Universidad Zaragoza
1542

Trabajo Fin de Máster

Sistema multi-cámara para la reconstrucción volumétrica de objetos 3D

Multi-camera systems for volumetric reconstruction of 3D objects

Autor

Enrique Hernández Murillo

Directores

Gonzalo López Nicolás

Rosario Aragüés Muñoz

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2019

AGRADECIMIENTOS

CONSIDERO que esta parte de la memoria es la más importante para mí. Durante todo el año que hemos estado embarcados en este trabajo, siempre me he sentido acompañado y es por ello, que reservo esta página para dedicar unas palabras a aquellas personas que se han convertido en sólidos pilares en mi futuro profesional.

A Gonzalo López-Nicolás: Porque sin su implicación, este trabajo no habría sido posible.

A Rosario Aragüés: Por su eterna paciencia a la hora de repetirme una y otra vez las cosas sin ningún reproche de por medio.

A mi madre, Luisa: Que tanto le quiero, porque su lucha me enseña lo que importa en esta vida y me transmite la alegría de vivir.

A mi padre, Enrique: Que tanto le quiero, por haberse ocupado, que no preocupado, de mí; siempre acompañándome y alentándome hasta el final.

A mis amigos y compañeros del laboratorio: En especial a Rafael que es mi pastor y a Marcos, Nacho y Pablo, que me han acompañado a lo largo de esta ardua travesía. A Víctor, Borja, Marta, Alba, Celia, Daniel y Ana siempre conmigo. A Luis, mi hermano que me escucha y me entiende. A ti también Pilar, sabes sacarme una sonrisa en los momentos oportunos.

Sistema multi-cámara para la reconstrucción volumétrica de objetos 3D

RESUMEN

Este trabajo se centra en el estudio y posterior implementación de un algoritmo distribuido para tareas de percepción de objetos deformables, así como el desarrollo de la arquitectura software apropiada para su implementación. Este algoritmo se encuentra dentro del campo del control cooperativo de sistemas multi-agente. Dentro de este marco de trabajo, la reconstrucción volumétrica de objetos es esencial en la manipulación robótica. Construir el modelo 3D de un objeto es un problema complejo que involucra aspectos como el modelado, el control, la percepción o la planificación.

El principal objetivo consiste en desarrollar, una arquitectura software que reconstruya un objeto en movimiento sujeto a deformaciones, mediante un equipo de robots, de manera que se perciba instantáneamente los cambios producidos en la superficie del objeto. La realización de esta tarea requiere la disposición del conjunto de todos los diferentes puntos de vista que cubran la superficie del objeto. Una sola cámara tardaría demasiado tiempo en la percepción de la superficie global de puntos alrededor del objeto sujeto a deformaciones, sin embargo, un escenario multi-cámara permite desempeñar la reconstrucción del objeto de una forma eficiente. Además, debido a las limitaciones de la cámara, como el campo de visión limitado o las auto-oclusiones, es esencial utilizar una estrategia de configuración eficaz para seleccionar las vistas adecuadas que proporcionen más información sobre el objetivo.

En este trabajo, se empiezan explicando conceptos básicos de la teoría de grafos, así como algunos preliminares matemáticos empleados en los sistemas multi-agente, y se presentan de forma detallada los algoritmos de consenso en tiempo discreto a través de simulaciones. Posteriormente se detalla una arquitectura multi-cámara basada en el ROS, así como la implementación de una estrategia de consenso, que permite la reconstrucción del objeto deformable. Para validar y analizar las propuestas desarrolladas, se llevan a cabo diversas simulaciones comparando el comportamiento

de varias estrategias, así como su correcto funcionamiento.

Este trabajo se encuentra enmarcado en el proyecto europeo COMMANDIA que aborda es el diseño e implementación de sistemas multi-robot para tareas de percepción de objetos deformables. Parte de este trabajo se presentó en las jornadas de jóvenes investigadores del *I3A* (Instituto Universitario de Investigación en Ingeniería en Aragón) de 2019. También se ha publicado un artículo en el congreso internacional *ETFA2019* (Emerging Technologies and Factory Automation).

Este trabajo ha sido apoyado por el proyecto *COMMANDIA SOE2/P1/F0638 (Programa Interreg Sudoe, FEDER) y PGC2018-098719-B-I00 (MCIU/AEI/FEDER, UE).*

Índice

Capítulo 1

Introducción

1.1. Motivación

Hoy en día, en la industria manufacturera, las tareas de percepción resultan imprescindibles en procesos que involucran la manipulación de objetos. Este trabajo tiene como objetivo principal la reconstrucción y la percepción a lo largo del tiempo de un objeto desconocido *a priori* que se mueve y deforma en el tiempo, mediante el control cooperativo de un sistema multi-agente. El proceso de construcción de un modelo 3D de un objeto real, conocido como reconstrucción volumétrica es esencial en la manipulación robótica.

En este trabajo se presenta una implementación de algoritmos distribuidos en una red de agentes autónomos para tareas de percepción como la reconstrucción 3D de un objeto. El objetivo es distribuir los agentes de forma coordinada alrededor del centroide de un objeto. Además, el principal foco de interés radica en la percepción de las esquinas del objeto deformable. Debido a las observaciones parciales de los objetivos por parte del grupo de agentes, se discuten algunas estrategias de consenso medio dinámicas de tiempo discreto que permiten al grupo de agentes seguir el centroide (promedio) del objeto mediante un conjunto de entradas de referencia. También se estudia y analiza cómo cambia el límite superior del error de seguimiento. Aquí, el límite superior de las entradas de control garantiza la convergencia asintótica del problema del consenso. En aras de la simplicidad, se considera que el equipo de agentes se encuentra bajo una red cuya topología de comunicación está modelada por un grafo no dirigido.

El problema del consenso multi-robot se basa en el control de varias unidades dinámicas que comparten información para alcanzar un estado de acuerdo, mediante el movimiento en equipo y a través de una comunicación local del sistema. Generalmente, un sistema multi-agente proporciona un mejor rendimiento de la tarea, así como una reducción del coste computacional, con respecto a un sistema con un solo agente. El

objetivo del control cooperativo de un sistema multi-agente es dirigir un equipo de agentes autónomos, colaborando los unos con los otros en una tarea, para alcanzar un rendimiento del grupo en común. Algunas veces, en los sistemas multi-agente se producen retrasos de información y pérdidas de la comunicación entre los agentes. En este trabajo se asume que esto no sucede. Los sistemas multi-agente se basan en la teoría matemática de grafos; concretamente la matriz *Laplaciana* del grafo permite determinar la convergencia y estabilidad del modelo del grafo en los algoritmos de consenso.

El algoritmo de consenso es una parte esencial en el control cooperativo y se define por ser un equipo de agentes alcanzando la convergencia a un conjunto de metas comunes mediante la colaboración con cada uno de los otros a través de una red de comunicaciones. Se ha decidido que el sistema multi-agente siga una estrategia distribuida, ya que aunque en este trabajo se plantean pequeños grupos de robots (6 cámaras), el objetivo es que el trabajo sea escalable y no esté limitado el número de robots ni el tamaño del objeto. En el caso de que el mundo fuera suficientemente pequeño, se puede cambiar el grafo de forma que todos los agentes estén comunicados entre sí dando lugar a un sistema centralizado.

Como ejemplo de aplicación del consenso en un sistema multi- agente, en las Fig. (1.1) y (1.2) se muestra una red de agentes que pueden comunicarse y compartir información entre ellos. Partiendo de este grafo de comunicaciones y sin el soporte de ningún agente externo, la meta es conseguir un acuerdo cooperativo del centro geométrico del grupo de objetivos. Esta posición se obtiene mediante el consenso de los agentes y en este ejemplo los agentes seguirán diferentes trayectorias dependiendo del movimiento del grupo de objetivos.

En cuanto a la tarea de percepción, por un lado, la visión activa podría ser tomada como una alternativa viable. [1]. Esta técnica requiere el uso de un sensor basado en la visión montado sobre un manipulador o robot móvil, que proporciona una imagen densa de datos de entrada en 3D. Los métodos de visión activa abordan el problema de generar un modelo volumétrico completo del objeto, tan rápido como sea posible en tiempo real, resolviendo el problema de la siguiente mejor vista (en inglés *next-best-view*). Por lo tanto, necesitan calcular una secuencia de vistas (ubicaciones) de las cámaras (posición y orientación) alrededor del objeto. Con el fin de reducir el número de puntos de vista posibles, pueden limitar el volumen del entorno a una esfera o un cilindro alrededor del objeto. Sin embargo, uno de los principales

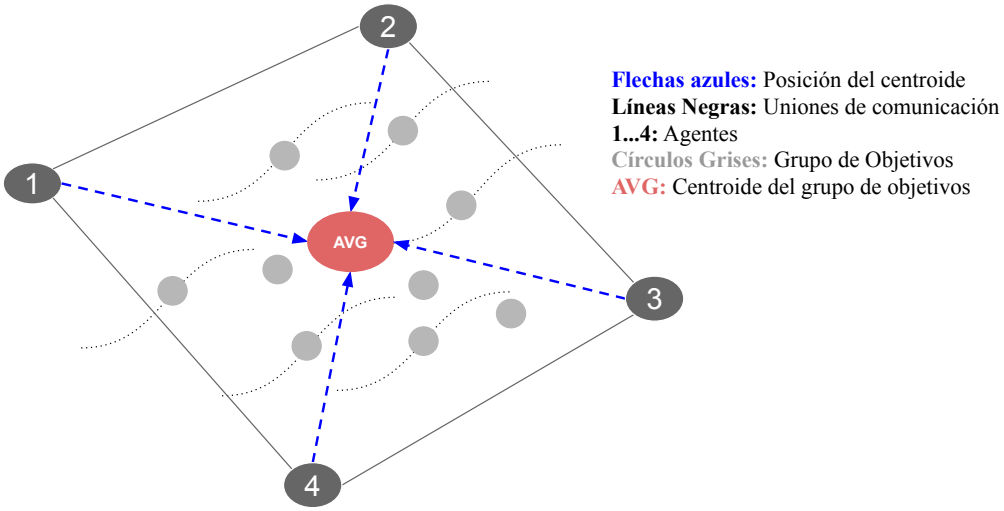


Figura 1.1: *Consenso promedio dinámico en un instante k de tiempo. Los agentes etiquetados con número del 1 al 4, realizan el tracking del promedio (AVG) de las posiciones de los objetivos en movimiento (círculos).*

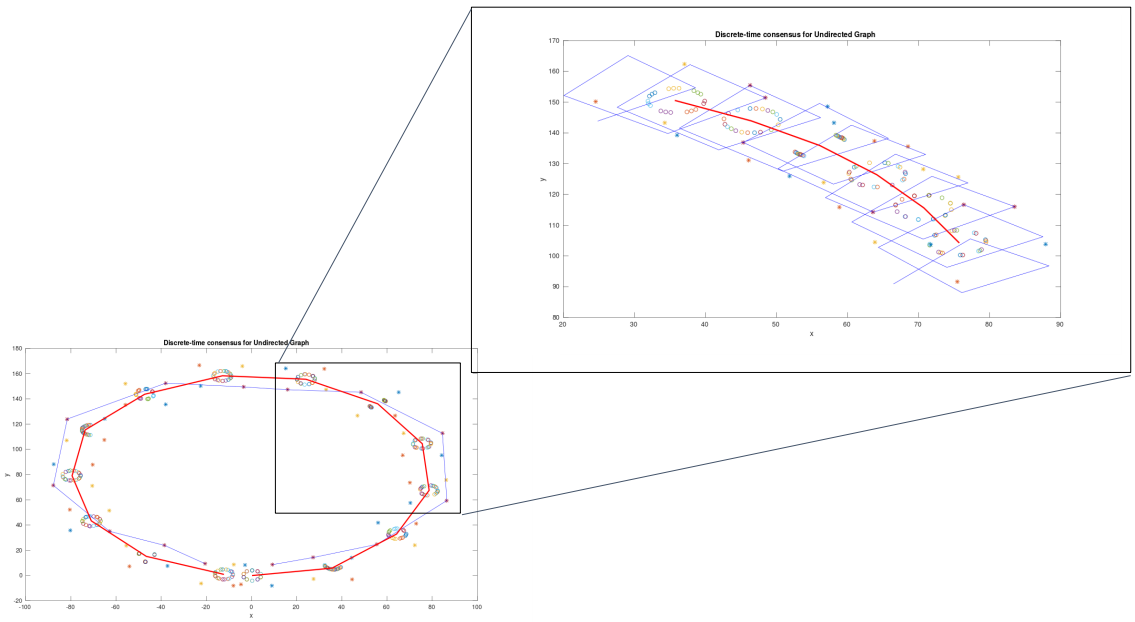


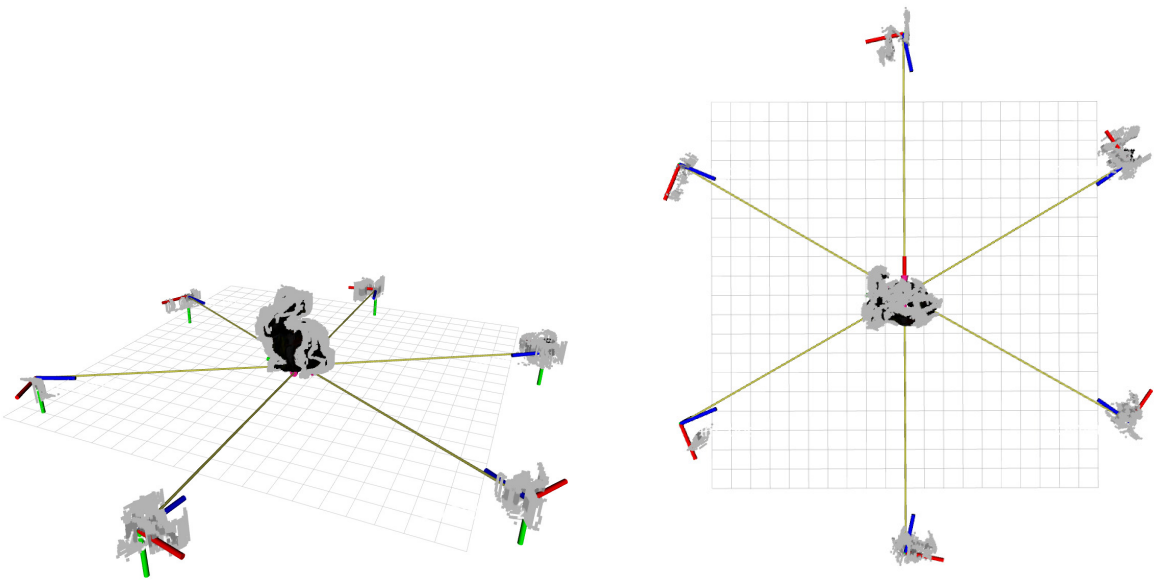
Figura 1.2: *Consenso promedio dinámico a lo largo de una trayectoria circular. En la imagen inferior izquierda se representa la trayectoria del centroide del objeto (línea sólida roja), ∘: posiciones de los objetivos, ×: posiciones de los agentes alrededor del centroide del objeto y la trayectoria que sigue el agente 1 (línea sólida azul). En la imagen superior derecha se muestra una ampliación para una mejor visualización.*

desafíos que implica la visión activa son los objetos deformables o móviles, lo que requiere una percepción instantánea y completa de la toda la superficie del objeto. No obstante, el principal inconveniente de la visión activa, es el tiempo necesario para seguir la secuencia de vistas de las cámaras. Entonces, ¿cómo puede ser algo percibido desde múltiples puntos de vista a la vez? La respuesta natural podría ser fusionar la información de varias cámaras en un mapa volumétrico de todo el objeto en cada momento (Fig. 1.3). Además, el sistema multi-cámara podría ser una buena elección si el modelo es deformable o el objeto se mueve. Proponemos colocar varias cámaras según una estrategia de formación distribuida, alrededor del objeto en movimiento para rodearlo y seguirlo. Esta estrategia asegura que el sistema multi-robot pueda realizar una percepción completa del objetivo a lo largo de su movimiento, ver Fig.(2.1).

En este trabajo, se considera el problema de la reconstrucción de un objeto *a priori* desconocido que se deforma y se mueve (no está limitado espacialmente). Se aborda el problema de la reconstrucción 3D instantánea usando un mapa volumétrico basado en probabilidades de ocupación del espacio, construido en tiempo real. Además, se extiende el problema a un escenario multi-agente. Se propone y evalúa una arquitectura de sistema multi-cámara para el mecanismo de posicionamiento de las cámaras que modele objetos arbitrarios en 3D. El escenario multi-robot permite elaborar un algoritmo de consenso que proporcione una estrategia y mejore el proceso de seguimiento de un objeto deformable o en movimiento. La aproximación propuesta que se presenta en este trabajo es un herramienta versátil y modular para la tarea específica de reconstrucción multi-sensorial de un objeto de interés. Sin embargo, existe un extenso rango de aplicaciones posibles y numerosos problemas de carácter tecnológico pueden beneficiarse de la arquitectura propuesta.

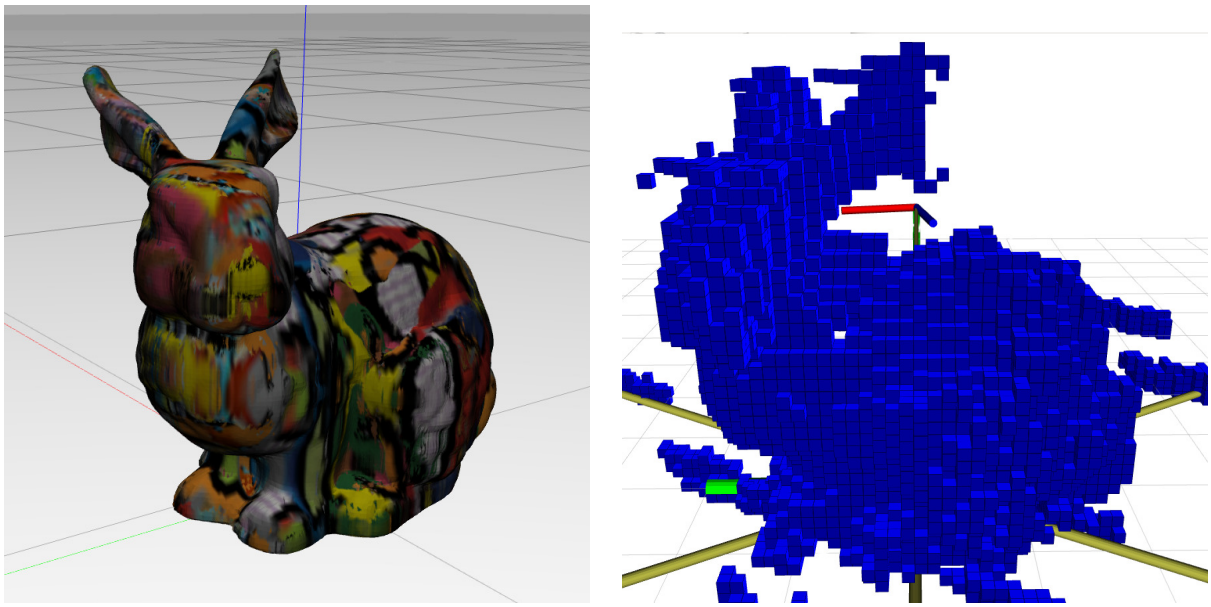
1.2. Objetivos

Los objetivos principales de este trabajo son dos, el primero es el estudio de un algoritmo de consenso en el marco de sistemas multi-agente. Y el segundo, es diseñar e implementar una arquitectura modular para llevar a cabo la reconstrucción instantánea de un objeto haciendo uso del algoritmo previamente estudiado a través de una estrategia multi-robot. La versatilidad y modularidad son las principales ventajas de la herramienta tecnológica propuesta. Dentro de este marco, la arquitectura de nuestro método utiliza tanto un sistema genérico basado en ROS (Robot Operating System) como una interfaz basada en *Matlab* para la estrategia. Además, evaluamos



(a) Vista en Perspectiva (b) Vista en Planta

Figura 1.3: *Ilustraciones de dos vistas diferentes del objeto reconstruido (Conejo de Standford) y las 6 cámaras. Se visualizan las representaciones finales (nube de puntos) de la escena y la formación multi-cámara que encierra el objeto. La figura muestra la salida del sistema como una estructura de datos visualizada por Rviz.*



(a) Modelo del Conejo de Standford (b) Celdas de Voxels Ocupadas

Figura 1.4: *Modelo de salida de nuestro sistema genérico para el objeto (Conejo de Standford). La subfigura a) muestra el modelo sintético del conejo, mientras que la subfigura b) muestra el objeto 3D reconstruido con los vóxeles ocupados.*

la totalidad de los mapas volumétricos del objeto en simulaciones realizadas en *Gazebo*.

En particular, los objetivos de este Trabajo Fin de Máster son los siguientes:

- El desarrollo e implementación de estrategias colaborativas multi-robot para tareas de percepción.
- Estudio de la materia relacionado con los problemas de consenso estáticos y dinámicos, así como de tareas multi-robot de percepción y tracking de un objeto.
- Implementación y análisis de un algoritmo de consensos
- Puesta en marcha de un equipo multi-robot, compuesto por varios pares de cámaras estéreo en el entorno de simulación *Gazebo*. Realización de varias tareas robóticas, como la construcción y mezcla de mapas adquiridos por un equipo compuesto por varios robots.
- Evaluación de las estrategias y desarrollo de métodos para percibir colaborativamente objetos del entorno y construir representaciones de los mismos.
- Validar los resultados obtenidos visualmente mediante simulación.

Las herramientas software que se han empleado han sido varias y el sistema operativo que le ha dado soporte ha sido Ubuntu 16.04. Para la implentación del protocolo de consenso se ha utilizado el programa MATLAB y aunque existen numerosas funciones para trabajar con grafos, en este trabajo no se ha requerido de ninguna de ellas. Para el diseño del nodo que comunica el algoritmo con la arquitectura multi-robot se requiere la versión de Matlab 2018 o superior, que permite ejecutar servicios en Gazebo, como mover un objeto por ejemplo. En cuanto a la implementación del sistema multi-agente, se ha llevado a cabo en ROS Kinetic, en lenguaje C++. Si bien es cierto, que con el editor gedit uno puede editar todo tipo de ficheros, en este trabajo se ha empleado Eclipse, que entre otras muchas ventajas, permite una visualización más cómoda y una organización plausible cuando se manejan numerosos paquetes en ROS.

1.3. Organización de la memoria

La memoria está organizada en 6 capítulos de la siguiente manera:
Una vez se ha introducido el problema y los objetivos que se van a abordar, así como el marco de trabajo en el que se encuentra este trabajo, en el Capítulo 2

se presentan los conceptos matemáticos para la comprensión de los problemas de control cooperativo. Por ejemplo, la teoría de grafos y la matriz Laplaciana, ya que tienen una aplicación directa en los algoritmos de consenso. Además, en este capítulo también se introducen diversos conceptos involucrados en la tarea de percepción visual y reconstrucción 3D.

En el Capítulo 3, se introduce formalmente el problema de interés, así como el algoritmo empleado, analizando sus propiedades de acuerdo al ratio de convergencia. Además, se presentan las estrategias estudiadas y se describe el funcionamiento del algoritmo de seguimiento para el equipo multi-robot. Se exponen las ecuaciones y se ilustran las estrategias mediante algunos ejemplos. Además, con motivo de explicar mejor su funcionamiento, se realiza una comparación de los resultados obtenidos con cada una.

El Capítulo 5 describe detalladamente la arquitectura del sistema multi-robot y se describen los módulos software sobre los que está implementada. Puesto que se abordan dos tareas independientes, en una primera parte se describe el software involucrado en la percepción instantánea y reconstrucción 3D del objeto, y a continuación se desarrollan las partes pertinentes al seguimiento y encapsulamiento del objetivo en cuestión. Finalmente, se expone la comunicación de la arquitectura de todo el sistema modular. Más aún, en este capítulo se resume el proceso de implementación de la arquitectura software sobre una plataforma de simulación. En este capítulo, se describen de forma abreviada los diversos programas que dan soporte a la herramienta y se detallan los pasos para ponerla en marcha. Además de lo anterior, se hacen algunos comentarios sobre el código que han permitido solventar los problemas técnicos acarreados.

A continuación, se llevan a cabo la simulación del el mecanismo de posicionamiento de la cámara y las simulaciones que prueban el rendimiento de la estrategia multi-robot en el Capítulo 6. La primera simulación, combina ambas tareas, la reconstrucción volumétrica y el mecanismo de posicionamiento de la cámara, con el objetivo de mostrar un escenario en el que sólo opere un robot. En la segunda simulación se he extendido el método a un entorno multi-robot e implementa una estrategia para reducir el coste computacional y el tiempo en generar el modelo $3D$.

Finalmente, en el Capítulo 7 resumimos las conclusiones extraídas del trabajo, así como, futuras vías de investigación y nuevas fronteras de aplicación de la herramienta.

A continuación, se muestra un diagrama ilustrativo (Fig.(1.5)) de los diferentes componentes implicados en este trabajo.

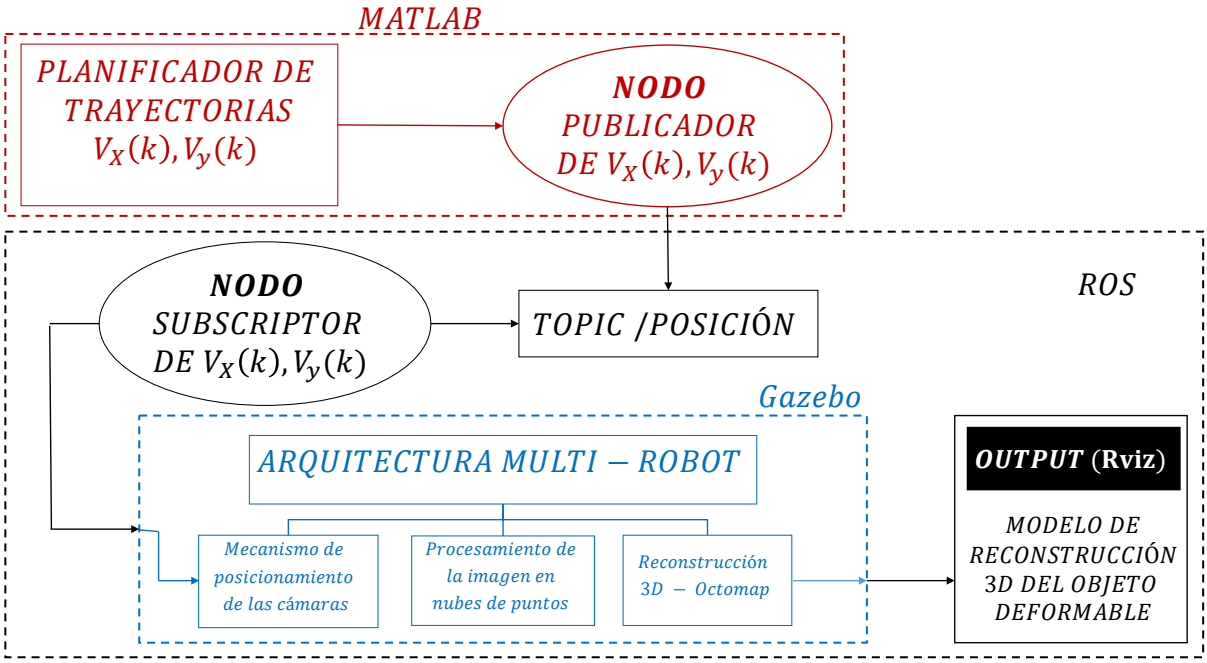


Figura 1.5: *Diagrama de bloques del sistema global. En rojo, se encuentran los dos programas de MATLAB; el planificador implementa el algoritmo de consenso promedio dinámico que calcula la posición de las cámaras (agentes); el segundo se comunica con ROS y publica en un topic las posiciones que le envía el planificador. En negro, se encuentra la arquitectura que tiene como soporte software ROS; aquí, se encuentra implantado el nodo subscriptor que obtiene las posiciones de las cámaras y se las comunica a Gazebo. Por último, en azul, se encuentra el simulador Gazebo, donde se sitúan las cámaras según las posiciones calculadas, se procesan las imágenes de éstas y se obtiene un modelo 3D del objeto. La salida del sistema, es un modelo 3D del objeto deformable en cada instante de tiempo.*

1.4. Publicaciones

A raíz de este trabajo se ha presentado un póster sobre la arquitectura implementada en ROS y se ha publicado bajo el título *Volumetric Object Reconstruction in Multi-Camera Scenarios* en la revista Jornada de Jóvenes Investigadores del I3A, vol. 7 (Actas de la VIII Jornada de Jóvenes Investigadores del I3A) 06 Junio de 2019, como se puede ver en [2]. Además, se ha escrito el artículo [3] publicado en la conferencia 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) en 2019, titulado *Multi-camera architecture for perception strategies*, páginas 1799-1804.

Capítulo 2

Fundamentos y Estado de la Materia

Este capítulo describe los fundamentos básicos para abordar el problema del consenso en los sistemas multi-agente así como parte de la literatura relacionada con este tema. Adicionalmente, se introducen algunos conceptos básicos respectivos a tareas de percepción multi-agente.

2.1. Conceptos Previos

2.1.1. Notación preliminar

En esta subsección se introduce la notación y terminología empleada a lo largo del documento y se definen algunos conceptos básicos de la teoría de grafos.

Notación: Se considera que $\mathbb{R}$ y$\mathbb{Z}_{\geq 0}$ denotan el conjunto de números reales y enteros, respectivamente. Para $\mathbf{v} \in \mathbb{R}^d$, $\|\mathbf{v}\| = \sqrt{\mathbf{v}^\top \mathbf{v}}$ se denota como la norma euclídea estándar. La transpuesta de una matriz $\mathbf{A}$ es $\mathbf{A}^\top$. El vector $\mathbf{1}$ representa un vector N-dimensional con todos los elementos iguales a uno, y $\mathbf{I}$ representa la matriz de identidad con la dimensión $N \times N$. Sea $\mathbf{\Pi}_N = \mathbf{I}_N - \frac{1}{N}\mathbf{1}_N\mathbf{1}_N^\top$. Cuando está claro en el contexto, no especificamos las dimensiones de la matriz. Para una señal dinámica $\mathbf{u}$, se indica con $\|\mathbf{u}\|_{ess}$, la norma suprema (esencial), es decir, $\|\mathbf{u}\|_{ess} = \sup\{\|\mathbf{u}(k)\|, k \geq 0\}$. En un sistema en red, distinguimos las variables locales de cada agente pero con un superíndice, por ejemplo, p^i es la variable local del agente $i \in \{1, ..., N\}$. Representamos el vector agregado de las variables locales p^i, $i \in \{1, ..., N\}$, por $\mathbf{p} = (p^1, ..., p^N)^\top \in \mathbb{R}^N$.

2.1.2. Teoría de grafos

La teoría de grafos es un área de las matemáticas que puede usarse para modelar y estudiar la comunicación de una red de un sistema multi-agente. En este trabajo,

se entiende el comportamiento y conexión de estos sistemas a través de enlaces del grafo de comunicaciones, que define la interacción entre los agentes. Esta rama abarca el conocimiento del comportamiento de la red, así como de sus propiedades. La información entre los agentes puede ser enviada de un agente a otro (unidireccional), o en ambos sentidos (bidireccional). En este capítulo, se introducen algunos conceptos básicos de la teoría de grafos para más tarde, centrar la atención en las matrices comúnmente usadas en los sistemas multi-agente, que son la matriz de adyacencia y la matriz Laplaciana.

A continuación, se introducen conceptos preliminares matemáticos sobre los grafos y las matrices de la teoría de grafos involucradas en el consenso.

Definición de Grafo: Un grafo es una representación gráfica de una red de nodos o vértices y un conjunto de arcos o enlaces que representan la conectividad de la red. La interacción entre dos nodos se indica mediante un arco, y puede ser unidireccional o bidireccional. La representación matemática de un grafo es $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, donde $\mathrm{V} = \{v_1, v_2, ..., v_n\}$ es el conjunto de vértices o nodos, mientras que E, denominado como (i, j), indica la conexión un par de nodos i, j, adyacentes entre sí. El grado del vértice i denota el número de conexiones, en el grafo, que tiene i y es igual al número de vecinos de este nodo. El conocimiento de los vecinos de un vértice i, es decir $N_i = \{j : (i, j) \in \mathrm{E}\}$ es fundamental en los algoritmos de consenso.

Grafos "No Dirigidos": Un grafo indirecto es aquel cuyos nodos están conectados por enlaces indirectos, es decir, la comunicación entre cada par de nodos es bidireccional y no tiene orientación. En otras palabras, el par (i, j) es el mismo que (j, i). Por simplicidad, se supone que todos los nodos tienen el mismo peso y pueden transmitir la misma cantidad de información. De aquí en adelante, en la memoria todos los grafos son indirectos. En la Fig.(2.1) se puede apreciar un ejemplo de grafo indirecto con $\mathrm{V} = \{1, 2, 3, 4, 5, 6\}$ y $\mathrm{E} = \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1)\}$, con respecto a uno directo o *"dirigido"*.

