Martinez
15/09


11 oct 10
18 oct 10
25 oct 10
1 nov 10
8 nov 10
15 nov 10
22 nov 10
29 nov 10
6 dic 10
D L M M J V S D L M M J V S D L M M J V S D L M M J V S D L M M J V S D L M M J V S D L M M J V S D L M M J V S D L M M J V S D L M M J V S
Pablo Archimowicz[25%];Santiago Martinez[25%]
Pablo Archimowicz[75%];Santiago Martinez[75%]


13 dic 10
20 dic 10
27 dic 10
3 ene 11
10 ene 11
17 ene 11
24 ene 11
31 ene 11
7 feb 11
Pablo Archimowicz;Santiago Martinez
Pablo Archimowicz[75%];Santiago Martinez[75%]
Pablo Archimowicz[25%];Santiago Martinez[25%]
Pablo Archimowicz[25%];Santiago Martinez[25%]
Pablo Archimowicz;Santiago Martinez
Pablo Archimowicz;Santiago Martinez
Pablo Archimowicz;Santia
13/01


14 feb 11
21 feb 11
28 feb 11
7 mar 11
14 mar 11
21 mar 11
28 mar 11
4 abr 11
11 abr 11
L M M J V S D
ago Martinez
Pablo Archimowicz[50%];Santiago Martinez[50%]
Pablo Archimowicz[50%];Santiago Martinez[50%]
Pablo Archimowicz;Santiag
14/03
15/02

8 Conclusiones

8.1 Conclusiones Generales

En el presente proyecto se estudiaron los fundamentos y conceptos básicos de la colaboración en comunidades, tanto biológicas como robóticas. En particular, se hizo énfasis en un subconjunto de estas, los enjambres, los cuales se destacan del resto debido a su interacción entre agentes y agente-entorno.
Se presentó una técnica formal para la toma de decisiones por parte de robots, mediante la optimización de una función de costos. La misma es muy utilizada y por tanto existe mucha literatura al respecto, lo cual ayudó a generar la noción del tamaño del campo de aplicación de las conductas estudiadas. Para esto, fueron estudiados los conceptos fundamentales de optimización. En forma detallada, se presentaron las técnicas de programación lineal y la teoría básica que yace por detrás, conceptos de programación no lineal y algunos algoritmos destacados.
Como vínculo interesante entre las conductas de enjambre y la toma de decisiones de los agentes robóticos mediante optimización, se estudió muy detalladamente el algoritmo "Particle Swarm Optimization" (PSO), el cual simula un enjambre de partículas para conseguir el óptimo de una función de costos. El algoritmo PSO se encuentra presente en una parte importante de la literatura dedicada a problemas de colaboración robótica.
Es interesante plantear el problema de varios robots que actúan como enjambre, cada uno con una función de costos asociada para determinar su acción. Al momento de minimizar esta función cada agente podría hacerlo mediante PSO. De esta forma, se puede concebir un enjambre en 2 niveles de abstracción diferentes. Estos se manifiestan en la conducta de enjambre que se ejecuta en la realidad, por parte del conjunto de agentes, y también en las reglas de decisión para la acción a tomar, por parte de cada agente, en forma individual.
Se estudiaron un conjunto de herramientas y lenguajes de software para la simulación de algoritmos, implementación de los mismos y presentación de resultados. Particularmente, se estudiaron herramientas basadas en MatLab para ejecutar las simulaciones, modificar sus parámetros y presentar de forma gráfica la evolución del enjambre. También se estudiaron bibliotecas de software basadas en el lenguaje C para un robot particular, Khepera III, mediante las cuales el mismo maneja sensores infrarrojos y ultrasónicos, además de una cámara.
En relación a lo anterior, se estudió el robot Khepera III. Se relevaron sus recursos, tanto hardware (cantidad y tipo de sensores y periféricos) como software (cantidad de recursos para cómputo y almacenamiento, sistema operativo, bibliotecas disponibles, etc.). En forma complementaria, se estudiaron ejemplos provistos por el fabricante y se propusieron y realizaron en forma exitosa un conjunto de implementaciones sim-

ples con la finalidad de determinar los requerimientos necesarios para el simulador requerido.
Se propuso un modelo abstracto de un robot en diferentes niveles de información. El modelo describe la interacción del agente con el entorno, desde el punto de vista del comportamiento del primero. Con esta concepción, el modelo recibe magnitudes cinéticas y espaciales (como la posición del agente, su velocidad, etc.) y entrega las magnitudes sensoriales que sus capacidades le permita. Los diferentes niveles, cinco en este caso, modelan el nivel de información que puede poseer un robot del entorno. Los mismos varían desde un conocimiento absoluto del mismo hasta un nivel de información puramente local (mediante sensores y cámara), con intercambio de datos globales no determinísticos (como el permiso que otorga un líder a moverse, mediante comunicación inalámbrica por broadcast).
Se diseñó e implementó un simulador orientado a la prueba y depuración de conductas colectivas del tipo ejecutado por un enjambre de agentes. El mismo fue implementado en MatLab y en forma acorde al modelo abstracto formulado. Se definieron las capacidades del agente simulado y las reglas de comportamiento del entorno de simulación. Las capacidades del agente simulado son un subconjunto de las poseídas por el robot Khepera III, pensando en una futura implementación de las estrategias planteadas. Se diseñó e implementó una interfaz gráfica para presentar la evolución del entorno de simulación, así como para la modificación de parámetros fundamentales (dimensiones del entorno, cantidad y dimensiones de los agentes, umbrales de visión de los sensores, etc.).
Se propuso un problema "representativo" a resolver (formación en línea recta) por un enjambre de robots y se propusieron algoritmos que resuelven el problema, para cada nivel de información del modelo abstracto de robot utilizado. Para el nivel de información más incompleto, se probó y depuró el algoritmo planteado y se recabaron datos variando parámetros fundamentales del enjambre. El algoritmo propuesto hace uso de un conjunto de estrategias, conceptos y modelos colaborativos-organizativos diferentes, como lo son el barrido del entorno, la estimación, la imitación, la competición y el libre albedrío. Cabe destacar que todos los puntos mencionados son problemas en si mismos.

8.2 Conclusiones Particulares

En nuestra opinión, la ejecución de tareas mediante colaboración robótica, puede resultar atractiva en diversos ámbitos. Nos resulta muy recomendable la utilización de robots en tareas peligrosas para humanos, como rescates en derrumbes, incendios, minería, etc. Además, resulta atractiva la ejecución de tareas repetitivas por parte de robots, como la producción en serie, debido al alto rendimiento de los mismos, además de no sufrir consecuencias debido a lo monótono de la tarea, como puede suceder en humanos. Existen, además, tareas las cuales no podrían siquiera ser ejecutadas por humanos, debido a las condiciones ambientales no aptas para la vida (como la exploración submarina y espacial) donde resulta esencial la utilización de robots.
Creemos que los enjambres de robots, particularmente, tienen cabida en diversas áreas. Por un lado, las tareas que consisten en una cantidad alta de subtareas iguales y simples (como clasificar cierto número de objetos iguales según un criterio), el in-

cremento en la cantidad de agentes realizándola reduciría en forma considerable el tiempo necesario para culminarla. Por otro lado, en tareas de caracterización de un entorno desde cierta perspectiva (como la realización de un modelo térmico o solar del entorno, la realización de Path Planning o SLAM), un aumento en la velocidad y confiabilidad de los datos obtenidos puede obtenerse mediante la dispersión de un número alto de agentes.
Desde otra óptica, el comportamiento simple de cada agente permitiría utilizar robots de muy bajos recursos y por tanto baratos. Por esto y por no poseer dependencia con cada robot en particular (sino con el enjambre), no resultaría importante la pérdida de agentes. Esto, a nuestro entender, refuerza la elección de un enjambre para la realización de todas las tareas anteriormente mencionadas.
A partir de la experiencia adquirida en este proyecto, entendemos que la complejidad de un problema a resolver mediante robótica, podría descomponerse en dos partes. Una de ellas es intrínseca al problema en sí y es con la que uno normalmente estima la complejidad del mismo. La otra parte está asociada al nivel de información que los robots pueden sensar del entorno. Esta última condiciona fuertemente la complejidad del algoritmo, al punto de convertir un problema sencillo (desde el punto de vista de la parte intrínseca) a uno sin solución. A modo de ejemplo, encomendar a un robot la tarea de encontrar un elemento luminoso, en un entorno libre de obstáculos y relativamente pequeño, puede resultar una tarea simple. Sin embargo, la misma resulta imposible de realizar si el robot no posee sensores de luminosidad. Esto pudo apreciarse en el problema abordado, donde la complejidad de los algoritmos propuestos aumentó conforme disminuía el nivel de información del entorno.
A diferencia de otro tipo de técnicas para la resolución de problemas con robots, entendemos que esta técnica obliga a la utilización de un simulador, tanto para la depuración como la validación de un algoritmo. Por un lado, la posibilidad de trabajar con un número relativamente alto de robots, impide la reprogramación de todos en forma periódica y simultánea, cuando se realizan depuraciones a los comportamientos. Por otro lado, como el comportamiento deseado emerge del comportamiento individual, para validar el comportamiento adoptado por el enjambre, es necesario que el mismo evolucione por cierto tiempo relativamente grande, lo que puede resultar impracticable en la realidad. Mediante un simulador puede acelerarse el proceso, obteniéndose resultados en forma mucho más rápida.
En lo que respecta al problema abordado (formación en línea recta), entendemos que el mismo conjuntamente con la solución propuesta es representativo, debido a que a causa del escaso nivel de información del entorno (en el último nivel del modelo formulado) tuvo que ser descompuesto en subproblemas los cuales pueden abordarse utilizando diferentes estrategias y modelos de organización. Ejemplos de estas conductas son la imitación, competencia y libre albedrío. Cada una, además, utilizando estrategias de estimación.
El algoritmo que propusimos surge de la descomposición del problema en tres subproblemas menores, siendo cada uno de estos resuelto mediante las conductas mencionadas anteriormente.
Observamos que la organización por competencia (donde los agentes compiten por el liderazgo para luego comandar al enjambre) resulta en una organización sumamente estructurada y ordenada. Esto provoca que, mediante esta organización, puedan abordarse tareas complejas (como terminar en una configuración con características geométricas estrictas, como una recta, a partir de una disposición espacial suma-

mente desordenada). Sin embargo, el costo de esta se ve reflejado en el tiempo requerido.
En contraposicón a la competencia, observamos que la organización descentralizada y sin líderes (en la cual todos lo agentes se mueven en forma simultánea) resulta en una estructura sumamente rápida y muy escalable, ya que no se perciben diferencias temporales a medida que crece el enjambre. En contraposición, la estructura es más desordenada y por lo tanto es más dificil abordar problemas complejos con este tipo de organización. Sin embargo podemos intuir que, generalmente, el comportamiento individual es más simple comparado al caso anterior.
Por último, queremos destacar el potencial que tiene la colaboración. Supóngase una situación en la cual se desea encontrar cierto objeto en un entorno determinado. Para esto, es posible utilizar el algoritmo de formación planteado, como un paso previo a la búsqueda. Una vez formados, los robots barrerían el espacio tratando de encontrar el objeto. De este modo, el conjunto de robots se puede concebir como un hiperrobot, con un incremento notable en la cantidad de información que puede adquirir del entorno y con la capacidad de procesamiento aumentada tantas veces como robots se utilicen.