Matrices de Adyacencia y Grado: La matriz de adyacencia **A** es una excelente forma de definir la comunicación de un grafo **G**, ya que proporciona información en lo que se refiere a los agentes y sus conexiones. En esta matriz se asigna un uno a los arcos conectados y cero a los demás. Por tanto, la matriz de adyacencia de un grafo indirecto **A** es de orden $\mathrm{N} \times \mathrm{N}$ donde $\mathrm{N} = |\mathrm{V}|$ es el número de nodos del grafo, por tanto se denota como

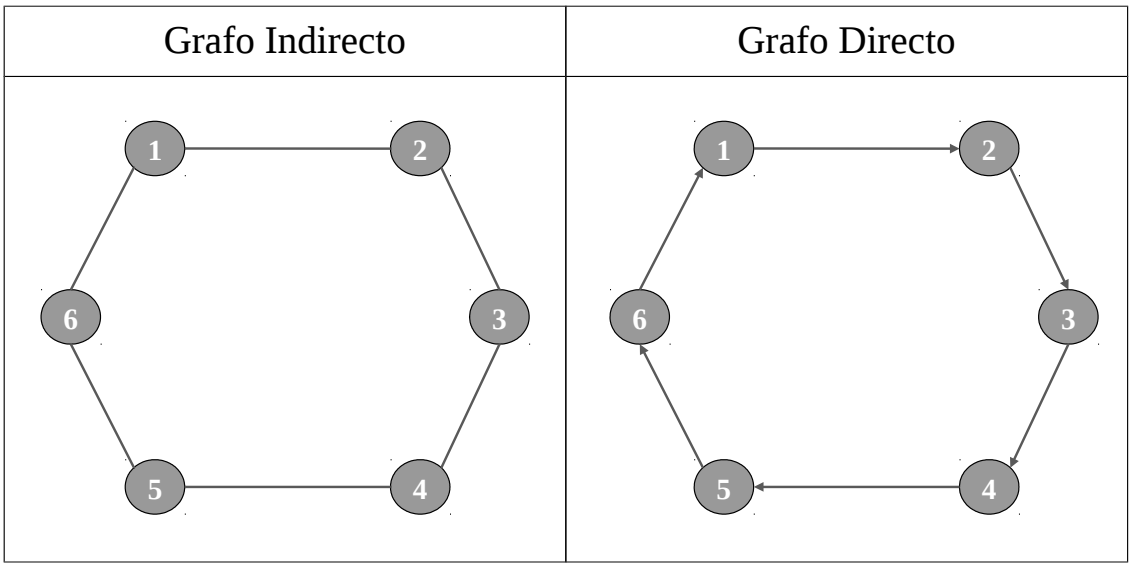


Figura 2.1: *Grafo Indirecto vs. Directo.*

$$[\mathbf{A}]_{ij} = \begin{cases} 1 & \text{si } (v_i, v_j) \in \text{ E} \\ 0 & \text{si no} \end{cases}$$

Puesto que los arcos son bidireccionales, en los grafos indirectos la matriz de adyacencia $\mathbf{A}$ es siempre simétrica. Para la Fig. 2.1, la matriz de adyacencia del grafo indirecto se representa,

$$[\mathbf{A}]_{ij} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

En un no dirigido, el grado de un nodo se denota como d^i, y equivale al número de nodos adyacentes a v_i en el grafo. La matriz diagonal de grado, cuya notación es $\mathbf{D}$, de un grafo $\mathbf{G}$, es una matriz diagonal que contiene el número de vecinos del nodo v_i, es decir, sus elementos diagonales son iguales al grado de cada nodo. Una vez más, la dimensión de esta matriz es N × N.

$$[\mathbf{D}]_{ij} = \begin{cases} N^i & \text{si } i = j \\ 0 & \text{si no} \end{cases}$$

Las entradas de la diagonal de esta matriz se calculan con la suma de los elementos de las filas o columnas de la matriz de adyacencia y el resto de elementos cero. Por

ejemplo, de nuevo para el grafo indirecto de la Fig 2.1, la matriz de grado es

$$[\mathbf{D}]_{ij} = \begin{bmatrix} d(1) & 0 & \cdots & 0 \\ 0 & d(2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d(n) \end{bmatrix} = \begin{bmatrix} 2 & 0 & \cdots & 0 \\ 0 & 2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 2 \end{bmatrix}$$

Matriz Laplaciana: Para finalizar, la representación de un grafo G es representada por la matriz Laplaciana **L**, que es una matriz N × N. Para un grafo indirecto la matriz Laplaciana se construye como

$$\mathbf{L} = \begin{cases} d^i & \text{si } i = j \\ -1 & \text{si } i \neq j \text{ y } (v_i, v_j) \in \text{ E} \\ 0 & \text{si no} \end{cases}$$

Además, también puede ser representada como

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

Puesto que la matriz de adyacencia **A** es simétrica y matriz de grado **D** es diagonal, el Laplaciano de un grafo **L** resulta ser también una matriz simétrica. Al igual que se hace anteriormente, para el grafo indirecto de la Fig. 2.1

$$[\mathbf{L}]_{ij} = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & -1 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ -1 & 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

Las propiedades de esta matriz son:

- Para un grafo no dirigido, **L** es positiva semi-definida, es decir, los valores propios de $\mathbf{A} + \mathbf{A^T}$ son no-negativos y para todo $\mathbf{x} \in \mathcal{R}^N$, $\mathbf{x^T L x} \geq 0$.

- **L** tiene asociados n valores propios no-negativos y reales ($0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N$), cuyo valor valor propio mínimo es cero. El segundo valor propio λ_2 es conocido como la conectividad algebraica de la red, esto es, es una medida del rendimiento/velocidad de los algoritmos de consenso.

- En **L**, la suma de los elementos de cada fila es cero. Por tanto, $\mathbf{L1} = 0$, donde **1** es un vector columna de unos $\mathbf{1} = [1 \cdots 1]$.

2.2. Estado de la materia

En este trabajo se aborda el problema del consenso promedio dinámico para un red de agentes autónomos y objetivos móviles. A partir de un set de señales variables en el tiempo, que sirven de entrada para los agentes, el problema trata el estudio previo y posterior análisis de la convergencia de un algoritmo distribuido que permita al grupo de agentes realizar el seguimiento del promedio de las señales, correspondientes al centroide de un objeto, a partir de la información de sus vecinos. Este problema resulta de interés en numerosas aplicaciones que requieren el consenso de información dinámica percibida por múltiples agentes. Algunas de las aplicaciones son coordinación multi-robot [4] o seguimiento distribuido [5].

En los últimos años los problemas de consenso se han estudiado en profundidad. El foco principal de estos estudios ha sido el caso estático, donde los agentes llegan a un consenso a partir de una función dependiente de las condiciones iniciales de los agentes. La literatura que recopila este foco de investigación véase, por ejemplo, [6], [7], [8], [9], [10] y sus referencias. En contraposición, en lo que se refiere al consenso dinámico, la literatura es escasa, si bien, en los últimos años ha tenido un fuerte auge. En lo que respecta a nuestro tema de interés, en [11] se propone un algoritmo de consenso promedio dinámico que con una determinada inicialización realiza tareas de seguimiento, con error en estado estacionario nulo. También, en [12], los autores parten del algoritmo de consenso estático de [13] para diseñar un modelo generalizado que permita el seguimiento del promedio de entradas cuyas derivadas están limitadas y que difieren por un ruido gaussiano de media cero. En este contexto, los algoritmos mencionados están diseñados en tiempo continuo y funcionan para redes con una topología de grafos fija, conectada y no dirigida. Adicionalmente, en [14], mediante un análisis de la estabilidad de las entradas con respecto al estado, se propone un algoritmo proporcional-integral para resolver el problema del consenso dinámico que, desde cualquier condición inicial, converge con un error de estado estacionario distinto de cero si las señales varían lentamente en el tiempo, y exactamente si las señales son estáticas. Además, en el trabajo desarrollado en [15] se diseña un tipo de algoritmos de consenso promedio dinámico en tiempo discreto cuyo análisis de convergencia se basa en las propiedades de estabilidad de entrada a salida en presencia de perturbaciones externas. Con una inicialización adecuada de los estados, los esquemas propuestos pueden rastrear, con un error de estado estacionario limitado, el promedio de las variables de entrada temporales cuya diferencia de enésimo orden está limitada. Si la diferencia de enésimo orden es asintóticamente nula, las estimaciones del promedio

convergen al promedio real asintóticamente con un retraso de un paso.

A continuación, se describen brevemente algunos conceptos relativos al problema del consenso promedio estático y dinámico, así como, una presentación a estos algoritmos en tiempo discreto.

2.2.1. Algoritmos de consenso

A lo largo de la historia de la informática han surgido diversas ramas que hoy en día siguen en desarrollo, uno de ellos, es el *consensus problem* dentro del área de la computación distribuida. Dada una red de *agentes*, donde el *sistema multi-agente* es un sistema dinámico, nos referimos al consenso como un estado en el que se alcanza un acuerdo respecto a una meta común o un valor de interés dependiente del estado de todos los agentes de la red. Este estado de acuerdo se alcanza mediante la interacción del grupo a través de enlaces de comunicación o sensores.

Para que los agentes alcancen el consenso, tienen que tener la misma forma de comunicación entre ellos. El intercambio de información entre un agente y todos sus vecinos de la red se rige por una regla o protocolo de interacción, se llama *algoritmo de consenso*. Gracias al algoritmo de consenso, un agente actualiza su información basándose en los datos recibidos de un agente vecino. En la Fig. 2.2 se muestran diversas representaciones gráficas de un algoritmo de consenso.

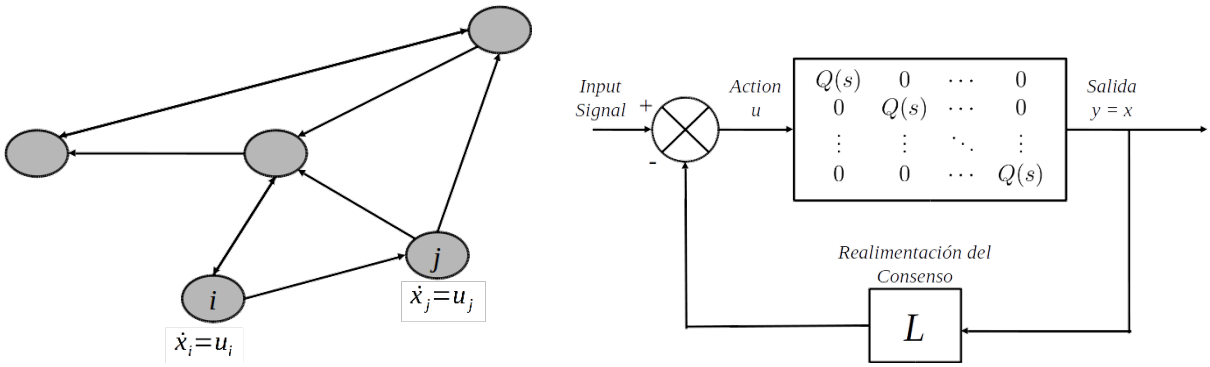
En lo que respecta a este capítulo, se comienza con una breve descripción de un *static average consensus* y se estudia su convergencia. Esta primera aproximación sirve como introducción previa al análisis de los algoritmos dinámicos. Más adelante, se describe el *dynamic average consensus algorithm* equivalente, así como el análisis de su convergencia. Finalmente, se presenta el algoritmo dinámico en tiempo discreto para la dinámica del integrador simple. Nótese que en algunas términos se mantiene la expresión original del inglés para facilitar su comprensión, ya que en castellano la traducción no es literal.

Introducción al *dynamic average consensus* (Consenso promedio dinámico)

En este apartado del capítulo se expone de manera resumida la formulación del problema del *dynamic average consensus* llevada a cabo en [16].

Se considera un grupo de N agentes capaces de

1. Enviar y recibir información con otros agentes.



(a) (b)

Figura 2.2: *Diferentes formas de representar los algoritmos de consenso: (a) red cuya dinámica de los agentes es un integrador simple. En ella, se aprecia como el agente i comunica su estado x_i a su vecino, el agente j; y en (b) se representa el diagrama de bloques de un sistema multi-agente dinámico, todos ellos con funciones de transferencia $Q(s) = \mathbf{1}/s$.*

2. Almacenar información obtenida por el agente.

3. Llevar a cabo cálculos computacionales de forma local.

La primera consideración del problema que hay que tener en cuenta es que cada agente tiene acceso a una señal de referencia local, $u^i(k) : \mathrm{N} \rightarrow \mathrm{R}$ en tiempo discreto y su equivalente en continuo. Esta señal puede ser la salida de un sensor del agente, o incluso la salida de otro algoritmo. Dentro de este marco de trabajo, el problema del *dynamic average consensus*, consiste en el diseño del algoritmo o protocolo que permita a los agentes por separado, seguir el *average* variable en el tiempo de las señales de referencia, expresado como

$$tiempo - continuo: \quad \mathbf{u}^{avg}(t) := (\frac{1}{N}\sum_{i=1}^{N} u^i(t))\mathbf{1},$$

$$tiempo - discreto: \quad \mathbf{u}^{avg}(k) := (\frac{1}{N}\sum_{i=1}^{N} u^i(k))\mathbf{1}$$

A continuación, se describen alguna de las propiedades deseadas en el diseño de soluciones algorítmicas para el *dynamic average consensus*:

- **escalabilidad**, de forma que el coste computacional llevado a cabo por cada agente, no crezca con respecto al tamaño de la red
- **robustez** a las perturbaciones de los sistemas físicos, como retrasos en las comunicaciones o pérdida de paquetes de información
- **precisión**, quiere decir que el algoritmo converge exactamente al *average* o, alternativamente, la distancia entre la medida estimada y la real del *average* está garantizada dentro de una cota.

A lo largo de esta memoria, se pretende explicar cómo la convergencia del algoritmo depende de la conectividad de la red y el ratio de cambio de la señal de referencia de cada agente.

Primer paso: *Static Average Consensus* (consenso promedio estático)

EXISTEN una amplia variedad de algoritmos de consenso en función de su marco de aplicación. Sin embargo, todos ellos comparten una cualidad común en su diseño. La idea de que los agentes inicien su estado de acuerdo con su propio valor de referencia y lo vayan ajustando en base a alguna realimentación lineal, que tenga en cuenta la diferencia entre su estado de acuerdo y el de sus vecinos. Los algoritmos presentan la siguiente forma

$$\begin{aligned} tiempo-continuo: &\quad \dot{x}_i(t) = -\sum_{j=1}^{N} a_{ij}(x^i(t) - x^j(t)), \\ tiempo-discreto: &\quad x_i(k+1) = x^i(k) - \sum_{j=1}^{N} a_{ij}(x^i(k) - x^j(k)), \end{aligned} \tag{2.1}$$

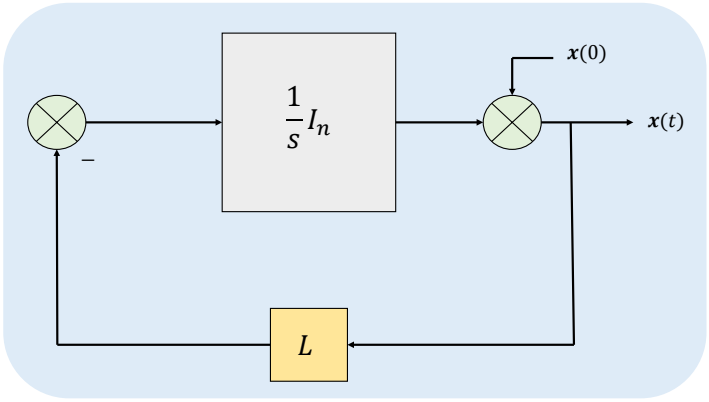
para $i \in \{1, \cdots, N\}$, donde la inicialización $x^i(0) = u^i$ es constante para las dos formas del algoritmo. Como se menciona en 2.1.2, $a_{ij} = [A]_{ij}$ es la *matriz de adyacencia* del grafo. Empleando una notación más compacta, en forma vectorial, los algoritmos de consenso estático se pueden representar mediante la *matriz Laplaciana* como

$$\begin{aligned} tiempo-continuo: &\quad \dot{x}_i(t) = -\mathbf{L}\mathbf{x}(t), \\ tiempo-discreto: &\quad x_i(k+1) = (\mathbf{I} - \mathbf{L})\mathbf{x}(k), \end{aligned} \tag{2.2}$$

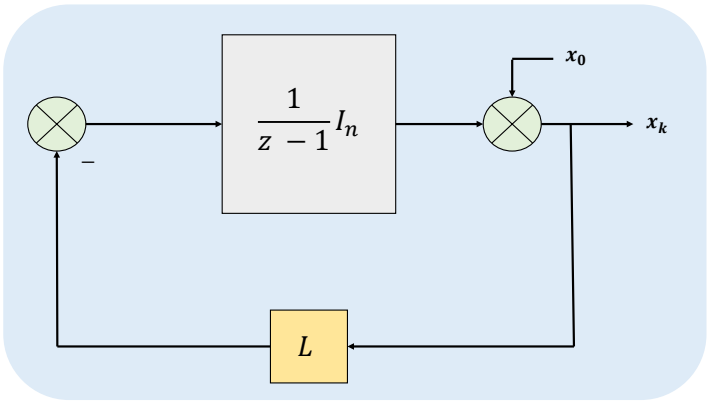
con $\mathbf{x}(0) = \mathbf{u}$. De aquí en adelante, se supondrá un grafo de comunicaciones fijo y por tanto, el sistema será *LTI* (*linear time-invariant*). El diagrama de bloques de los *static average consensus algorithms* definidos previamente se muestra en la Fig. 2.3. Como se puede ver en (2.2), la dinámica de estos algoritmos reside en el bucle de realimentación negativo, cuyo término de realimentación combina la matriz Laplaciana y un integrador simple ($1/s$ en tiempo-continuo y $1/(z-1)$ en tiempo discreto). En los *static average consensus algorithms*, la señal de referencia se introduce al sistema como condición inicial del estado del integrador y es invariante en el tiempo. Con una apropiada inicialización, el error de estos algoritmos puede converger a cero.

Convergencia de los *static average consensus algorithms*

SEA un grafo de comunicaciones constante a lo largo del tiempo y la señal de referencia u^i de cada agente $i \in \{1, \cdots, N\}$ un escalar constante. Entonces, para



(a) Continuous-time static average consensus.



(b) Discrete-time static average consensus.

Figura 2.3: *Se muestran dos diagramas de bloques correspondientes a los static average consensus algorithms (2.2). Aquí, las señales de entrada se establecen como condiciones iniciales, es decir,* $\mathbf{x}(0) = \mathbf{u}$ *en tiempo continuo y* $\mathbf{x}_0 = \mathbf{u}$ *para su equivalente en tiempo discreto.*

los algoritmos de consenso (2.1) se dan los siguientes resultados de convergencia [16]:

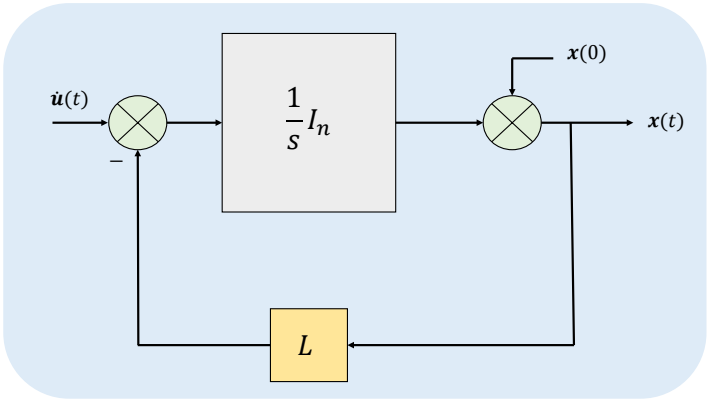
- Tiempo Continuo: Para todo $t \to \infty$, cada estado de acuerdo $x^i(t), i \in \{1, \cdots, N\}$ del algoritmo (2.1) converge a u^{avg} con un ratio exponencial mayor a λ_2, el menor valor propio distinto a cero de $\mathbf{L}$.

- Tiempo Discreto: Para todo $k \to \infty$, cada estado de acuerdo $x_k^i, i \in \{1, \cdots, N\}$ del algoritmo (2.1) converge a u^{avg} con un ratio exponencial mayor a $\rho \in (0, 1)$, proporcionando que la matriz Laplaciana satisfaga $\rho = \|\mathbf{I_N} - \mathbf{L} - \mathbf{1_N}\mathbf{1_N^T}/N\|_2 < 1$.

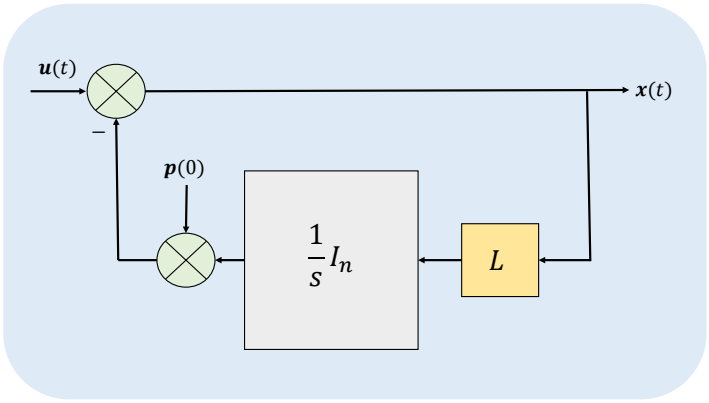
Un paso más: *dynamic average consensus*

HASTA ahora se ha visto como en los *static average consensus algorithms* las señales de referencia entran al algoritmo como condiciones iniciales y a lo largo de la ejecución de éste permanecen constantes. Por tanto, frente a señales que varían en el tiempo, el algoritmo no es capaz de llevar a cabo el *tracking* de ellas. Aquí, está claro que para poder realizar la tarea, es necesario alimentar de forma continua las

señales como entradas al sistema dinámico. Gracias a esto, el sistema puede reaccionar a los cambios producidos en las señales, sin tener que reinicializarse. Esta observación, se muestra en la Fig.(2.4).



(a) Continuous-time dynamic average consensus.



(b) Alternativa al Continuous-time dynamic average consensus.

Figura 2.4: *Se muestran dos diagramas de bloques correspondientes al dynamic average consensus algorithm (2.3) y (2.4). Mientras que las señales de referencia se aplican como condiciones iniciales para los algoritmos estáticos de consenso, las señales de referencia se aplican aquí como entradas al sistema. Aunque ambos sistemas son equivalentes, el sistema (a) requiere explícitamente la derivada de las señales de referencia, y el sistema (b) no requiere derivar las señales de referencia.*

Considerando la señales estáticas como una función escalón dinámica, el algoritmo, en el cual los valores de referencia consensuados por los agentes entran como entradas externas, resulta

$$\begin{aligned} \dot{\mathbf{x}}(t) = -\mathbf{L}x(t) + \dot{\mathbf{u}}(t), \quad & x^i(0) = u^i(0), \\ u^i(t) = h(t), \quad & i \in \{1, \cdots, N\}, \end{aligned}$$

donde $h(t)$ es una función variable en el tiempo. A partir de la ecuación anterior, en

[16] se propone el siguiente *dynamic average consensus algorithm*,

$$\dot{x}^i(t) = -\sum_{j=1}^{N} a_{ij}(x^i(t) - x^j(t)) + \dot{u}^i(t), \quad i \in \{1, \cdots, N\}, \tag{2.3}$$

$$x^i(0) = u^i(0), \tag{2.4}$$

Como se muestra en [11], si cada señal de entrada $\mathbf{u}^i, i \in \{1, \cdots, N\}$, tiene una transformada de Laplace cuyos polos están en el plano medio izquierdo y como máximo un polo cero (señales asintóticamente constantes), entonces todos los agentes cuyo protocolo sea (2.3) seguirán asintóticamente $\mathbf{u}^{avg}(t)$ con error cero.

El error de *tracking* del agente i se denota por

$$\mathbf{e}^i(t) = \mathbf{x}^i(t) - u^{avg}(t), \quad i \in \{1, \cdots, N\},$$

Convergencia de los *dynamic average consensus algorithms*

Sea un grafo de comunicaciones constante a lo largo del tiempo y sea la señal de referencia u^i de cada agente $i \in \{1, \cdots, N\}$ variable en el tiempo, con

$$\sup_{t \geq 0} \|(I_N - \frac{1}{N} 1_N 1_N^T)\dot{u}(t)\| = \gamma < \infty. \tag{2.5}$$

Entonces, las trayectorias de (2.3) y (2.4) están limitadas y satisfacen

$$\lim_{t \to \infty} \|x^i(t) - u^{avg}(t)\| \leq \frac{\gamma(\infty)}{\lambda_2}, \quad i \in \{1, \cdots, N\}, \tag{2.6}$$

de donde se obtiene $\sum_{j=1}^{N} x^j(t_0) = \sum_{j=1}^{N} u^j(t_0)$, y por tanto $\sum_{j=1}^{N} x^j(t) = \sum_{j=1}^{N} u^j(t)$ para $t \in [t_0, \infty)$.

Finalmente: Consenso en Tiempo Discreto

La implementación de algoritmos en tiempo continuo conlleva la continua comunicación entre agentes. En la práctica, debido a las restricciones en el ancho de banda de las comunicaciones, no se puede satisfacer este requerimiento. Como solución a este problema, se implementan en tiempo discreto, donde la comunicación entre agentes solo se produce en saltos de tiempo discretos. La principal diferencia radica en el ratio en el que sus estimaciones convergen al promedio de las señales de referencia.

Se propone una forma de expresar el algoritmo en tiempo discreto en [6]

$$x^i(k+1) = x^i(k) + \delta \sum_{j=1}^{N} a_{ij}(x^j(k) - x^i(k)). \tag{2.7}$$

De una forma más compacta, la dinámica del sistema discretizado se representa como

$$x(k+1) = \mathbf{P}x(k) \tag{2.8}$$

con $\mathbf{P} = \mathbf{I} - \delta$ ($\mathbf{I}$ es la matriz identidad) y $\delta > 0$ es el tamaño de paso. La ecuación (2.7) y $\mathbf{P} = \exp(-\delta \mathbf{L})$ son un caso especial cuya comunicación se produce entre vecinos de primer orden. Generalmente, la matriz $\mathbf{P}$ se llama *Perron matrix* de un grafo $\mathbf{G}$ con el parámetro δ.

2.2.2. Percepción y reconstrucción de un objeto deformable

El siguiente punto trata de describir brevemente conceptos básicos de las tareas de percepción multi-robot.

Existen diversos procesos robóticos, como el seguimiento de un objetivo (del inglés *tracking*), la percepción y la manipulación entre otros. Lo que es común a todos ellos es que en los tres la reconstrucción tridimensional de objetos reales es una tarea fundamental. El modelo del objeto se puede obtener utilizando una representación precisa y *a priori* construida del objeto,por ejemplo, un modelo CAD. Este método es el menos eficiente de todos debido a la necesidad de un profesional que lo modele.

Debido a que la observación exhaustiva consume mucho tiempo, si se quiere implementar la tarea eficientemente, es esencial seleccionar los puntos de vista que proporcionen más información. Una posible solución para aplicaciones robóticas, es un algoritmo de planificación de vistas que pueda encontrar automáticamente estas vistas (localizaciones) donde situar el sensor [17]. Para obtener una reconstrucción completa del objeto, los sensores deben ser colocados en diferentes puntos de vista que cubran toda la superficie del objeto.

Percepción basada en la visión

En las aplicaciones robóticas, la percepción visual del objeto que es manipulado es indispensable. Para ello, es útil tener información 3D de la escena, y estos datos se pueden obtener de múltiples tipos de sensores como cámaras *RGBD*, cámaras estéreo, etc. Las técnicas de visión $3D$ estéreo obtienen información sobre los objetos del entorno a partir de imágenes en tres dimensiones. Para obtener esta información 3D, se generan y utilizan nubes de puntos, como se muestra en la Fig. 1.3. Una nube de puntos es una representación tridimensional de puntos basada en una estructura de datos. Conceptualmente, se utilizan nubes de puntos para modelar tanto la forma como la ubicación de un objeto digitalmente en un sistema de coordenas tridimensional.

Procesado 3D del entorno: Como resultado de toda la información del entorno observada por los sensores $3D$, una nube de puntos con mucha información es generada. Para procesar esta información, se desarrollan varios algoritmos para construir un

modelo $3D$ completo de la escena. Uno de estos algoritmos es el *Octree*, y la herramienta software que lo soporta se llama *Octomap* [18].

OctoMap: Un marco de trabajo 3D basado en octree

OCTOMAP es un software de código abierto para el mapeado tridimensional. En este trabajo se ha hecho uso de la implementación de *Octomap* basada en [1], que utiliza una eficiente estructura de datos con *Octree*, lo que permite la representación volumétrica del modelo de forma compacta en la memoria y múltiples resoluciones de los mapas del modelo en cuestión. Usando una estimación probabilística de la ocupación, esta aproximación es capaz de representar modelos 3D volumétricamente, que incluyen áreas libres y desconocidas.

Representación probabilística: Los sensores de los robots realizan la construcción de mapas en 3D midiendo las distancias del entorno respecto al objeto a reconstruir. Estas mediciones se presentan con incertidumbre. Además, puede haber mediciones aleatorias causadas por objetos dinámicos o reflejos. Para generar un modelo preciso a partir de estas mediciones, la incertidumbre se aborda de forma probabilística. De esta manera, fusionando todas las mediciones realizadas, se obtiene una estimación robusta del objeto. Nótese que la fusión probabilística de la información sensorial permite la implementación de múltiples sensores y, en consecuencia, un escenario multi-robot.

Modelado 3D Completo: *Octomap* construye el mapa volumétrico de un objeto a partir de una nube de puntos como una rejilla 3D basada en una estructura *octree* basada en *vóxels*. Un *vóxel* es un cubo que contiene nubes de puntos. Estos cubos se colocan como una rejilla a lo largo de toda la nube de puntos para obtener una resolución más baja y homogénea que la inicial. Como se ha comentado en la sección 2.2.2, para generar un modelo completo del objeto, Octomap recibe como datos de entrada nubes de puntos, y proporciona como salida la información para el modelo $3D$ que se quiere visualizar (Fig. 2.1). *Octomap* es capaz de representar tanto las áreas ocupadas como el espacio libre en un entorno tridimensional. Además es actualizable. Esto significa que diferentes sensores pueden aportar información al mismo mapa, es decir, *Octomap* fusiona la información sensorial tal y como se menciona en el párrafo anterior.

Problema de la siguiente-mejor-vista

LA planificación de las vistas de los sensores basadas en los datos de los sensores 3D es a menudo conocido como el problema de la planificación de la siguiente mejor

vista NBV [17]. El punto clave de NBV es la decisión que debe tomar el robot de localizar un sensor basado en visión en la vista que le proporcione mayor información del objetivo. A pesar de toda la investigación sobre este tema como se menciona en [1] y [19], es necesario disponer del modelo predefinido de el objeto que se pretende representar.

En trabajos de investigación, es típico que cada investigador haga sus propias suposiciones sobre el problema con el que trata de lidiar. Sin embargo, cuando se trata del problema de la NBV, siempre hay dos hipótesis que son comunes a todos ellos. Una de ellas es que el problema de la NBV se encuentra dentro de un espacio de trabajo formado por un conjunto de vistas, las cuales han sido muestreadas sobre un modelo predefinido del espacio. Este modelo suele representarse mediante una figura geométrica (cilindro, esfera), en cuyo interior se encuentra el objeto objetivo. La otra hipótesis, es que estas aproximaciones no tienen en cuenta que el rendimiento del sensor depende de la distancia a la superficie del objetivo. Además, estas aproximaciones no pueden aplicarse a objetos cuyo tamaño exceda el campo de visión de la cámara (FOV).

Por tanto aquí se propone un sistema multi-cámara que incorpore una estartegia distribuida para la percepción y el seguimiento del objeto $3D$.

Capítulo 3

Formulación del problema del *tracking* y percepción de un objeto deformable basado en consenso

En este capítulo, se formalizan y se proporcionan las formulaciones de ambos problemas de interés. El primero, consiste en el estudio y análisis de los algoritmos del *tracking* de un objeto deformable usando métodos de consenso en sistemas multiagente, como se explica a continuación. El segundo, se refiere al desarrollo de un método para la percepción y reconstrucción tridimensional del objeto deformable. Finalmente, se lleva a cabo el estudio del algoritmo empleado.

3.1. Problema del *tracking* de un objeto deformable basado en consenso multi-agente

Dada una red de un equipo de agentes en $\mathbb{R}^2$ y un grupo de objetivos en $\mathbb{R}^2$, consideremos un objeto deformable que tiene que ser percibido por los agentes. Aquí, cada objetivo representa una esquina del objeto deformable. Se define un límite para la deformación del objeto,

Definición 3.1.1. *Llamamos R_{max}, la deformación máxima del objeto.*

Esto significa que R_{max} será el movimiento relativo de cada uno de estos objetivos con respecto al centroide del objeto deformable. Además, para percibir el objeto deformable, el equipo de agentes debe ser colocado a su alrededor a una distancia de seguridad, como lo llamamos d_s, para no chocar con el objeto. Por lo tanto, en este trabajo caracterizamos y analizamos cuidadosamente cómo debería ser ese d_s según los diferentes casos.

A continuación, se hace un breve recordatorio de algunos de los conceptos del Capítulo 2 aplicados a este trabajo. Primero se establece la topología de comunicación

del grupo de agentes y después se describe el marco del problema de consenso multi-robot.

Topología del grafo de comunicaciones de los agentes

Sea $G = (V, E)$ un *grafo no dirigido* que modele la comunicación en un conjunto de V de N agentes (nodos), es decir, $(i, j) \in E \Leftrightarrow (j, i) \in E$. En el conjunto V, los agentes están etiquetados con un índice $i \in \{1, \cdots, N\}$. Que E sea el conjunto de bordes (i, j), donde un borde es la comunicación entre vecinos y $\mathbf{A} = [a_{ij}] \in \mathbb{R}^{n \times n}$ es la matriz ponderada de *adyacencia*. Para un arco $(i, j) \in E$, j se llama un vecino de i. Consideraremos a un vecino como un agente i que comparte una arco con el agente j en cada momento, es decir, $(i, j) \in E$. Que N_i sea también el conjunto de vecinos del agente i, denotado por $N_i = \{j \in V : (i, j) \in E\}\ \forall i \in 1, \cdots, n$.

Suposición 3.1.1. *(Grafo de comunicación entre los agentes.) G es un ciclo no dirigido conectado donde los agentes están ubicados alrededor de los objetivos y la red de comunicaciones coincide con la posición de los agentes.*

En un grafo no dirigido $a_{ij} = a_{ji}$ para todos los $i, j \in V$. El *grado* de un agente i, se define como $d^i = \sum_{j=1}^{N} a_{ji}$. A continuación, la matriz de grado $\mathbf{D} = [d_{ij}]$ del grafo no dirigido G es $\mathbf{D} = Diag(d^1, ..., d^N) \in \mathbb{R}^{N \times N}$, por $i \in V$. La matriz *Laplaciana* asociada al grafo es $\mathbf{L} = \mathbf{D} - \mathbf{A}$, y también la definimos como:

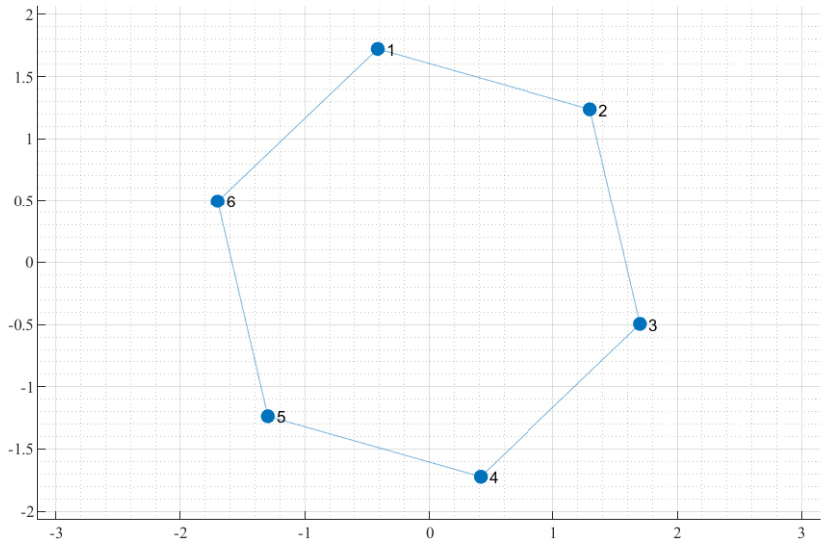


Figura 3.1: *Ejemplo de grafo indirecto para un grupo de seis agentes.*

La matriz Laplaciana asociada al gráfico es $\mathbf{L} = \mathbf{D} - \mathbf{A}$, y se define como:

$$\mathbf{L} = [l_{ij}] = \begin{cases} \sum_{s=1, j \neq i}^{m} a_{is} & j = i \\ -a_{ij} & j \neq i \end{cases}$$

Se ha de tener en cuenta que, $\mathbf{L}\mathbf{1}_n = 0$. Si se presta especial atención a la estructura de $\mathbf{L}$, al menos uno de los valores propios de $\mathbf{L}$ es cero mientras que el resto tiene partes reales no negativas. Para un grafo conectado no dirigida, cero es un valor propio simple de $\mathbf{L}$. Los valores propios de $\mathbf{L}$ se indican con λ_i y los ordenamos como $\lambda_1 = 0 < \lambda_2 \leq \lambda_3 \leq ... \leq \lambda_n$, para cualquier $i \in \{1, ..., n\}$.

Marco de trabajo

En este contexto, cada objetivo se etiqueta con un índice $s \in \{1, ..., m\}$ y el objeto es una representación genérica de un sólido con un comportamiento dinámico que sigue una trayectoria denotada como $\mathbf{q}_T(k)$ a lo largo del plano 2D. De ahora en adelante, $x_T^s(k)$, $s \in \{1, ..., m\}$ es la posición del objetivo s al instante $k \in \mathbb{Z}$ correspondiente a la coordenada x del vector $\mathbf{q}_T(k)$. Aunque el problema está en un marco 2D, la notación en la que analizamos y expresamos las ecuaciones y vectores a lo largo de los Capítulos 4 y 5 es en 1D. Sin embargo, todas las simulaciones presentes en el Capítulo 6 están enmarcadas en un espacio de trabajo 2D aplicando el método 1D por separado a las coordenadas x e y del problema.

El promedio del grupo de todos los objetivos es el centroide del objeto deformable, $x_T^{avg}(k) = \frac{1}{m}\sum_{s=1}^{m} x_T^s(k)$, where $x_T^s(k)$, donde $x_T^s(k)$ denota el vector de posiciones de cada objetivo $\forall s \in \{1, ..., m\}$. Se define una matriz de percepción cuya representación es la siguiente

$$\mathbf{M} = [m_{is}] = \begin{cases} 1 & \textit{agente i ve el objetivo s} \\ 0 & \textit{si no lo ve} \end{cases}$$

Una consideración importante del problema a tener en cuenta es que $\mathbf{M}$ cambia dependiendo del caso, como veremos más adelante.

Cada agente se identifica por el índice $i \in \{1, ..., N\}$. El vector $\mathbf{x}(k)$ almacena las posiciones de cada agente i al instante k. La posición de cada agente $x_A^i(k)$ depende de varios parámetros y se representa como,

$$x_A^i(k) = x^i(k) + R_s^i(k) \tag{3.1}$$

donde $x^i(k)$ denota la posición estimada asintóticamente en coordenadas absolutas por el agente i del centroide y $R_s^i(k)$ es la distancia relativa entre el centroide y la posición en la que queremos localizar al agente para que no choque con el objeto. Debido a que se emplea un protocolo lineal para la estimación del centroide, el valor de consenso estimado asintóticamente para cada agente $x^i(k)$, no es exactamente el mismo que éste. El valor estimado se ubicará en una región circular alrededor del centroide cuyo radio

es equivalente al error cometido en la estimación llamada previamente d_s, es decir, es la diferencia entre la posición real del centroide $x_T^{avg}(k)$ y la estimación de cada agente $x^i(k)$. En este trabajo, la distancia relativa entre el agente y el centroide como

$$R_s^i(k) = R_{max}(k) + d_s^i(k) \tag{3.2}$$

donde $R_{max}(k)$ es la deformación máxima del objeto (3.1.1) y la distancia de seguridad d_s es el error de seguimiento cometido en la estimación del centroide, como vemos en la Fig. (3.2).

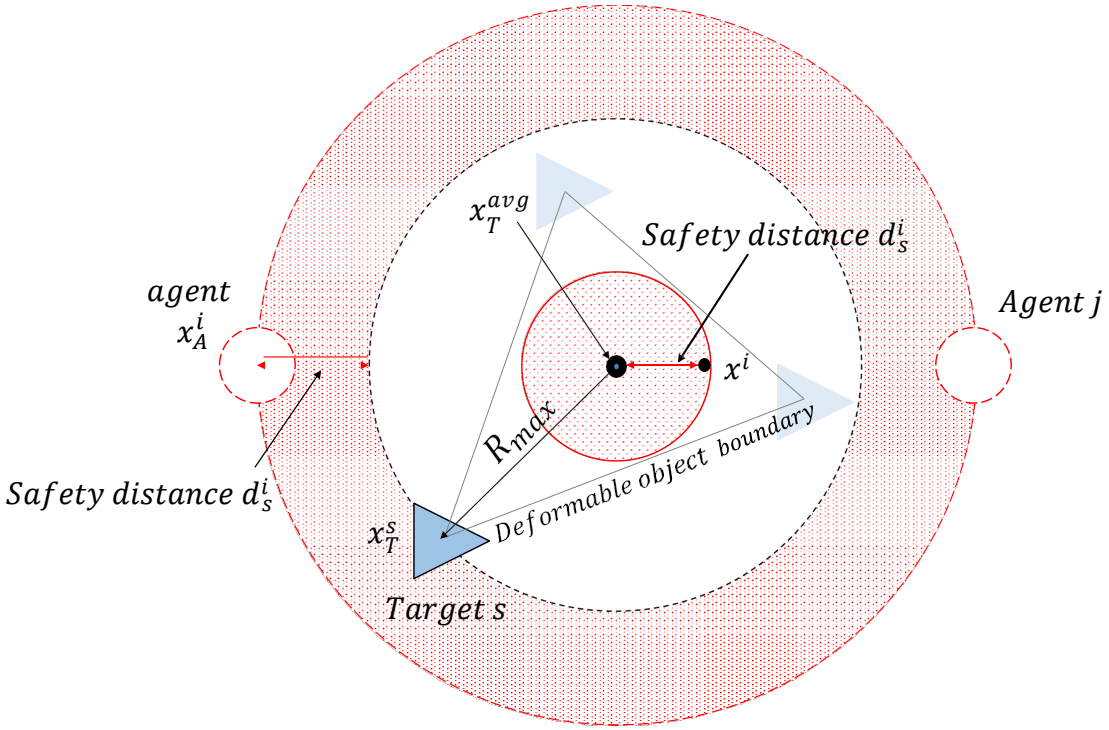


Figura 3.2: *Definición del problema. Un equipo de dos agentes debe seguir a un grupo de tres objetivos. Los objetivos representan los vértices de un objeto deformable. Los agentes deberán situarse en torno a una circunferencia de radio mínimo* $R_{max} + d_s$.

En este contexto, cada agente $i \in \{1, ..., N\}$ tiene acceso a una señal de entrada de referencia variable en el tiempo $u^i : \mathbb{Z}_{\geq 0} \to \mathbb{R}$ que representa la trayectoria temporal seguida por cada objetivo s visto por un agente, es decir,

$$u^i(k) = K \sum_{s=1}^{m} \mathbf{M}_{is} x_T^s(k),$$

donde K es una expresión a diseñar a continuación, el número de agentes es n y el número de objetivos es m. Tanto las posiciones de los agentes como las de los objetivos se definen en un sistema de referencia global arbitrario.

Sea $u^{avg}(k)$ una cantidad para la cual la red de agentes debe seguir asintóticamente su evolución hasta alcanzar un consenso dinámico; un ejemplo es considerado en [20]

donde un equipo de vehículos debe encontrar el centro de masa variable en el tiempo de una red de vehículos móviles. Este trabajo centra su atención en el consenso medio dinámico, es decir, en el problema del seguimiento de la media variable en el tiempo de los términos u^i, llamado el centro de la formación de los objetivos. Por lo tanto, se define la cantidad que deseamos que sea rastreada por el equipo de agentes como:

$$u^{avg}(k) = \frac{1}{n}\sum_{i=1}^{n} u^i(k) = \frac{1}{n}\sum_{i=1}^{n}(K\sum_{i=1}^{n}\sum_{s=1}^{m}\mathbf{M}_{is}x_T^s(k)) \tag{3.3}$$

Problema: *Dado un conjunto de m objetivos móviles, donde cada uno sigue una trayectoria diferente $x_T^s(k)$ y un equipo multi robot de n agentes cuya dinámica es (3.1), sobre una red con topología de interacción fija. El objetivo principal es implementar un algoritmo de consenso medio dinámico que los agentes puedan utilizar, para alcanzar asintóticamente el promedio de las señales de entrada de referencia $u^{avg}.(k)$ correspondiente al centroide del objeto deformable, permaneciendo al mismo tiempo en una región circular (con radio R_s^i), alrededor del valor de consenso.*

Para conseguir su objetivo, se debe caracterizar un límite superior para las señales de entrada que eviten la colisión de agentes con el objeto, cuya superficie está definida por los objetivos. Además, el equipo de agentes debe seguir al grupo de objetivos en movimiento que están rodeados en todo momento como se mueven por el entorno.

La convergencia del algoritmo se alcanza si la desviación relativa máxima entre cualquier entrada de referencia y el centroide del objeto está limitada por algún parámetro llamado R_s. El error de estado estacionario del algoritmo de consenso promedio dinámico es $\lim_{k\to\infty}|x^i(k) - u^{avg}(k)|$ del que se hablará más adelante. Esta medida se interpreta como la distancia entre el estado de consenso de cada agente y el valor de consenso que nos interesa lograr. El resultado será un error de estado estacionario cero o menor que (o igual a) un límite determinado que es estudiado para diferentes casos de percepción. Para una visualización general de este problema, la notación se representa en la Fig. 3.3.

3.2. Problema de percepción y reconstrucción

PARA el planteamiento del problema de percepción y reconstrucción del objeto deformable se considera que el objeto a partir del cual se genera un modelo 3D inicialmente rígido y está situado en un entorno vacío. El objeto es una representación genérica de un sólido estático en el plano del suelo. Para la tarea de reconstrucción volumétrica, se define que cada robot incorpora un par de sensores basados en la visión.

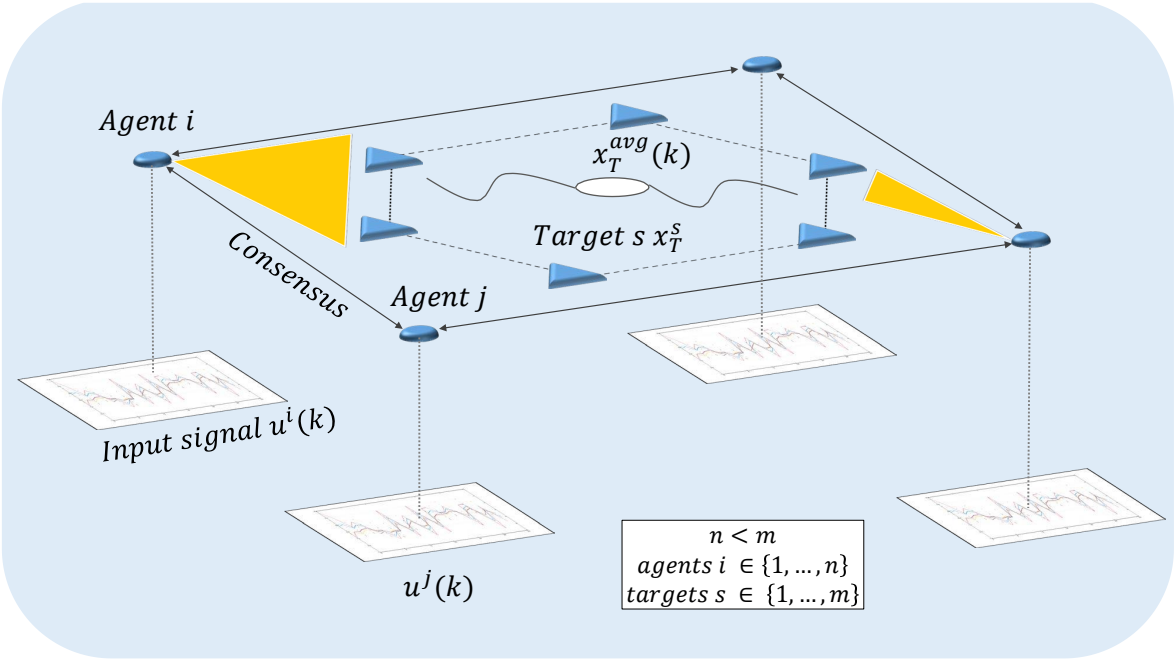


Figura 3.3: *Notación de la formulación del problema para cuatro agentes y seis objetivos. Los círculos azules representan los agentes, mientras que los triángulos son los objetivos. En círculo blanco del centro se corresponde con el centroide de la formación. Cada agente puede ver uno o más de un objetivo, campo de visión en amarillo, y recibe la posición de los objetivos mediante una señal de entrada u*

Estos sensores proporcionan la información sensorial necesaria para llevar a cabo esta tarea. En términos generales, la información sensorial se traduce como imágenes del objeto de destino. Para construir el mapa volumétrico del objeto es necesario adquirir varias vistas alrededor del objeto de destino para generar imágenes de toda su superficie. Por lo tanto, es necesario disponer de un mecanismo de posicionamiento de las cámaras que las coloque en estas vistas.

Dentro de este marco, los pares de cámaras estéreo se colocan alrededor del objeto de destino según la estrategia que se elija. La estrategia que consideramos consiste en una formación circular que encierra el objeto objetivo. Así, obtenemos un conjunto de imágenes que cubren toda la superficie del objeto y le dan una visión general del mismo.

Problema 2: *Según el planteamiento realizado, el marco de nuestro problema es un objeto que presenta un comportamiento dinámico, del cual pretendemos construir un modelo 3D instantáneamente, para observar los cambios producidos en su superficie. El primer objetivo, es diseñar una arquitectura multi-cámara, en la que tengamos múltiples puntos de vista alrededor del objeto. El segundo objetivo, es encontrar una estrategia para la posición y movimiento de las cámaras, que nos permita mover el sistema de*

percepción siguiendo al objetivo sin perderlo de vista.

3.3. Estudio del algoritmo de consenso

En esta sección, se presenta un *dynamic average consensus algorithm* distribuido con error acotado en estado estacionario para señales de entrada arbitrarias y variables en el tiempo. No sólo se presentan diferentes parámetros de diseño para controlar el tamaño del error, sino que también se caracterizan los requisitos del tiempo de muestreo para implementaciones en tiempo discreto.

Como se menciona en el Capítulo 2, la topología de la interacción de red es fija y modelada por un grafo no dirigido G que define la forma en que los agentes se comunican con sus vecinos, además, la transmisión de la información no está sujeta a retrasos de información. Aquí, en la red modelada previamente, el objetivo principal es utilizar un algoritmo distribuido que lleve a cada agente a realizar un seguimiento asintótico de $\frac{1}{m}\sum_{s=1}^{m} x_T^s(k)$. Para hacer esto, comenzamos con un algoritmo de consenso dinámico propuesto en [21], en tiempo discreto, inicializado en $zi(0) \in \mathbb{R}$ y $v^i(0) \in \mathbb{R}$ con $\sum_{i=1}^{n} v^i(0) = 0$ y que resuelve el problema:

$$v^i(k+1) = v^i(k) + \delta\alpha\beta \sum_{j=1}^{N} a_{ij}(x^i(k) - x^j(k)), \tag{3.4}$$

$$x^i(k+1) = x^i(k) + \Delta u^i(k) - \delta\alpha(x^i(k) - u^i(k)) - \delta\beta \sum_{j=1}^{N} a_{ij}(x^i(k) - x^j(k)) - \delta v^i(k), \tag{3.5}$$

donde para $i \in \{1, \cdots, N\}$, x^i, $v^i \in \mathbb{R}$ son variables asociadas con el agente i y $\Delta u^i(k) = u^i(k+1) + u^i(k)$. También, $\mathbf{L}$ es el Laplaciano del grafo indirecto $\mathbf{G}$ que modela la topología de la interacción. El algoritmo emplea los dos últimos términos de la ecuación (3.5) como una realimentación integral proporcional (PI) para establecer el acuerdo entre el conjunto de agentes y vecinos, mientras estos agentes, debido a los dos primeros términos de (3.5), se están moviendo hacia sus respectivas señales de entrada. Bajo condiciones ideales en la topología de las comunicaciones, el resultado de implementar este algoritmo es que cada agente sigue eventualmente el *average* de todas las entradas en la red. Se pueden diferentes rendimientos del algoritmo en función del diseño de los parámetros $\alpha, \beta \in \mathbb{R}$.

Se ha de comentar, que la implementación de este algoritmo de forma iterativa en cada salto de tiempo k, requiere el acceso a un valor futuro, en $k+1$, de la entrada de referencia. Este requisito no es fácilmente proporcionable cuando la referencia de

entrada es muestreada de un proceso físico o es un resultado de otro algoritmo online. La solución que se propone en [21], evita la necesidad del conocimiento de valores futuros de la referencia de entrada. Esta solución consiste en la introducción de una variable intermedia $z^i(k) = x^i(k) - u^i(k)$ y como resultado el algoritmo queda:

$$\begin{aligned} v^i(k+1) &= v^i(k) + \delta\alpha\beta \sum_{j=1}^{N} a_{ij}(x^i(k) - x^j(k)), \\ z^i(k+1) &= z^i(k) - \delta\alpha z^i(k) - \delta\beta \sum_{j=1}^{N} a_{ij}(x^i(k) - x^j(k)) - \delta v^i(k), \\ x^i(k) &= z^i(k) + u^i(k), \end{aligned} \tag{3.6}$$

donde por $i \in \{1, ..., N\}, v^i, z^i, x^i$ son variables asociadas con el agente i, como se presenta en [16]. Las constantes $\alpha, \beta \in \mathbb{R}$ son parámetros de diseño para mejorar el rendimiento del algoritmo. Sin embargo, en aras de la simplicidad, en este trabajo se fijan en un valor fijo tal que:

$$\alpha = \beta = 1. \tag{3.7}$$

Para una comprensión completa de estos parámetros ver [21]. Tenga en cuenta que (3.6) es implementable sin el uso de entradas futuras.

A continuación se muestra que este sistema dinámico, bajo un grafo no dirigido, es estable y convergente. Si se supone que las diferencias de las entradas de los agentes de la red aseguran

$$\|\mathbf{\Pi}_N \mathbf{\Delta u}\|_{ess} = \underset{k\in\mathbb{Z}_{\geq 0}}{\text{máx}} \|(\mathbf{I}_N - (\frac{1}{N})(\mathbf{1}_N \mathbf{1}_N^\top))\mathbf{\Delta u}(\mathbf{k})\| \leq \gamma < \infty, \tag{3.8}$$

donde $\mathbf{\Delta u}(k) = \mathbf{u}(k) - \mathbf{u}(k-1)$, como se aprecia en [16, Corolario 4.2], la convergencia de (3.6) sobre los grafos no dirigidos está garantizada, para cualquier valor de $\alpha, \beta > 0$. Las trayectorias de (3.6) $\mathbf{G}$ están acotas y satisfacen

$$\lim_{k\to\infty} \left| x^i(k) - u^{avg}(k) \right| \leq \frac{\gamma}{\beta\lambda_2\delta}, i \in \{1, ..., N\}, \tag{3.9}$$

siempre que δ se haya elegido tal que $\delta \in (0, \text{mín}\{\alpha^{-1}, \beta^{-1}(d_{out}^{max})^{-1}\})$, donde $\delta \in \mathbb{R}_{>0}$ es el tamaño de muestreo. Aquí, $|x^i(k) - u^{avg}(k)|$ es el error de *tracking* en estado permanente y λ_2 es el segundo valor propio de la matriz Laplaciana.

Comentar que, en [21], para garantizar que el error en estado estacionario del *tracking* esté acotado, la diferencia de los vectores de las entradas de la red en el espacio de consenso están acotadas. Finalmente, en [21], se identifican las condiciones de las entradas y sus diferencias, bajo las cuales el algoritmo (3.6) converge asintóticamente

al consenso con error estacionario nulo. Las clases de entradas dependen del parámetro α que debe ser conocido por cada agente para obtener un error estacionario nulo. En el Capítulo 6 se comrprobará experimentalmente.

Capítulo 4

Propuesta de Estrategia Multi-Robot

En este capítulo se proponen diferentes estrategias para tratar de resolver el primer problema planteado en la sección anterior. Por un lado, se plantean tres situaciones diferentes para un sistema distribuido, es decir, no se conoce de antemano la trayectoria que seguirá el objeto en cuestión. Además, cada agente solo se comunica con sus vecinos, teniendo que llegar a un consenso con los agentes de toda la red. Por otro lado, se propone una estrategia centralizada, donde se calcula una formación geométrica alrededor del objeto de interés. Aquí, el *tracking* del objeto se lleva a cabo a partir del conocimiento previo de la trayectoria que sigue este.

4.1. Propuestas de estrategia multi-robot basada en *dynamic average consensus*

A continuación, se presentan tres casos diferentes bajo una percepción cambiante o fija de los objetivos para abordar el problema:

Red con percepción fija en el tiempo y $n = m$: Dados el mismo número de objetivos que de agentes y para una matriz de percepción $\mathbf{M}$ constante, la señal de entrada de cada agente i es

$$u^i(k) = x_T^i(k), \tag{4.1}$$

donde $u^{avg}(k) = x_T^{avg}(k)$ y x_T^i es la posición del objetivo i en cada instante k. Por tanto, un agente ve el mismo objetivo a lo largo de toda la trayectoria. El propósito de $\mathbf{M}$ es asignar los objetivos que están siendo vistos por cada agente i al instante k.

Red con percepción variable en el tiempo y $n = m$: En este caso, el número de agentes es el mismo que el número de objetivos, $n = m$, pero los agentes giran

alrededor del grupo de objetivos. Esto significa que la percepción varía con el tiempo, es decir, cada agente ve un objetivo y todos los objetivos se ven en cada instante k. Por lo tanto, la matriz de percepción $\mathbf{M}$ será variable en el tiempo y como resultado la acción de control también. Además, en $\mathbf{M}$ hay un número de unos que es exactamente igual al número de agentes y, por lo tanto, exactamente igual al número de objetivos. La acción es como,

$$u^i(k) = \sum_{s=1}^{m} \mathbf{M}_{is} x_T^s(k), \tag{4.2}$$

donde $\mathbf{u}^{avg}(k) = \mathbf{x}_T^{avg}(k)$. En est capítulo se explica cómo varían las condiciones que deben cumplir las señales de entrada y, por último, se proporciona una expresión que garantiza una cota de convergencia del algoritmo donde en algunos casos el error en estado estacionario es cero.

Red con percepción fija pero $n < m$: Ahora, cada objetivo se ve exactamente una vez, pero al ser el número menor de agentes ($n < m$), los agentes individuales pueden ver uno o más objetivos. Luego se considera la expresión (4.3) y se muestra cómo el promedio de las entradas $\mathbf{u}^{avg}$ es el mismo que en los anteriores casos propuestos. Por lo tanto, la matriz de percepción $\mathbf{M}$ será constante. Ahora, se propone la siguiente acción que tiene propiedades interesantes que mantiene el promedio,

$$u^i(k) = \frac{n}{m} \sum_{s=1}^{m} \mathbf{M}_{is} x_T^s(k). \tag{4.3}$$

Consideramos que es esta acción porque queremos que el centroide del objeto sea igual al promedio de las entradas a medida que probamos a continuación.

A continuación, se muestra un lema de la expresión propuesta anterior

Lemma 4.1.1. *(Promedio de (4.3).) Deje que G sea un grafo no dirigido sobre una red con percepción fija y $n < m$. Si las acciones se expresan de acuerdo con (4.3), el promedio de las acciones es igual al centroide de los objetivos.*

Demostración. Entonces, el centroide del objeto resulta como

$$\mathbf{u}^{avg}(k) = \frac{1}{n} \sum_{i=1}^{n} u^i(k) = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{n}{m} \sum_{s=1}^{m} \mathbf{M}_{is} x_T^s(k)\right) = \mathbf{x}_T^{avg}(k). \tag{4.4}$$

□

Recordar que el propósito de $\mathbf{M}$ es asignar los objetivos que están siendo vistos por cada agente i al instante k.

En la Fig. 4.1 se muestra un ejemplo de este último caso; los datos de la simulación se presentan en (4.1). La trayectoria que siguen los objetivos es la siguiente:

$$x_T^i(k) = 2 + \sin(w(k)k + \phi(k)) + b^i,$$
$$y_T^i(k) = 2 + \cos(w(k)k + \phi(k)) + b^i,$$

$\forall\, k \in \mathbb{Z}_{\geq 0}$ donde la frecuencia y la fase cambian aleatoriamente a lo largo del tiempo, como: $w \sim N(0, 0{,}25)$ y $\phi \sim N(0, (2{,}4674)^2)$. Aquí, el equipo de agentes percibe este proceso tomando muestras síncronas a través de señales de entrada.

El *bias* b^i en cada agente es fijo y su valor es $b^1 = -0{,}5, b^2 = 1, b^3 = 0{,}6, b^4 = -0{,}9, b^5 = -0{,}6, b^6 = 0{,}4, b^7 = -1, b^8 = 0{,}3, b^9 = 0{,}25, b^{10} = -0{,}85$. Aquí, cada agente, después de cada muestreo, trata de obtener el promedio de las mediciones antes del siguiente tiempo de muestreo. Por lo tanto, la entrada $u^i(k)$ permanece constante en $u^i(l)$ entre los tiempos de muestreo l y $l+1$. Fig. 4.1 muestra la evolución a lo largo del tiempo de los estados internos locales x^i y y^i generados por el algoritmo (3.6) con $\alpha = \beta = 1$. El ancho de banda de comunicación es $\delta = 0{,}25$, es decir, 4 Hz. En esta aplicación, la matriz de percepción es una matriz aleatoria y constante en el tiempo. Los resultados de la implementación de (3.6) en este caso proporcionaron un seguimiento perfecto después de algunas iteraciones.

Caso	n	m	k	δ	nº comunicaciones
1	4	10	30	0.25	1 iteración
2	4	10	30	0.25	3 iteración
3	4	10	30	0.25	10 iteración

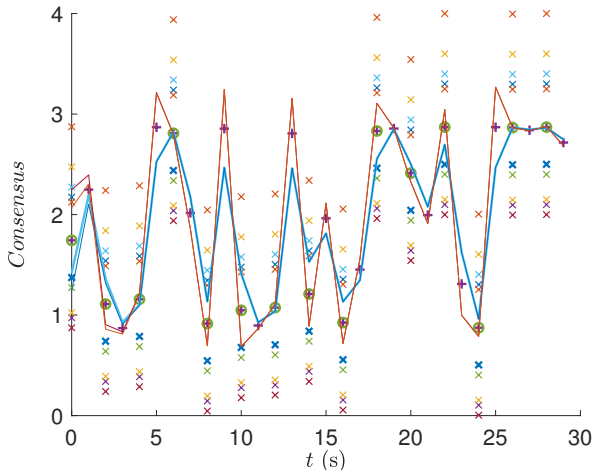
Tabla 4.1: Parámetros de la Fig.(4.1).

4.2. Estudio de la convergencia de la estrategia multi-robot

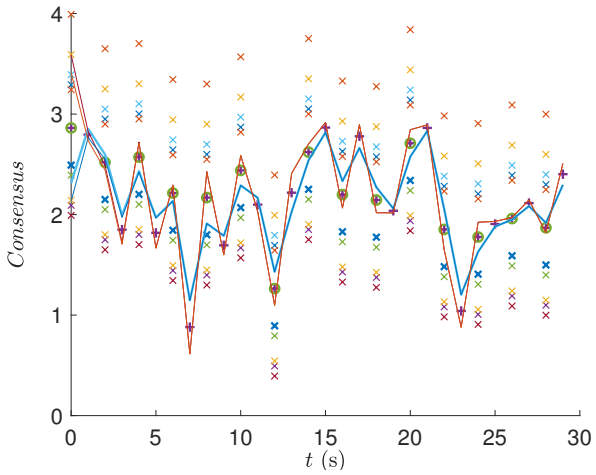
4.2.1. Caracterización de δ

En [16, Theorem S2] se proponen dos alternativas para calcular el tamaño del escalón admisible δ para (3.6). Entonces, entre estos dos, en términos de especificar los tamaños de paso apropiados δ que aseguren las propiedades de convergencia del algoritmo discreto, se usa uno que habla de λ_N.

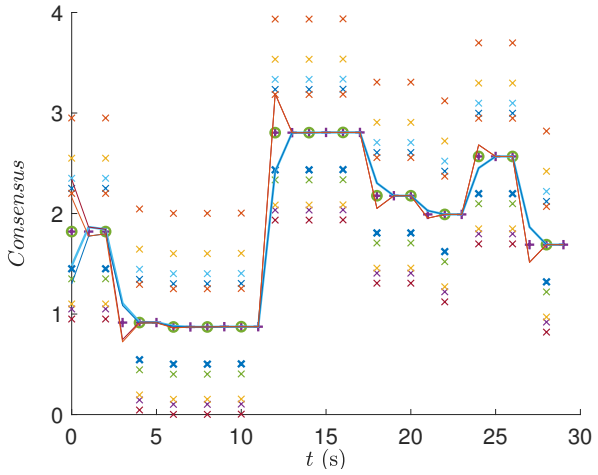
A continuación, se exploran los límites en el tamaño de paso δ que aseguran que (3.6) sea convergente y siga el centroide del objeto.



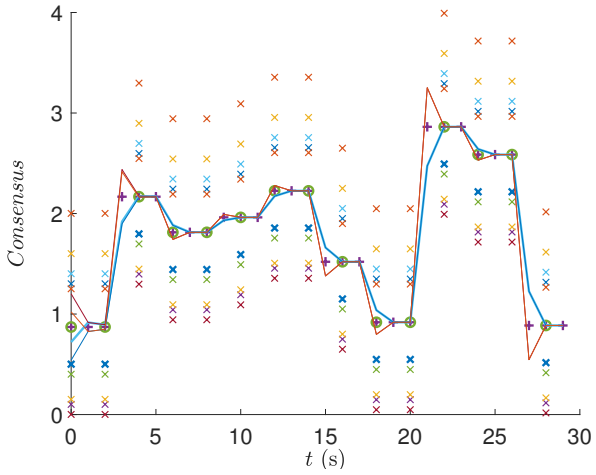
(a) *1 iteración en x*



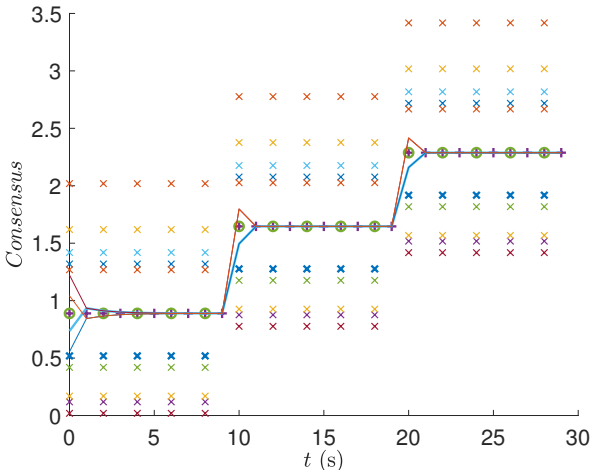
(b) *1 iteración en y*



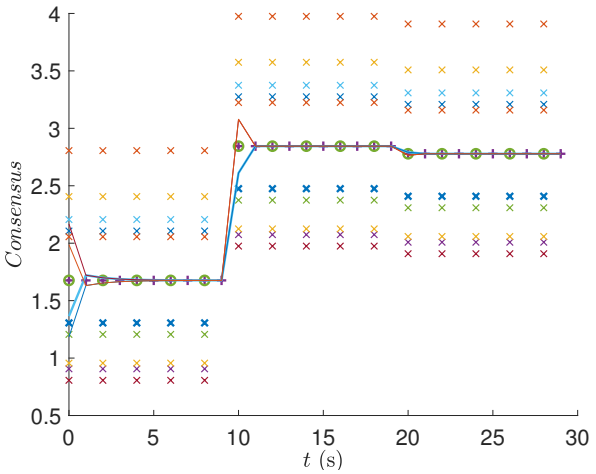
(c) *3 iteración en x*



(d) *3 iteración en y*



(e) *10 iteración en x*



(f) *10 iteración en y*

Figura 4.1: *Los valores de los parámetros de esta simulación se muestran en la tabla 4.1. Líneas sólidas: representan la historia temporal del estado de acuerdo de cada agente generado por el consenso promedio dinámico de (3.6); ×: puntos de muestreo en $m\Delta k$; ∘: la media en $m\Delta k$; +: la media de las señales de referencia. Como se ha comentado en los capítulos previos, bajo una inicialización apropiada, los agentes son capaces de rastrear $x_T^{avg}(k)$ con un pequeño error. La columna de la izquierda representa el eje x y la de la derecha el eje y. En este contexto, (a) y (b) es el consenso promedio dinámico con una iteración; (c) y (d) para tres iteraciones y finalmente, (e) y (f) para 10 iteraciones.*

Lemma 4.2.1. *(Caracterización del tamaño de paso del sistema.) Sea G un grafo no dirigido conectado con una topología de interacción fija definido en (3.1), con $\alpha = \beta = 1$ como en (3.7), el tamaño del paso es*

$$\delta_{accu} \in (0, \frac{1}{2}), \tag{4.5}$$

tal que $\delta_{accu} \in (0, \min\{\alpha^{-1}, 2\beta^{-1}(\lambda_N)^{-1}\})$, entonces el algoritmo en tiempo discreto (3.6) converge.

Demostración. Algoritmo (3.6) converge si $\delta_{accu} \in (0, \min\{\alpha^{-1}, 2\beta^{-1}(\lambda_N)^{-1}\})$, donde λ_N es el mayor valor propio de la matriz laplaciana y β se expresa en (3.7).En este caso, λ_N tiene una expresión para el grafo cíclico considerado dada por [22].

Para demostrarlo de forma teórica, en el trabajo desarrollado en [22] se definen las matrices *Tridiagonal Toeplitz* y *Circulant matrix*. Además, en [22, Lemma A.1] se calculan los valores propios de estas matrices, en particular λ_N. Los valores propios de $Circ_N(a, b, c)$ para $i \in \{1, ..., N\}$ se calculan como:

$$\lambda_i^{theo} = b + (a + c)\cos(\frac{i2\pi}{N}) + \sqrt{-1}(c - a)\sin(\frac{i2\pi}{N}), \tag{4.6}$$

dado un $\delta_{accu}^{theo} \in (0, \min\left\{\alpha^{-1}, 2\beta^{-1}(\lambda_N^{theo})^{-1}\right\}.)$

Se recuerda que en el supuesto de que el grafo de comunicaciones sea un ciclo, la matriz circulante coincide con el laplaciano del grafo G con $a = c = -1$ y $b = 2$. Por lo tanto, el valor propio λ_N calculado a partir de (4.6) coincide con el del ciclo que se propone en este trabajo y es $\lambda_N \leq 4$.

Por lo tanto, la convergencia de (3.6) está garantizada ya que $\delta_{accu} \in (0, \min\{\alpha^{-1}, 2\beta^{-1}(\lambda_N)^{-1}\}.)$

Por tanto, se obtiene (4.5) y se concluye la demostración. □

Tenga en cuenta que, como la topología de interacción del grafo es fija, el tamaño de paso permanecerá constante a pesar de la percepción de cambio de los objetivos.

4.2.2. Análisis de la convergencia de la estrategia propuesta

A continuación, se analiza las dimensiones de d_s cuyo valor es el mismo que el de la cota o *bound*, tal y como se ha comentado anteriormente. Para ello, debemos estudiar el comportamiento del parámetro presentado en este trabajo como γ, según

(3.8) para el $\mathbf{u}$ que proponemos. Recordar que se define la cota o *bound* (3.8), como la diferencia entre la estimación del centroide del objeto y su valor real para cada agente. Además, tal como se ha comentado en el Capítulo 3, existe una deformación máxima para el objeto llamada R_{max}, de modo que los objetivos siempre estén limitados a través de una región circular, es decir, a lo sumo los objetivos se mueven en un radio de R_{max}. Para percibir el objeto deformable, el equipo de agentes debe estar situado a su alrededor a una distancia suficiente, para no chocar con el objeto. En esta sección, se describe el método de deducción de la cota R_{max}.