8.3 Trabajo Futuro

Como comentamos, este proyecto es una primer aproximación a las conductas de colaboración, basadas en enjambres. Se comentaron y estudiaron diferentes técnicas y estratégias para resolver el problema formulado, pero destacamos que las mismas son problemas en sí mismos. Es por esto que merecen ser evaluadas exhaustivamente con el fin de optimizar el algoritmo planteado y generalizarlas para que puedan ser utilizadas en otros problemas. En particular, destacamos la técnica de estimación con el fin de conseguir una menor incertidumbre en los datos.
También es necesario evaluar la posibilidad de reformular, completa o parcialmente, el algoritmo propuesto, como un problema de optimización. Destacamos al estado 1, formarse, como un gran candidato a ser reformulado, ya que en éste el problema es de carácter *offline*. Por otro lado, también se podría pensar una reformulación para el estado 3, avance, donde se mantendría el proceso de estimación formulado pero la determinación de la velocidad y el ángulo del robot, se determinarían por medio de una función de costos.
Respecto al simulador *EdRo*, podría pensarse la posibilidad de agregarle más capacidades, como la posibilidad de introducir en el entorno obstáculos poligonales y dotar de cinemática tanto a objetivos como a obstáculos.

A Simulador - Manual de Usuario

A.1 Requerimientos

El computador en el que se desee utilizar el simulador debe poseer MatLab®R2009.a. o posterior. Cabe destacar que este requerimiento se debe a que el simulador fue diseñado para esta plataforma, pero puede ser posible que el mismo corra bajo una versión anterior.
Debido a que el simulador fue diseñado en MatLab®, puede ser ejecutado en cualquier arquitectura, ya que el simulador es interpretado y no compilado. Sin embargo, es deseable poseer ciertos recursos para el uso fluido del simulador. Como antecedente, el mismo se desempeñó en forma aceptable en una arquitectura con un procesador AMD Athlon^{TM}X2 de 2,1 GHz y con 3GB de memoria RAM.

A.2 Iniciar el programa

Para iniciar el simulador deben realizarse los siguientes pasos:

- Abrir MatLab®.
- Dirigirse al directorio donde se encuentran los archivos del simulador (utilizar el comando `cd ruta`).
- Ejecutar el archivo `Interfaz`.

Luego de esto se desplegará en pantalla la interfaz gráfica, como se muestra en la figura A.1.

A.3 Parámetros que se pueden modificar

En la imagen A.1 pueden observarse los diferentes parámetros que pueden ajustarse con el fin de realizar la simulación deseada. Los mismos son:

- Entorno - Tamaño del Entorno: Determina el ancho y el largo en centímetros que tendrá el espacio de simulación.
- Robot - Cantidad: Determina la cantidad de robots al momento de la simulación.
- Robot - Radio: Radio de los robots al momento de la simulación.
- Robot - Velocidad Máxima: Módulo de la velocidad máxima que puede alcanzar un robot.

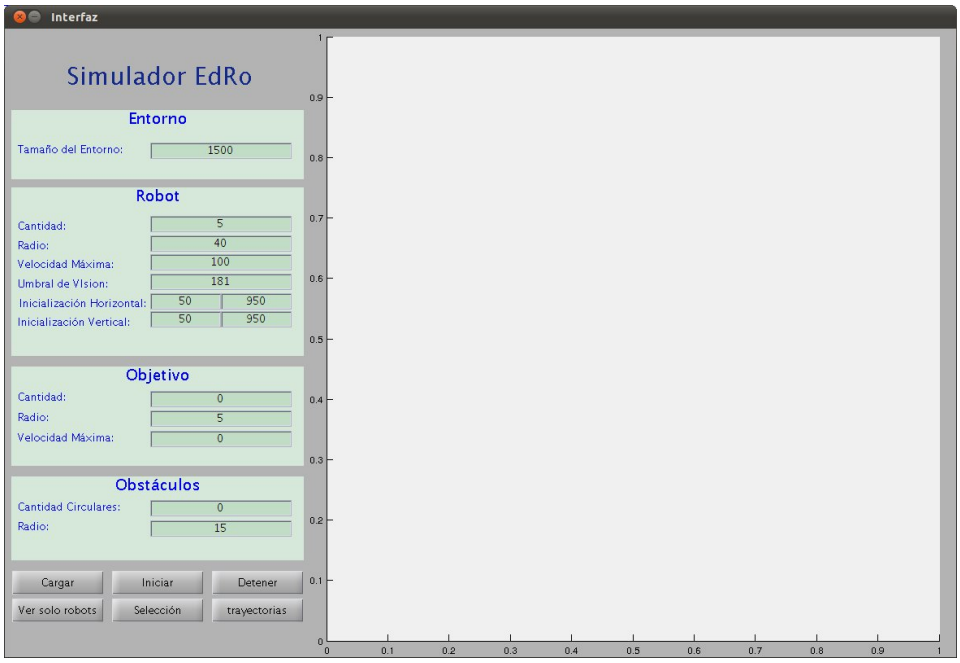


Figura A.1: Interfaz del simulador al momento de iniciar el programa.

- Robot - Umbral de visión: Rango de sensado IR del robot.
- Robot - Inicialización horizontal: En caso de inicialización aleatoria, determina los extremos horizontales donde puede encontrarse el robot.
- Robot - Inicialización vertical: En caso de inicialización aleatoria, determina los extremos verticales donde puede encontrarse el robot.
- Objetivo - Cantidad: Determina la cantidad de objetivos al momento de la simulación.
- Objetivo - Radio: Radio de los objetivos al momento de la simulación.
- Objetivo - Velocidad Máxima: Módulo de la velocidad máxima que puede alcanzar un objetivo.
- Obstáculo - Cantidad Circulares: Determina la cantidad de obstáculos circulares al momento de la simulación.
- Obstáculo - Radio: Radio de los obstáculos circulares al momento de la simulación.

A.4 Simulación

Para comenzar una simulación posicionando en forma automática tanto robots, como obstáculos y objetivos, debe presionarse el botón **Cargar** y posteriormente **Iniciar**. En caso de desearse posicionar manualmente a los robots, debe presionarse el botón **Selección** y a continuación presionar sobre el entorno en el lugar donde se desee ubicar a los agentes, por último presionar **Iniciar**.

Para pausar una simulación en curso se debe presionar el botón **Detener**. Para retomar la misma debe presionarse nuevamente **Iniciar**.
Otras opciones interesantes son las de ver únicamente lo que puede ser visto por los robots, debido a su umbral de visión y examinar las trayectorias realizadas por cada robot en forma simultánea. Estas acciones se realizan presionando los botones **ver solo robots** y **trayectorias** respectivamente.

B Simulador - Guía del Programador

B.1 Introducción

El simulador EdRo es una plataforma de simulación para generar, testear y depurar conductas de comportamiento colectivo para sistemas de modelos basados en agentes. En particular, el simulador permite experimentar conductas asociadas a la robótica de enjambre para la familia de robots khepera III [25].
El simulador permite seleccionar las características del entorno donde se ejecutará la simulación, como ser la dimensión del espacio, cantidad de obstáculos (circulares) y dimensión de los mismos así como la posibilidad de fijar objetivos físicos. Es posible también determinar la cantidad de agentes robóticos a utilizar, las dimensiones de los mismos, su ubicación en el espacio, sus características cinemáticas y características respecto al sensado del entorno, como son los umbrales de recepción de datos de sensores infrarrojos, adquisición y tratado de imágenes, etc.

B.2 Entorno

El entorno es la entidad de mayor dimensión del simulador, en ella se desarrolla la simulación y a ella pertenecen los obstáculos, los objetivos, los robots y el espacio de simulación.
Por esto, el mismo integra dos grandes grupos de elementos. Un grupo lo integran las normas físicas, es decir, la física de los elementos y las reglas de construcción físicas de una configuración de simulación (como ubicación no conflictiva de los elementos y la respuesta ante choques entre agentes). El otro grupo es integrado por las normas de comportamiento de los elementos presentes (algoritmos, procesamiento, etc.).

B.3 Espacio de Simulación

El espacio de simulación es en donde se ubican los elementos involucrados en la simulación, puede decidirse el "terreno". El mismo involucra también a los obstáculos y a los objetivos.
El espacio de simulación posee ciertas dimensiones las cuales pueden ser configurables y tiene bordes que pueden ser leídos por los robots mediante sensado infrarrojo. También es posible, desde el punto de vista del usuario, tener una visión global del

espacio de simulación, pudiendo observarlo en su totalidad u observar sólo las zonas donde los robots pueden hacerlo, para tener mayor comprensión del comportamiento.

B.3.1 Obstáculos

Los obstáculos son elementos que pueden adicionarse al espacio de simulación con el fin de darle complejidad al terreno. Los mismos no poseen comportamiento, por lo cual se encuentran fijos. La cantidad es configurable al igual que sus dimensiones (su diámetro, ya que son circulares).

B.3.2 Objetivos

Los objetivos son elementos similares a los obstáculos, pudiendo decidir sus dimensiones y su cantidad. Lo que diferencia unos de otros es su color, pudiendo ser identificados unos u otros mediante una cámara y no por sensado infrarrojo.