Nótese que el algoritmo que se usa en (3.6), presentado en [21], está específicamente diseñado para trabajar con la suposición $n = m$ tal que cada agente ve el mismo objetivo en cada instante k, es decir, $\mathbf{M}$ es constante (percepción fija.) Definimos $\mathbf{Delta}(k)$ como la distancia relativa en cada instante entre la señal de entrada y el centroide del objeto deformable, es decir,

$$\Delta^i(k) = u^i(k) - u^{avg}(k) = \sum_{s=1}^{m} \mathbf{M}_{is} x_T^s(k) - \frac{1}{n} \sum_{i=1}^{n} \sum_{s=1}^{m} \mathbf{M}_{is} x_T^s(k), \tag{4.7}$$

para $i \in \{1, \cdots, N\}$ y $s \in \{1, \cdots, m\}$. Comentar que a partir de ahora se expresará en forma vectorial, de tal manera que $\mathbf{\Delta} = \{\Delta^i, \cdots, \Delta^N\}$. Gráficamente, se interpreta este parámetro como el radio de un círculo cuyo centro es el promedio de las señales de entrada.

A continuación, se estudia el parámetro d_s (distancia de seguridad) y se analiza cómo cambia el parámetro R_{max} para cada una de nuestras implementaciones:

Percepción variable y $n = m$: El vector de posiciones de los objetivos vistos por los agentes es $\mathbf{u}(k)$.

El siguiente resultado muestra que para las señales de entrada cuya proyección ortogonal en el espacio de acuerdo está limitada, el algoritmo (3.6) proporciona una solución de nuestro problema para cualquier condición inicial dada.

Proposición 4.2.1. *(Algoritmo (3.6) resuelve el problema del consenso bajo una red con percepción variable en el tiempo y $n = m$.) Sea G un grafo no dirigido conectado. Se supone que las diferencias de las entradas de los agentes de la red aseguran*

$$\|\mathbf{\Pi_N \Delta u}\|_{ess} = \|\mathbf{\Delta}(k) - \mathbf{\Delta}(k-1)\|_{ess} = \gamma < 2R_{max}, \tag{4.8}$$

la convergencia de (3.6) sobre grafos no dirigidos está garantizada, si y solo si, para cualquier $\alpha, \beta > 0$, las trayectorias de (3.6) sobre G sin retardo de comunicación, inicializadas en $z^i(0) \in \mathbb{R}$ y $v^i(0) \in \mathbb{R}$ con $\sum_{i=1}^{N} v^i(0) = 0$ están limitadas y satisfacen

$$\lim_{k\to\infty} \left| x^i(k) - u^{avg}(k) \right| \leq \frac{2R_{max}}{\beta\lambda_2\delta_{accu}}, i \in \{1, ..., N\}, \tag{4.9}$$

proporcionado δ_{accu} tal que $\delta_{accu} \in (0, \min\{\alpha^{-1}, 2\beta^{-1}(\lambda_N)^{-1}\})$, donde $\delta_{accu} \in \mathbb{Z}_{>0}$ es el tamaño de muestreo.

Demostración. Como se demuestra en [21, Corolario 4.2], si

$$\|\mathbf{\Pi_N \Delta u}\|_{ess} = \|\mathbf{\Delta}(k) - \mathbf{\Delta}(k-1)\|_{ess} = \gamma < \infty \tag{4.10}$$

entonces

$$\lim_{k\to\infty} \left| x^i(k) - u^{avg}(k) \right| \leq \frac{\gamma}{\beta\lambda_2\delta_{accu}}, i \in \{1, ..., N\}, \tag{4.11}$$

así, se va a estudiar un valor que acota a γ.

Para ello, se aplica la desigualdad triangular, que es propiedad de las normas, para obtener

$$\gamma = \|\mathbf{\Delta(k) - \Delta(k-1)}\|_{ess} \leq \|\mathbf{\Delta}(k)\|_{ess} + \|\mathbf{\Delta}(k-1)\|_{ess}. \tag{4.12}$$

Entonces, si se llama $r(k) = \|\mathbf{\Delta}(k)\|_{ess}$ a la distancia máxima de todos los agentes desde el centroide de la formación en cada instante de k, la expresión anterior queda

$$\gamma = \max_{k\geq 0}\{\|\mathbf{\Delta}(k) - \mathbf{\Delta}(k-1)|\} \leq \max_{k\geq 0}\{r(k)\} + \max_{k\geq 0}\{r(k-1)\}. \tag{4.13}$$

Ahora, $R_{max} = \max_{k\geq 0}\{r(k)\}$. Recuerdese que se define R_{max} como la deformación máxima del objeto, es decir, es el radio máximo a lo largo de toda la trayectoria de la región circular formada por las posiciones relativas de los agentes respecto al centroide del objeto. Entonces

$$\gamma = \max_{k\geq 0}\{\|\mathbf{\Delta}(k) - \mathbf{\Delta}(k-1)|\} \leq 2R_{max} = \gamma^{Teo}. \tag{4.14}$$

entonces ahora (4.8) se obtiene simplemente teniendo en cuenta que $\beta = 1$ por simplicidad.

□

Notar que λ_2 se puede obtener como se indica en la demostración del lema 4.2.1.

Percepción fija con $n < m$: En este caso, se define $\hat{\mathbf{\Delta}}(k)$ como la distancia relativa en cada instante entre la señal de entrada y el centroide del objeto deformable, es decir,

$$\begin{aligned}\hat{\Delta}^i(k) &= \hat{u}^i(k) - \hat{u}^{avg}(k) = \\ &= \frac{n}{m}\sum_{s=1}^{m}\mathbf{M}_{is}x_T^s(k) - \frac{1}{n}\sum(\frac{n}{m}\sum_{s=1}^{m}\mathbf{M}_{is}x_T^s(k)),\end{aligned} \tag{4.15}$$

por $i \in \{1, \cdots, n\}$ y $s \in \{1, \cdots, m\}$. Comenta que a partir de ahora se expresará en forma vectorial, de tal forma que $\hat{\mathbf{\Delta}} = \{\hat{\Delta}^i, \cdots, \hat{\Delta}^n\}$.

El siguiente resultado muestra que para las señales de entrada cuya proyección ortogonal en el espacio de acuerdo está limitada, el algoritmo (3.6) proporciona una solución de nuestro problema para cualquier condición inicial dada.

Proposición 4.2.2. *(Algoritmo (3.6) resuelve el problema del consenso bajo una red con percepción variable en el tiempo y $n = m$.) Sea G un grafo no dirigido conectado. Se supone que las diferencias de las entradas de los agentes de la red aseguran*

$$\|\mathbf{\Pi_N}\hat{\mathbf{\Delta}}\mathbf{u}\|_{ess} = \|\hat{\mathbf{\Delta}}(k) - \hat{\mathbf{\Delta}}(k-1)\|_{ess} = \hat{\gamma} < 2\hat{R}_{max}, \tag{4.16}$$

la convergencia de (3.6) sobre grafos no dirigidos está garantizada, si y solo si, para cualquier $\alpha, \beta > 0$, las trayectorias de (3.6) sobre G sin retardo de comunicación, inicializadas en $z^i(0) \in \mathbb{R}$ y $v^i(0) \in \mathbb{R}$ con $\sum_{i=1}^{N} v^i(0) = 0$ están limitadas y satisfacen

$$\lim_{k\to\infty}\left|x^i(k) - u^{avg}(k)\right| \leq \frac{2\hat{R}_{max}}{\beta\lambda_2\delta_{accu}}, i \in \{1, ..., N\}, \tag{4.17}$$

proporcionado δ_{accu} tal que $\delta_{accu} \in (0, \min\{\alpha^{-1}, 2\beta^{-1}(\lambda_N)^{-1}\})$, donde $\delta_{accu} \in \mathbb{Z}_{>0}$ es el tamaño de muestreo.

Demostración. Si se desarrolla la expresión (4.16) en función de las posiciones de los objetivos, se obtiene

$$\begin{aligned}\hat{\gamma} &= \|\hat{\mathbf{u}}(k) - \hat{\mathbf{u}}(k-1) - \frac{1}{n}\sum_{i=1}^{n}(\hat{\mathbf{u}}(k) - \hat{\mathbf{u}}(k-1))\|_{ess} = \\ &= \|\frac{n}{m}\sum_{s=1}^{m}\mathbf{M}_{is}x_T^s(k) - \frac{1}{n}\sum_{i=1}^{n}(\frac{n}{m}\sum_{s=1}^{m}\mathbf{M}_{is}x_T^s(k)) - \\ &- \frac{n}{m}\sum_{s=1}^{m}\mathbf{M}_{is}x_T^s(k-1) - \frac{1}{n}\sum_{i=1}^{n}(\frac{n}{m}\sum_{s=1}^{m}\mathbf{M}_{is}x_T^s(k-1))\|_{ess} \\ &= |\frac{n}{m}|\|\mathbf{\Delta}(k) - \mathbf{\Delta}(k-1)\| = \frac{n}{m}\gamma,\end{aligned} \tag{4.18}$$

El siguiente resultado muestra la relación existente $\gamma = \frac{m}{n}\hat{\gamma}$, así, se va a estudiar un valor que acota a $\hat{\gamma}$.

Para ello, se aplica la desigualdad triangular, que es propiedad de las normas, para obtener

$$\hat{\gamma} = \|\hat{\mathbf{\Delta}}(\mathbf{k}) - \hat{\mathbf{\Delta}}(\mathbf{k-1})\|_{ess} \leq \|\hat{\mathbf{\Delta}}(k)\|_{ess} + \|\hat{\mathbf{\Delta}}(k-1)\|_{ess}. \tag{4.19}$$

Entonces, si llamamos $\hat{r} = \|\hat{\mathbf{\Delta}}\|_{ess}$ a la distancia máxima de todos los agentes del centroide de la formación en cada instante de k, la expresión anterior es

$$\hat{\gamma} = \max_{k\geq 0}\{\|\hat{\mathbf{\Delta}}(k) - \hat{\mathbf{\Delta}}(k-1)|\} \leq \max_{k\geq 0}\{\hat{r}(k)\} + \max_{k\geq 0}\{\hat{r}(k-1)\}. \tag{4.20}$$

Finalmente, se define $\hat{R}_{max} = \max_{k\geq 0}\{\hat{r}\}$ como el radio máximo a lo largo de toda la trayectoria de la región circular formada por las posiciones relativas de los agentes con respecto al centroide del objeto. Entonces

$$\hat{\gamma} = \max_{k\geq 0}\{\|\hat{\mathbf{\Delta}}(k) - \hat{\mathbf{\Delta}}(k-1)|\} \leq 2\hat{R}_{max} = \hat{\gamma}^{Teo}. \tag{4.21}$$

entonces ahora (4.16) se obtiene simplemente teniendo en cuenta que $\beta = 1$ por simplicidad.

El resultado anterior muestra que si las diferencias en las señales de entrada de la red satisfacen $\frac{m}{n}\hat{\gamma}$, el algoritmo (3.6) resuelve el problema 1. Actualmente, se está investigando la relación existente entre $2\hat{R}_{max}$ y $2R_{max}$.

□

Notar que λ_2 se puede obtener como se indica en la demostración del lema 4.2.1.

4.3. Ejemplo de estrategia multi-robot basada en un criterio geométrico

PARA que los casos considerados sean lo más amplios posibles, se ha decidido implementar un método centralizado basado en la suposición de que se conoce la trayectoria del objeto deformable, como se explica a continuación.

En una segunda aproximación, distribuimos un grupo de cámaras estéreo según una formación circular que encierra el objeto [23]. Esto significa que las cámaras

mantienen sus posiciones relativas en formación con respecto al objeto objetivo, al mismo tiempo que cumplen el objetivo dentro del campo de visión de todos. Se toma la decisión de distribuir los robots uniformemente a lo largo de la circunferencia formando un polígono regular. La forma del polígono depende del número de robots que desee tener.

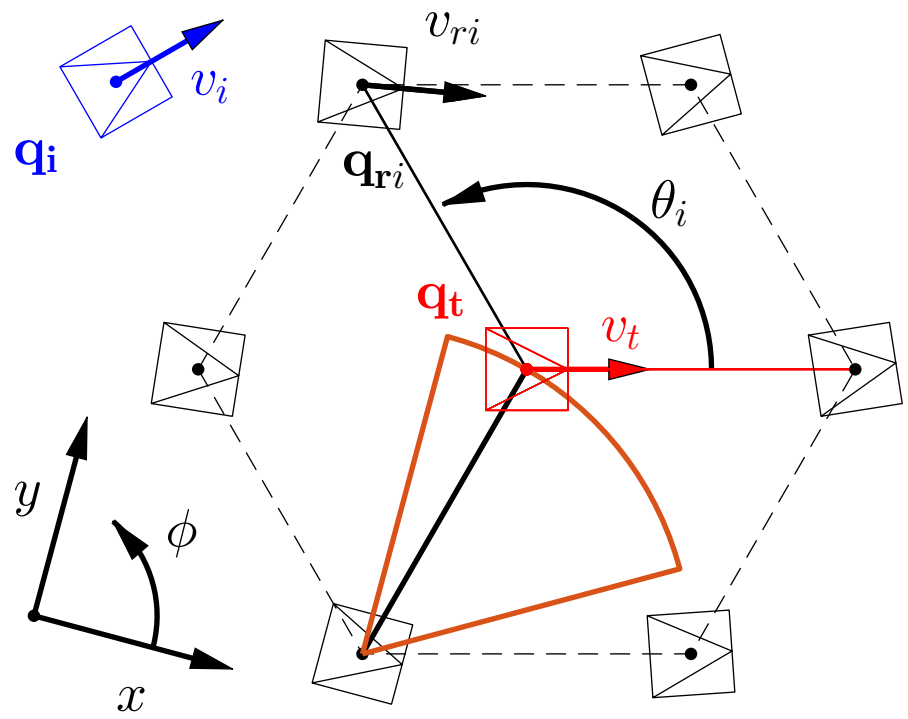


Figura 4.2: *Estrategia de formación de escala constante: Se muestran los parámetros implicados en la estrategia multi-robot. Se definen las trayectorias de referencia* $\mathbf{q_{ri}}$ *para los robots* $\mathbf{q_i}$ *de acuerdo al mantenimiento de una formación circular que incluya el objetivo* $\mathbf{q_t}$ *y cumplimiento de las restricciones del FOV. Cada robot* $\mathbf{q_i}$ *seguirá su trayectoria de referencia* $\mathbf{q_i}$*. También se muestra el FOV de una cámara fija sobre un robot.*

Dado un objetivo en movimiento que sigue una trayectoria previamente definida en $\mathbb{R}^2$, se considera su posición $q_t(t) = (x_t(t), y_t(t))^T$ y su orientación $\phi_t(t) \in \mathbb{R}$. Estas coordenadas se encuentran en un marco de referencia global, y también suponemos que el objetivo se mueve de acuerdo con la cinemática del monociclo:

$$\dot{x}_t = v_t \cos\phi_t, \quad \dot{y}_t = v_t \, sin\phi_t, \quad \dot{\phi}_t = w_t, \tag{4.22}$$

donde $v_t(t) \in \mathbb{R}$ y $\omega_t(t) \in \mathbb{R}$ son la velocidad lineal y angular del objetivo.

En consecuencia, para seguir y encerrar el objeto, también se considera los robots en $\mathbb{R}^2$. Su posición y orientación son $q_i(t) = (x_i(t), y_i(t))^T$ y $\phi_i.(t) \in \mathbb{R}$, con $i = 1, ..., N$. Estos robots siguen la cinemática del monociclo:

$$\dot{x}_i = v_i \, cos\phi_i, \;\; \dot{y}_i = v_i \, sin\phi_i, \;\; \dot{\phi}_i = w_i, \tag{4.23}$$

donde $v_i(t) \in \mathbb{R}$ y $\omega_i(t) \in \mathbb{R}$ son las velocidades lineal y angular de los robots.

La escala de la formación se define por su radio $d_i = d(t)$. También se considera que todos los robots están apuntando al objeto en todo momento y que el objeto está en el centro del círculo. Por lo tanto, el valor d es constante para todos los robots de la formación, equivalente a la distancia de cada robot del objetivo. Para ilustrar esto, se exponen las coordenadas de cada robot con respecto a la referencia del objetivo en movimiento:

$$x_{0i}(t) = d \, cos\phi_i, \;\; and \;\; y_{0i} = d \, sin\phi_i. \tag{4.24}$$

La figura 4.2 muestra un ejemplo de una estrategia multi-cámara bien definida en el caso de seis cámaras ($N = 6$).

Resumiendo, en este capítulo se han propuesto diversos casos de estrategias distribuidas para el *tracking* de un objeto deformable en movimiento. Además, se ha llevado a cabo su análisis de convergencia y se han descrito los requisitos que deben cumplir las señales de entrada para que error en estado estacionario del seguimiento sea cero. También se ha observado la influencia de ambos parámetros, γ y λ_2, en una medida de la distancia de la estimación de cada agente con respecto al promedio del conjunto de señales de entrada, como se comenta en (3.9) y se ha propuesto una cota superior para γ y $\hat{\gamma}$. Por último, se ha implementado una estrategia centralizada para resolver el problema del seguimiento del objeto deformable basada en un criterio geométrico.

Capítulo 5

Arquitectura e implementación del sistema

Este trabajo se divide en dos módulos bien definidos, dependiendo de la tarea que realicen. El objetivo de estos módulos es percibir en cada instante de tiempo, las variaciones en la superficie del objeto, ya sea porque se ha deformado o porque se está moviendo. Para abordar este problema, se ha dividido el problema en dos procesos separados y se ha empleado una arquitectura de sistema adicional para la tarea de reconstrucción. Nuestros sistemas se basan en el marco presentado en [19] y [23]; tal y como se ha introducido en el capítulo 1. Ambos sistemas pueden ser sustituidos por cualquier otro sin afectarse mutuamente, sólo adaptando la arquitectura del sistema. Conceptualmente, se trata de una arquitectura de sistema modular compuesta por diferentes subsistemas independientes que interactúan a través de interfaces de comunicación, como se muestra en la Fig. 5.2.

5.1. Percepción instantánea y reconstrucción 3D

Se ha abordado la tarea de reconstrucción como un proceso iterativo en cada instante; además, la arquitectura del módulo que implementa esta tarea es un sistema genérico basado en ROS. Se ha separado la tarea de reconstrucción multi-robot en tres partes (i) *reconstrucción de modelos volumétricos instantáneos*, (ii) *planificador de vistas multi-cámara* (Estrategia) y (iii) *mecanismo de posicionamiento multi-cámara*. En esta sección se describe una visión general de nuestro sistema, que se basa en [19].

- El **Sistema de Percepción** es responsable de la adquisición de datos y de su procesamiento. Los componentes que forman este sistema son el *módulo del sensor* y el *módulo de la percepción 3D*. En el escenario multi-cámara la salida son tantas nubes de puntos de puntos 3D observados como el número de cámaras que tenemos.

- El **Módulo de Representación del Mundo** registra todos los datos percibidos dentro del mapa, dando acceso al mapa actualizado. Las imágenes tomadas de iteraciones anteriores necesitan ser registradas y fusionadas para construir un modelo común. Este módulo también realiza la tarea de reconstrucción con *OctoMap* [18].

- El **Planificador del Control y el Movimiento** implementa el mecanismo de posicionamiento de la cámara que localiza el robot, y la *Interfaz del Robot* que proporciona la interfaz entre el *Módulo Interfaz de ROS* y el robot.

- El **Módulo Interfaz de ROS** lleva a cabo la planificación de trayectorias y envía la nueva vista al robot, que lo recibe mediante la *Interfaz del Robot*. Esta nueva vista es donde el robot debe mover el sensor correspondiente.

5.2. Seguimiento y encapsulamiento del objetivo

PARA proporcionar al **Módulo Interfaz de ROS** la nueva ubicación de cada cámara y percibir en cada momento los cambios experimentados por el objeto en el entorno, se requiere de un programa que siga una estrategia multi-robot que calcule una formación geométrica de los robots alrededor del objeto y sus respectivas trayectorias. Esta sección se ha implementado en un programa que presenta una arquitectura basada en Matlab. Como se menciona en el en el capítulo 1, el objetivo es encerrar y rastrear el objeto para construir el mapa volumétrico.

- El **Módulo de Planificación Multi-robot** calcula las coordenadas relativas (vistas) de los robots con respecto al objetivo. Como resultado, se genera una formación circular alrededor del objeto con las N cámaras. Por lo tanto, el resultado son N vistas en cada instante, cuya información se almacena en una estructura de datos. Nótese que una vez se han calculado todas las vistas a lo largo de una trayectoria, esta estructura tiene tantas filas como vistas (robots) y tantas columnas como instantes de tiempo. Se recorre esta estructura de datos generados y se envía la posición de cada robot al *Módulo Interfaz ROS*.

5.3. Comunicación entre módulos

EN cuanto a la comunicación entre ambos módulos, se desarrolla una arquitectura de sistema basada en *ROS*. Así, el *Módulo Interfaz de ROS* es capaz de recibir del *Módulo Planificador Multi-robot*, el id del modelo de cámara que ha de moverse de acuerdo a la estrategia definida, y su posición. Se emplea una comunicación basada en

"topics", con sus respectivos publicadores y suscriptores.

También consideramos que la percepción del objeto es instantánea, aunque el sistema toma N instantes (tantos como N robots hay) para formar un mapa *3D* completo. Notar que el sistema sólo mueve un robot por instante en un bucle, así que para terminar la formación geométrica alrededor del objeto con N robots, se necesitan N instantes para construir la percepción completa.

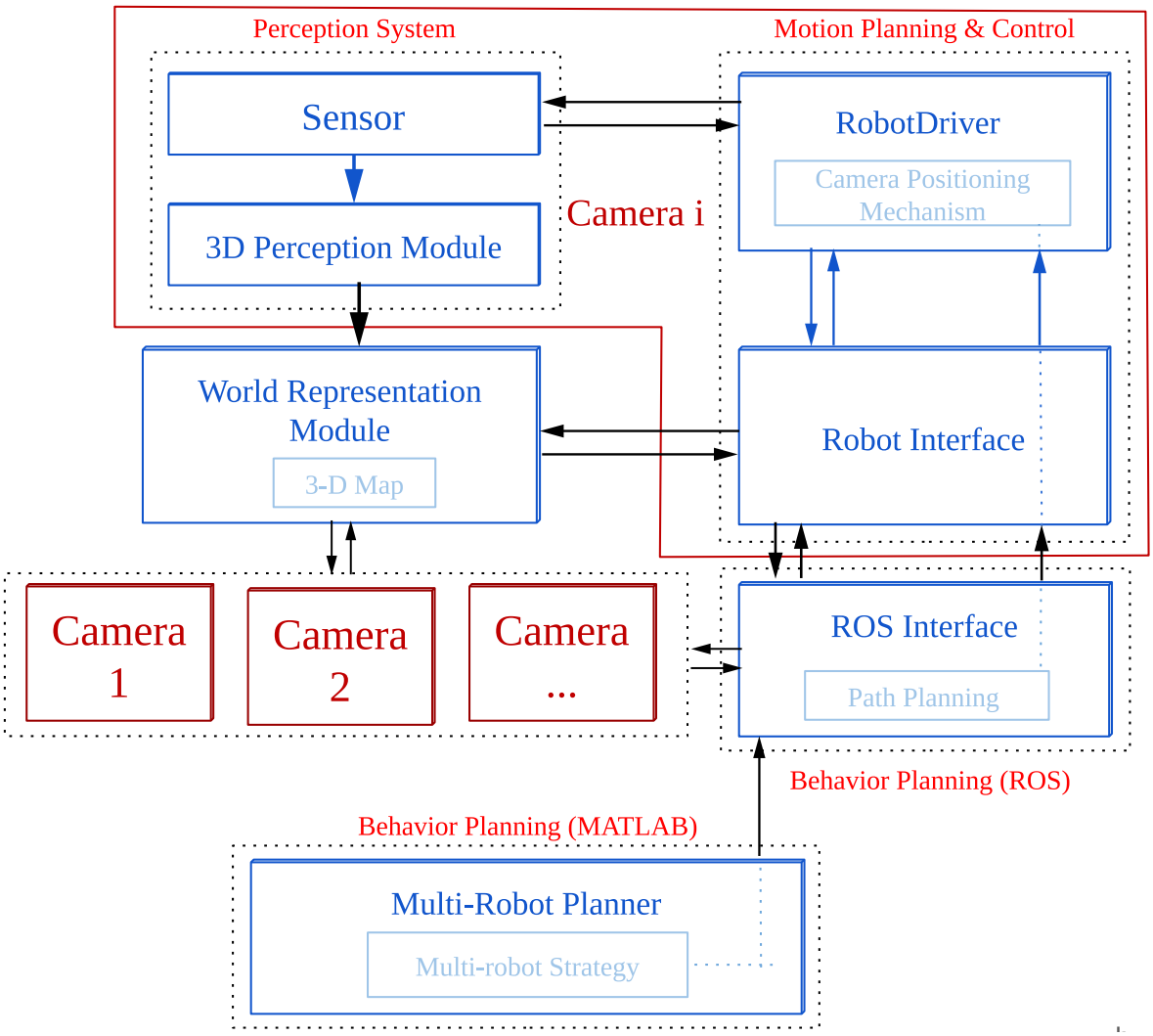


Figura 5.1: *Mapa conceptual del sistema: Se muestran los módulos principales y sus interfaces de comunicación (flechas). En la parte inferior de la imagen se encuentra el sistema basado en Matlab, que envía las ubicaciones calculadas al sistema basado en ROS. El Módulo Interfaz de ROS recibe las posiciones y establece comunicación con cada cámara. El Módulo Interfaz de Robot recibe su ubicación respectiva y el Mecanismo de Posicionamiento de la Cámara es responsable de colocarla en esa ubicación. En el centro de la imagen se encuentra el Módulo de Representación del Mundo, que genera el modelo 3D utilizando Octomap. Este módulo construye el mapa volumétrico a partir de los datos de los sensores de cada cámara.*

Algorithm 1 Reconstrucción Instantánea con Planificación de la Posición

Input: Número de robots (N) .
Output: Mapa volumétrico del objeto.

1: El *Planificador Multi-robot* calcula la siguiente posición de los robots.
2: El *Interfaz de ROS* establece comunicación con los robots.
3: El *Planificador Multi-robot* publica en un *"topic"* la posición i para el robot i.
4: El *Interfaz de ROS* se subscribe al *"topic"* y recibe la posición i del robot i.
5: **while** $t < t_{final}$ **do**
6: **for** $i = 1$ **to** N **do**
7: El *Interfaz de ROS* ordena al *Driver del Robot* que mueva el robot i a la posición i en *Gazebo*.
8: *Gazebo* obtiene las imágenes del sensor del robot.
9: El *Módulo de Representación del Mundo* procesa las imágenes de *Gazebo* y actualiza el *Octomap*.
10: El *Planificador Multi-robot* publica en un *"topic"* la posición i para el robot i.
11: El *Interfaz de ROS* se subscribe al *"topic"* y recibe la posición i del robot i.
12: **end for**
13: **end while**

5.4. Implementación del sistema en un entorno de simulación

Como se ha comentado anteriormente en el capítulo 1, uno de los objetivos de este trabajo es la puesta en marcha de un equipo multi-robot, compuesto por varios pares de cámaras estéreo, en un entorno de simulación (*Gazebo*) que presente unas condiciones lo más realistas posibles. De manera, que los problemas que surjan en ésta resulten similares a los de una implementación real.

Éste capítulo está dedicado a la hacer una breve introducción del software empleado, desde la puesta en marcha de un sólo robot hasta la extensión a un escenario multi-robot. Además, mencionar que el código es de libre acceso para todo aquél que lo solicite, desde los nodos implementados en ROS para la arquitectura del sistema, como los programas llevado a cabo en MATLAB para las simulaciones del algoritmo de control.

5.4.1. Presentación del software

Cada uno de los programas que se emplean están enfocados para cumplir una tarea determinada. Por un lado está el simulador del mundo real *Gazebo*. Se trata de una plataforma que actúa a su vez de motor físico para representar el comportamiento de los robots. Por otro lado, el sistema operativo que controla los robots y sobre el que está implementada la arquitectura del sistema es *ROS* (Robot Operating System).

Este meta-sistema incorpora las librerías y paquetes necesarios para la herramienta de reconstrucción *Octomap* y para el modelo de robot seleccionado. Para finalizar, el programa que soporta las estrategias multi-robot y se comunica con la plataforma de *ROS* es MATLAB (MATrix LABoratory). Las versiones empleadas en este trabajo son Gazebo 7 y ROS Kinetic en Ubuntu 16.04.

La elección de estos programas se debe al amplio rango de aplicaciones desarrollados a través de ellos, lo que permite disponer de librerías de software ya desarrollado. Además, existen numerosos métodos de interacción entre MATLAB y ROS y MATLAB con Gazebo. La descripción de los programas se detalla a continuación.

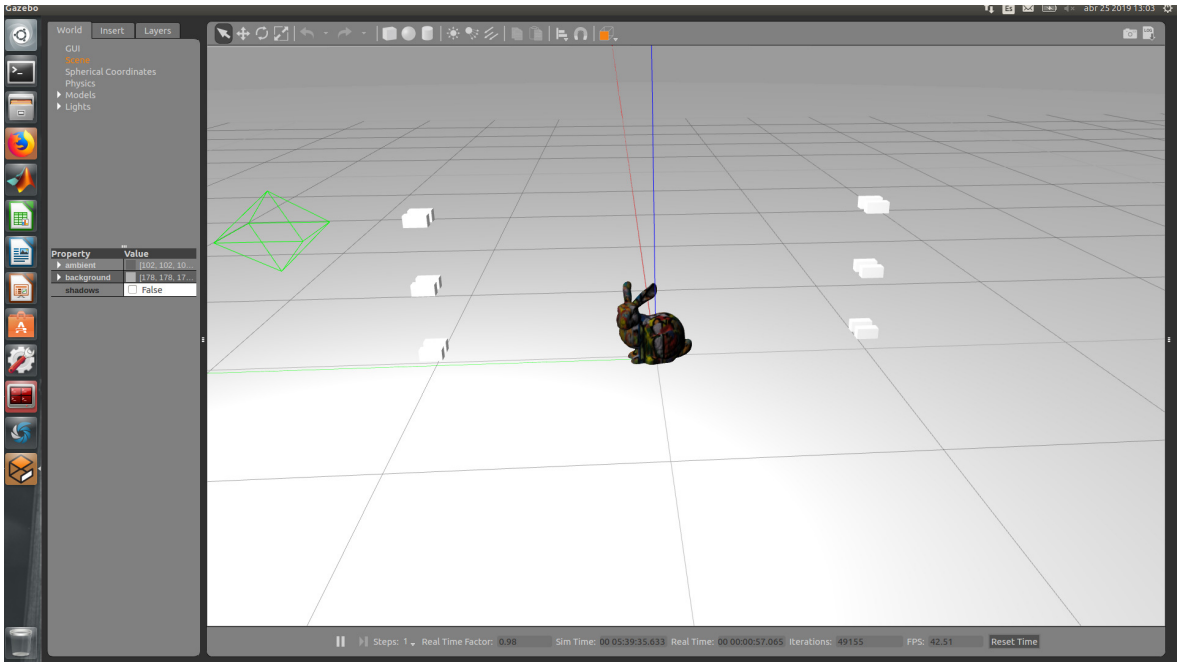


Figura 5.2: *Escenario de Gazebo de una de las simulaciones implementadas en este trabajo.*

Gazebo Nace con el nombre de Gazebo Project, es un simulador 3D, cinemático, dinámico y multi-robot que permite realizar simulaciones de robots articulados en entornos complejos, interiores o exteriores, realistas y tridimensionales. En cuanto a su integración en ROS, se ejecuta como un nodo más de ROS, poseyendo sus propios topics y servicios. El simulador también posee su propio stack denominado simulator gazebo, que se divide en una serie de paquetes con diferentes funcionalidades como gazebo msgs, gazebo plugins o gazebo worlds entre otros.

ROS Es la herramienta que da soporte al desarrollo software de los robots. Lo conforman un amplio rango de paquetes de libre uso y el sistema operativo en sí.

Gracias a esto, es un foco de desarrollo no sólo para investigadores sino para cualquier usuario que quiera disponer de él.

Los grafos son la base que da soporte a su arquitectura, donde los nodos son los programas encargados de la ejecución de los procesos. La comunicación entre ellos se lleva a cabo de tres formas distinta. Una de ellas, se basa en la interacción publicador-subscriptor, donde la información se transmite cuando uno de ellos publica un cierto tipo de mensaje y el otro se subscribe a este. Otra es mediante los servicios, donde uno de los nodos solicita al otro nodo que realice una cierta tarea. Por último, está la comunicación a través de acciones, de la cuál no se ha hecho uso en esta memoria.

Matlab Es un soporte software matemático con lenguaje de programación propio y capaz de implementar todo tipo de datos, algoritmos, redes neuronales, etc. Además, es capaz de comunicarse con otros soportes informáticos, como se ha hecho en este trabajo con ROS.

Todas las simulaciones del Capítulo 6 se han llevado a cabo en este programa, como se explica más adelante. Más aún, es la herramienta en la cuál se implementa el algoritmo de consenso.

5.4.2. Implementación de un par de cámaras estéreo

En esta sección se introduce brevemente como se ha llevado a cabo la implementación en *Gazebo* del par de cámaras estéreo.

Los nodos se han escrito en *C++* y se describen en el lenguaje *XML*. Se lanzan a *Gazebo* mediante *ROS* a través de archivos *.launch*. Además, se crea un fichero con la descripción de la estructura y representación gráfica del par de cámaras estéreo con extensión *.urdf*. En Fig. (6.24) se muestra mediante la herramienta *Rqt* el conjunto de nodos y topics de todo el sistema.