B.3.3 Robots

Son los elementos más complejos presentes en el espacio de simulación. Los mismos se rigen tanto por normas físicas como de conducta.
Desde el punto de vista físico los robots son circulares, indicando su frente con la cámara que poseen. Los mismos poseen:

- nueve sensores IR
- una cámara
- un transmisor-receptor inalámbrico

Sensores IR
El agente robótico simulado posee nueve sensores infrarrojos ubicados a
[10° 45° 80° 135° 180° 225° 280° 315° 350°], como puede observarse en la figura (5.4). Los mismos son capaces de medir distancias a un objeto opaco, ya sea un obstáculo, un objetivo u otro robot.
El haz de "visión" de cada sensor infrarrojo es modelado como un segmento el cual está comprendido en la recta radial al robot que contiene al sensor y de una longitud igual al umbral de visión que este tipo de sensores posee (ver figura B.1).
Desde el punto de vista de su conducta, los robots son capaces de ejecutar un algoritmo precargado para decidir su comportamiento. Para ello, el algoritmo puede hacer uso de "servicios" precargados en el robot que hacen uso del hardware para sensado que él posee.

B.4 Interfaz gráfica

La interfaz gráfica es la forma de presentar los elementos involucrados y su evolución.

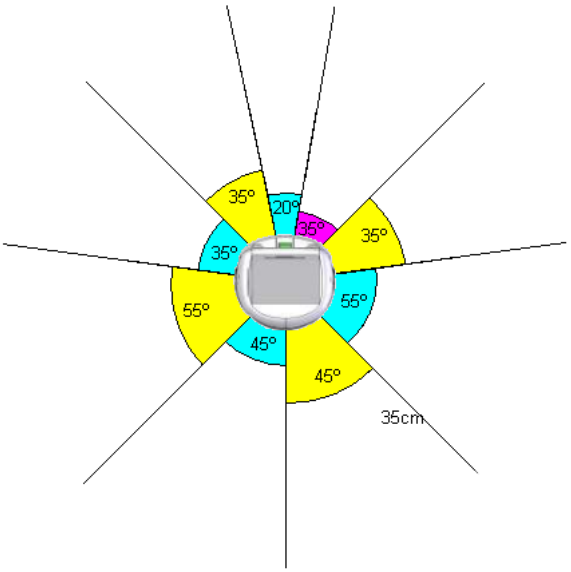
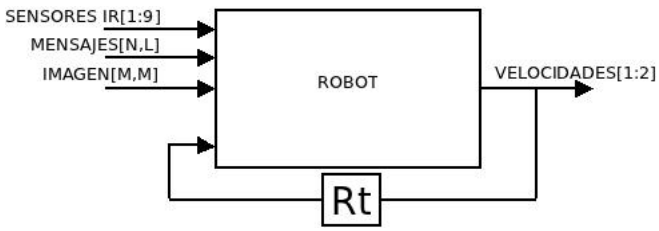


Figura B.1: Modelado de haz infrarrojo y disposición de los sensores alrededor del robot.

B.5 Alcance del Sistema

B.5.1 Planteamiento del problema

El problema radica en poder simular de manera práctica algoritmos en un enjambre de robots. Será necesario poder variar dichos algoritmos con la particularidad de que todos deberán al final devolver una velocidad a cada agente. El simulador deberá soportar la existencia de obstáculos y objetivos en el entorno y tendrá que ser robusto ante choques de los diferentes objetos (robots, objetivos y obstáculos). El robot deberá cumplir el siguiente modelo abstracto:



B.5.2 Justificación

Dentro del marco del proyecto, era necesario realizar simulaciones de los algoritmos planteados para resolver problemas mediante un enjambre de robots. Durante una búsqueda muy exhaustiva se determinó que los simuladores libres que existían hasta el momento, eran poco compatibles con el concepto de enjambre, donde cada robot posee el mismo algoritmo sencillo y no recibe información del exterior, salvo la de sus propios sensores o sus compañeros. Es por esto que se decidió que sería necesario realizar un simulador a medida, que permita ejecutar en cada agente por

separado un mismo algoritmo, que posea las características del modelo abstracto creado para el Khepera III (comentado anteriormente) y que permita crear dentro del entorno ciertos objetos.

B.6 Objetivos

B.6.1 Objetivo general

Crear un simulador capaz de ejecutar un mismo algoritmo en cada robot y desplegar en pantalla el estado actual del entorno.

B.6.2 Objetivos específicos

- Modelar un robot según el modelo abstracto del mismo, donde el comportamiento sea adaptable y posea dimensiones.
- Modelar un objetivo como un objeto con dimensiones.
- Modelar un obstáculo como un objeto con dimensiones.
- Modelar un entorno dinámico que permita manejar robots, obstáculos y objetivos, además de ser robusto ante choques, que se modelarán como elásticos.
- Realizar una configuración que maneje los parámetros del sistema (radio del robot, umbral de visión de sus sensores, etc.).
- Realizar una interfaz donde pueda ser modificada la configuración, y se pueda observar el estado del entorno y sus objetos.

B.7 Diseño del simulador EdRo

B.7.1 Diagrama de clases

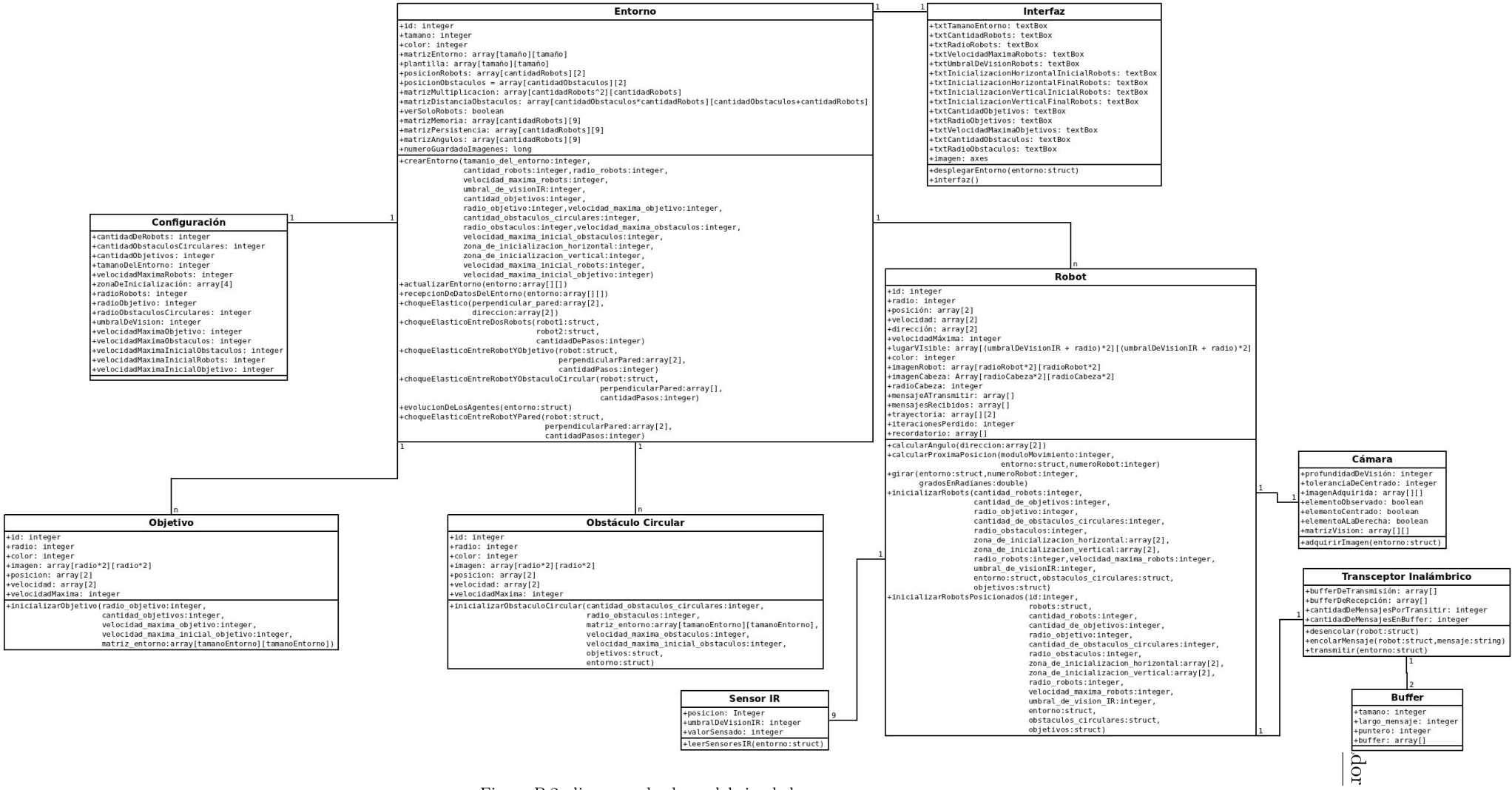


Figura B.2: diagrama de clases del simulador.

1 *Entorno*

El entorno es la clase que contiene a todo el simulador, esta clase posee: robots, objetivos, obstáculos, etc.
Atributos:

- `integer id`: identificador único.
- `integer tamano`: El dibujo del entorno será una matriz cuadrada de este tamaño.
- `integer color`: Es un número entero que representa el color que tomará el entorno al dibujarlo sobre la interfaz gráfica.
- `array[tamano][tamano] matrizEntorno`: Es una matriz cuadrada del tamaño del entorno, en donde se dibujará el valor correspondiente a lo que haya en cada parte del mismo, por ejemplo, en el lugar de un robot se pondrá en esta matriz el entero correspondiente a un robot, en donde no haya nada se pondrá el color correspondiente al entorno, etc.
- `array[tamano][tamano] plantilla`: Es una matriz entorno que tiene dibujado en ella, los obstáculos y objetivos (que son fijos) y de fondo el entorno. De esta manera la construcción de la matriz entorno es más rápida, partiendo de esta plantilla se dibujan solamente los robots y objetos que tienen movimiento.
- `array[cantidadRobots][2] posicionRobots`: Es una matriz que guarda la posición actual de los robots en el entorno para agilizar los cálculos.
- `array[cantidadObstaculos][2] posicionObstaculos`: Es una matriz que guarda la posición actual de los obstáculos en el entorno para agilizar los cálculos.
- `array[cantidadRobots^2][cantidadRobots] matrizMultiplicacion`:
  Es una matriz fija cuya característica principal es que multiplicada por la matriz de `posicionRobots`, devuelve una matriz que representa la distancia entre robots.
- `array[cantidadObstaculos*cantidadRobots]`
  `[cantidadObstaculos*cantidadRobots] matrizDistanciaObstaculos`: Es una matriz que multiplicada por otra matriz de la forma `[posicionRobots:posicionObstaculosCirculares]` devuelve como resultado la distancia entre los robots y los obstáculos, donde las primeras `cantidadRobot` entradas corresponden al primer obstáculo, es decir la distancia del robot 1, 2, 3...`cantidadRobot` al primer obstáculo y en forma consecutiva para cada obstáculo.
- `boolean verSoloRobots`: Es un booleano que representa si en la interfaz gráfica se verá únicamente lo que los robots logran a ver.
- `array[cantidadRobots][9] matrizMemoria`: Es una matriz que representa un recuerdo de aquello que los robots han sensado últimamente, ordenado por cada sensor.