5.4.3. Extensión a un escenario multi-robot

Para lanzar en *Gazebo* múltiples robots, en este caso pares de cámaras estéreo, se han modificado los ficheros con extensión *.launch* tal y como se muestra en el Anexo B. Concretamente, en este trabajo, se ha tenido que rediseñar la arquitectura, implementado un nodo interfaz de comunicaciones por cada par de

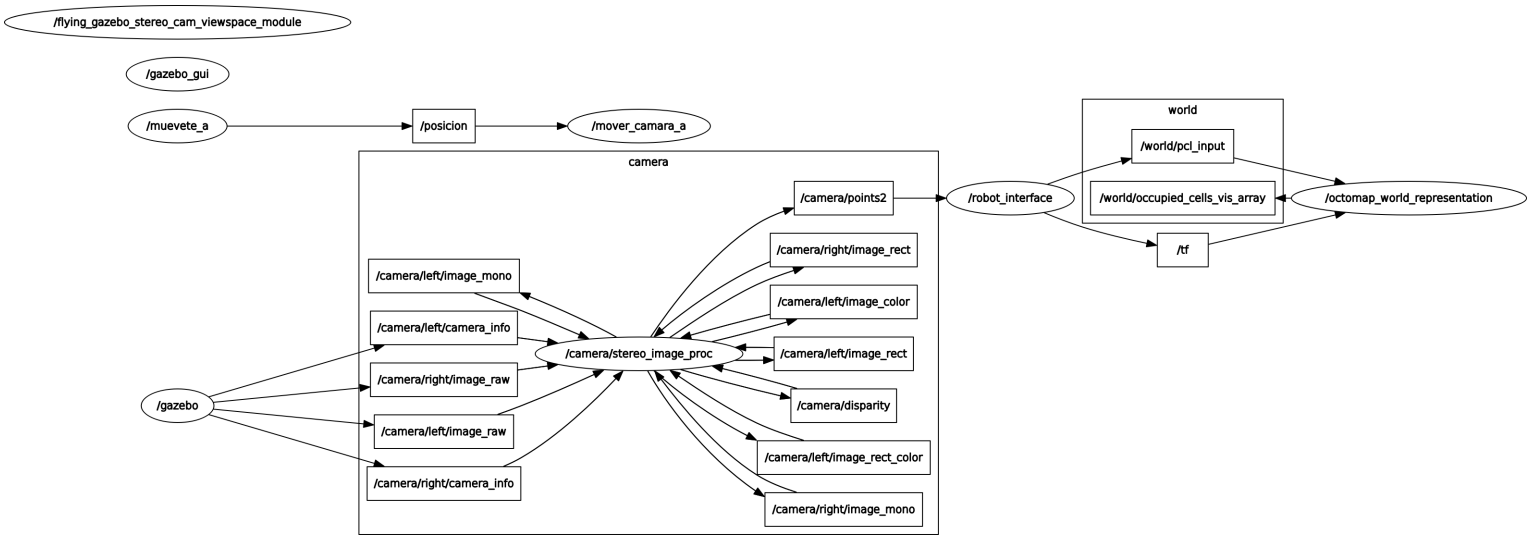


Figura 5.3: *Conjunto de nodos y topics de la arquitectura diseñada para una cámara. Nodos:* ***gazebo*** *es el nodo del simulador;* ***muevete a*** *representa el nodo que publica las posiciones de la cámara, se lanza desde Matlab;* ***mover posicion a****, es el nodo encargado de suscribirse al topic* ***posicion****, además, controla la localización de la cámara en la posición indicada las cámaras en Gazebo. A modo de ejemplo, en C se muestra el código de este nodo. Dentro del grupo camera (rectángulo) se encuentra el nodo* ***stereo image proc*** *que transforma las imágenes de la cámara en una nube de puntos. Esta nube de puntos se publica en un mensaje al topic* ***points2****. Es aquí, cuando interviene un nodo interfaz, (****robot interface****), entre el modelo de la cámara y ROS. Este nodo, ofrece sus servicios para la transmisión del mensaje que contiene la nube de puntos. Por último, es el nodo* ***octomap world representation*** *quién recibe el mensaje del nodo interfaz de la cámara, para construir el modelo del objeto que se quiere representar.*

cámaras introducido. El ejemplo que se muestra en la Fig.(5.4) ilustra el conjunto de nodos y topics requeridos para dos pares de cámaras.

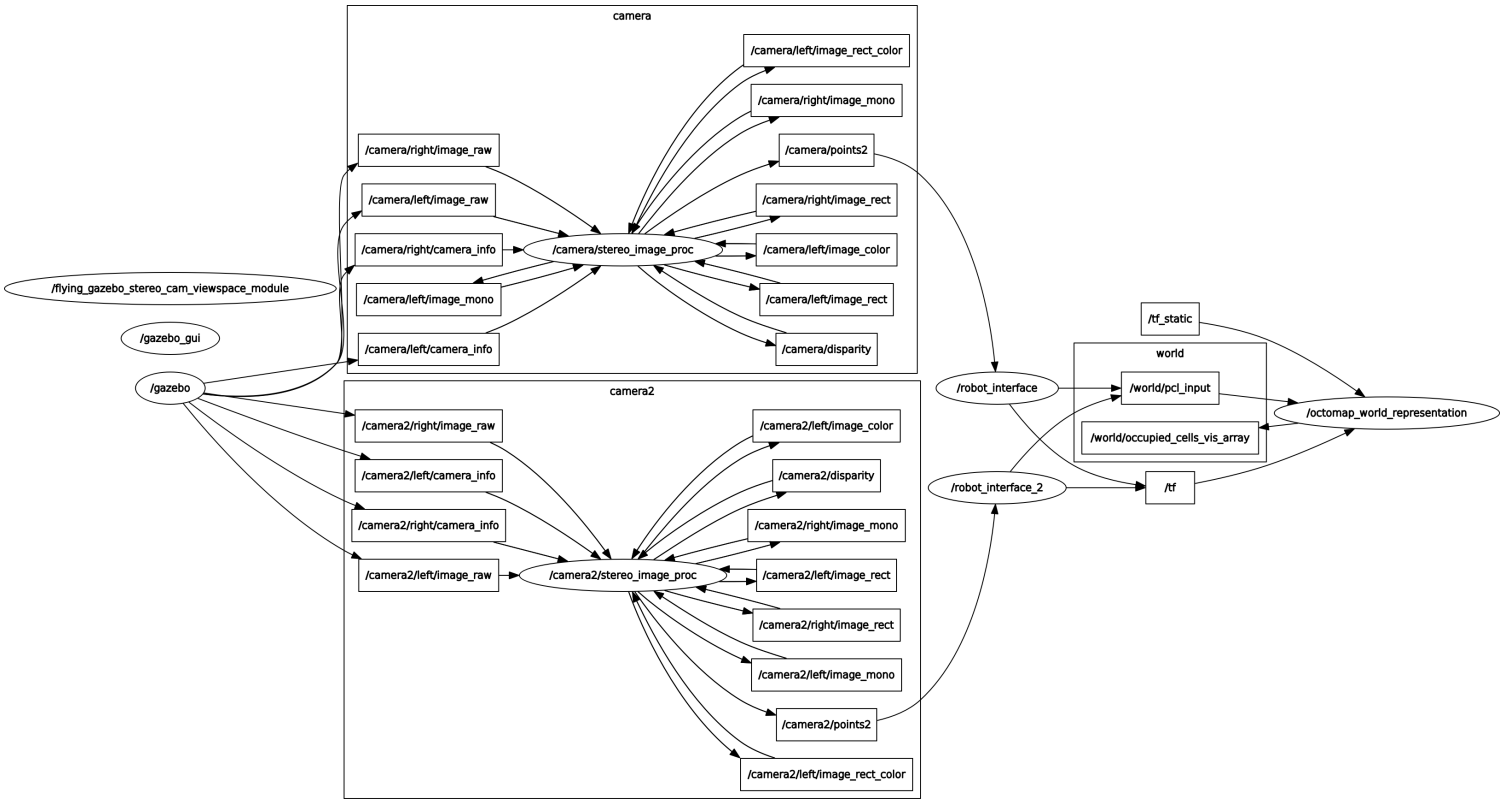


Figura 5.4: *Arquitectura de nodos y topics en ROS para dos robots Notase los dos espacios de trabajo para cada una de las cámaras.*

Capítulo 6

Simulaciones y evaluación

En este capítulo se presentan las simulaciones del trabajo. Por un lado, se describen el algoritmo de consenso evaluado extensivamente y en profundidad en *MATLAB* con objetos ideales (m puntos); por otro lado, se desarrolla un sistema y una arquitectura para evaluar en un entorno realista (*Gazebo*), las tareas de percepción y construcción del objeto en movimiento. presentan las simulaciones para las . Entre los objetivos del capítulo, se encuentran la explicación de las simulaciones realizadas y el análisis de los resultados obtenidos. Así, el cometido de algunas simulaciones es la ilustración del comportamiento de la arquitectura y el algoritmo implementados, mientras que otras están destinadas al análisis de los cambios de comportamiento del sistema al modificar ciertos parámetros de este.

6.1. Simulaciones de la estrategia de consenso

En esta sección, se van a estudiar los tres situaciones de consenso previamente planteadas en el Capítulo 4, vía simulación. La estructura de la sección es la siguiente, se presenta tres subsecciones, cada una equivalente a un caso; dentro de cada caso, se muestran seis simulaciones, cada una correspondiente a una trayectoria del grupo de objetivos diferente. En el primer caso, además, se realiza un análisis del comportamiento del algoritmo al cambiar diversos parámetros.

El marco de trabajo de las simulaciones es el siguiente: Se considera un proceso descrito por una trayectoria que varía en el tiempo. Un grupo de N agentes con la topología de comunicación mostrada en la Fig. (3.1) monitorea este proceso mediante la toma de muestras síncronas, cada una de acuerdo a la trayectoria del caso que se esté estudiando. Después de cada muestreo, cada agente desea obtener el promedio de las mediciones a través de la red antes del siguiente tiempo de muestreo. Entre los tiempos de muestreo m y $m+1$, la entrada $u^i(k)$ se fija en $u^i(m)$. Puesto que el interés

de este trabajo es que el algoritmo (3.6) converja en una iteración, se ha fijado $m = 1$. Los casos estudiados son:

1. Red con percepción fija en el tiempo y $n = m$.
2. Red con percepción variable en el tiempo y $n = m$.
3. Red con percepción fija pero $n < m$.

Caso	n	m	k	δ_2^{accu}	nº comunicaciones	α	β	λ_2
1	6	6	300	0.25	1 iteración	1	1	4
2	6	6	300	0.25	1 iteración	1	1	4
3	4	10	30	0.25	1 iteración	1	1	4

Tabla 6.1: Parámetros comunes a todas las simulaciones. Aquí, n es el número de agentes y m el número de objetivos que delimitan las esquinas del objeto deformable. El instante de tiempo discreto se representa con k. δ_2^{accu} es el stepsize caracterizado para el grafo de este trabajo. Los parámetros α y β modifican el rendimiento del algoritmo como se vera más adelante. Por último, λ_2 corresponde al segundo valor propio. Se ha añadido este elemento porque interviene en el cálculo del error del *tracking*.

En la tabla 6.1 se muestran los valores de los parámetros de las simulaciones.

Estos tres casos se proponen con el objetivo de ilustrar el comportamiento del algoritmo ante dos cambios en la estructura del sistema (número de agentes distinto al de objetivos y rotación de los agentes) y de cara al futuro, como pasos previos a una estrategia de *orbitación* general.

En lo que respecta a las simulaciones, se dividen en tres tipos de trayectorias según su funcionalidad:

1. **Trayectorias Tipo 1**: Todos los objetivos coinciden con la posición de su centroide al seguir la misma trayectoria. La trayectoria que se estudia es de la forma:

$$x_T^s(k) = v_{xt}(k),$$
$$y_T^s(k) = v_{yt}(k),$$

donde v_{xt} y v_{yt} son rectas y $s \in \{1, \cdots, m\}$.

2. **Trayectorias Tipo 2**: Se incluye un término de entrada *bias (b)*, cuyos valores son $b^1 = -0{,}5, b^2 = 1, b^3 = 0{,}6, b^4 = -0{,}9, b^5 = -0{,}6, b^6 = 0{,}4, b^{)} = -1, b^8 = 0{,}3, b^9 = 0{,}25, b^{10} = -0{,}85$. Cada objetivo sigue una trayectoria distinta y las trayectorias propuestas son:

a) La primera simulación de este tipo de trayectorias es

$$x_T^s(k) = v_{xt}(k) + b^s,$$
$$y_T^s(k) = v_{yt}(k) + b^s,$$

donde b es *bias* de entrada.

b) La segunda y última trayectoria de este tipo es

$$x_T^s(k) = v_{xt}(k) + b^s + k,$$
$$y_T^s(k) = v_{yt}(k) + b^s + k,$$

donde k es el instante de tiempo.

3. **Trayectorias Tipo 3**: Este tipo de trayectorias son exponenciales y su objetivo es explorar los límites de convergencia del algoritmo llevándolo al extremo e incluso forzándolo a perder el *tracking* del centroide.

a) La primera simulación de este tipo de trayectorias es

$$Y = \exp(k/0{,}25),$$
$$x_T^s(k) = Y(k),$$
$$y_T^s(k) = Y(k),$$

donde k es el instante de tiempo.

b) La segunda trayectoria de este tipo es

$$x_T^s(k) = v_{xt}(k) + b^s k,$$
$$y_T^s(k) = v_{yt}(k) + b^s k,$$

donde k es el instante de tiempo.

c) La tercera y última trayectoria es tal que

$$x_T^s(k) = v_{xt}(k) + \exp(s * k),$$
$$y_T^s(k) = v_{yt}(k) + \exp(s * k),$$

donde k es el instante de tiempo.

A continuación, debido a la similitud de las gráficas para todos los casos y sus tipos de trayectorias, se ha creído conveniente su explicación en este párrafo. En los tres casos se representan seis ventanas, donde la columna de la izquierda corresponde a la coordenada x y la derecha a la coordenada y.

– **Gráficas (a) y (b):** Muestran el resultado de la simulación utilizando el algoritmo de consenso de tiempo discreto (3.6) con $\alpha = \beta = 1$. El ancho de banda de comunicación es de 4 Hz, es decir, $\delta = 0,25$ segundos. Líneas sólidas: las curvas rojas representan la historia temporal del estado de acuerdo de cada agente generado por el consenso promedio dinámico de (3.6)]; $\times$: puntos de muestreo en $m\Delta k$; $\circ$: la media en $m\Delta k$; $+$: la media de las señales de referencia a $k\delta$. Como se ha comentado en los capítulos previos, bajo una inicialización apropiada, los agentes son capaces de rastrear $x_T^{avg}(k)$ con un pequeño error.

– **Gráficas (c) y (d):** La línea verde sólida representa el valor del $bound = f(\gamma)$ mientras que el resto de las líneas corresponden al error entre los estados de acuerdo del algoritmo y la media de entrada. El algoritmo tiene garantizada la converegencia cuando este error es inferior a la línea verde.

– **Gráficas (e) y (f):** Evaluación del rendimiento del algoritmo con respecto a las señales de entrada que varían en el tiempo y que son diferentes entre sí. En primer lugar, la línea discontinua representa el radio máximo de la circunferencia que define la región donde se ubicarán la estimación de los agentes del centroide del objeto. Podemos apreciar el valor conservador de $2R_{max}$. Por otro lado, la línea verde sólida representa el valor del $bound = f(\gamma)$. Finalmente, la línea sólida roja representa el valor de γ, estudiado en capítulos anteriores.

Para finalizar, comentar que uno de los objetivos principales de estas simulaciones es estudiar, entender y verificar como se comporta γ en cada uno de los casos y sus trayectorias.

6.1.1. Caso 1. Percepción fija (M constante) y $n = m$

Trayectoria Tipo 1. Recta igual para todos los agentes sin offset.

Por un lado, el parámetro γ se obtiene a partir de la diferencia de las distancias relativas de las acciones $u^i(k)$ respecto a la media de todas ellas $u^{avg}(k)$. Para este tipo de trayectorias, en los dos primeros casos, el parámetro γ es cero, como se aprecia en las tablas (6.2) y (6.8), esto es debido a que al no existir un offset en la trayectoria, la señal de entrada de cada agente en el instante k coincide, ergo su media tendrá el mismo valor que la señal de entrada y por tanto la diferencia entre estas es cero. Por otro lado, la cota superior $2R_{max}$ que debe cumplir el error cometido depende de γ, por ello, este parámetro también es nulo para este tipo de trayectorias. Esto se traduce en un error en estado estacionario cero.

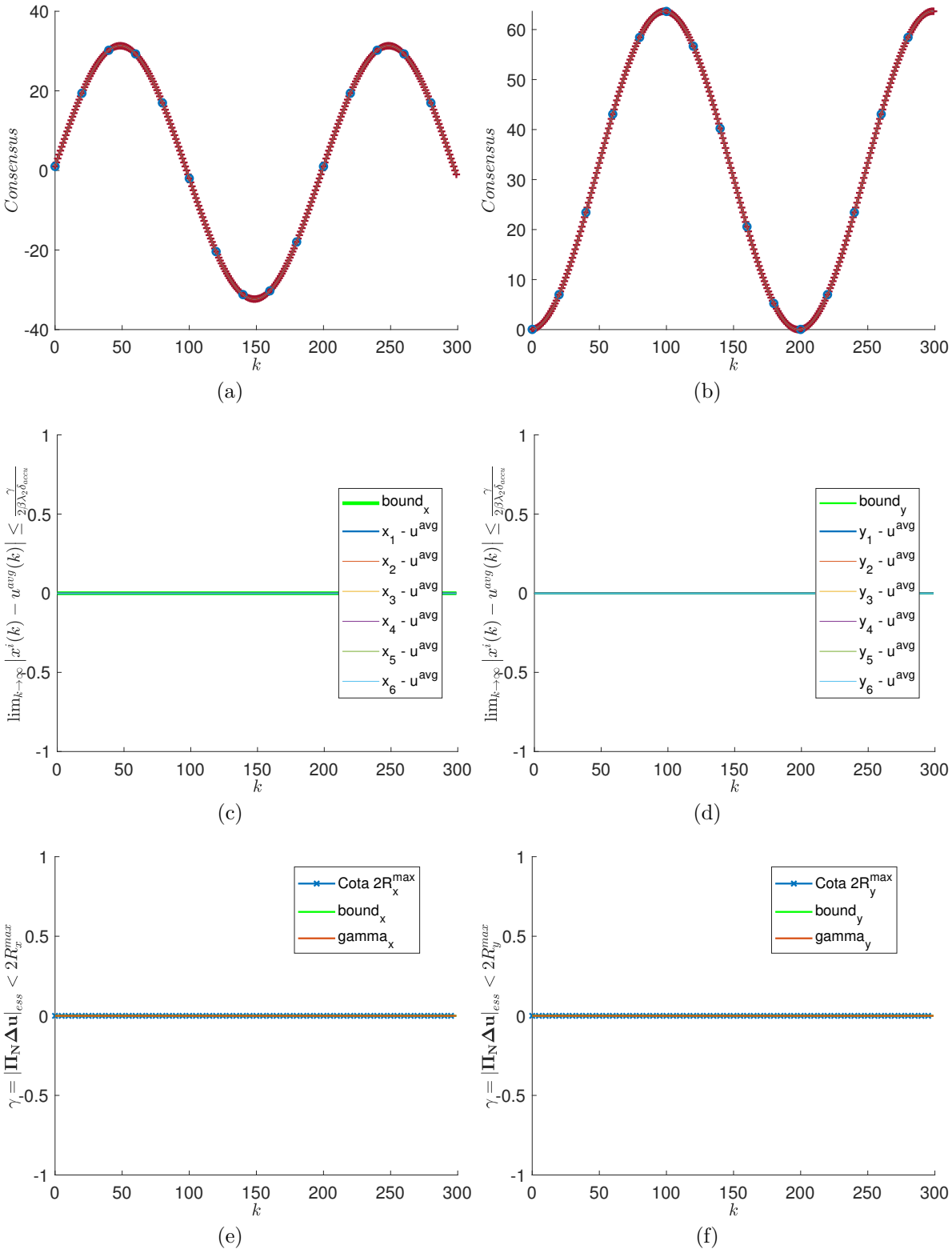


Figura 6.1: *Caso 1, trayectoria tipo 1. Corresponde con la tabla 6.2. El error en estado estacionario es nulo, veáse en (c) y (d) las líneas sólidas de colores en cero.*

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	$2R_x^{max}$	$2R_y^{max}$
0	0	0.99	0.99	0.03	0.03	0	0	0	0
299	299	25.54	25.54	13.52	13.52	0	0	0	0

Tabla 6.2: Caso 1, trayectoria tipo 1. Corresponde con la Fig. (6.1). En esta tabla se muestra el valor de diferentes variables relevantes. Las variables x e y corresponden al promedio de los valores de consenso estimados por el grupo de agentes, se aprecia que coinciden con el valor promedio de las posiciones de los objetivos x_T^{avg} e y_T^{avg}. También se ha creído conveniente la introducción de γ en ambas coordenadas, por su relación directa con el error cometido en la estimación. Finalmente, se añade a la tabla la cota diseñada $2R_{max}$.

Trayectorias Tipo 2. Recta para todos los agentes más un offset distinto para cada agente.

Simulación (a). Offset constante en el tiempo. Para este tipo de trayectorias, en este caso, el parámetro γ no es cero en un ámbito realista; sin embargo, como se aprecia en la tabla (6.3), en las simulaciones es nulo, porque aunque en la trayectoria se añade un término a modo de offset (b^i) constante, de manera que la señal de entrada de cada agente en el instante k no coincida con ninguno de sus vecinos, en las simulaciones se ha impuesto una cota inferior (0.1) a γ. Por tanto, la diferencia entre cada agente y el promedio de las señales de entrada es cero. Por otro lado, al no imponer este límite a la cota superior $2R_{max}$, este parámetro es no nulo para este tipo de trayectorias.

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	$2R_x^{max}$	$2R_y^{max}$
0	0	1.99	1.99	1.03	1.03	0	0	13.71	13.71
299	299	55.54	55.54	43.52	43.52	0	0	13.71	13.71

Tabla 6.3: Caso 1, trayectoria tipo 2a. Corresponde con la Fig. (6.2).

Simulación $\alpha = 0$ Y $\alpha = 50$ Como se aprecia en la Fig.(6.3), el incremento del parámetro α conlleva a un error en estado estacionario que no es cero durante más tiempo, es decir, tarda un mayor número de iteraciones en alcanzar asintóticamente el consenso dinámico.

Simulación $\beta = 0{,}01$ y $\beta = 50$ El incremento del parámetro β conlleva a una reducción del error en estado estacionario, que no es cero. Cuando se aumenta β, el algoritmo (3.6) tarda un menor número de iteraciones en alcanzar asintóticamente el consenso dinámico.

Simulación $\delta = 0{,}25$ y $\delta = 0{,}5$ El incremento del parámetro δ conlleva a la no convergencia del algoritmo, como se muestra en (c) y (d), donde el error en estado

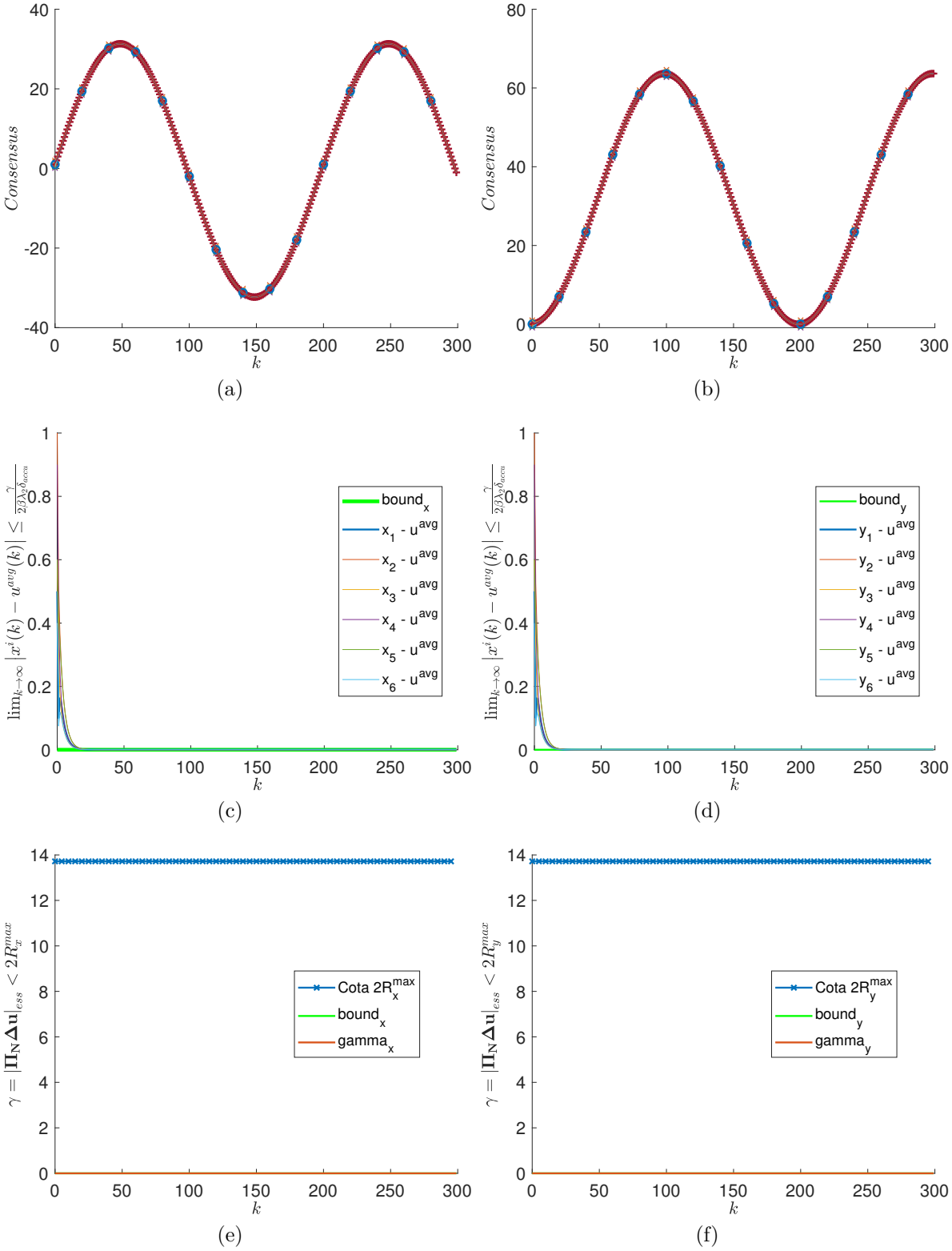


Figura 6.2: *Caso 1, trayectoria tipo 2a. Los datos de esta simulación corresponden a los de la tabla 6.3. Notase que en (a), debería verse una línea sólida por cada agente. El valor del bias es demasiado pequeño para que se puedan apreciar diferencias entre estas líneas.*

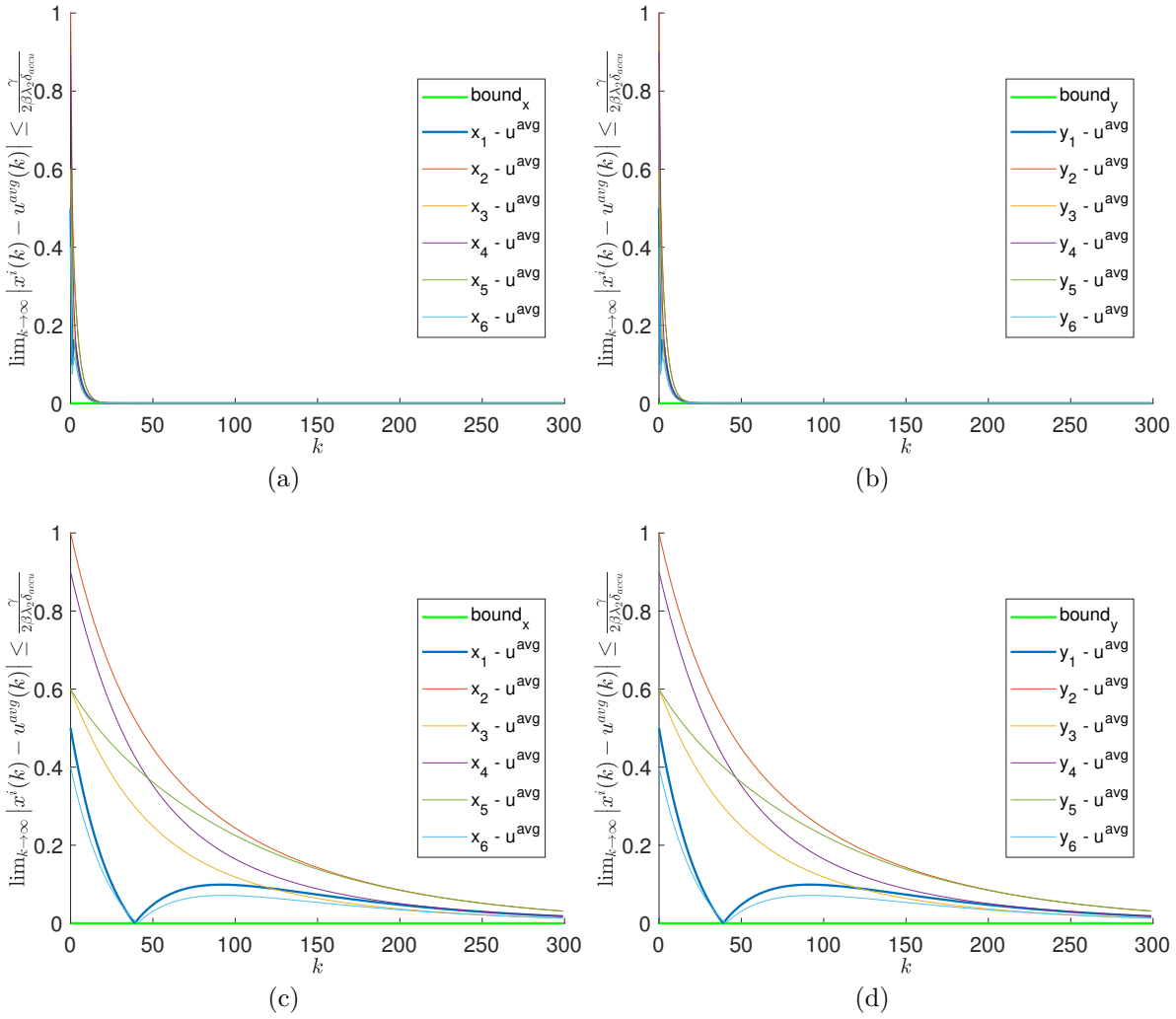


Figura 6.3: *Caso 1, trayectoria tipo 2a. Parámetro de diseño α para la regulación del rendimiento del algoritmo. Simulaciones (a) y (b) corresponden a un $\alpha = 0$ y las simulaciones (c) y (d) a un $\alpha = 50$. En esta simulación, se extrae la siguiente conclusión: El incremento del parámetro α conlleva a un error en estado estacionario que no es cero durante más tiempo.*

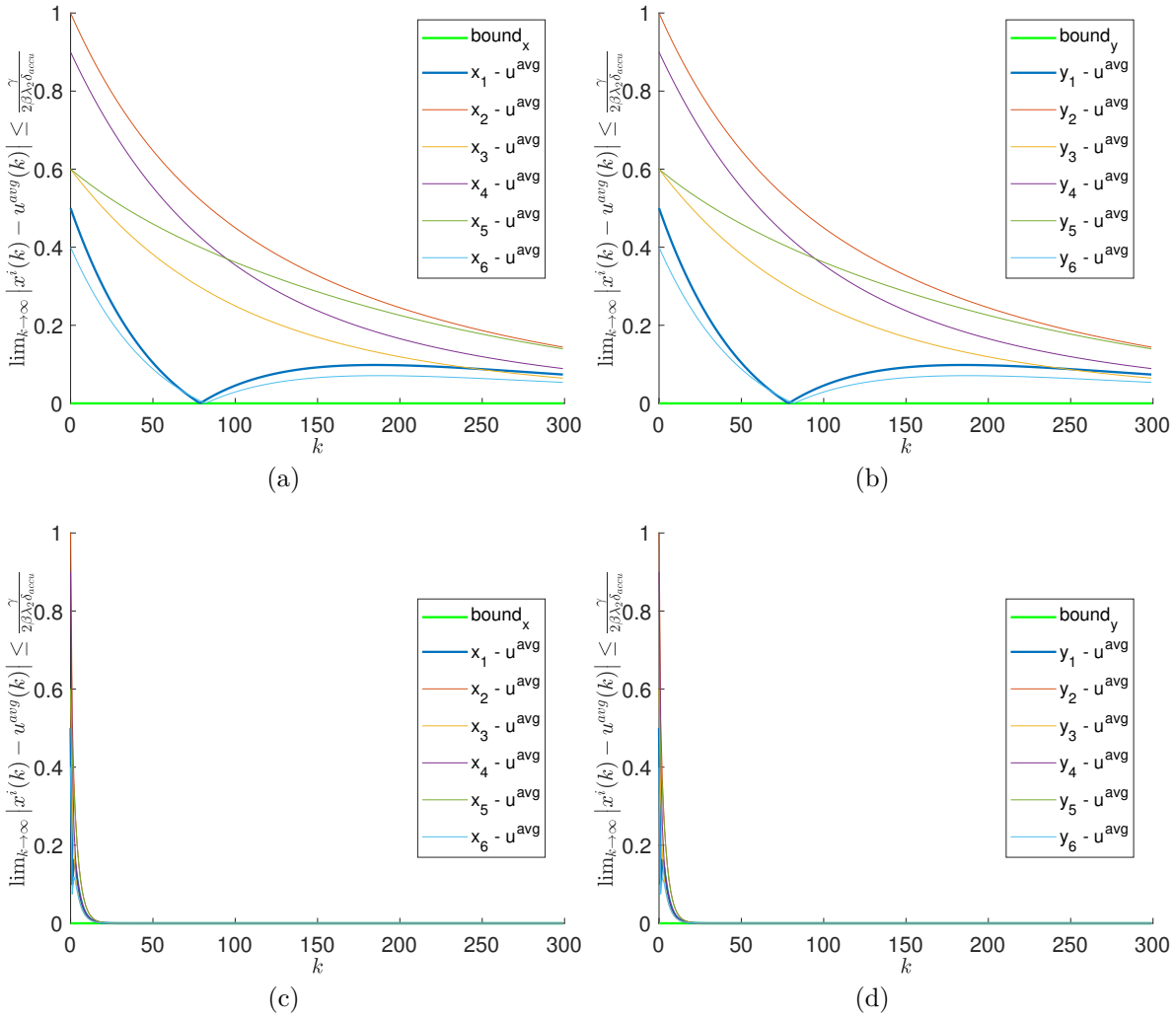


Figura 6.4: *Caso 1, trayectoria tipo 2a. Parámetro de diseño β para la regulación del rendimiento del algoritmo. Simulaciones (a) y (b) corresponden a un $\beta = 0{,}01$ y las simulaciones (c) y (d) a un $\beta = 50$. En esta simulación, se extrae la siguiente conclusión: El incremento del parámetro β conlleva a una reducción del error en estado estacionario, que no es cero.*

estacionario para cada agente queda por encima del límite de convergencia (línea verde sólida.)

Simulación $n = m = 6$ y $n = m = 10$ El incremento del número de agentes en este caso en concreto, no influye ni en la convergencia del algoritmo (el error en estado estacionario está por debajo de la cota de convergencia, línea verde sólida, una vez alcanzado el consenso), ni en la velocidad de convergencia, ya que visualmente en la gráficas se alcanza el consenso en el mismo instante de tiempo.

Simulación (b). Offset variable en el tiempo. Al igual que en la trayectoria anterior, para este tipo de trayectorias, en este caso concreto, el parámetro γ es cero, como se aprecia en la tabla (6.4), esto es debido a que aunque en la trayectoria se añaden dos términos constantes a modo de offset (b^i) y (k), de manera que la señal de entrada de cada agente en el instante k no coincida con ninguno de sus vecinos. Por tanto la diferencia entre estas es cero. Aquí, la cota superior $2R_{max}$ tampoco es nula para este tipo de trayectorias. Esto se traduce en un error en estado estacionario cero en las simulaciones.

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	$2R_x^{max}$	$2R_y^{max}$
0	0	1.99	1.99	1.03	1.03	0	0	13.71	13.71
299	299	55.54	55.54	43.52	43.52	0	0	13.71	13.71

Tabla 6.4: Caso 1, trayectoria tipo 2b. Se correponde con la Fig.(6.7).

Trayectorias Tipo 3. Exponenciales.

Este tipo de trayectorias exponenciales permiten forzar el algoritmo hasta la pérdida de la convergencia. No tienen sentido por tanto, para ninguna de las simulaciones (a), (b) y (c), en los tres casos, hablar del parámetro γ y de la cota superior diseñada R_{max}. En la tabla (6.5) se muestra como estos parámetro toman valores demasiado elevados. Aunque en el caso (b), si que parezca que tienen un sentido físico, tanto el diseño de la trayectoria (se multiplica el offset b^i por un valor variable k) como las gráficas (ver Fig. (6.21), gráficas (c) y (d)), nos muestran que el error de seguimiento incrementa, i.e., el algoritmo diverge.