- `array[cantidadRobots][9] matrizPersistencia`: Representa la cantidad de iteraciones que se ha recordado el valor de `matrizMemoria` sin refrescar dicha entrada.
- `array[cantidadRobots][9] matrizAngulos`: Recuerda el ángulo girado del robot, para saber hacia dónde se encuentra el valor guardado en `matrizMemoria`. Lo que se realiza es lo siguiente: cada vez que el robot X sensa un nuevo valor con el sensor Y, se guarda en la entrada `matrizAngulos[X][Y] = anguloSensores[Y]` y cada vez que el robot gire, se restará a todas las entradas el ángulo de giro, de esta manera se sabrá que el ángulo de lo sensado por el robot X con el sensor Y, no necesariamente fue sensado en la dirección en la que se encuentra dicho sensor actualmente, sino que se encuentra en el ángulo que está guardado en `matrizAngulo[X][Y]`.
- `long numeroGuardadoImagenes`: Número entero que recuerda cuántas imágenes fueron guardadas en el disco duro. Esto se utiliza cuando se quiere guardar en disco duro las imágenes de la interfaz gráfica, de manera de ir numerándolas consecutivamente.

Funciones:

- `crearEntorno(integer tamanio_del_entorno, integer cantidad_robots, integer radio_robots, integer velocidad_maxima_robots, umbral_de_visionIR, cantidad_objetivos, integer radio_objetivo, velocidad_maxima_objetivo, cantidad_obstaculos_circulares, radio_obstaculos, velocidad_maxima_obstaculos, velocidad_maxima_inicial_obstaculos, zona_de_inicializacion_horizontal, zona_de_inicializacion_vertical, velocidad_maxima_inicial_robots, velocidad_maxima_inicial_objetivo)`: Inicializa el entorno, inicializando robots, objetivos y obstáculos. Genera un entorno vacío y un conjunto de robots con posiciones y direcciones aleatorias, un conjunto de obstáculos circulares con direcciones y velocidades aleatorias. Para lograr esto, crea los siguientes parámetros:
  - `matriz_entorno`: Es un entorno gráficamente vacío, simplemente es la estructura que representa al mismo y sus elementos. Básicamente es una matriz cuadrada de tamaño `tamanio_del_entorno` que tiene ceros en todas sus entradas.
  - `posicion_robots`: Es la matriz que luego se usará para guardar la posición de los robots para poder acceder rápidamente y calcular la distancia entre ellos. Si esta matriz no fuera creada, habría que realizar una iteración por todos los robots cada vez que sea necesario conocer la ubicación de todos los agentes. Básicamente es una matriz de ceros de `cantidad_robots` $\times$ 2.
  - `posicion_obs_circ`: Al igual que la matriz anterior, esta será usada con el mismo propósito pero para los obstáculos. Básicamente es una matriz de ceros de `cantidad_obstaculos_circulares` $\times$ 2.

- `matriz_multiplicacion`: Es una matriz que multiplicada por `posicion_robots` devuelve la distancia entre ellos, es de tamaño `cantidad_robots^2×cantidad_robots`
- `matriz_dist_obst`: Cumple la misma función que la matriz anterior, pero con los obstáculos.
- `matriz_memoria`: Es una matriz que recuerda para cada robot el último sensado exitoso de cada sensor, es de tamaño `cantidad_robots` × `cantidad_sensores`
- `matrizPersistencia`: Es una matriz que recuerda para cada robot hace cuántas iteraciones fue el último sensado exitoso de cada sensor.
- `matrizAngulos`: Recuerda para cada robot el ángulo que forma la cabeza del robot con el último sensado exitoso de cada sensor. Luego crea la estructura entorno, inicializando los objetivos llamando a la función `inicializar_objetivos`. Inicializa los obstáculos por medio de `inicializar_obstaculos_circulares` y también los robots haciendo uso de `inicializar_robots`.

Para finalizar hace uso de la función `recepcion_de_datos_del_entorno` para generar la `matriz_entorno` con los elementos dibujados en ella.

- `actualizarEntorno(array[][] entorno)` Actualiza la posición de los robots en función de la velocidad que poseen. En caso de presentarse una colisión, la misma será modelada como un choque elástico. En general se divide el tiempo en `cantidad_pasos`. De esta manera se realiza el cálculo de a un paso y se deja un margen de píxeles en los que se considera que si dos objetos se encuentran dentro de ese margen han colisionado. Para esto se definen los siguientes parámetros:
  - `cantidad_pasos`: Explicado anteriormente.
  - `margen`: Explicado anteriormente.
  - `pausa`: Es el tiempo en segundos que se esperará a la interfaz gráfica para que dibuje en pantalla, luego de cada paso.
  - `matriz_diferencias`: Es la multiplicación de la `matriz_multiplicacion` (comentada en el la función `crear_entorno`) y la matriz `posicion_robots`. Quedará ordenada de la siguiente manera, las primeras `cantidad_robots` filas serán del primer robot, las siguientes `cantidad_robots` filas serán del segundo robot (y así sucesivamente), dentro de las filas de cada robot, las entradas serán la resta de las posiciones del robot en cuestión con cada uno de los robots. Tendrá 2 columnas que serán la coordenada X y la coordenada Y respectivamente. A modo de ejemplo, si hay 5 robots la entrada $(3, 1)$ será la resta de las coordenadas X del robot 1 (dado que se está en las primeras 5 filas) y el robot 3; la entrada $(9, 2)$ será la resta de las coordenadas Y del robot 2 (dado que se está en las segundas 5 filas) y el robot 4 (dado que las entrada 6 será la distancia entre el robot

2 y el 1, y así sucesivamente hasta llegar a la entrada 9 que dará como resultado el robot 4).

- `choco_en_este_paso`: Es un vector de unos de tamaño `cantidad_robots` que tendrá ceros a medida que choque un robot. A modo de ejemplo si chocara el robot 3, la entrada `choco_en_este_paso(3)` pasaría a valer cero.

- `distancias`: Es una matriz similar a `matriz_diferencias` pero con los obstáculos. A modo de ejemplo, si hay 5 robots, la entrada $(6, 1)$ será la diferencia de las coordenadas X entre el robot 1 y el obstáculo 2 (las primeras `cantidad_robots` filas pertenecen al obstáculo 1, las siguientes al obstáculo 2 y así sucesivamente)

El algoritmo, como fue mencionado anteriormente, consiste en calcular las colisiones. Para esto se recorre la `cantidad_pasos` determinadas anteriormente, y en cada iteración se realiza lo siguiente:

- Se calculan las colisiones entre robots: Dada la `matriz_diferencias`, todas aquellas entradas que posean como módulo un número menor a 2 veces el radio de los robots más el margen estipulado de antemano, han chocado. Se crean los vectores `robot_1` y `robot_2` que poseen en cada una de sus entradas robots que han chocado entre ellos. A modo de ejemplo, la entrada 3 de ambos vectores, dará como resultado el número de dos robots que han chocado entre ellos.

- Se calculan las colisiones entre robots y obstáculos: Dada la matriz distancias, todas aquellas entradas que posean como módulo un número menor a el radio de los robots más el radio de los obstáculos más el margen estipulado de antemano, han chocado. Se crean los vectores `indices_robots` e `indices_obstaculos` que poseen en cada una de sus entradas robots que han chocado con obstáculos respectivamente. A modo de ejemplo, la entrada 3 de ambos vectores, dará como resultado el número del robot y el obstáculo que han chocado.

- Se calculan las colisiones con el objetivo: Dada las posiciones de los robots y la posición del objetivo, si la distancia entre estos es menor a el radio de los robots más el radio del objetivo más el margen estipulado de antemano, han chocado. Se crea la matriz `indices_objetivo` que posee en cada una de sus entradas el número de aquellos robots que han colisionado con el objetivo.

- Se calculan las colisiones con las paredes: Dada las posiciones de los robots, aquellas posiciones que su coordenada X sea menor al margen (mayor al tamaño del entorno menos el margen), han colisionado con la pared izquierda (pared derecha). Aquellas posiciones que su coordenada Y sea menor al margen (mayor al tamaño del entorno menos el margen), han colisionado con la pared inferior (pared superior).

Para finalizar se realiza una recorrida por todos los robots, dentro de ella se realiza lo siguiente:

- Se cuestiona si dentro del vector `robot_1` hay alguna entrada que sea el robot que se está recorriendo, en caso afirmativo se llama a la función `choqueelastico_entre_dos_robots`, pasando como parámetros los 2 robots que han colisionado y la cantidad de pasos. Esta función variará la velocidad de ambos, según un choque elástico (ver descripción de la función), se colocará en `choco_en_este_paso` en la entrada de cada robot un cero, se modificarán las posiciones de dichos robots en la matriz posiciones que pertenece al entorno (la posición de cada robot ya estará modificada por la función llamada anteriormente).

- Se cuestiona si todavía el robot en cuestión no ha colisionado. En caso afirmativo, se vuelve a preguntar si dicho robot colisiona con el objetivo. En caso afirmativo se llamará a la función `choque_elastico_entre_robot_y_objetivo` pasando como parámetros el robot que ha colisionado, la dirección de la colisión (proveniente de la resta de la posición del robot con la posición del objetivo) y la cantidad de pasos. Esta función variará la velocidad de ambos según un choque elástico (ver descripción de la función). Se colocará en `choco_en_este_paso` en la entrada de este robot un cero, se modificará la posición del robot en la matriz posiciones que pertenece al entorno (la posición del robot ya estará modificada por la función llamada anteriormente).

- Se realiza el mismo cuestionamiento con los obstáculos circulares (llamando a la función `choque_elastico_entre_robot_y_obstaculo_circular`) y con cada una de las paredes (llamando a la función `choque_elastico_entre_robot_y_pared`)

- Para finalizar se actualizan las posiciones y velocidades del robot en cuestión y se guarda su trayectoria y se sale de la iteración de cada robot.

Por último se llama a la función `recepcion_de_datos_del_entorno` para actualizar la matriz entorno, se dibuja la misma en la interfaz gráfica, se espera un tiempo de pausa y se sale de la iteración de la `cantidad_pasos`.

- `recepcionDeDatosDelEntorno(array[][] entorno)` Actualiza la matriz descriptiva del entorno para presentar el entrono actual gráficamente. Para esto se parte de la planilla del entorno con los objetos quietos, se realiza una copia llamada `entornoNuevo` y se comienzan a colocar los objetos móviles. También se crea un entorno "negro" que será utilizado en el caso que la opción de ver solo robots esté activa. Para esto se crea una matriz `entornoNegro` de ceros del mismo tamaño del entorno y se irá llenando con unos en las zonas visibles. Se recorren todos los robots haciendo lo siguiente en cada uno:

  - Se dibuja sobre el `entornoNuevo` la imagen del robot centrada en la posición del robot.

  - Se dibuja sobre el `entornoNuevo` la cabeza del robot centrada en la posición de la cabeza del robot.

- Se coloca la circunferencia de lo que es visible por el robot, en el entorno negro, centrada en la posición del robot en cuestión. Se debe tener en cuenta, que si el robot está muy cerca de la pared, lo que logra ver no es una circunferencia, sino que será esta misma circunferencia recortada.
- Se recorren los obstáculos circulares y se coloca la imagen del obstáculo de a una en el `entornoNuevo`, centrada en la posición del obstáculo.
- Se recorren los objetivos y se coloca la imagen del objetivo de a una en el `entornoNuevo`, centrada en la posición del objetivo.
- Para finalizar se cuestiona si está activada la opción `verSoloRobots`. En caso afirmativo se procederá a multiplicar entrada a entrada las matrices `entornoNuevo` y `entornoNegro`. Esto devolverá un entorno de ceros en los lugares que no son visibles y lo que había en `entornoNUevo` en los lugares que si lo son.

- `choqueElastico(array[2] perpendicular_pared, array[2] direccion)`: Esta función calcula el ángulo de salida luego de un choque, el modelo físico utilizado es el de un choque completamente elásticos. Para esto utiliza dos parámetros

  - `direccion`: Es la dirección en la que se estaba moviendo el objeto que choca. En el caso particular de un robot, es el vector que une su centro con su cabeza.
  - `perpendicular_pared`: Es un vector perpendicular a la tangente del elemento con el que se colisionó. En el caso particular de dos robots es el vector que une el centro del robot con el centro del robot al que choca.

  En general el algoritmo calcula el `angulo_incidente`, que es la diferencia entre el ángulo de la dirección en la que se mueve el robot menos el ángulo de la perpendicular a la pared. El ángulo reflejado será la diferencia entre el ángulo del vector perpendicular a la pared con el `angulo_incidente`. Esto funcionará en la mayoría de los casos, salvo cuando colisionan dos robots, dado que son dos objetos en movimiento. Uno de ellos puede estarse moviendo en dirección contraria a la del choque (ver figura B.3.a) o no (ver figura B.3.b). Si se observa detenidamente, la única diferencia entre ambos casos es que cuando el robot es chocado (se encontraba moviéndose en dirección contraria al choque), el ángulo resultante es igual al ángulo resultante en el caso que el robot choca (se encontraba moviéndose en dirección al choque), con una diferencia de π grados. Para encontrar el caso en el que es necesario sumarle π, se verá el caso borde (ver figura B.4) donde el robot se encuentra en el límite de chocar o ser chocado. Observando el segmento de recta azul de dicha figura, la misma tendrá un módulo mayor en caso que el robot sea chocado y menor en el caso que el robot choque, el módulo límite será entonces $\sqrt{4r^2 + 16/25r^2}$. En caso de ser mayor no se deberá adicionar π, en caso contrario si se deberá hacer.

- `choqueElasticoEntreDosRobots(struct robot1, struct robot2, integer cantidadDePasos)`: Esta función modela el choque elástico entre

dos robots, recibe como parámetros los dos robots que han colisionado y la cantidad de pasos en la que se está integrando (ver descripción de función actualizar entorno para detalles sobre este parámetro). El algoritmo calcula primero el choque para el robot 1, para esto halla la perpendicular al choque como la resta de las posiciones de ambos robots, lo que resulta en un vector que debe apuntar hacia afuera del robot en cuestión. Una vez obtenido esto, llama a la función `choque_elastico` para calcular el ángulo reflejado, una vez determinado dicho ángulo, se calcula nueva velocidad, posición y dirección. Para finalizar, realiza lo mismo pero para el robot 2 (la perpendicular debe apuntar hacia afuera del robot 2 en este caso).

- `choqueElasticoEntreRobotYObjetivo(struct robot, array[2] perpendicularPared, integer cantidadPasos)`: Esta función modela el choque elástico entre un robot y un objetivo, recibe como parámetros el robot que ha colisionado, la perpendicular al choque y la cantidad de pasos en la que se esta integrando (ver descripción de función actualizar entorno para detalles sobre este parámetro). El algoritmo modifica el módulo de la perpendicular al choque para que sea 2 veces el radio del robot, de manera de poder utilizar luego la función `choque_elastico_entre_robot_y_pared`.
- `choqueElasticoEntreRobotYObstaculoCircular(struct robot, array[] perpendicularPared, integer cantidadPasos)`: Esta función modela el choque elástico entre un robot y un obstáculo, recibe como parámetros el robot que ha colisionado, la perpendicular al choque y la cantidad de pasos en la que se esta integrando (ver descripción de función actualizar entorno para detalles sobre este parámetro). El algoritmo modifica el módulo de la perpendicular al choque para que sea 2 veces el radio del robot, de manera de poder utilizar luego la función `choque_elastico_entre_robot_y_pared`, que realizará la tarea que necesita resolver esta función.
- `evolucionDeLosAgentes(struct entorno)` Esta función es la encargada de proporcionar al robot la próxima velocidad según un algoritmo deseado por el usuario. Se puede observar el algoritmo utilizado para la alineación y el avance en el resto de la documentación.
- `choqueElasticoEntreRobotYPared(struct robot, array[2] perpendicularPared,integer cantidadPasos)`: Esta función modela el choque elástico entre un robot y una pared, recibe como parámetros el robot que ha colisionado, la perpendicular a la pared y la cantidad de pasos en la que se esta integrando (ver descripción de función actualizar entorno para detalles sobre este parámetro). El algoritmo modifica el módulo de la perpendicular al choque para que sea 2 veces el radio del robot, de manera de poder utilizar luego la función `choque_elastico`, que devuelve el ángulo de salida luego del choque. Por último actualiza la velocidad posición y dirección del robot.

2 *Objetivo*

El objetivo es un tipo de objeto que pueden o no pertenecer al entorno. Atributos:

(a) (b)

Figura B.3: Este es el caso en el que dos robots colisionan. En el caso (a) el robot de arriba es colisionado, es decir, se mueve contraria al choque. En el caso (b) ambos robots se dirigen hacia el punto de choque. β es el ángulo de la dirección del choque y α es la diferencia entre la dirección que llevaba el robot y el ángulo de la dirección del choque. El vector rojo será la dirección que tomará el robot luego del choque.

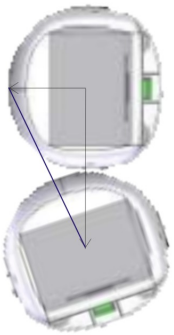


Figura B.4: Este es el caso límite entre chocar y ser chocado, dado que el robot de arriba se encuentra en dirección perpendicular al choque. Aquí se puede observar que si el segmento azul tiene módulo mayor a este caso, entonces el robot fue chocado, en caso contrario el robot ha chocado.

- `integer id`: Identificador único.
- `integer radio`: Número entero que en centímetros representa el radio del objetivo.
- `integer color`: Número entero que representa el color que tomará el objetivo al dibujarlo en la interfaz gráfica.
- `array[radio*2][radio*2] imagen`: Matriz de unos en una circunferencia de radio `radio` y ceros en el resto.
- `array[2] posicion`: Vector que representa la posición del objetivo en las coordenadas cartesianas del entorno.
- `array[2] velocidad`: Vector que representa la velocidad del objetivo en coordenadas cartesianas como las del entorno pero centradas en el objetivo.
- `integer velocidadMaxima`: Número que representa el módulo máximo de la velocidad que puede tomar el objetivo.

Funciones:

- `inicializarObjetivo (integer radio_objetivo, integer cantidad_objetivos, integer velocidad_maxima_objetivo, integer velocidad_maxima_inicial_objetivo, array[tamanoEntorno][tamanoEntorno] matriz_entorno)`: Inicializa el estado de los objetivos, dándole una posición y velocidad aleatoria. Para realizar dicha acción, crea la imagen del objetivo como una matriz del color del objetivo en una circunferencia del radio del objetivo y unos en el resto de las entradas.

Obstáculo: Los obstáculos son un tipo de objetos que pueden pertenecer al entorno. Atributos:

- `integer id`: Identificador único
- `integer radio`: Número que en centímetros representa el radio del obstáculo.
- `integer color`: Número que representa el color que tomará el obstáculo al dibujarlo en la interfaz gráfica.
- `array[radio*2][radio*2] imagen`: Matriz de unos en una circunferencia de radio `radio` y ceros en el resto.
- `array[2] posicion`: Vector que representa la posición del centro del obstáculo en las coordenadas cartesianas del entorno.
- `array[2] velocidad`: Vector que representa la velocidad del obstáculo en coordenadas cartesianas iguales a las del entorno pero centradas en el obstáculo.
- `integer velocidadMaxima`: Número entero que representa el módulo máximo de la velocidad que puede tomar el obstáculo.

Funciones:

- `inicializarObstaculoCircular (integer cantidad_obstaculos_circulares, integer radio_obstaculos, array[tamanoEntorno][tamanoEntorno] matriz_entorno, integer velocidad_maxima_obstaculos, integer velocidad_maxima_inicial_obstaculos, struct objetivos, struct entorno)`: Inicializa el estado de los obstáculos, dándole una posición y velocidad aleatoria. Para realizar dicha acción, crea la imagen del obstáculo como una matriz del color del mismo en una circunferencia del radio del obstáculo y unos en el resto de las entradas.