Simulación (a). Exponencial sin offset. Teóricamente, este tipo de trayectoria, al no tener offset, debería converger con error en estado estacionario nulo, puesto que la diferencia entre la acción de cada agente y el promedio de todas ellas es cero.

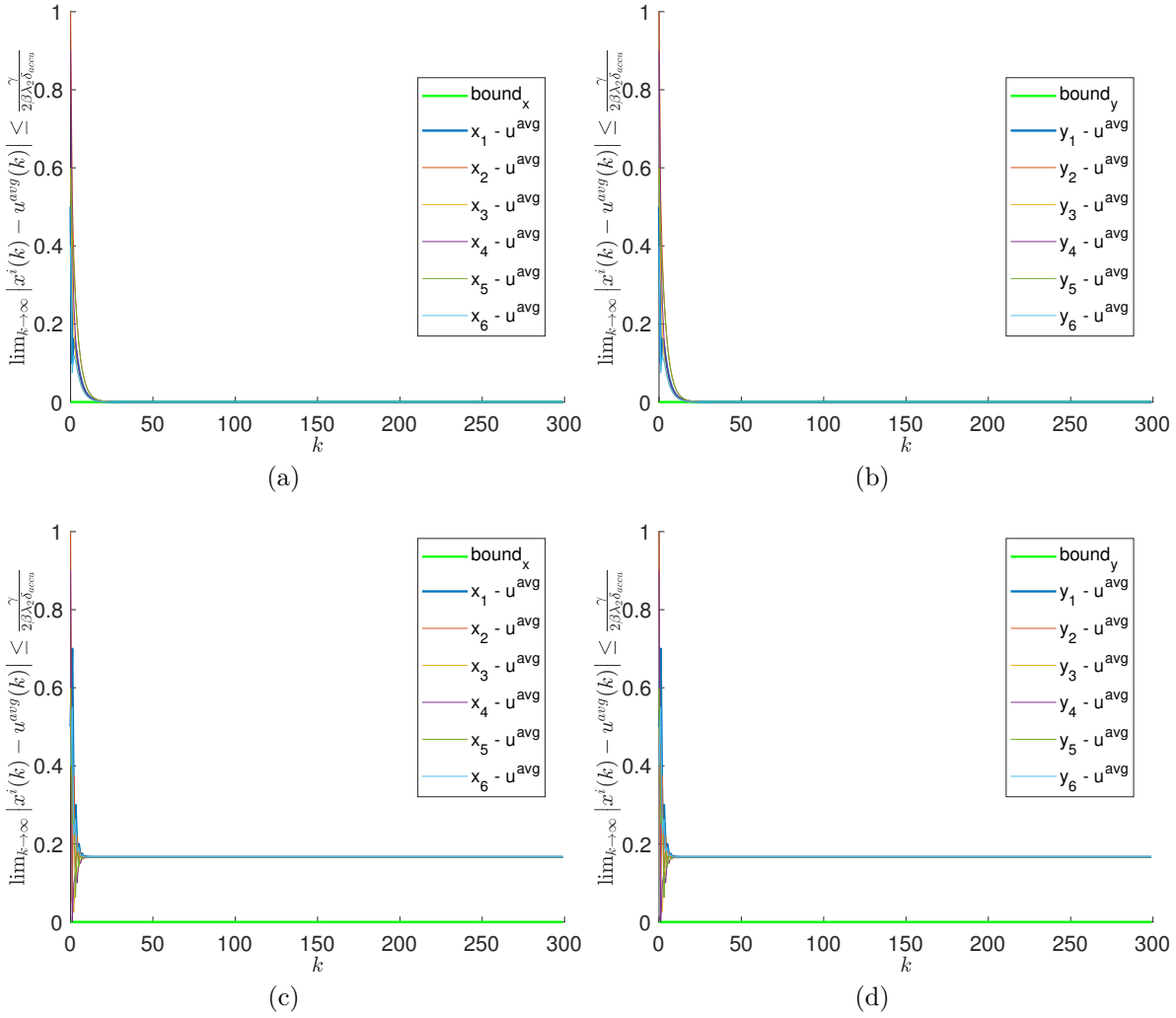


Figura 6.5: *Caso 1, trayectoria tipo 2a. Estudio de la influencia del ancho de banda de comunicación o stepsize δ sobre la convergencia del algoritmo 3.6. Simulaciones (a) y (b) corresponden a un $\delta = 0,25$ segundos, es decir, es de 4 Hz y las simulaciones (c) y (d) a un $\delta = 0{,}5$ segundos, es decir, es de 2 Hz. Se ha elegido una valor de 0.5 por ser el límite del intervalo para el que δ asegura la convergencia del algoritmo. Sorprendentemente, el incremento del parámetro δ hasta el límite del intervalo $\delta \in \{0, \text{mín}\{1, 0{,}5\}\}$ conlleva a la convergencia del algoritmo a un valor distinto de cero (error estacionario de 0.2), pero que no se encuentra dentro del límite de convergencia, como se muestra en (c) y (d), donde el error en estado estacionario para cada agente queda por encima del límite de convergencia (línea verde sólida.)*

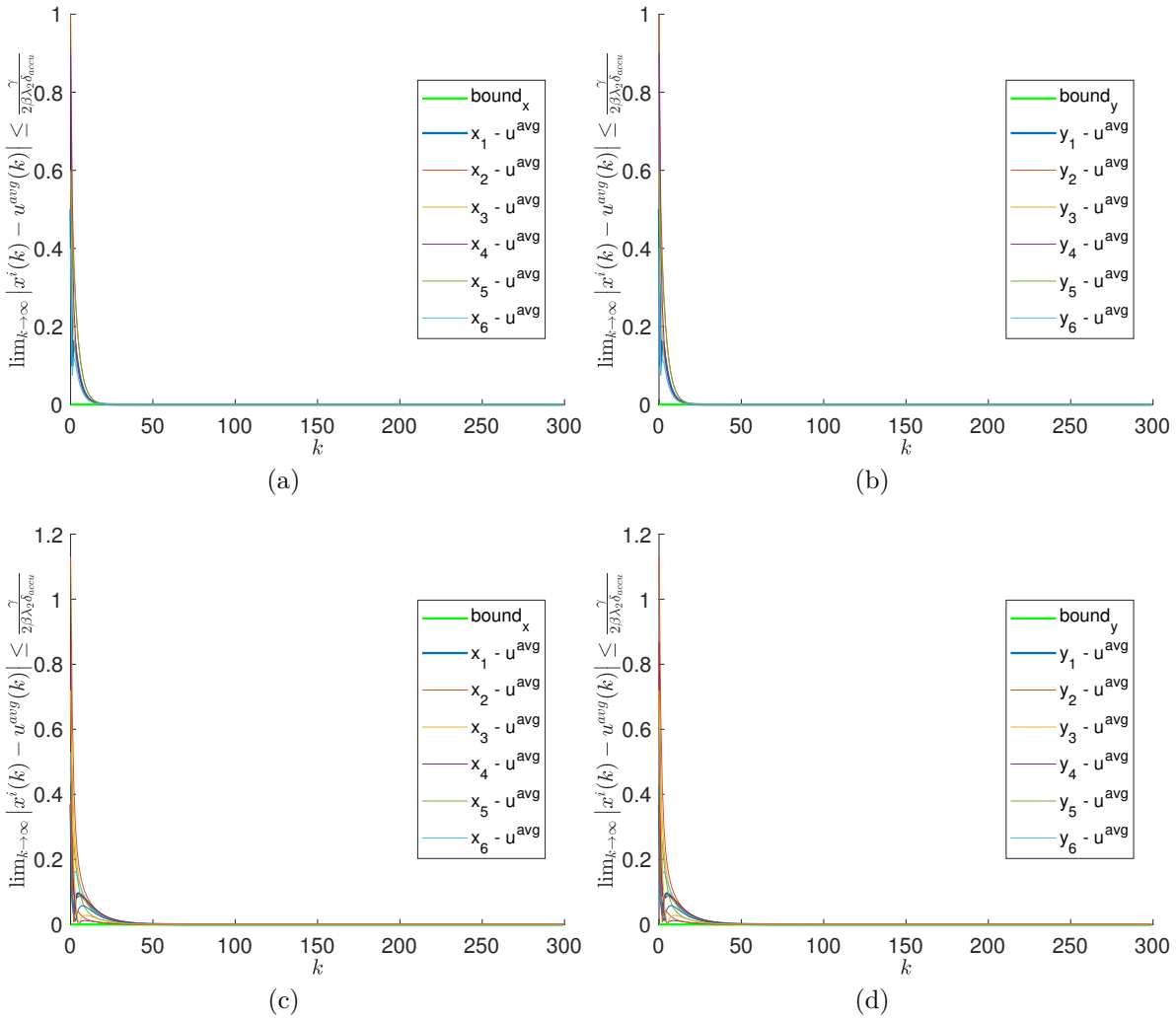


Figura 6.6: *Caso 1, trayectoria tipo 2a. Estudio de la influencia del número de agentes y objetivos (n y m) sobre la convergencia del algoritmo 3.6. Simulaciones (a) y (b) corresponden a un sistema donde* $n = m = 6$ *y las simulaciones (c) y (d) a uno donde* $n = m = 10$*. En esta simulación, se extrae la siguiente conclusión: El incremento del número de agentes en este caso en concreto, no influye ni en la convergencia del algoritmo (el error en estado estacionario está por debajo de la cota de convergencia, línea verde sólida, una vez alcanzado el consenso), ni en la velocidad de convergencia, ya que visualmente en la gráficas se alcanza el consenso en el mismo instante de tiempo.*

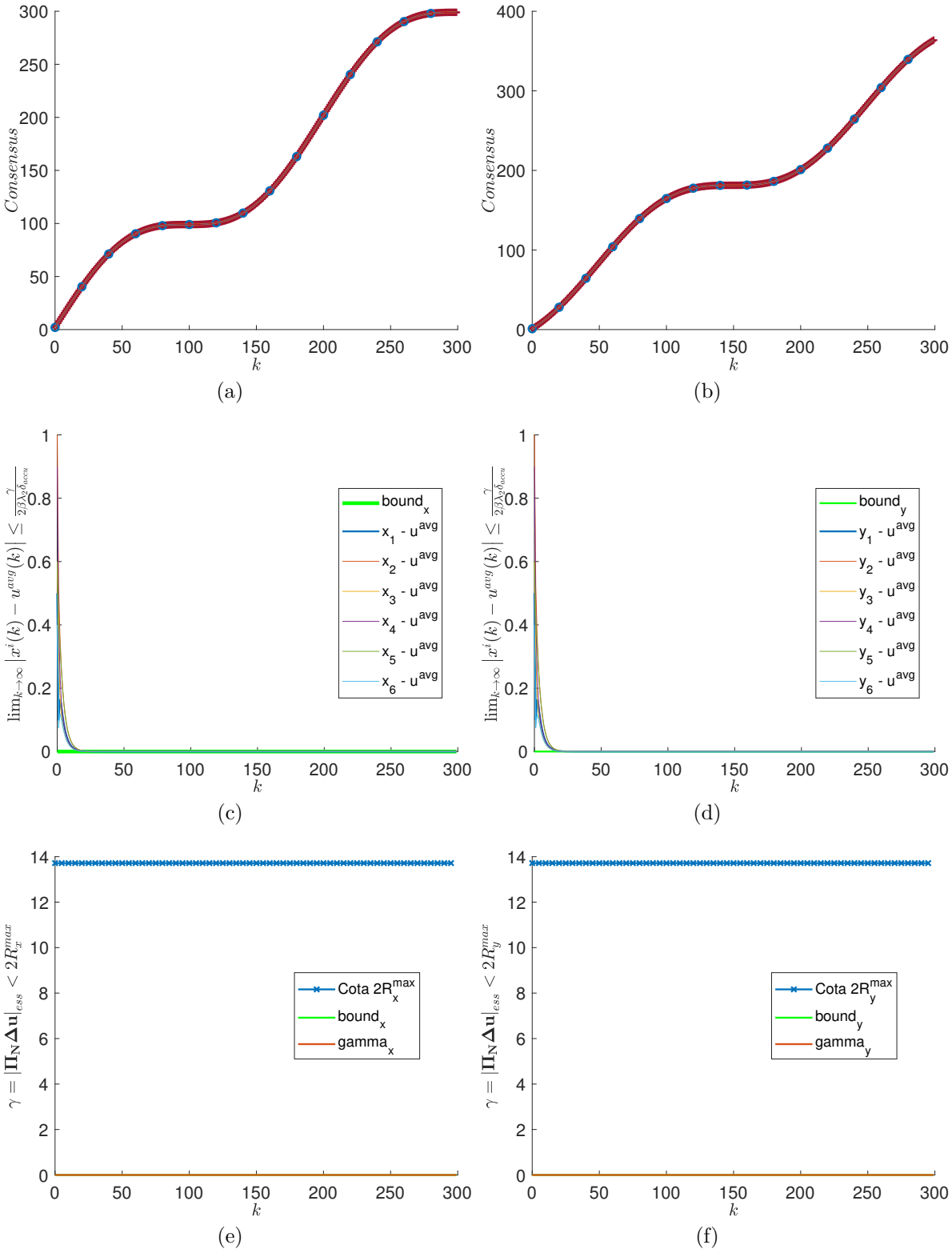


Figura 6.7: *Caso 1, trayectoria tipo 2b. Ver tabla (6.4). Como se aprecia en las gráficas (c) y (d) no hay error de tracking.*

Sorprendentemente, como se aprecia en la tabla 6.5, el valor de γ es demasiado elevado, lo cual conlleva a un posible problema numérico de implementación.

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	$2R_x^{max}$	$2R_y^{max}$
0	0	1	1	1	1	∞	∞	∞	∞
299	299	2.38e+50	2.3e+50	2.3e+50	2.3e+50	∞	∞	∞	∞

Tabla 6.5: Caso 1, trayectoria tipo 3a. Esta tabla se corresponde con la Fig.(6.8).

Simulación (b). Recta mas offset variable en el tiempo. Trayectoria con un offset variable en el tiempo y cuyo valor entre dos instantes de tiempo difiere en una cantidad considerable. En la tabla 6.6 se aprecia como γ es constante a lo largo de toda la simulación. Se debe a que en las gráficas (e) y (f), sólo se representa el máximo valor y no el historial a lo largo de cada instante de la simulación. Notase además, que bajo esta trayectoria, el algoritmo no diverge, como se puede apreciar en la Fig.(6.9) en las gráficas (e) y (f), donde γ se estabiliza.

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	$2R_x^{max}$	$2R_y^{max}$
0	0	0.99	0.99	0.03	0.03	6.85	6.85	∞	∞
299	299	25.54	25.54	13.52	13.52	6.85	∞	∞	∞

Tabla 6.6: Caso 1, trayectoria tipo 3b. Esta tabla se corresponde con la Fig.(6.9).

Simulación (c). Exponencial con offset variable en el tiempo. Trayectoria de tipo exponencial con offset variable en el tiempo ($s * k$, $\forall s \in \{1, \cdots, m\}$), aún mayor que el tipo 3b de trayectoria.

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	$2R^x max$	$2R^y max$
0	0	107.10	107.10	106.13	106.13	∞	∞	∞	∞
299	299	∞	∞	∞	∞	∞	∞	∞	∞

Tabla 6.7: Caso 1, trayectoria tipo 3c. Se corresponde con la Fig.(6.10).

6.1.2. Caso 2. Percepción variable (M variable) y $n = m$

Trayectoria Tipo 1. Recta igual para todos los agentes sin offset.

De nuevo, según el razonamiento llevado a cabo para este tipo de trayectorias en el caso 1 el parámetro γ es cero, como se aprecia en las tablas (6.8). Esto es debido a que al no existir un offset en la trayectoria, la señal de entrada de cada agente en el instante k coincide, ergo su media tendrá el mismo valor que la señal de entrada y por

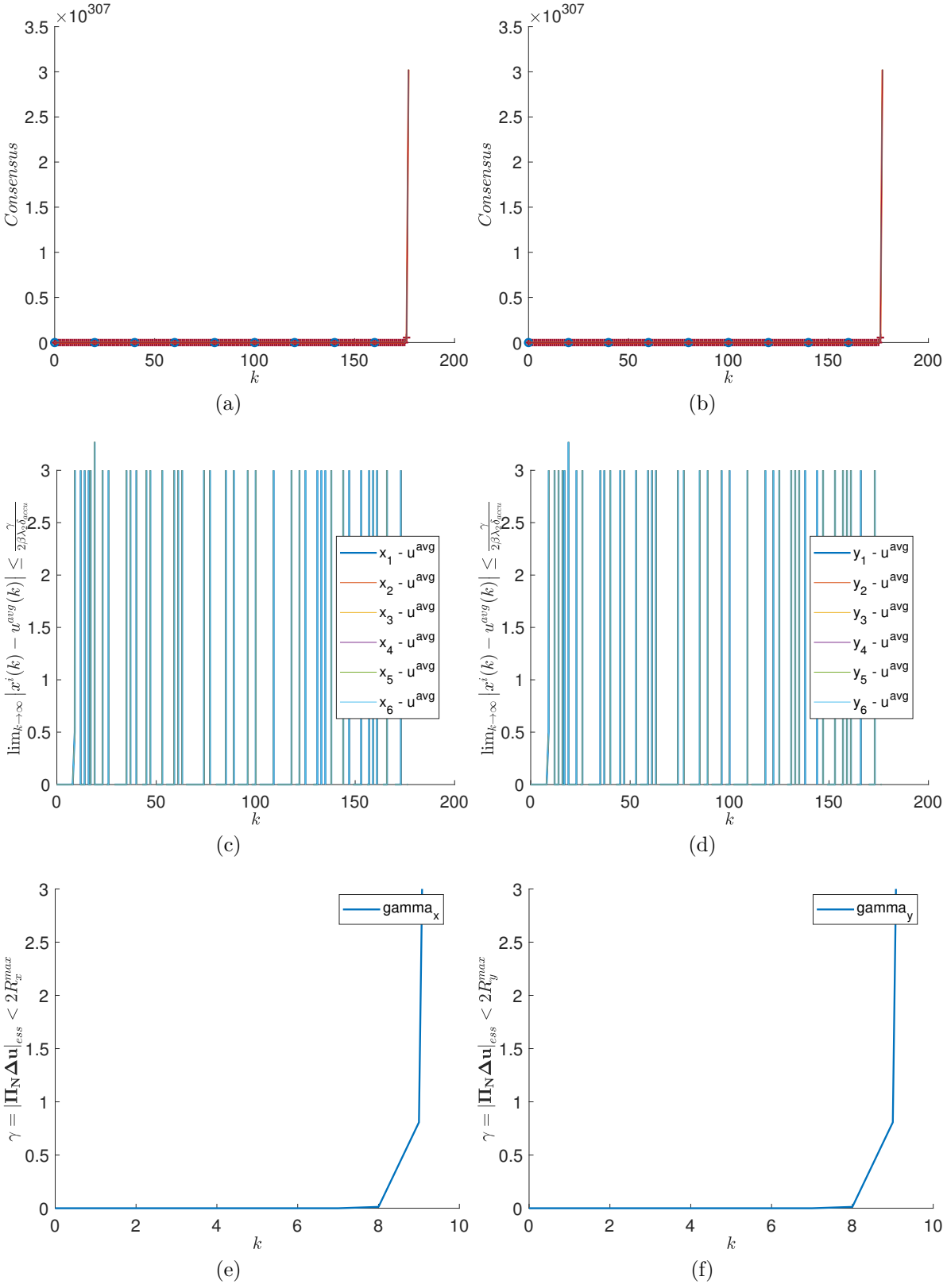


Figura 6.8: *Caso 1, trayectoria tipo 3a. En las gráficas (a) y (b) se aprecia la pérdida de consenso antes de llegar al instante* $k = 200$*, también (e) y (f) muestran como* γ *diverge. Ver tabla 6.5*

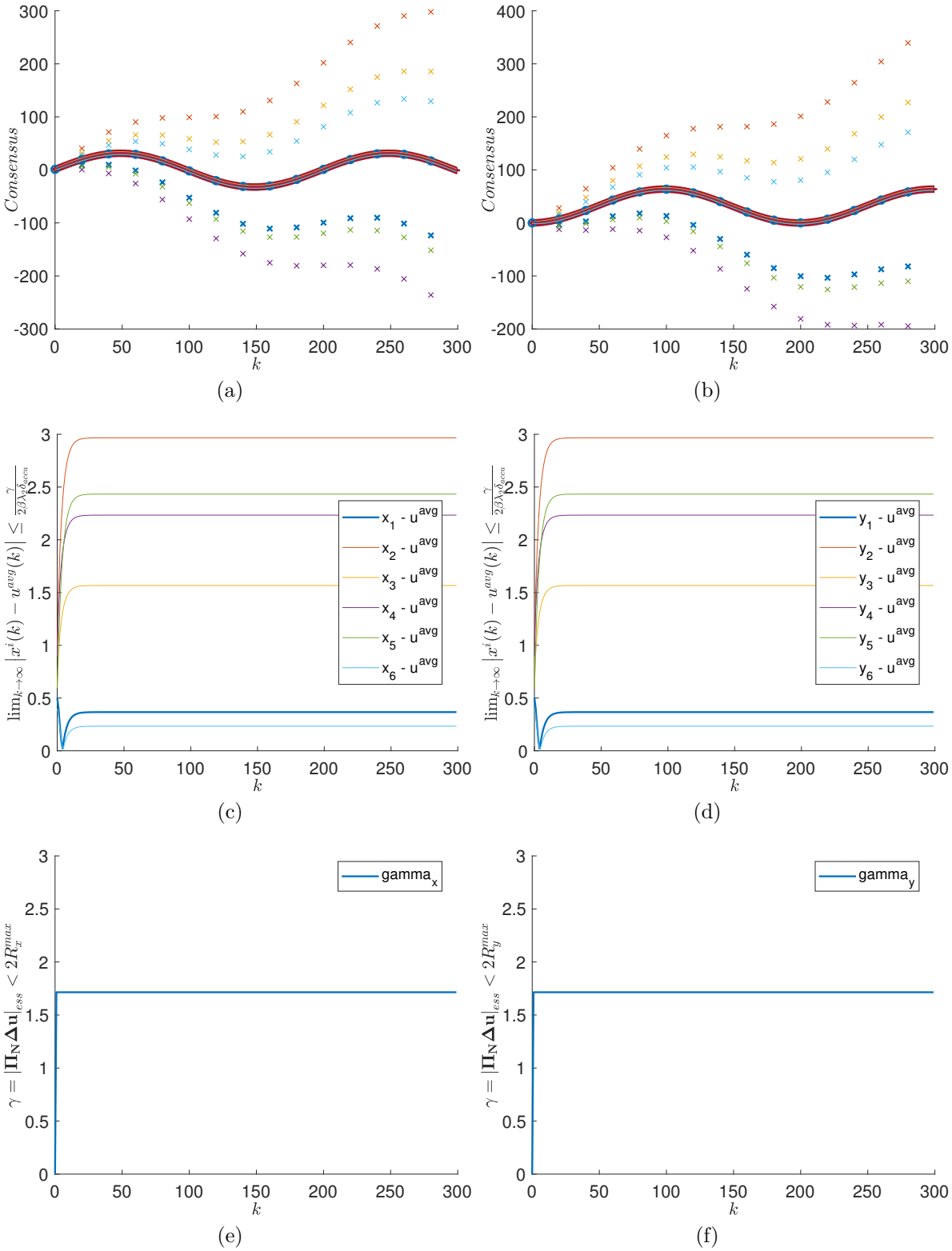


Figura 6.9: *Caso 1, trayectoria tipo 3b. Esta figura se corresponde con la tabla 6.6. Comentar que en las gráficas (c) y (d) se ha optado por no representar el bound* $= \frac{\gamma}{2\beta\lambda_2\delta_{accu}}$ *para que se pueda apreciar mejor visualmente la convergencia del error en estado estacionario que comete cada agente en un valor. Se puede comprobar numéricamente, sustituyendo los parámetros de las tablas 6.6 y 6.1, que el bound* $=$ 3,425 *y por tanto el error (máximo en 3) está por debajo de esta cota, luego el algoritmo converge.*

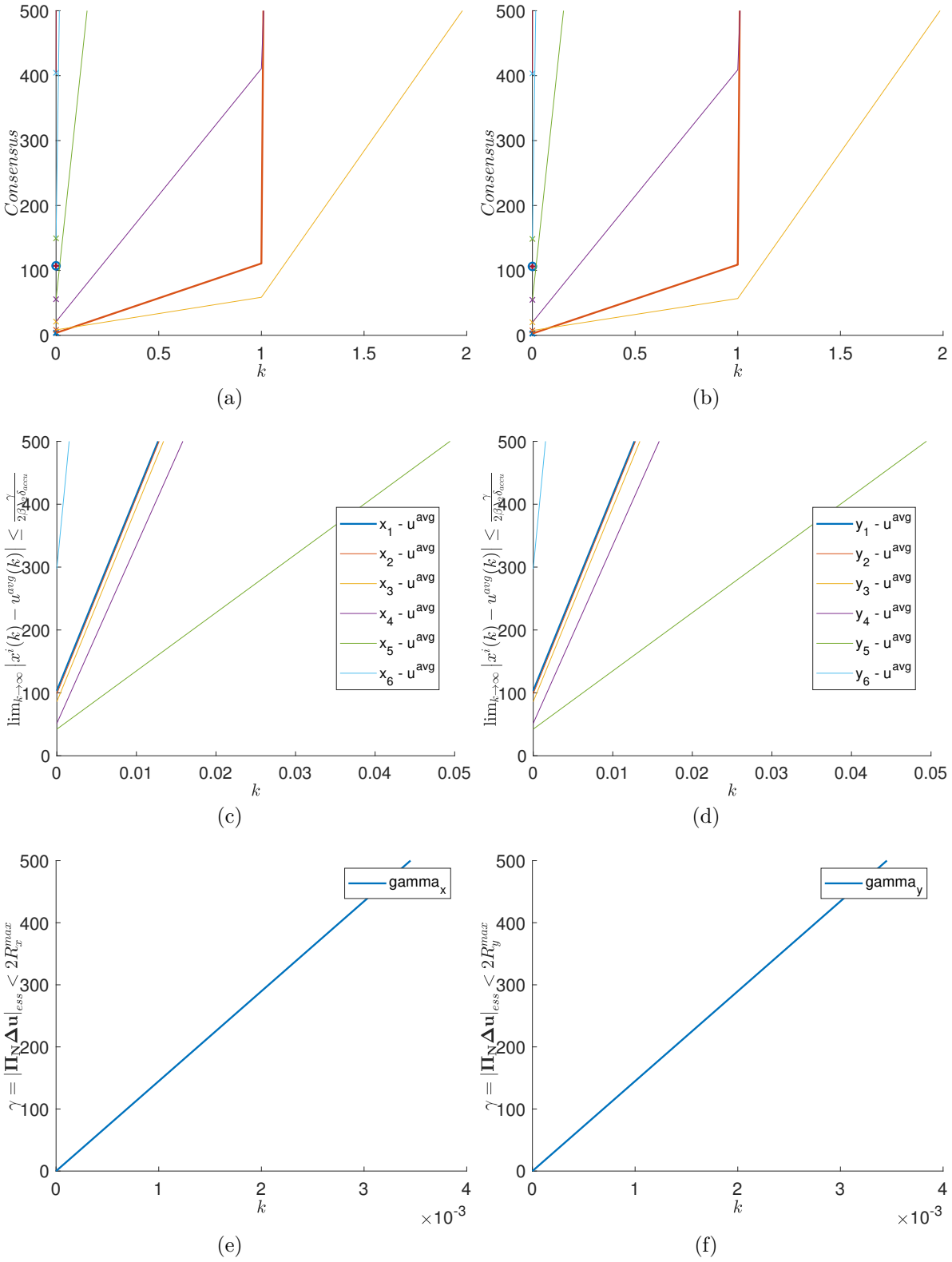


Figura 6.10: *Caso 1, trayectoria tipo 3c. Se corresponde con la tabla 6.7.*

tanto la diferencia entre estas es cero. Por otro lado, la cota superior $2R_{max}$ que debe cumplir el error cometido depende de γ, por ello, este parámetro también es nulo para este tipo de trayectorias. Esto se traduce en un error en estado estacionario cero.

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	γ_x^{rot}	γ_y^{rot}	$2R_x^{max}$	$2R_y^{max}$
0	0	0.99	0.99	0.03	0.03	0	0	0	0	0	0
299	299	-0.99	-0.99	63.65	63.65	0	0	0	0	0	0

Tabla 6.8: Caso 2, trayectoria tipo 1. Se corresponde con la Fig.(6.11).

Trayectorias Tipo 2. Recta para todos los agentes más un offset distinto para cada agente.

Comentar, que en este caso se introduce un nuevo parámetro en las tablas correspondiente al parámetro $2R^{rot}$ resultante de la rotación de los agentes.

Simulación (a). Offset constante en el tiempo. Para este tipo de trayectorias, en ambos casos 2 y 3, el parámetro γ no es cero, como se aprecia en las tablas (6.9) y (6.15). Ya que aunque la trayectoria es igual que en el caso 1, la matriz de percepción $\mathbf{M}$ en este caso cambia en el tiempo, por lo que las diferencias de las señales de entrada para un mismo agente en dos instantes consecutivos de tiempo, no serán constantes, esto es, el offset cambia para cada agente de un instante a otro. Por ello, la diferencia de cada agente con el promedio de las señales de entrada no es nula. Por otro lado, como se ha comentado en el caso 1, para este tipo de trayectorias, la cota superior $2R_{max}$ tampoco es nula para este tipo de trayectorias. Esto se traduce en un error en estado estacionario no nulo. Nótese, que el parámetro $2R^{rot}$ represente la cota diseñada cuando los agentes rotan alrededor del centroide, mientras que $2R^{max}$ es la cota cuando la percepción es constante y $n = m$.

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	$2R_x^{rot}$	$2R_y^{rot}$	$2R_x^{max}$	$2R_y^{max}$
0	0	0.99	0.99	0.03	0.03	3.48	3.48	6.85	6.85	6.85	6.85
299	299	-0.99	-0.99	63.65	63.65	3.48	3.48	6.85	6.85	6.85	6.85

Tabla 6.9: Caso 2, trayectoria tipo 2a. Se corresponde con la Fig.(6.12).

Simulación (b). Offset variable en el tiempo. Al igual que en la trayectoria anterior, en este tipo de trayectorias, en ambos casos 2 y 3, el parámetro γ no es cero, como se aprecia en las tablas (6.10) y (6.16).Ya que aunque la trayectoria es igual que en el caso 1, la matriz de percepción $\mathbf{M}$ en este caso cambia en el tiempo, por lo que las diferencias de las señales de entrada para un mismo agente en dos instantes

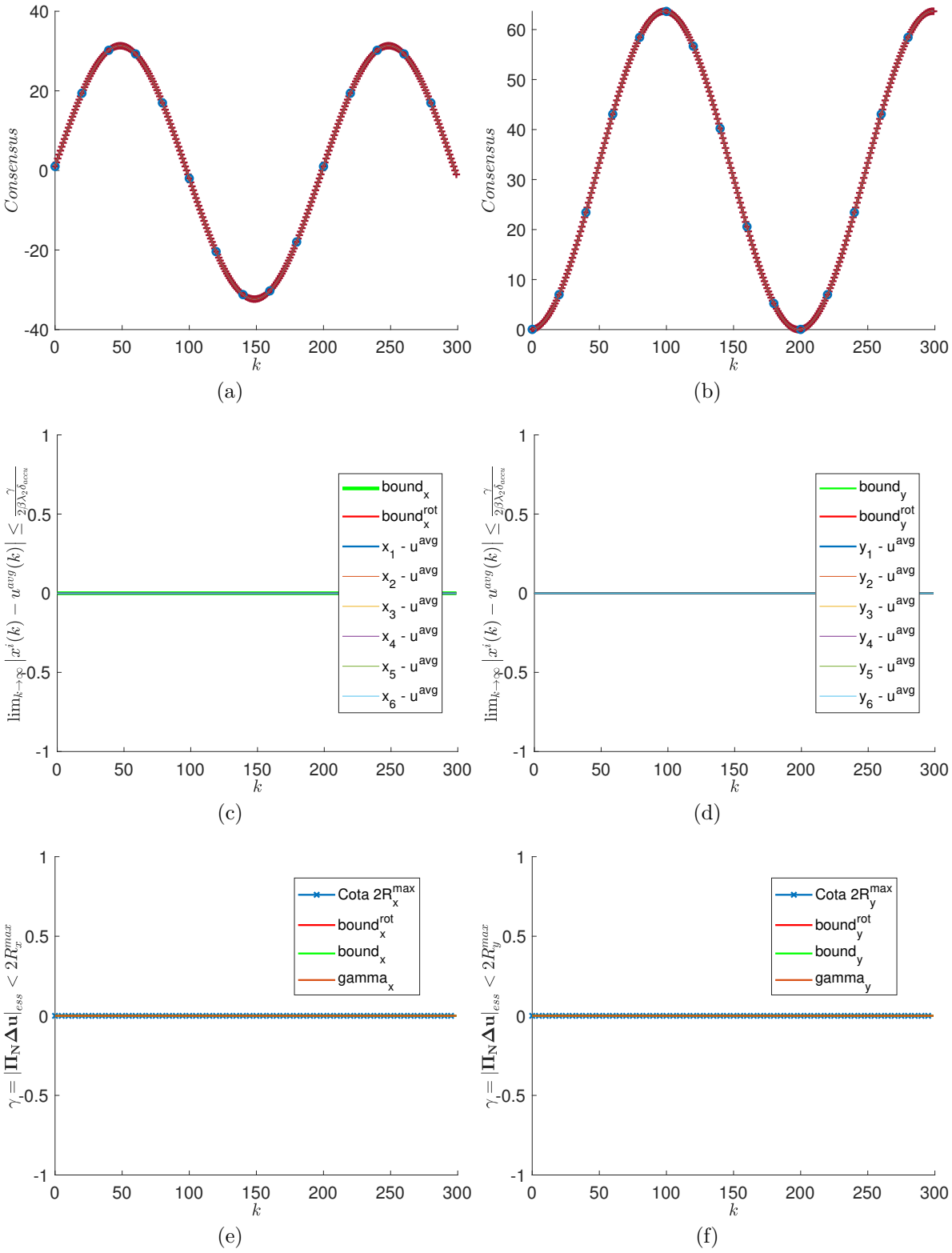


Figura 6.11: *Caso 2, trayectoria tipo 1. Se corresponde con la tabla 6.8. Se verifica que* γ *y* $2R_{max}$ *valen cero porque no hay error en estado estacionario.*

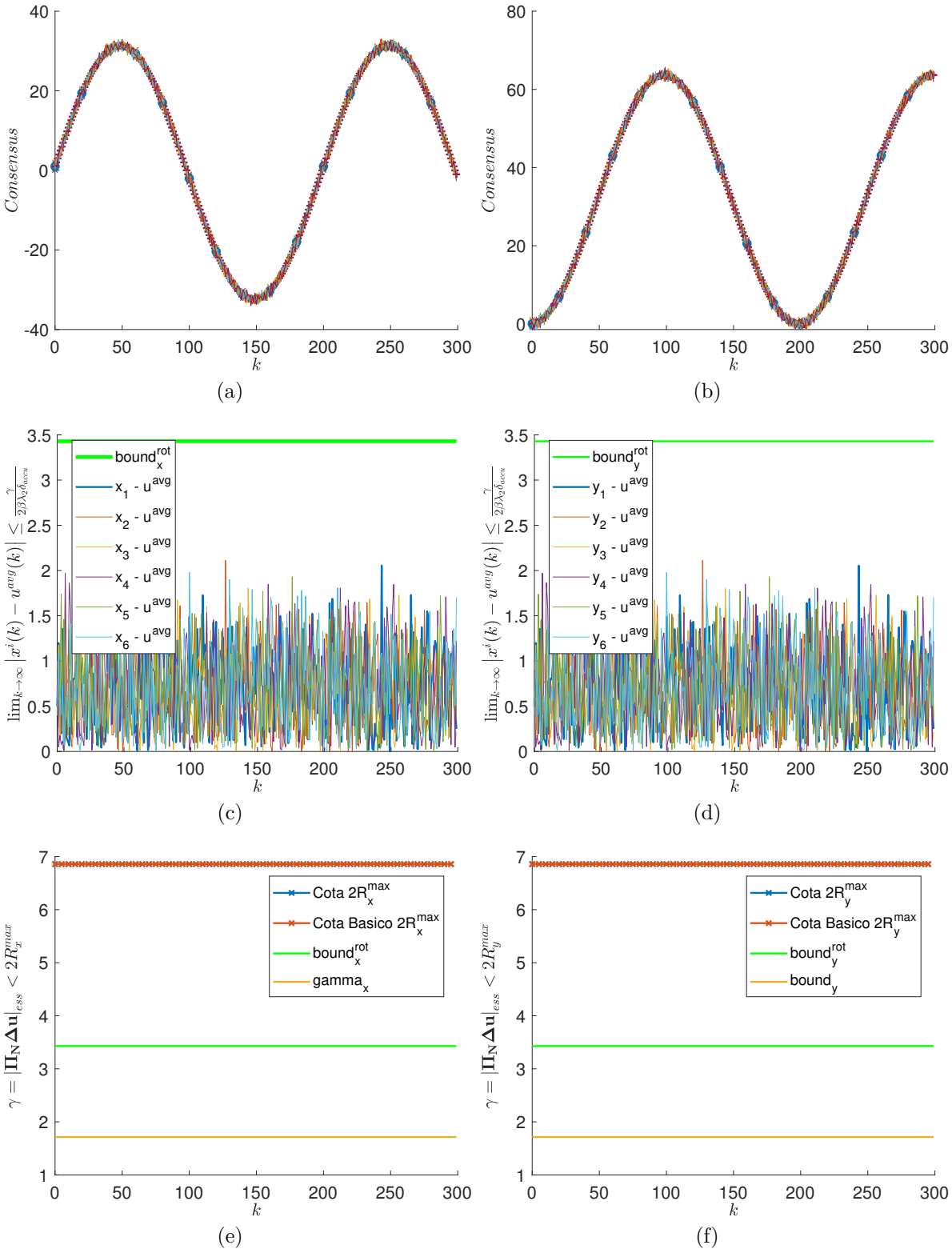


Figura 6.12: *Caso 2, trayectoria tipo 2a. Esta simulación se corresponde con los datos de la tabla 6.9. Gráficas (c) y (d) muestran el error de tracking cometido (líneas sólidas) y la cota superior (línea verde sólida). Nótese que la convergencia del algoritmo está garantizada ya que el error cometido se encuentra por debajo de la cota. Gráficas (e) y (f) muestran la cota diseñada por nosotros para ambos casos (caso 1 línea discontinua en azul y caso 2 línea discontinua en rojo).*

consecutivos de tiempo, no serán constantes, esto es, el offset cambia para cada agente de un instante a otro. Por ello, la diferencia de cada agente con el promedio de las señales de entrada no es nula. Por otro lado, como se ha comentado en el caso 1, para este tipo de trayectorias, la cota superior $2R_{max}$ tampoco es nula, lo que se traduce en un error en estado estacionario distinto de cero.