Cámara: Es una clase que representa la cámara del robot. Atributos:

- `integer profundidadDeVisi\'on`: Es un número que en centímetros representa hasta qué distancia se puede distinguir el objetivo en la imagen de una cámara.
- `integer toleranciaDeCentrado`: Es un número entero que en grados representa qué tan alejado del centro de la cámara se puede encontrar un objeto para decidir que el mismo está centrado.

- `array[][] imagenAdquirida`: Es una matriz que representa la imagen adquirida por la cámara.
- `boolean elementoObservado`: Es un número booleano que determina si el objetivo fue observado por la cámara.
- `boolean elementoCentrado`: Es un número booleano que determina si el objetivo fue observado en el centro de la imagen según la tolerancia aceptada.
- `boolean elementoALaDerecha`: Número booleano que determina si el objetivo se encuentra fuera de la tolerancia hacia la derecha.
- `array[][] matrizVision`: Es una matriz de unos en un cono de 60 grados y ceros en el resto. Representa el lugar donde la cámara toma imágenes.

Funciones:

- `adquirirImagen (struct entorno)`: Esta función actúa como el procesamiento de la imagen. La idea es determinar si hay un objetivo a la vista y en caso afirmativo determinar si el mismo se encuentra sobre la derecha, la izquierda o en el centro, con cierta tolerancia en este último caso.

Buffer: Es una clase que representa el buffer de un transeptor inalámbrico. Atributos:

- `integer tamano`: Número entero que representa el tamaño del buffer.
- `integer largo_mensaje`: Número entero que representa el largo que debe tener un mensaje.
- `integer puntero`: Número entero que representa el próximo mensaje a leer en el buffer.
- `array[] buffer`: Vector de mensajes.

Transceptor Inalámbrico Es una clase que representa la transmisión $WiFi$ del robot. Atributos:

- `array[] bufferDeTransmisi\'on`: Es una matriz de mensajes a transmitir.
- `array[] bufferDeRecepci\'on`: Es un arreglo de mensajes recibidos.
- `integer cantidadDeMensajesPorTransitir`: Número entero que representa los mensajes que quedan por enviar.
- `integer cantidadDeMensajesEnBuffer`: Número entero que representa los mensajes que hay en el buffer.

Funciones

- `desencolar (struct robot)`: Esta función devuelve el último mensaje del buffer transmisor y lo elimina del mismo.
- `encolarMensaje (struct robot, string mensaje)`: Esta función agrega un mensaje al buffer transmisor y mueve el puntero al lugar correspondiente.

- `transmitir (struct entorno)`: Esta función recibe todos los mensajes a transmitir por los robots y los envía a los robots (salvo al que lo envió), dejándolos en su buffer de recepción.

SensorIR: Esta clase representa a cada uno de los sensores que posee el robot.
Atributos:

- `Integer posicion`: Es un número que en grados representa la distancia angular a la cabeza del robot.
- `integer umbralDeVisionIR`: Es un número entero que en centímetros representa el límite de visión de los sensores.
- `integer valorSensado`: Es un número que en centímetros representa la distancia al objeto sensado.
- `leerSensoresIR (struct entorno)`: Esta función carga en el vector de sensores de cada robot, la distancia a la que cada uno de los sensores mida. Para realizar esto, se va recorriendo la matriz que representa al entorno, por medio de rectas en la dirección de cada sensor. En caso de detectar algo, se detiene el recorrido del segmento de recta que obtuvo un resultado y se guarda en el vector de sensores IR el módulo de dicho segmento de recta en el lugar correspondiente al sensor en cuestión. En caso de no detectar nada se seguirá recorriendo hasta que el módulo del segmento de recta sea mayor al umbral de visión, en dicho caso no se ha detectado nada por ese sensor y se registrará esto en el vector de los sensores IR en el lugar correspondiente.

Configuración: Esta clase posee aquellos parámetros variables del simulador.
Atributos:

- `integer cantidadDeRobots`: Es la cantidad de robots con la que se simulará.
- `integer cantidadObstaculosCirculares`: Es la cantidad de obstáculos circulares con la que se simulará.
- `integer cantidadObjetivos`: Es la cantidad de objetivos con la que simulará.
- `integer tamanoDelEntorno`: Es el tamaño que tomará el entorno durante la simulación.
- `integer velocidadMaximaRobots`: Es el módulo máximo que podrá tomar la velocidad de un robot.
- `array[4] zonaDeInicializaci\'on`: Es la zona en la que se inicializarán los robots en el caso de optar por la inicialización automática que posicionará a los mismos en forma aleatoria.
- `integer radioRobots`: Es el radio de los robots.
- `integer radioObjetivo`: Es el radio del objetivo.
- `integer radioObstaculosCirculares`: Es el radio de los obstáculos circulares.

- `integer umbralDeVision`: Es la distancia máxima a la que los sensores del robot logran medir.
- `integer velocidadMaximaObjetivo`: Es el módulo máximo que podrá tomar la velocidad de un objetivo.
- `integer velocidadMaximaObstaculos`: Es el módulo máximo que podrá tomar la velocidad de un obstáculo.
- `integer velocidadMaximaInicialObstaculos`: Es el módulo máximo que podrá tomar la velocidad inicial de un obstáculo.
- `integer velocidadMaximaInicialRobots`: Es el módulo máximo que podrá tomar la velocidad inicial de un robot.
- `integer velocidadMaximaInicialObjetivo`: Es el módulo máximo que podrá tomar la velocidad inicial de un objetivo.

Interfaz: Es la clase que mantiene el contacto con el usuario, se encargará de cargar la configuración y de mostrar en pantalla la situación actual del entorno.
Atributos:

- `textBox txtTamanoEntorno`: Es un cuadro de texto donde el usuario podrá ingresar el tamaño que tomará el entorno durante la simulación.
- `textBox txtCantidadRobots`: Es un cuadro de texto donde el usuario podrá ingresar la cantidad de robots que habrán durante la simulación.
- `textBox txtRadioRobots`: Es un cuadro de texto donde el usuario podrá ingresar el radio de los robots.
- `textBox txtVelocidadMaximaRobots`: Es un cuadro de texto donde el usuario podrá ingresar el módulo máximo de la velocidad de los robots.
- `textBox txtUmbralDeVisionRobots`: Es un cuadro de texto donde el usuario podrá ingresar la distancia máxima a la que los sensores del robot logran medir.
- `textBox txtInicializacionHorizontalInicialRobots`: Es un cuadro de texto donde el usuario podrá ingresar el límite inferior de la zona horizontal donde se inicializarán los robots, en caso de elegir la opción automática.
- `textBox txtInicializacionHorizontalFinalRobots`: Es un cuadro de texto donde el usuario podrá ingresar el límite superior de la zona horizontal donde se inicializarán los robots, en caso de elegir la opción automática.
- `textBox txtInicializacionVerticalInicialRobots`: Es un cuadro de texto donde el usuario podrá ingresar el límite inferior de la zona vertical donde se inicializarán los robots, en caso de elegir la opción automática.
- `textBox txtInicializacionVerticalFinalRobots`: Es un cuadro de texto donde el usuario podrá ingresar el límite superior de la zona vertical donde se inicializarán los robots, en caso de elegir la opción automática.

- `textBox txtCantidadObjetivos`: Es un cuadro de texto donde el usuario podrá ingresar la cantidad de objetivos que habrán durante la simulación.
- `textBox txtRadioObjetivos`: Es un cuadro de texto donde el usuario podrá ingresar el radio de los objetivos.
- `textBox txtVelocidadMaximaObjetivos`: Es un cuadro de texto donde el usuario podrá ingresar el módulo máximo de la velocidad de los objetivos.
- `textBox txtCantidadObstaculos`: Es un cuadro de texto donde el usuario podrá ingresar la cantidad de obstáculos que habrán durante la simulación.
- `textBox txtRadioObstaculos`: Es un cuadro de texto donde el usuario podrá ingresar el radio de los obstáculos.
- `axes imagen`: Es la imagen donde se irá desplegando periódicamente el estado del entorno.

Funciones:

- `desplegarEntorno (struct entorno)`: Esta función se encarga de desplegar en pantalla la matriz imagen del entorno.
- `interfaz ()`: Esta función se encarga de inicializar la interfaz gráfica con sus componentes. Dentro de ellas se encuentran algunos de los parámetros del simulador (la configuración), la imagen del entorno y los botones de cargar, iniciar, detener, selección, trayectorias y ver solo robots.
  Cuando el usuario realiza click sobre el botón cargar, se leerá la configuración y se inicializará el entorno posicionando a los objetos de manera aleatoria. En cambio si se activa el botón de selección, se realizará lo mismo salvo que el usuario elegirá, realizando click sobre la imagen del entorno, donde posicionar a los robots.
  Para iniciar el simulador y que los objetos comiencen a moverse, se deberá hacer click sobre iniciar, lo que comenzará el loop infinito entre `evolucion_de_los_agentes` y `actualizar_entono`. En caso de querer interrumpir la ejecución se deberá activar el botón de detener.
  La opción de ver solo robots, se utilizará en caso de que se desee observar en la imagen del entorno, solo aquellos objetos que se encuentren en el rango de visión de los robots. Esta función deberá negar el valor actual de `verSoloRobots` perteneciente al entorno, de esta manera, en caso de que se quiera volver a observar todo el entorno, bastará con volver a realizar click sobre dicho botón.
  Por último está la opción de ver las trayectorias, que desplegará en pantalla una gráfica de las mismas.

Robot: Esta clase representa a un robot, según el modelo abstracto manejado anteriormente
Atributos:

- `integer id`: Número de identificación único.
- `integer radio`: Representa el radio del robot.

- `array[2] posici\'on`: Representa la posición del robot (x, y) en el entorno.
- `array[2] velocidad`: Representa la velocidad actual del robot en las coordenadas cartesianas del entorno.
- `array[2] direcci\'on`: Representa la dirección de la cabeza del robot, en coordenadas cartesianas en la dirección y sentido de las coordenadas del entorno, pero centradas en el robot.
- `integer velocidadM\'axima`: Representa el máximo módulo de velocidad que puede tomar el robot.
- `array[(umbralDeVisionIR + radio)*2] [(umbralDeVisionIR + radio)*2]lugarVIsible:` Es una matriz que tiene unos en una circunferencia de radio `(umbralDeVisionIR + radio)*2` y el resto ceros.
- `integer color`: Es un número entero que representa el color que tomará el robot al representarlo en la interfaz gráfica.
- `array[radioRobot*2][radioRobot*2] imagenRobot`: Es una matriz que lleva el número de color en una circunferencia de radio `radioRobot2` y el resto ceros.
- `Array[radioCabeza*2][radioCabeza*2] imagenCabeza`: Es una matriz de número `colorCabeza` en una circunferencia de radio `radioCabeza` y el resto ceros.
- `integer radioCabeza`: Es un entero que representa el radio de la cabeza del robot (normalmente `radioRobot`/5).
- `array[] mensajeATransmitir`: Es un array de Strings de los mensajes que quiere transmitir el robot y debe transmitir el transceptor inalámbrico.
- `array[] mensajesRecibidos`: Es un array de strings de los mensajes que ha recibido el transceptor inalámbrico y los ha dejado a disposición del robot.
- `array[][2] trayectoria`: Es una matriz cuyas entradas son las posiciones por las que ha pasado el robot.
- `integer iteracionesPerdido`: Es un número entero que representa la cantidad de iteraciones en las que el robot no ha visto a nadie consecutivamente.
- `array[] recordatorio`: Es un array de strings libre, el robot puede guardar allí lo que decida.

Funciones:

- `calcularAngulo (array[2] direccion)`: Esta función devuelve el ángulo de un vector entre 0 y 2π. Para esto calcula el arco-tangente del cociente de la segunda coordenada y la primera. En caso de que la primer coordenada sea negativa, le sumará π al resultado.

- `calcularProximaPosicion (integer m\'oduloMovimiento, struct entorno, integer numeroRobot)`: Esta función calcula la velocidad del robot en función del módulo del movimiento, suponiendo que el robot se moverá en la dirección de su cabeza. Para esto calcula el vector orientación como la resta entre la posición de la cabeza y la posición del robot, de ahí calcula el ángulo de la cabeza y en conjunto con el módulo del movimiento devuelve la velocidad.

- `girar (struct entorno, integer numeroRobot, double gradosEnRadianes)`: Esta función modifica la posición de la cabeza del robot según el ángulo que se le transmite como parámetro.

- `inicializarRobots (integer cantidad_robots, integer cantidad_de_objetivos, integer radio_objetivo, integer cantidad_de_obstaculos_circulares,integer radio_obstaculos, array[2] zona_de_inicializacion_horizontal, array[2] zona_de_inicializacion_vertical, integer radio_robots, integer velocidad_maxima_robots, integer umbral_de_visionIR, struct entorno,struct obstaculos_circulares, struct objetivos)`: Inicializa el estado de los robots, dándole una posición, dirección y velocidad aleatoria. Se crea la matriz de ceros del lugar visible que se inicializará con unos dentro de la circunferencia de visión del robot. Se crearán las matrices con las imágenes del robot y de la cabeza del mismo y se inicializará la cámara con el cono de visión correspondiente a la misma.

- `inicializarRobotsPosicionados (integer id, struct robots, integer cantidad_robots, integer cantidad_de_objetivos, integer radio_objetivo, integer cantidad_de_obstaculos_circulares, integer radio_obstaculos, array[2] zona_de_inicializacion_horizontal, array[2]zona_de_inicializacion_vertical, integer radio_robots, integer velocidad_maxima_robots, integer umbral_de_vision_IR, struct entorno, struct obstaculos_circulares, struct objetivos)`: Inicializa el estado de los robots, dándole una dirección y velocidad aleatoria, recibiendo como parámetro la posición que deberá tomar, la cual será verificada de manera de no encontrarse colisionando o sobre otro objeto. Se crea la matriz de ceros del lugar visible, que se inicializará con unos dentro de la circunferencia de visión del robot. Se crearán las matrices con las imágenes del robot y de la cabeza de mismo y se inicializará la cámara con el cono de visión correspondiente a la misma.

B.7.2 Diagrama de secuencias

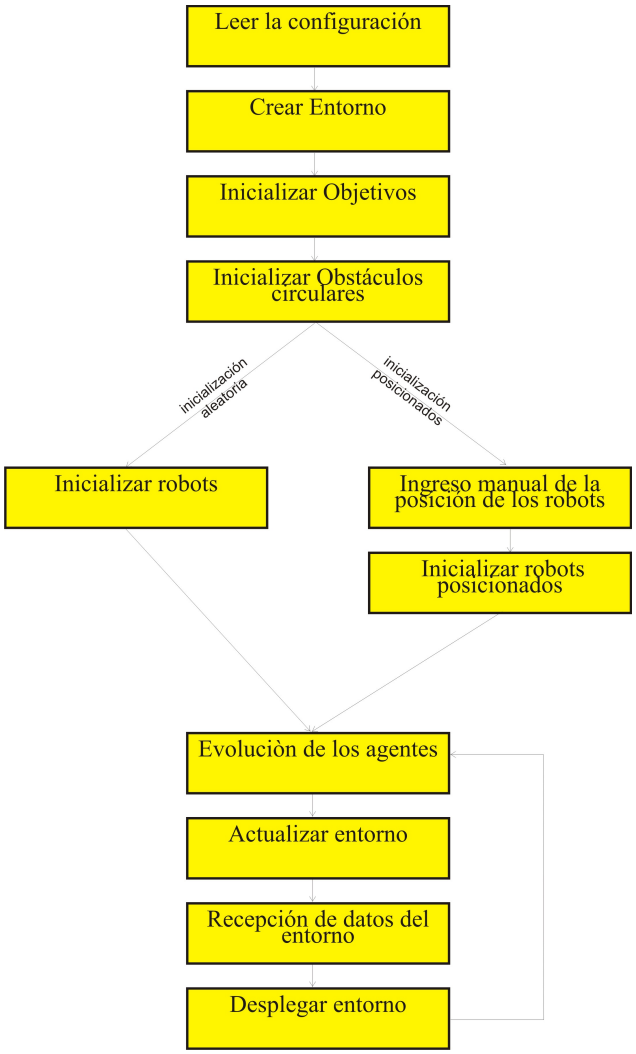


Figura B.5: Diagrama de secuencias.

C Robot Khepera III

C.1 Especificaciones

- Procesador: DsPIC 30F5011 de 60MHz, ampliable a 600MHz con el *KoreBot* II
- RAM: 4KB en DsPIC, 64MB en la extensión *KoreBot*
- Flash: 66KB en DsPIC, 32MB en la extensión *KoreBot*
- Movimiento: 2 servomotores DC con encoders incrementales (aproximadamente 22 pulsos por *mm* de movimiento del robot)
- Velocidad: $0.5\frac{\mathrm{m}}{\mathrm{s}}$ máximo
- Sensores: 9 sensores infrarrojos de proximidad y luz ambiental con un rango máximo de $30cm$. 2 sensores infrarrojos inferiores para aplicaciones de seguimiento de linea y 5 sensores de ultra-sonido con un rango de 20cm a 4m
- I/O: varias entradas y salidas con la extensión *KoreIO*.
- Alimentación: Adaptador de corriente intercambiable, batería de polímero de litio (1400mAH)
- Autonomía: 8 horas, en continuo movimiento, sin *KoreBot*.
- Comunicación: Puerto serie, comunicación USB de hasta 115kbps con el *KoreBot*, Wireless Ethernet con el *KoreBot* y tarjeta WiFi.
- Bus de extensión: Modulos de expansión pueden ser agregados al robot usando *KB-250 bus*.
- Tamaño: diámetro 130mm, altura 70mm.
- Peso: 690g aprox.
- Carga útil: 2000g aprox.
- Simulador: V-REP
- Entorno de desarrollo para aplicaciones autónomas: GNU C/C++

- Software para el control remoto vía sujeción o radio: LabVIEW®, (para PC, MAC o SUN) utilizando RS232. MATLAB ®(en PC, Mac, Linux o SUN) utilizando RS232. SysQuake ®(en PC, Mac, Linux o SUN) utilizando RS232. Programas de dominio público. Cualquier otro software que permita la comunicación RS232.

C.2 Pequeñas implementaciones

C.2.1 No caerse de la mesa

Esta implementación consiste de un sencillo algoritmo de prueba de los sensores inferiores del robot.
El problema que el robot debe resolver es poder moverse sobre una mesa a cierta altura, sin caerse de la misma.
Se utilizaron sus sensores inferiores en modo proximidad de manera que el robot pueda conocer, en cualquier momento, la distancia al piso. En caso de que se encontrara en medio de la mesa, esa distancia será pequeña. Sin embargo, al encontrarse en el borde la lectura se hará sobre el piso, por lo que será mayor.
En base a esto, el algoritmo consiste en un loop infinito donde se mide con los sensores inferiores en modo distancia. Si el valor medido es pequeño, se decide seguir a una velocidad constante de módulo $V1$ en línea recta, en caso contrario, se toma la decisión de girar (el robot gira sobre sí mismo). Lo primero se logra dándole la misma velocidad $V1$ a cada rueda y en el mismo sentido. Lo segundo se consigue dándole a las ruedas una velocidad de módulo $V2$ y en sentido contrario, durante un tiempo T. Estos tres parámetros deben ser fijados de antemano con ciertos cuidados:

- La velocidad $V1$ no puede ser demasiado alta, ya que una vez que se detecta un borde, el robot aún debe tener tiempo suficiente para procesar los datos y detenerse. Esto es una restricción debido a la ubicación de los sensores, los cuales no se encuentran en el borde del mismo, sino a unos centímetros de las ruedas). $V1$ fue determinada experimentalmente.

- La velocidad V_2 y el tiempo T fueron obtenidos también de manera experimental de forma tal que el robot, luego de girar, quede posicionado en una dirección que, a priori, se supone que no apunta hacia un borde cercano.

Podría parecer que, por ejemplo, en una mesa rectangular, girar entre 45° y 135°, no solucionaría el problema si el robot se encontrara en un vértice de la misma (ver figura C.1). Sin embargo, si la velocidad V_1 estuvo fijada de manera correcta, el robot volverá a sensar una distancia grande y volverá a girar, por lo que no supone una gran complicación.