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	$2R_x^{rot}$	$2R_y^{rot}$	$2R_x^{max}$	$2R_y^{max}$
0	0	1.99	1.99	1.03	1.03	3.48	3.48	6.85	6.85	6.85	6.85
299	299	299	299	363	363	3.48	3.48	6.85	6.85	6.85	6.85

Tabla 6.10: Caso 2, trayectoria tipo 2b. Se corresponde con la Fig.(6.13).

Trayectorias Tipo 3. Exponenciales.

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	γ_x^{rot}	γ_y^{rot}	$2R_x^{max}$	$2R_y^{max}$
0	0	1	1	1	1	∞	∞	∞	∞	∞	∞
299	299	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Tabla 6.11: Caso 2, trayectoria tipo 3a. Se corresponde con la Fig.(6.14). Se aprecia la pérdida de la convergencia total.

Simulación (a). Exponencial sin offset.

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	γ_x^{rot}	γ_y^{rot}	$2R_x^{max}$	$2R_y^{max}$
0	0	0.99	0.99	0.03	0.03	3.73	3.73	6.85	6.85	∞	∞
299	299	-0.99	-0.99	63.65	63.65	3.73	3.73	6.85	6.85	∞	∞

Tabla 6.12: Caso 2, trayectoria tipo 3b. Se corresponde con la Fig.(6.15). Se aprecia la convergencia del algoritmo como en el caso anterior, para la misma trayectoria.

Simulación (b). Recta mas offset variable en el tiempo.

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	γ_x^{rot}	γ_y^{rot}	$2R_x^{max}$	$2R_y^{max}$
0	0	1	1	1	1	∞	∞	∞	∞	∞	∞
299	299	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Tabla 6.13: Caso 2, trayectoria tipo 3c. Se corresponde con la Fig.(6.16). Se aprecia la pérdida de la convergencia total.

Simulación (c). Exponencial con offset variable en el tiempo.

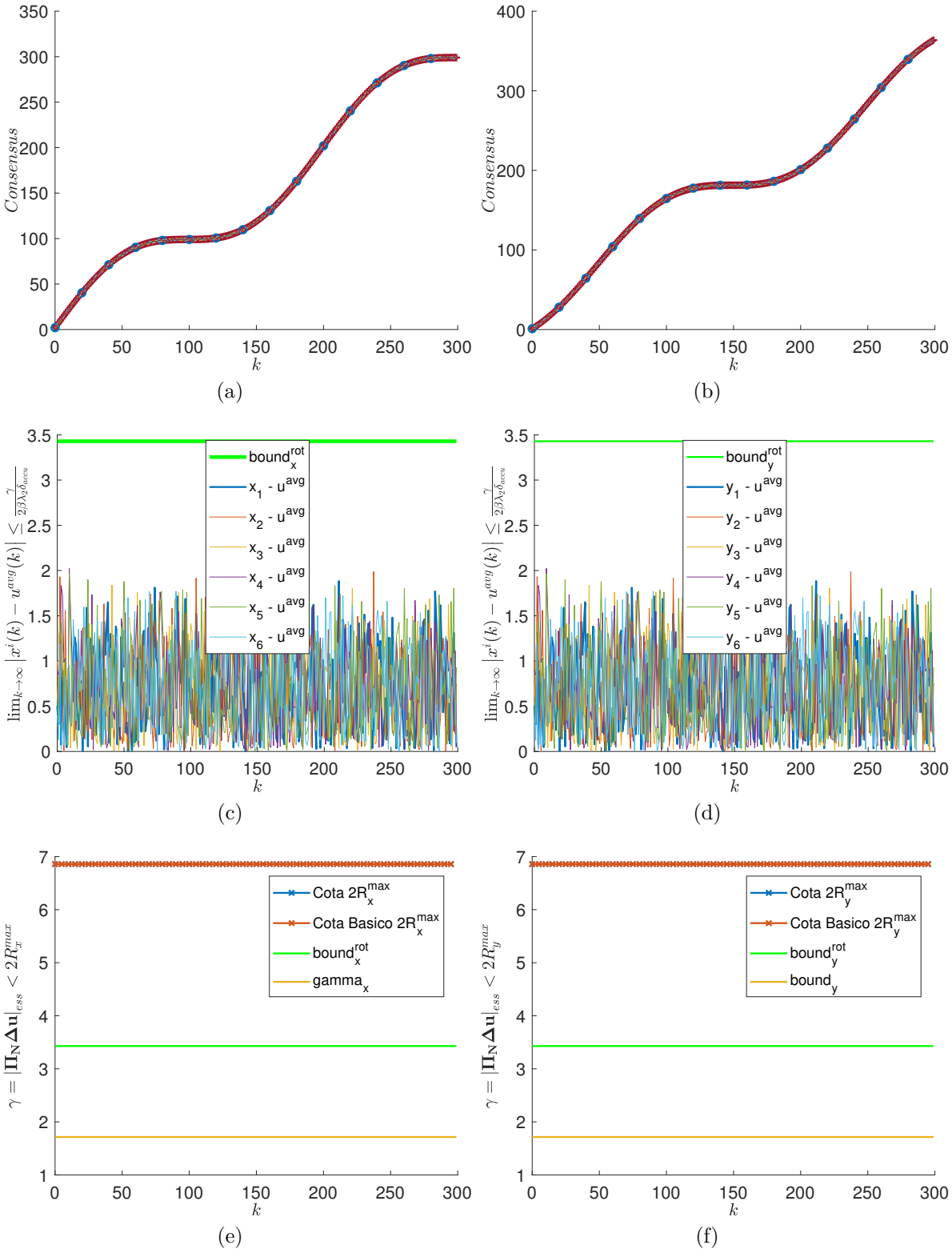


Figura 6.13: *Caso 2, trayectoria tipo 2b. Se corresponde con la tabla 6.10. Gráficas (c) y (d) muestran el error de tracking cometido (líneas sólidas) y la cota superior (línea verde sólida). Nótese que la convergencia del algoritmo está garantizada ya que el error cometido se encuentra por debajo de la cota. Gráficas (e) y (f) muestran la cota diseñada por nosotros para ambos casos (caso 1 línea discontinua en azul y caso 2 línea discontinua en rojo).*

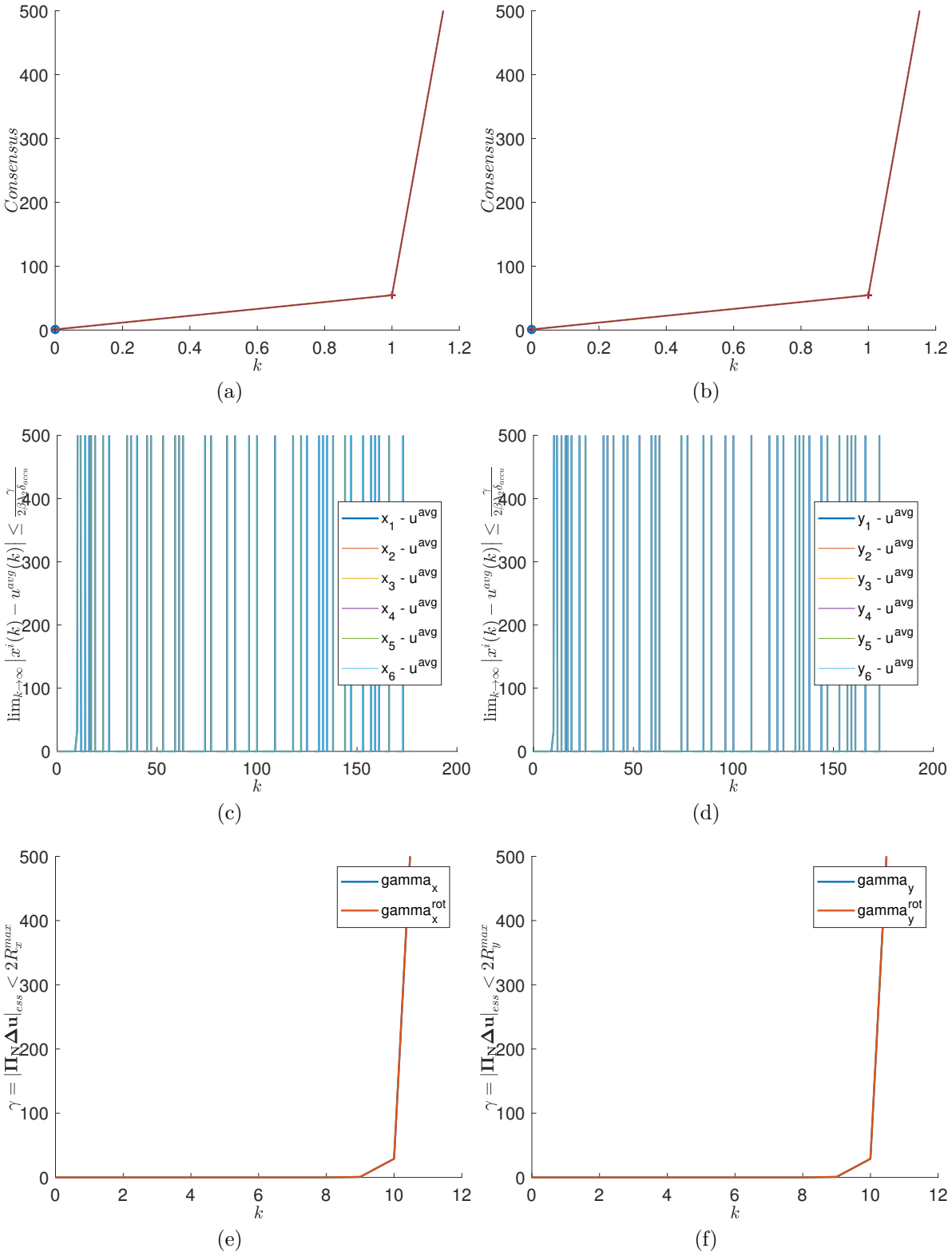


Figura 6.14: *Caso 2, trayectoria tipo 3a. Se corresponde con la tabla 6.11. De nuevo, mismo comentario que en el caso 1, para la misma trayectoria.*

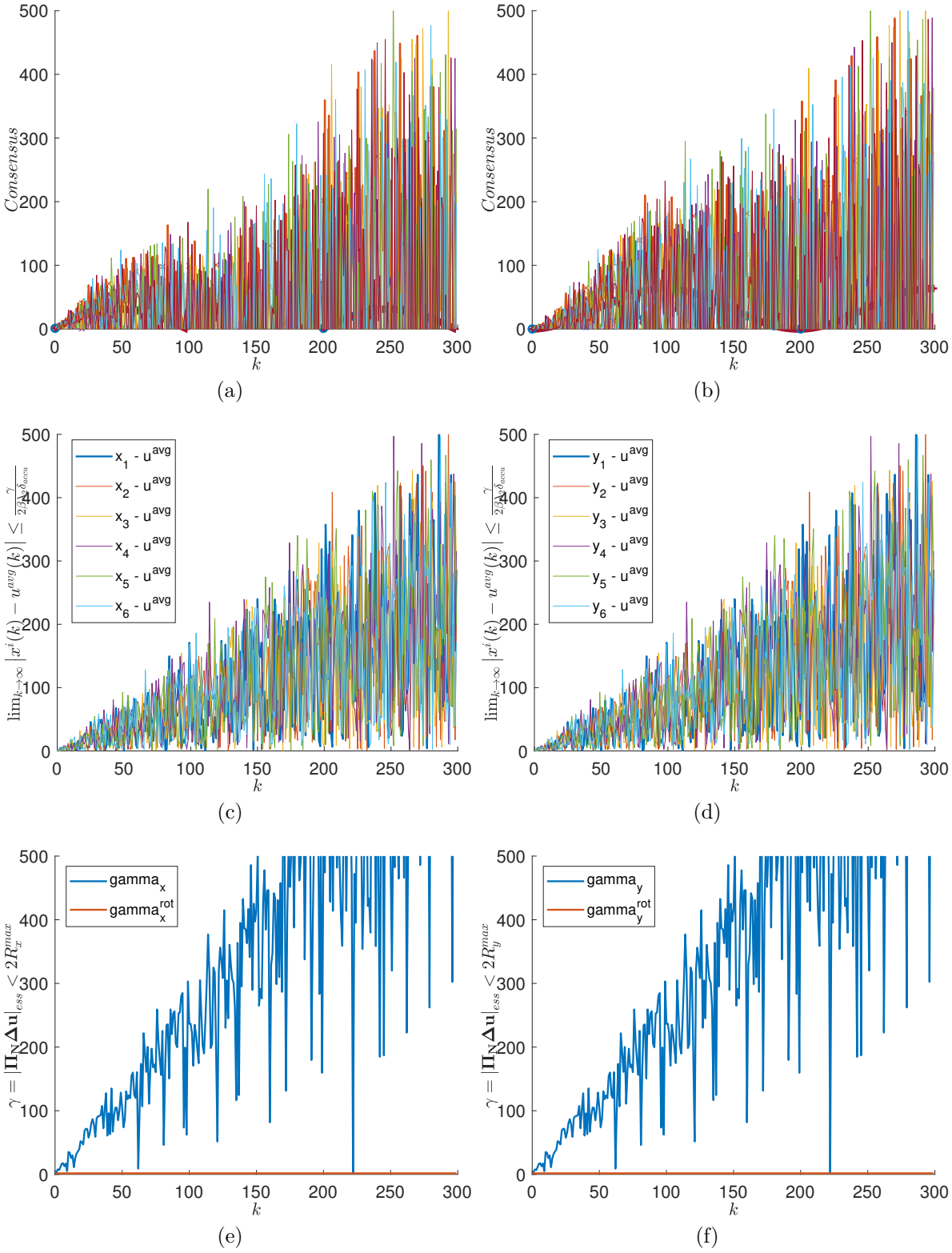


Figura 6.15: *Caso 2, trayectoria tipo 3b. Se corresponde con la tabla 6.12.*

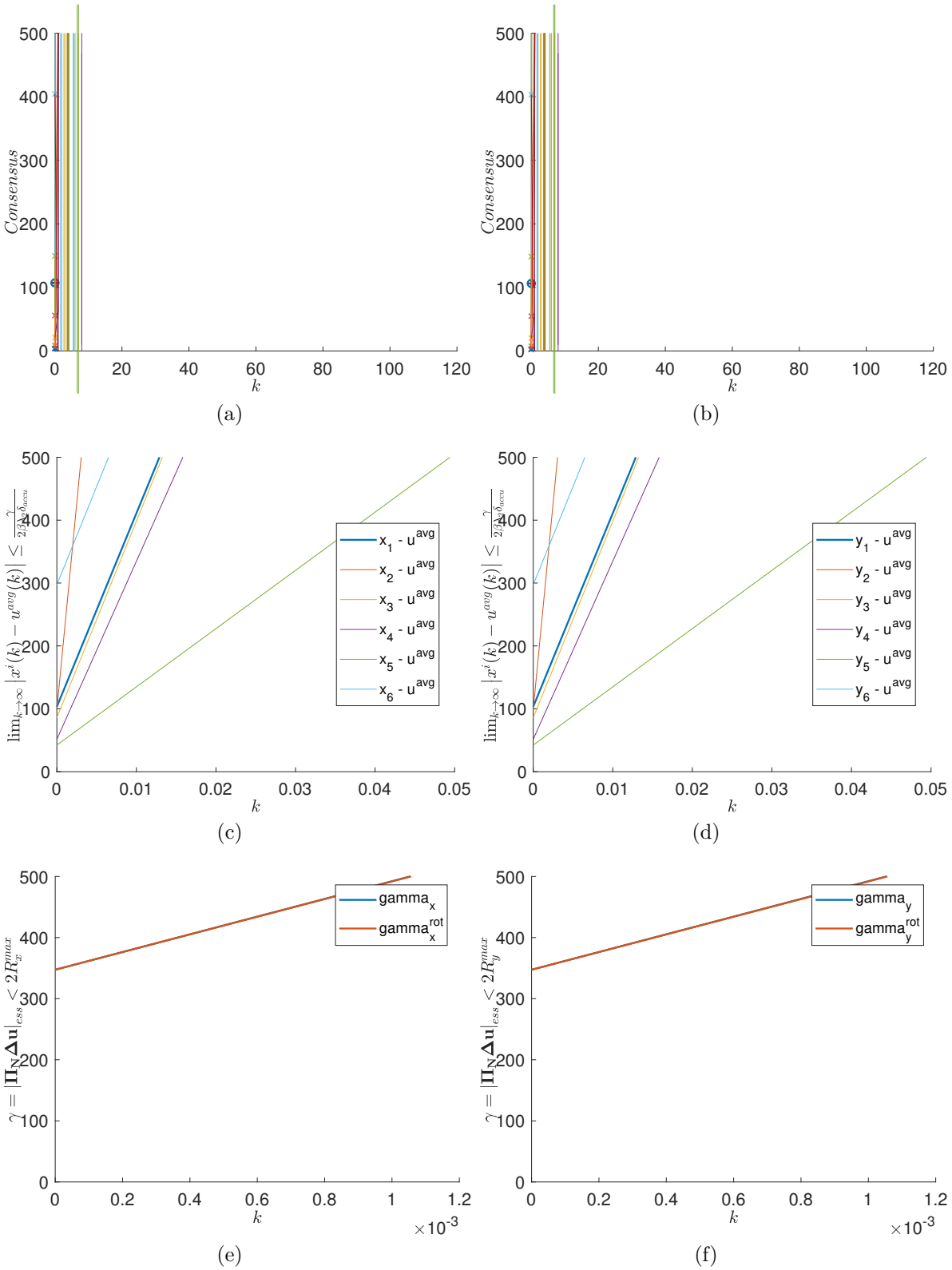


Figura 6.16: *Caso 2, trayectoria tipo 3c. Se corresponde con la tabla 6.13. De nuevo, mismo comentario que en el caso 1, para la misma trayectoria.*

6.1.3. Caso 3. Percepción fija (M) constante y $n < m$

Trayectoria Tipo 1. Recta igual para todos los agentes sin offset.

En este caso y para este tipo de trayectorias, a diferencia de los dos casos previos el parámetro γ no es cero, como se aprecia en las tablas (6.14), debido a que a pesar de no existir un offset en la trayectoria, el diseño de la input propuesta es diferente a los casos previos, tal como se comentó en el Capítulo (4). Ahora cada agente accede a una señal de entrada promedio (parámetro $K = \frac{N}{m}$), ya que la percepción es parcial y un agente puede ver más de un objetivo. Esto quiere decir que la señal de entrada de cada agente en el instante k no coincide con su media y por tanto la diferencia entre estas no es cero. De igual forma, la cota superior $2R_{max}$ que debe cumplir el error cometido depende de γ, por ello, este parámetro tampoco es nulo para este tipo de trayectorias, en este caso concreto. Esto se traduce en un error en estado estacionario distinto de cero.

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	$2R_x^{max}$	$2R_y^{max}$
0	0	0.99	0.99	0.03	0.03	0.79	0.64	40.86	21.63
29	29	25.54	25.54	13.52	13.52	0.79	0.64	40.86	21.63

Tabla 6.14: Caso 3, trayectoria tipo 1. Se corresponde con la Fig.(6.17).

Trayectorias Tipo 2. Recta para todos los agentes más un offset distinto para cada agente.

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	$2R_x^{max}$	$2R_y^{max}$
0	0	0.86	0.86	-0.09	-0.09	0.79	0.64	41.19	21.97
29	29	25.41	25.41	13.39	13.39	0.79	0.64	41.19	21.97

Tabla 6.15: Caso 3, trayectoria tipo 2a. Se corresponde con la Fig.(6.18).

Simulación (a). Offset constante en el tiempo.

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	$2R_x^{max}$	$2R_y^{max}$
0	0	1.86	1.86	0.90	0.90	1.59	1.44	89.19	69.96
29	29	55.41	55.41	43.39	43.39	1.59	1.44	89.19	69.96

Tabla 6.16: Caso 3, trayectoria tipo 2b. Se corresponde con la Fig.(6.19).

Simulación (b). Offset variable en el tiempo.

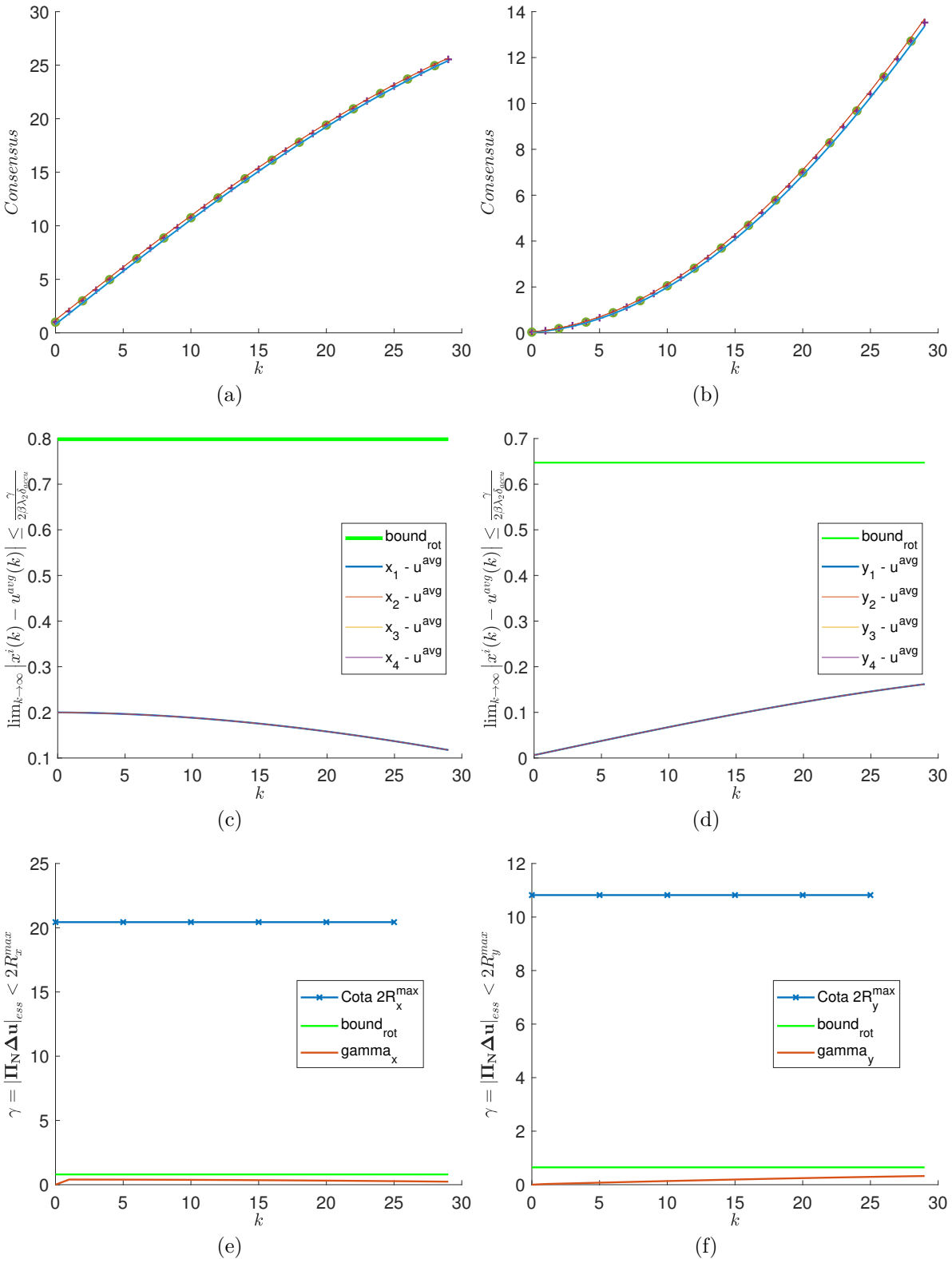


Figura 6.17: *Caso 3, trayectoria tipo 1. Se corresponde con la tabla 6.14. En este caso, se ha de comentar que en la gráfica (d) se aprecia una cierta tendencia a la divergencia. Sería necesario analizar si esta divergencia desestabiliza la simulación.*

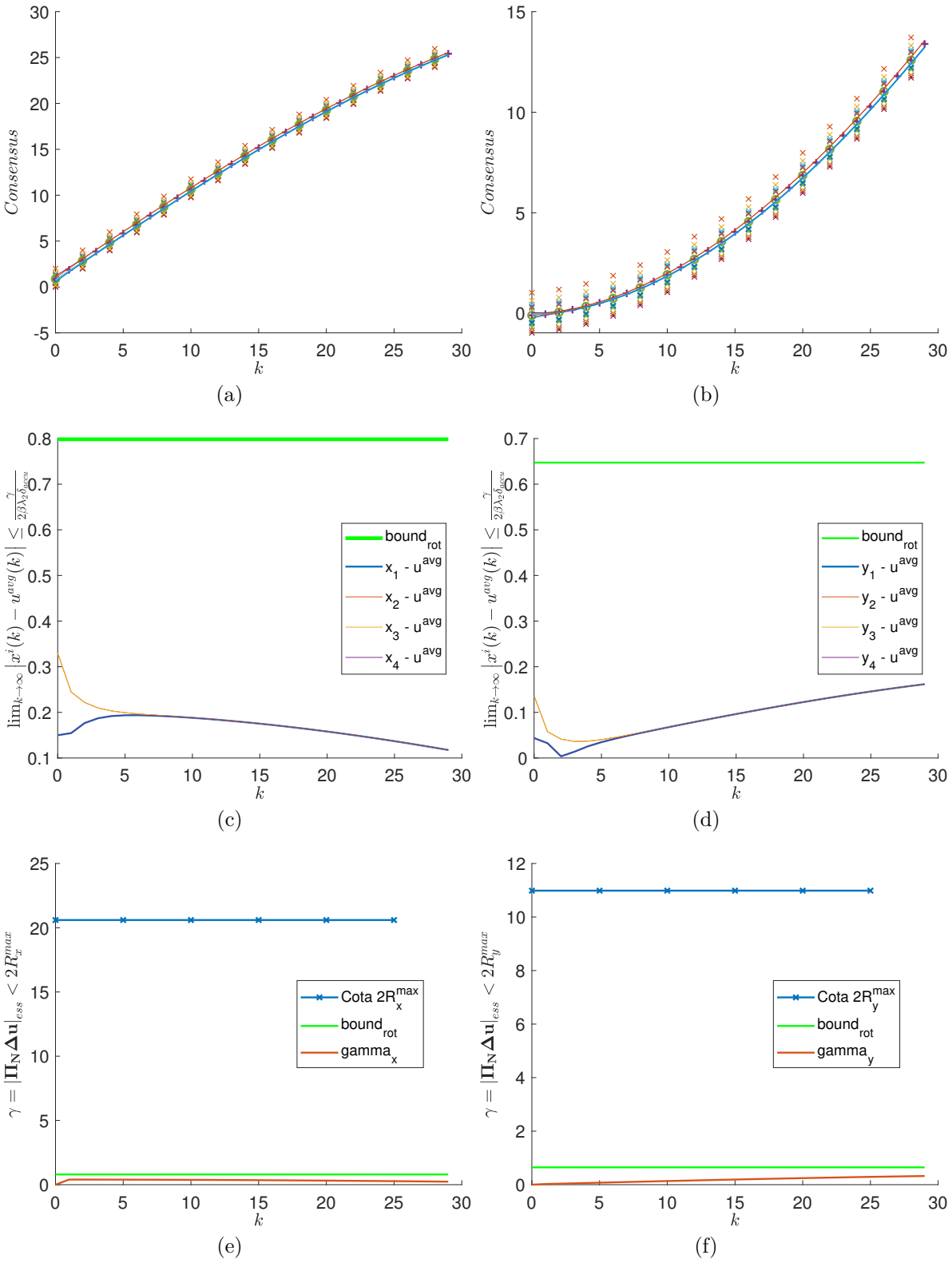


Figura 6.18: *Caso 3, trayectoria tipo 2a. Esta gráfica de simulación se corresponde con la tabla 6.15. En este caso, se ha de comentar para la gráfica (d) una cierta tendencia a la divergencia. Sería necesario analizar esta divergencia desestabiliza la simulación.*

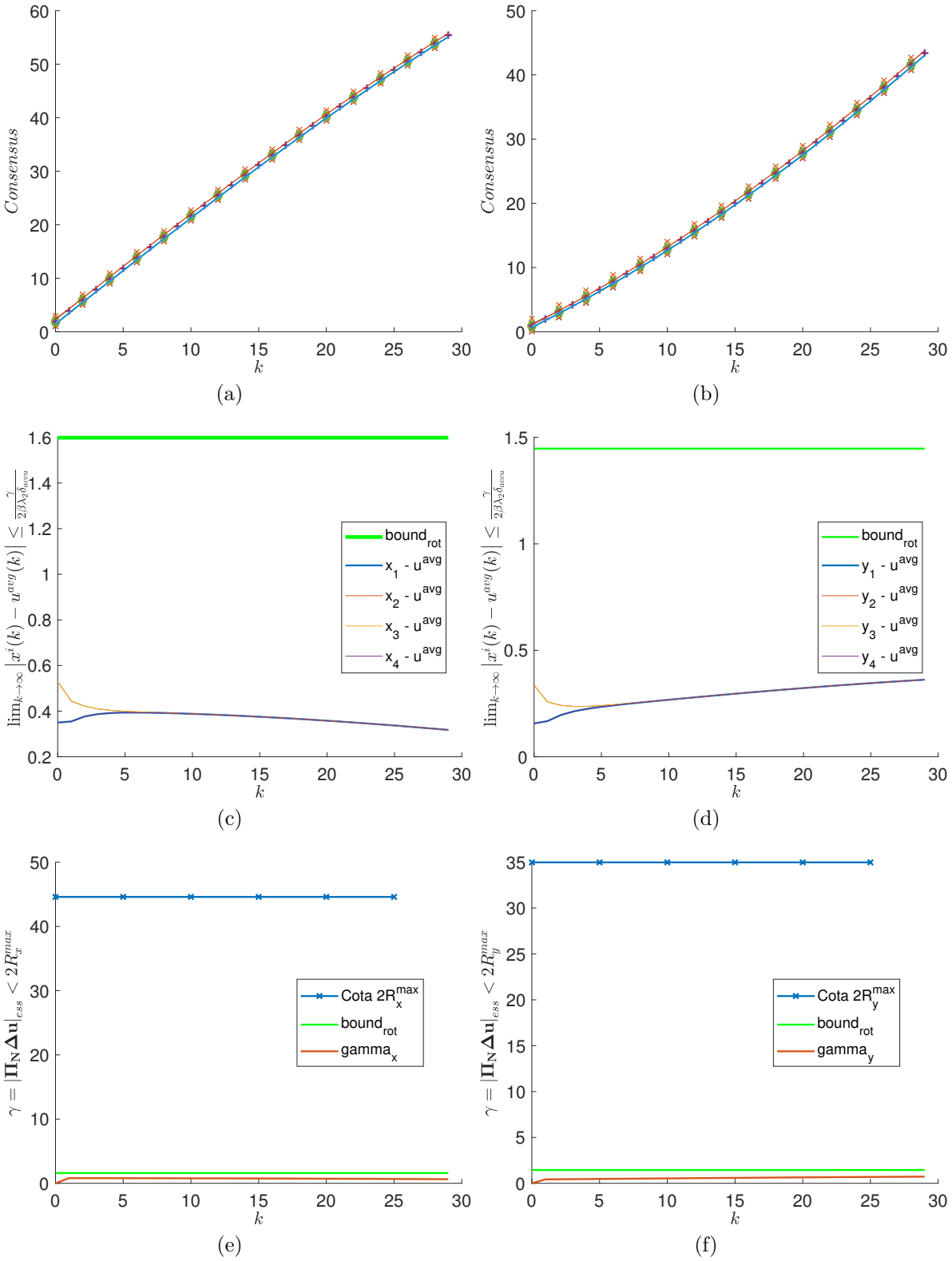


Figura 6.19: *Caso 3, trayectoria tipo 2b. Esta gráfica de simulación se corresponde con la tabla 6.16. En este caso, se ha de comentar para la gráfica (d) una cierta tendencia a la divergencia. Sería necesario repetir la simulación hasta un* $k = 200$ *para dejar alcanzar al algoritmo su estado de convergencia y evaluar de nuevo los resultados.*

Trayectorias Tipo 3. Exponenciales

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	$2R_x^{max}$	$2R_y^{max}$
0	0	1	1	1	1	1.8e+50	1.8e+50	3.8e+50	3.8e+50
29	29	2.38e+50	2.3e+50	2.3e+50	2.3e+50	1.8e+50	1.8e+50	3.8e+50	3.8e+50

Tabla 6.17: Caso 3, trayectoria tipo 3a. Se corresponde con la Fig.(6.20). Tanto en las gráficas como en los valores numéricos se aprecia una clara tendencia a la divergencia.