C.2.2 No salir del doyo

En este algoritmo, el robot debe hacer es moverse sobre una doyo, sin salir del mismo.
Puede observarse que el problema presenta una gran similitud con el anterior. En este caso se utilizaron los sensores inferiores en modo luminosidad, ya que el doyo utilizado es de color negro con una franja blanca en su borde. Cuando el robot mida

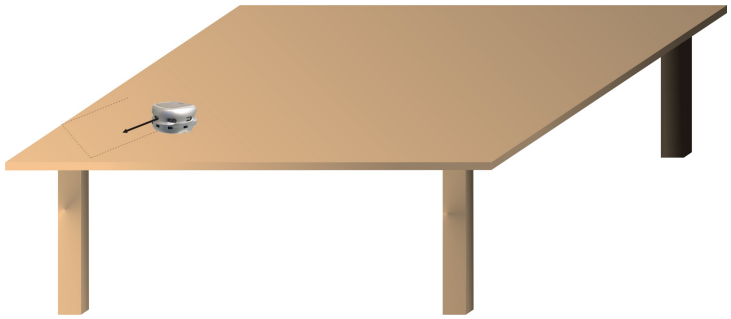


Figura C.1: Imagen descriptiva del comportamiento deseado.

un nivel de luminosidad bajo, será porque se encuentra lejos del borde y sucederá lo contrario cuando se encuentre cerca.
El algoritmo diseñado es casi idéntico al anterior, con la diferencia de que la medida realizada es a través del sensor en modo luminosidad. Al igual que en el algoritmo anterior, si no encuentra el borde, el robot sigue avanzando en forma rectilínea y en caso contrario se detiene y gira sobre sí mismo. Una característica a destacar es que en este caso se detecta el borde antes de llegar al mismo y por lo tanto las limitaciones en la velocidad son despreciables, en comparación con la velocidad máxima que puede tomar el robot por sus características de diseño.

C.2.3 Seguidor de línea

En este caso, el problema que debe resolver el robot es avanzar por un camino de un color dado (llamado línea) en un entorno de un fondo de color constante que contraste con el del camino.
Para lograr esto, se utilizaron dos sensores que el robot posee en su parte inferior, ambos ubicados en el centro, uno al lado del otro. Utilizando los sensores en modo luminosidad, según los valores medidos, se puede saber si el robot está completamente sobre la línea, del lado izquierdo o del lado derecho de la misma.
En este caso, se utilizó un fondo blanco y una línea negra, por lo tanto, para niveles bajos de luminosidad, el robot estará ubicado sobre la línea y en caso contrario estará sobre la superficie blanca. Para seguir la línea, el robot deberá seguir en forma rectilínea cuando el camino sea recto y doblar cuando el camino se curve. Se debe tener en cuenta que no solo el robot tendrá que girar cuando la línea se curve, sino también cuando la línea sea recta, pero el mismo no se encuentre alineado a ella. Para esto, el robot deberá girar y realinearse.
Por lo tanto, en la siguiente tabla se presentan los casos posibles y se proponen las acciones a tomar en función de los mismos:

Luminosidad sensor derecho	Luminosidad sensor izquierdo	Situación	Acción a tomar
Bajo	Bajo	EL robot se encuentra sobre la línea	Seguir en línea recta
Alto	Bajo	Solo la rueda izquierda está sobre la línea	Girar a la derecha
Bajo	Alto	Solo la rueda derecha está sobre la línea	Girar a la izquierda
Alto	Alto	El robot perdió a la línea	Retroceder hasta que cambie la situación

El algoritmo diseñado repite el esquema de los algoritmos anteriores, difiriendo en las acciones a tomar. Si el robot mide un valor bajo en ambos sensores, el mismo continúa en línea recta. Si una medida resulta en un valor alto y otra en uno bajo, el robot gira según cuál sea la posición del sensor que realizó la medida baja. En el caso en que ambas medidas resulten en niveles altos, el robot retrocede hasta encontrar nuevamente la línea. Esta última acción está justificada bajo la hipótesis que el robot anteriormente se encontraba sobre la línea.

Bibliografía

[1] Unimate, primer robot de uso industrial programable. Utilizado por General Motors. URL: http://www.gm.com/experience/education/9-12/making_vehicles/robotsandpeople.jsp

[2] Luis Merino1, Jesús Capitán2, Aníbal Ollero2. Robótica cooperativa e integración con sensores en el ambiente. Aplicaciones en entornos urbanos. Grupo de Robótica, Visión y Control. Escuela Politécnica Superior, Universidad Pablo de Olavide, Crta. Utrera km. 1, Sevilla. Escuela Superior de Ingenieros, Universidad de Sevilla, Camino de los Descubrimientos s/n, Sevilla

[3] Palauqui, J. -., & Laufs, P. (2011). Phyllotaxis: In search of the golden angle. Current Biology, 21(13), R502-R504. Retrieved from www.scopus.com

[4] Prasad, K., Grigg, S. P., Barkoulas, M., Yadav, R. K., Sanchez-Perez, G. F., Pinon, V., et al. (2011). Arabidopsis PLETHORA transcription factors control phyllotaxis. Current Biology, 21(13), 1123-1128. Retrieved from www.scopus.com

[5] Okabe, T. (2011). Physical phenomenology of phyllotaxis. Journal of Theoretical Biology, 280(1), 63-75. Retrieved from www.scopus.com

[6] Moreira Arana, C. E., Equihua Zamora, M. E., & Negrete Martínez, J. (2004). EXPERIMENTOS EN AUTOORGANIZACIóN. (Spanish). Acta Zoológica Mexicana, 20(3), 107-125. Retrieved from EBSCOhost.

[7] Universidad de Bristol, Inglaterra. URL: http://www.bris.ac.uk/

[8] Laboratorio de "Bio-Ingeniería e Inteligencia en Sistemas Autónomos" de la Universidad de bristol. URL: http://www.brl.ac.uk/projects.html

[9] Proyecto "Swarm-bots". URL: http://www.swarm-bots.org/index.php?main=1

[10] Comisión Europea CORDIS. URL: http://cordis.europa.eu/home_en.html

[11] Dorigo, M.; Maniezzo, V.; Colorni, A.; , "Ant system: optimization by a colony of cooperating agents" Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on , vol.26, no.1, pp.29-41, Feb 1996 doi: 10.1109/3477.484436 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=484436&isnumber=10246

[12] Kennedy, J.; Eberhart, R.; , "Particle swarm optimization," Neural Networks, 1995. Proceedings., IEEE International Conference on , vol.4, no., pp.1942-1948 vol.4, Nov/Dec 1995 doi: 10.1109/ICNN.1995.488968 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=488968&isnumber=10434

[13] Kennedy, J.; Eberhart, R.C.; , "A discrete binary version of the particle swarm algorithm," Systems, Man, and Cybernetics, 1997. 'Computational Cybernetics and Simulation'., 1997 IEEE International Conference on , vol.5, no., pp.4104-4108 vol.5, 12-15 Oct 1997 doi: 10.1109/ICSMC.1997.637339 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=637339&isnumber=13793

[14] D.E. Luenberger, Programación Lineal y no Lineal, Addisson-Wesley Iberoamericana, 1989.

[15] Ellips Masehian, and Davoud Sedighizade, Classic and Heuristic Approaches in Robot Motion Planning - A Chronological Review. Proceedings of World Academy of Science, Engineering And Technology Volume 23 August 2007

[16] Li Lu; Dunwei Gong; , "Robot Path Planning in Unknown Environments Using Particle Swarm Optimization," Natural Computation, 2008. ICNC '08. Fourth International Conference on , vol.4, no., pp.422-426, 18-20 Oct. 2008 doi: 10.1109/ICNC.2008.923 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4667318&isnumber=4667228

[17] Mao Yang; Chengfeng Li; Yantao Tian; , "Flocking for Swarm Robot System: Distributed Coadaptive Control and Optimization," Information Engineering and Computer Science, 2009. ICIECS 2009. International Conference on , vol., no., pp.1-4, 19-20 Dec. 2009 doi: 10.1109/ICIECS.2009.5365540 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5365540&isnumber=5362514

[18] Kalman; Rudolph Emil; , "A New Approach to Linear Filtering and Prediction Problems", Transactions of the ASME–Journal of Basic Engineering, Volume = 82, Number = Series D, Pages = 35–45, 1960.

[19] Watanabe, Y.; Yuta, S.; , "Position estimation of mobile robots with internal and external sensors using uncertainty evolution technique,Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on , vol., no., pp.2011-2016 vol.3, 13-18 May 1990 doi: 10.1109/ROBOT.1990.126301 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=126301&isnumber=3534

[20] Lieu My Chuong; Lam Siew Kei; Srikanthan, T.; , "High-level delay estimation technique for porting C-based applications on FPGA,Ïndustrial Electronics, 2008. ISIE 2008. IEEE International Symposium on , vol., no., pp.1991-1996, June 30 2008-July 2 2008 doi: 10.1109/ISIE.2008.4677118 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4677118&isnumber=4676877

[21] Xiucai Ji; Zhiqiang Zheng; Hui Zhang; , Robot Position Convergency in Simultaneous Localization and Mapping,"Mechatronics and Automation, 2007. ICMA 2007. International Conference on , vol., no., pp.320-325, 5-8 Aug. 2007 doi: 10.1109/ICMA.2007.4303562 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4303562&isnumber=4303488

[22] Chang, H.J.; Lee, C.S.G.; Hu, Y.C.; Yung-Hsiang Lu; , "Multi-robot SLAM with topological/metric maps,Ïntelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on , vol., no., pp.1467-1472, Oct. 29 2007-Nov. 2 2007 doi: 10.1109/IROS.2007.4399142 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4399142&isnumber=4398944

[23] Pugh, J.; Martinoli, A.; , "Inspiring and Modeling Multi-Robot Search with Particle Swarm Optimization," Swarm Intelligence Symposium, 2007. SIS 2007. IEEE , vol., no., pp.332-339, 1-5 April 2007 doi: 10.1109/SIS.2007.367956 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4223193&isnumber=4223144

[24] Bruce A. Carlson, Paul B. Crilly, Janet C. Rutledge, "Communication Systems: An introduction to Signals and Noise in Electrical Communication", 4th. Edition, McGraw-Hill, 2002. ISBN: 0-07-112175-7.

[25] URL: http://www.k-team.com/mobile-robotics-products/khepera-iii