Simulación (a). Exponencial sin offset.

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	$2R_x^{max}$	$2R_y^{max}$
0	0	0.86	0.86	-0.09	-0.09	1.02	0.88	54.89	37.97
29	29	21.64	21.64	9.62	9.62	1.02	0.88	54.89	37.97

Tabla 6.18: Caso 3, trayectoria tipo 3b. Se corresponde con la Fig.(6.21).

Simulación (b). Recta más offset variable en el tiempo.

t	k	x	x_T^{avg}	y	y_T^{avg}	γ_x	γ_y	$2R_x^{max}$	$2R_y^{max}$
0	0	3.4854e+03	3.4854e+03	3.4844e+03	3.4844e+03	∞	∞	∞	∞
29	29	∞	∞	∞	∞	∞	∞	∞	∞

Tabla 6.19: Caso 3, trayectoria tipo 3c. Se corresponde con la Fig.(6.22).

Simulación (c). Exponencial con offset variable en el tiempo. Para finalizar, algunas de las conclusiones que se han extraído son las siguientes:

- La sorprendente convergencia del algoritmo para trayectorias no triviales.
- La divergencia para algunas trayectorias del caso de parcepción parcial de los objetivos.
- Se concluye la existencia de ciertas tendencias en las gráficas que darán lugar a un estudio futuro.
- Se ha estudiado la robustez para tres casos distintos de posicionamiento de los agentes alrededor del centroide. Aquí, la estrategias que ofrece mayores garantías de convergencia es la implementada en el caso 1, con una percepción constante y $n = m$. Entre las estrategias implementadas en el caso 2 y 3, es más robusta la primera, cuando los agentes rotan y $n = m$.

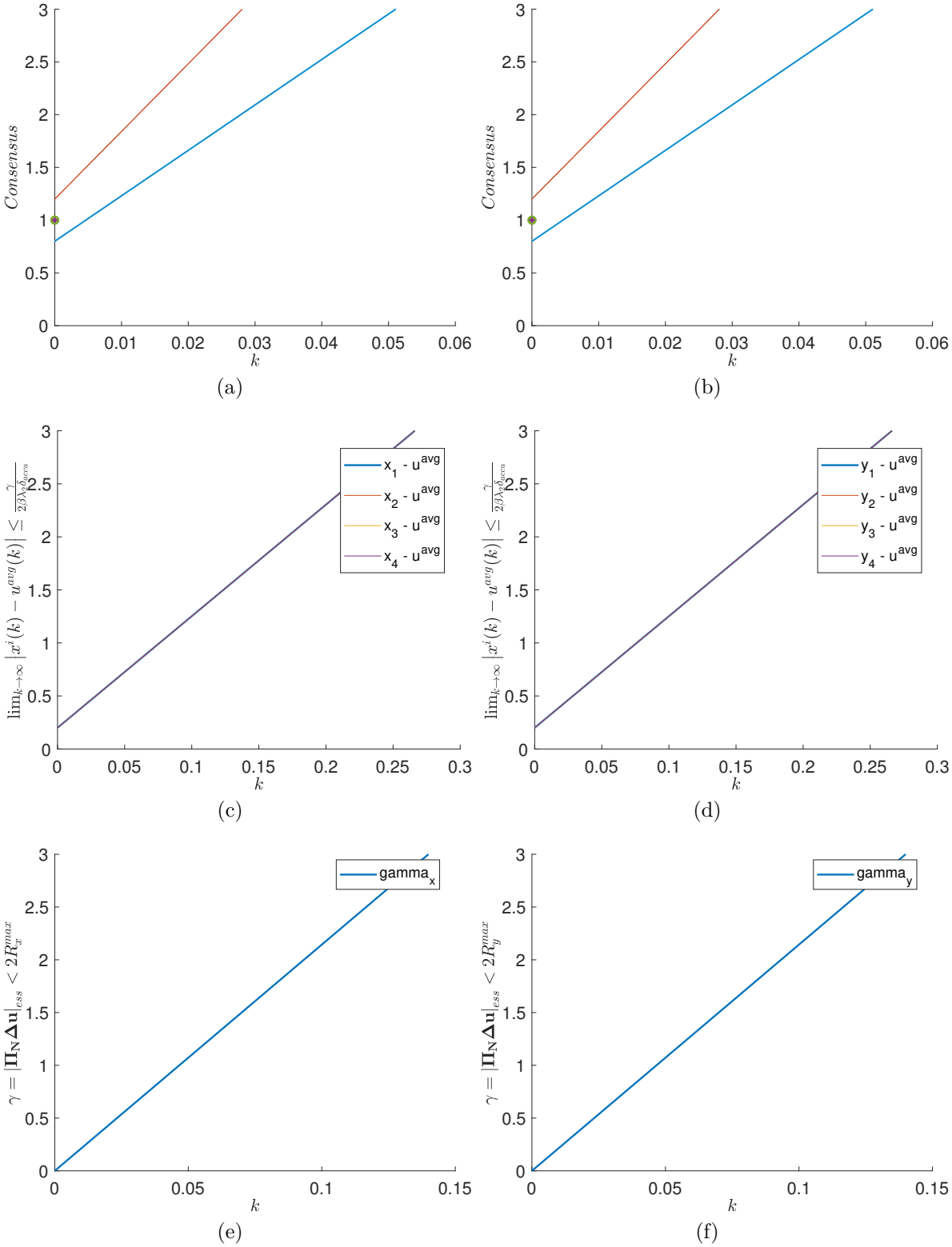


Figura 6.20: *Caso 3, trayectoria tipo 3a. Esta gráfica de simulación se corresponde con la tabla 6.17. Evidente tendencia a la divergencia en todas las gráficas. En la gráfica (a) y en la (b), sólo se ve un punto en el instante inicial relativo al centroide del grupo de objetivos. No se ven el resto porque la posición de los objetivos toma valores muy elevados y en la simulación se ha limitado el eje y de la gráfica hasta 3.*

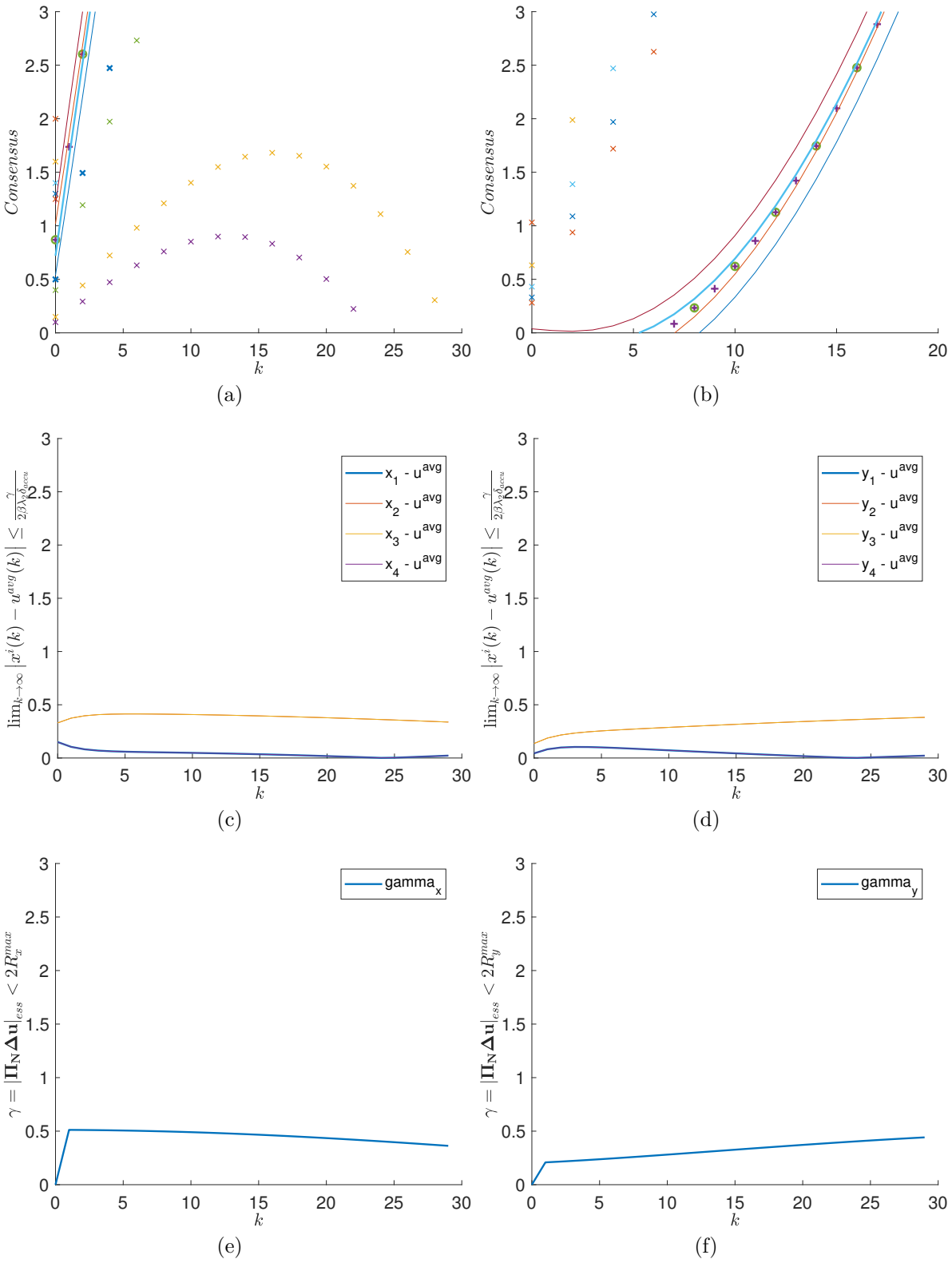


Figura 6.21: *Caso 3, trayectoria tipo 3b. Esta gráfica de simulación se corresponde con la tabla 6.18. Como se ha comentado en casos anteriores, el algoritmo diverge para este tipo de trayectorias, véase (d), donde el error de tracking aumenta.*

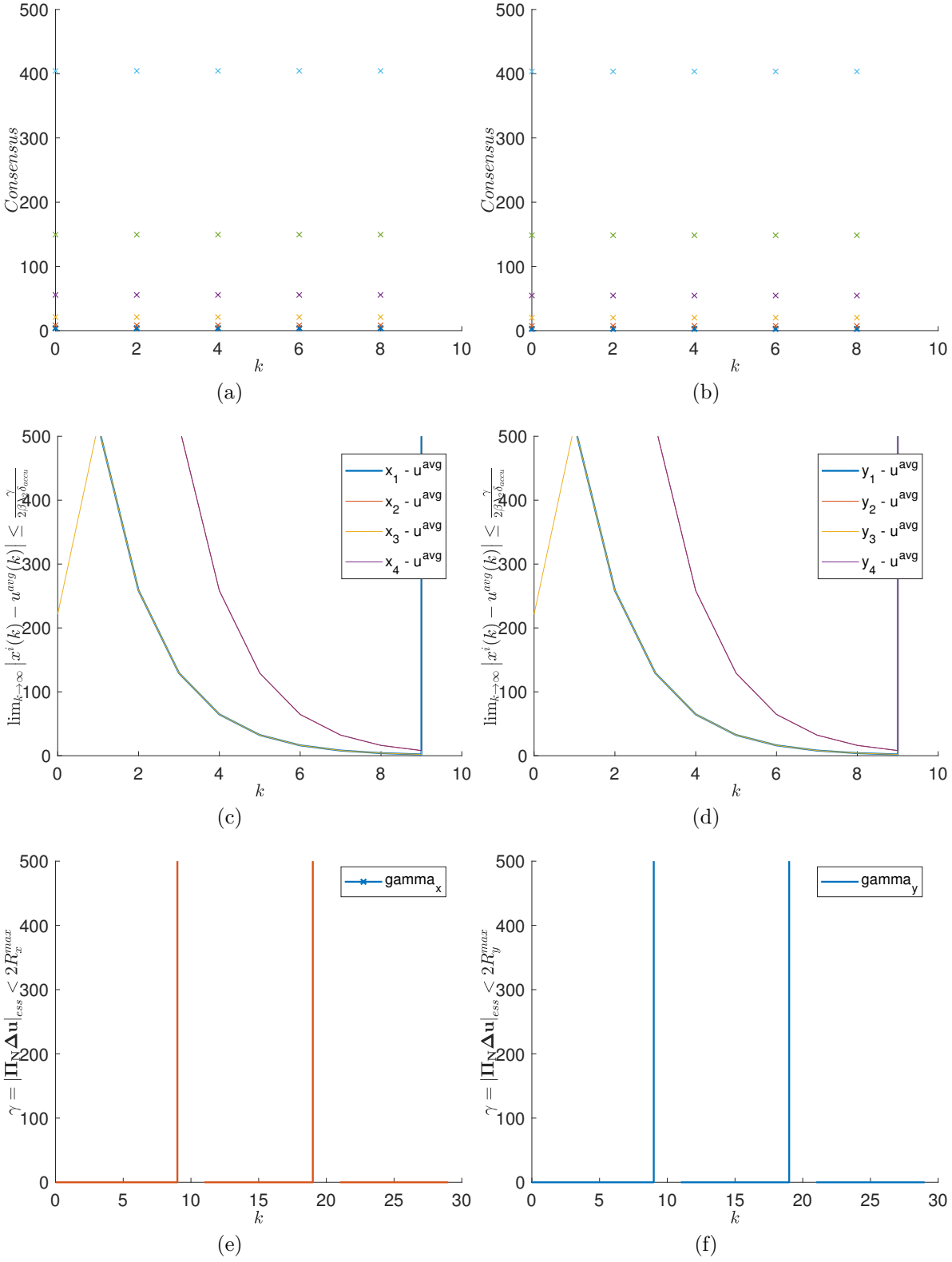


Figura 6.22: *Caso 3, trayectoria tipo 3c. Esta gráfica de simulación se corresponde con la tabla 6.19. El algoritmo diverge.*

En la sección anterior se ha estudiado y analizado la propiedad de robustez del sistema de forma teórica, se hace necesario una simulación sobre un programa más realista para elegir la estrategia de seguimiento adecuada que garantice la total convergencia del algortimo, con especial atención a la forma en que los agentes se distribuyen alrededor del centroide y si rotan o no.

6.2. Simulaciones de percepción y construcción

En esta sección se describen las simulaciones realizadas sobre la plataforma software *ROS-Gazebo*. La primera simulación combina ambas tareas, la reconstrucción volumétrica y el mecanismo de posicionamiento de la cámara, para ilustrar un escenario en el que sólo opera un robot. En la segunda simulación se extiende el método a un entorno multi-robot y se implementa una estrategia que permita reducir el coste computacional y el tiempo de generación del modelo 3D.

6.2.1. Puesta a punto (set-up) de la simulaciones

Las simulaciones están diseñadas para comprobar que cada módulo funciona independientemente de los demás, de forma que si surge un problema se pueda abordar fácilmente o si se desea realizar cambios, resulte en un proceso sencillo.

Los dos casos simulados se explican a continuación:

1. Primero se simula un robot compuesto por un par de cámaras estéreo y el modelo del Conejo de Standford, de forma que el mecanismo de posicionamiento ubique el robot en posiciones arbitrarias. El objetivo de esta simulación es construir y comprobar el correcto funcionamiento del algoritmo de reconstrucción de forma manual.

2. En segundo lugar, se simula un equipo formado por 6 robots y se implementa una estrategia que ubique los robots según un criterio geométrico. En este caso, se dota de movimiento al conejo, y se comprueba que el equipo siga, encierre y reconstruya el modelo 3D del objetivo.

La simulación de la escena de la reconstrucción incluye un objeto colocado en un mundo vacío (Fig. 6.23.) y un sensor de profundidad que proporciona información de cada vista (cámara stéreo). En la figura se muestran dos ventanas, la más grande representa el entorno de *Gazebo*, la otra corresponde al visualizador *Rviz* donde se construye el modelo 3D. Aquí, *Rviz* es una herramienta que ofrece *ROS* para la

visualización de variables y parámetros del entorno de simulación. El modelo utilizado para el objeto está disponible en línea: el conejo de Stanford (Fig. 2.1).

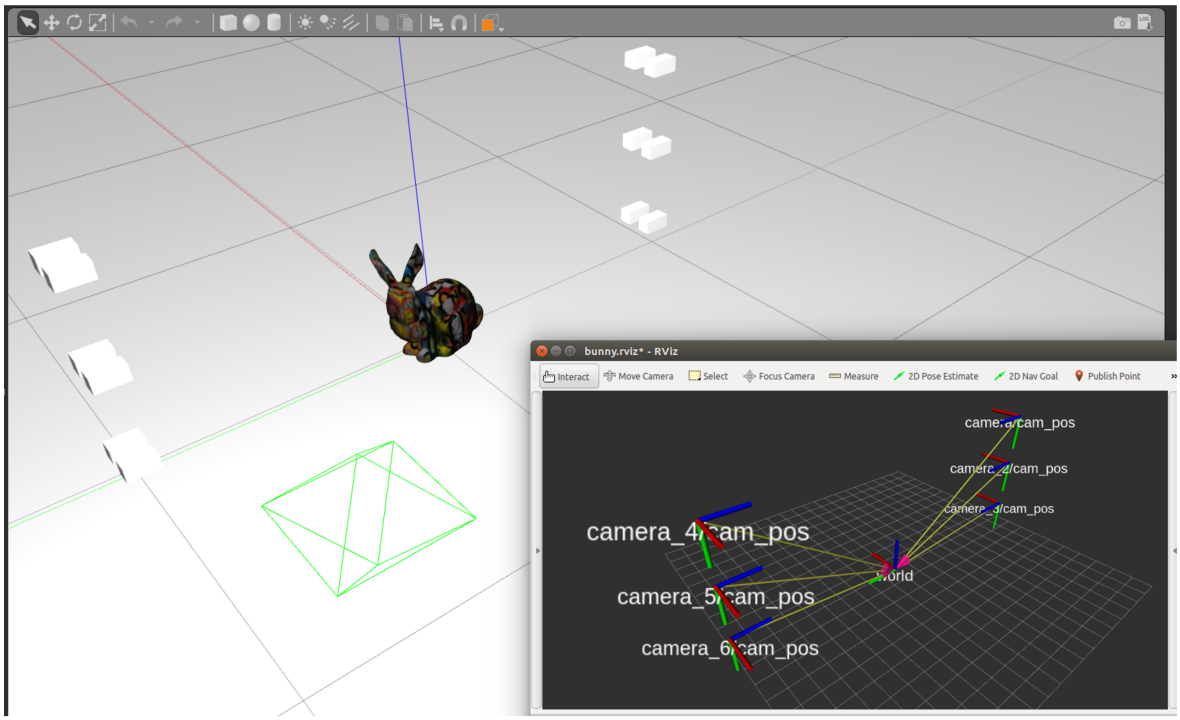


Figura 6.23: *Escenario de reconstrucción de la simulación: Tanto el conejo como los pares de cámaras estéreo son simulados en Gazebo y mostrados por Rviz.*

Alrededor del objeto se genera el sistema robótico. Cada robot, es una cámara estéreo de vuelo libre con 6 DoF. La cámara estéreo es un sensor RGB-D que obtiene una imagen de la situación del escenario. Todas las simulaciones se realizan en Gazebo, en este entorno el procesamiento estéreo se puede realizar mediante *ROS* (*Robot Operating System*). Ambas simulaciones comienzan situando los sensores en una vista predefinida elegida. La salida del sensor es una nube de puntos, que es redirigida por el *Módulo Interfaz del Robot*, a través de varios servicios, al *Módulo de Representación del Mundo*, compuesto por el *Octomap*. El mapa volumétrico y probabilístico se basa en una aproximación realizada por [1]. En este método, cada par de cámaras estéreo comparten la nube de puntos generada, con el objetivo de que se fusionen e integren estos mensajes en el mapa. El planificador de posicionamiento de la cámara se basa en [23], que localiza las cámaras según una formación hexagonal (Fig. 4.2).

Ejemplo de reconstrucción volumétrica En primer lugar, se va a a explicar cómo se implementan en la presente aplicación los conceptos presentados en el Capítulo 2. El objetivo es modelar un objeto 3D utilizando un robot (una cámara estéreo). Para

el proceso de reconstrucción, se utiliza el mapa volumétrico de probabilidades basado en una malla de *vóxels*, implementado como aproximación del *Octomap*, así como el mecanismo de posicionamiento de la cámara, desarrollado a partir del *Módulo Interfaz de ROS* basado en [19]. Se aproxima este mecanismo mediante una implementación basada en *ROS*, en comunicación con los N robots.

En este marco, el objeto es a priori desconocido y limitado espacialmente, ya que no se deforma ni se mueve. Partiendo de un sistema que iteraba a lo largo de un conjunto de vistas para obtener la siguiente mejor posición para colocar la cámara, se elimina el bucle secuencial del algoritmo. Nuestro objetivo es recibir vistas de un algoritmo externo, que no tiene que seleccionar la mejor vista, la estrategia será discutida más adelante. Una vez recibida la ubicación (posición y orientación), el método coloca el sensor en él, tomando una imagen para el mapa volumétrico. Para construir el mapa volumétrico completo del objeto de destino, *Octomap* necesita nubes de puntos como datos de entrada. Estas nubes de puntos se obtienen procesando un conjunto de imágenes tomadas por varias cámaras. Como salida, *Octomap* devuelve una estructura de datos del objeto de destino que se visualiza en *Rviz*. Tener en cuenta que para no perder de vista el objeto en ningún momento, se impone una restricción geométrica para que la orientación de la cámara apunte siempre hacia el centro del objeto.

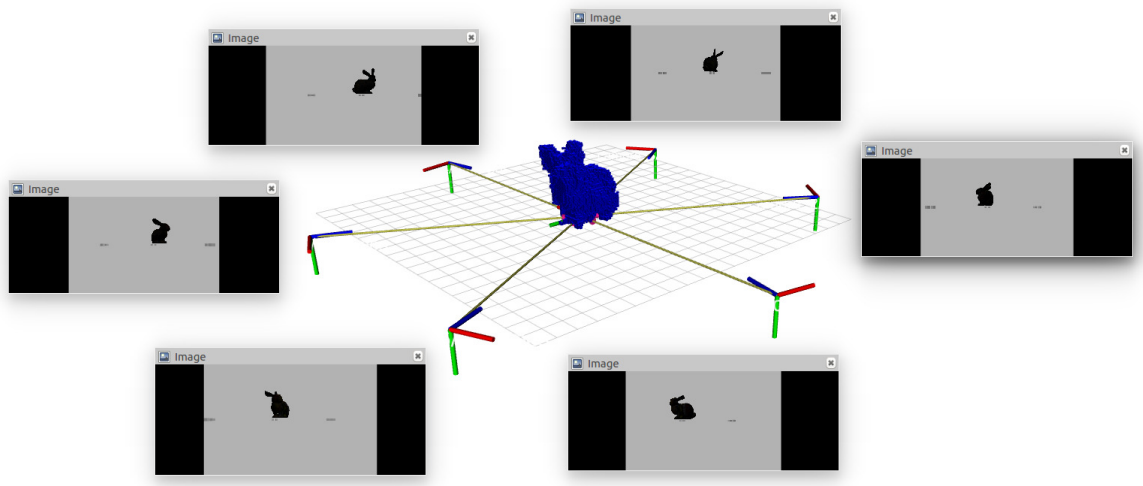
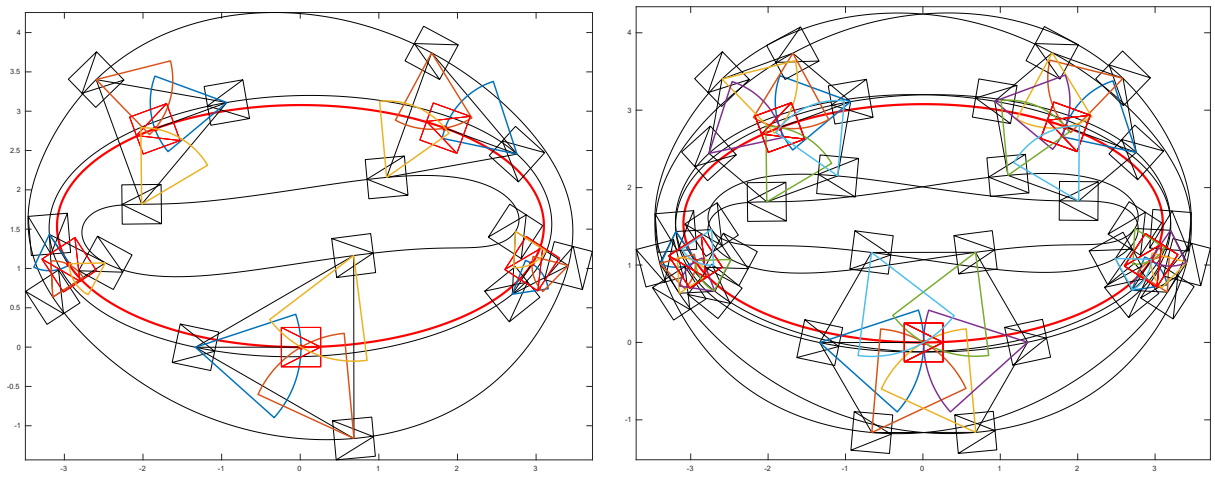


Figura 6.24: *Simulación de la reconstrucción: Se observa el modelo completo del conjunto de datos del conejo de Stanford, donde los vóxels ocupados están pintados de azul. También se visualiza la formación de forma hexagonal multi-cámara. En la tarea de percepción instantánea de un objeto, el campo de visión de cada cámara cubre una porción de la superficie del objeto en cada instante de tiempo. Al fusionar estas observaciones, el objeto se percibe plenamente.*



(a) $N = 3$ (b) $N = 6$

Figura 6.25: *La imagen muestra una simulación que sigue y encierra un objetivo con movimiento elipsoidal. Se muestran dos casos de izquierda a derecha: el número de robots elegidos es $N = 3$ y $N = 6$, respectivamente. La simulación representa el movimiento del objetivo y los robots que lo rodean. Además, el polígono de la formación y las cuñas (FOV) de cada robot también están representados durante algunos instantes de tiempo.*

Ejemplo de un escenario multi-cámara En segundo lugar, se describe la simulación propuesta para un escenario multi-robot, utilizando el algoritmo de planificación basado en [23]. Aunque el número de cámaras que se pueden utilizar en cada ejecución del programa es variable, en esta simulación se considera un equipo formado por $N = 6$ agentes, formando un hexágono. La formación circular se observó en la Fig. 4.2. Para simular cambios en el modelo del conejo, se impone que siga una trayectoria elíptica predefinida. Como explica en el Capítulo 4, se calculan las trayectorias del objeto objetivo y del equipo multi-cámara, así como los valores de d para llevar a cabo la tarea. Una vez calculadas las trayectorias, se realiza un control del seguimiento. Para calcular las posiciones en las que se debe mover, cada par de cámaras necesita estimar su ubicación relativa con respecto al objeto de destino (véase la Fig. 6.24). En la Fig. 6.25 se presenta un ejemplo de la estrategia comentada, con un objeto que sigue una trayectoria elipsoidal.

Generalmente, el comportamiento que presenta la herramienta software es satisfactorio. Algunos de los problemas encontrados han sido que a la hora de seguir la trayectoria del conejo, se ha tenido que ampliar el área de cobertura del *Octomap*. Además, para poner en movimiento el conejo, se ha tenido que implementar un controlador **PID** que regule su velocidad. También, se han encontrado dificultades a la hora de sincronizar la arquitectura basada en *ROS* con la de *Matlab*.

Capítulo 7

Conclusiones y líneas futuras

7.1. Conclusiones

Este capítulo aúna las conclusiones extraídas del presente Trabajo Fin de Máster, así como, nuevos horizontes de aplicación y líneas futuras.

En este trabajo hemos presentado un sistema multi-robot cuyo objetivo es ser capaces de realizar el seguimiento coordinado colaborativo y la construcción del modelo 3D de un objeto deformable que se mueve en el entorno. La realización de estas tareas requiere un conjunto de robots para cubrir toda la superficie del objeto ya que que uno solo tarda demasiado en recorrer todas estas posiciones. Para ello, se presenta una implementación de algoritmos distribuidos en una red de agentes autónomos. El objetivo es distribuir los agentes de forma coordinada alrededor del centroide del objeto. Debido a las observaciones parciales de los objetivos por parte del grupo de agentes, se discuten algunas estrategias de consenso medio dinámicas de tiempo discreto que permiten al grupo de agentes seguir el centroide (promedio) del objeto mediante un conjunto de entradas de referencia. También se estudia y analiza cómo cambia el límite superior del error de seguimiento. Aquí, el límite superior de las entradas de control garantiza la convergencia asintótica del problema del consenso. Por último se desarrolla una arquitectura multicámara basada en el Sistema Operativo del Robot.

En primer lugar, se ha estudiado un algoritmo distribuido de consenso, analizando la caracterización del tamaño de paso y la convergencia del mismo en labores de seguimiento. Además se han propuesto dos estrategias alternativas de percepción del objeto y se ha llevado a cabo el análisis de su convergencia. En las simulaciones llevadas a cabo en *MATLAB*, se demuestra la convergencia asintótica del algoritmo en tareas de seguimiento. Además, se extraen conclusiones en lo que se refiere a diferentes parámetros del algoritmo y su convergencia. Finalemente, se muestra como la cota que se propone limita superiormente el error de seguimiento cometido para cada una

de las estrategias.

En segundo lugar, se ha desarrollado un algoritmo de reconstrucción instantánea basado en [23], el cuál está basado en una aproximación multi-robot. Se ha dividido el método en dos módulos independientes: la reconstrucción volumétrica multi-cámara y la estrategia multi-cámara. También se ha propuesto una aproximación para la conexión entre ambos, lo que garantiza la adaptabilidad de la arquitectura del sistema modular a otras plataformas robóticas. Dado un sólido con un comportamiento dinámico que sigue un camino, el algoritmo de reconstrucción instantánea considera el problema la percepción completa del objeto y la reconstrucción probabilística en un modelo $3D$ de ese objeto en cada instante de tiempo, mediante N cámaras estéreo colocadas a su alrededor. También se ha considerado el tipo de trayectorias que encierran y que siguen el rastro de un objetivo en movimiento con un sistema multi-robot mientras se cumplen las restricciones de movimiento y *FOV*. Se ha elegido una formación circular que encierra el objeto objetivo debido a un criterio de simplicidad y ausencia de oclusiones. La arquitectura propuesta de formación es general y cualquier otra estrategia de formación puede ser fácilmente probada aprovechando el diseño modular de la implementación basada en *ROS*.

Se ha conseguido implementar el sistema multi-robot sobre el simulador *Gazebo*, donde las simulaciones se evalúan visualmente, focalizando la atención sobre si la estrategia adoptada para el acuerdo de las cámaras alrededor del objeto produce el modelo de objeto, sin tener en cuenta la exactitud de la observación. El rendimiento de la estrategia implementada puede ser visualmente evaluado a través de la librería de *ROS* llamada *Rviz*, donde el seguimiento del objeto dinámico llevado a cabo por el sistema multi-cámara y la reconstrucción volumétrica correspondiente del modelo es mostrado.

Finalmente, se ha conseguido implementar en un mismo bloque lo anterior comentado, ver Fig. (1.5) en el Capítulo 1. Se emplea el algoritmo de consenso distribuido para obtener la posición de los agentes alrededor del centroide, estas posiciones se envían a través de un programa a la arquitectura multi-robot, quién es la encargada de situar las cámaras en las posiciones recibidas y procesar la información para crear el modelo de reconstrucción 3D del objeto deformable.

Cabe decir que este trabajo se encuentra enmarcado en un proyecto europeo cuyo principal foco de investigación es el diseño e implementación de sistemas multi-robot

para tareas de percepción de objetos deformables. Más aún, parte de este trabajo se presentó en las jornadas de jóvenes investigadores del *I3A* (Instituto Universitario de Investigación en Ingeniería en Aragón), como se puede ver en [2]. También se ha publicado el artículo [3] en el congreso *ETFA2019* (Emerging Technologies and Factory Automation), celebrado en Zaragoza. En un futuro próximo se espera publicar un artículo en lo concerniente al problema del consenso presentado en este trabajo.

7.2. Líneas futuras

A partir de este trabajo, las líneas de investigación en las que focalizar nuestras investigaciones son muchas.

En cuanto a la parte del algoritmo de consenso, uno de los objetivos de cara al futuro es el desarrollo de una estrategia general, donde el número de agentes sea menor al de objetivos y estos orbiten alrededor del centroide, con una percepción parcial de los objetivos. A parte de esta línea, existen muchas otras, por ejemplo, hasta ahora, se ha trabajado con un protocolo lineal en el que el valor estimado asintóticamente y el real difieren en un error, en un futuro, cabría la posibilidad de emplear un algoritmo distribuido no lineal que nos permitiera obtener una medida precisa del centroide del objeto. Además, el modelo del sistema se puede ir haciendo más complejo, introduciendo retrasos en las comunicaciones o implementado un grafo de comunicaciones variable.

Respecto a las direcciones futuras en el campo de la reconstrucción volumétrica instantánea, la construcción de objetos dinámicos utilizando múltiples robots incluye muchos temas diferentes. Algunos de estos aspectos son compartidos entre las estrategias anteriores, que podrían resumirse como sigue: mejoras en la eficiencia del marco del sistema y evaluación de diferentes estrategias multi-robot.

Capítulo 8

Bibliografía

[1] Jeffrey Delmerico, Stefan Isler, Reza Sabzevari, and Davide Scaramuzza. A comparison of volumetric information gain metrics for active 3d object reconstruction. *Autonomous Robots*, 42(2):197–208, Feb 2018.

[2] E. Hernandez-Murillo„ R. Aragues, and G. Lopez-Nicolas. Volumetric object reconstruction in multi-camera scenarios. *Jornada de Jóvenes Investigadores del I3A*, page vol. 7 (Actas de la VIII Jornada de Jóvenes Investigadores del I3A), 06 Junio 2019.

[3] E. Hernandez-Murillo„ R. Aragues, and G. Lopez-Nicolas. Multi-camera architecture for perception strategies. pages 1799–1804, Sept 2019.

[4] P. Yang, R. A. Freeman, and K. M. Lynch. Multi-agent coordination by decentralized estimation and control. *IEEE Transactions on Automatic Control*, 53(11):2480–2496, Dec 2008.

[5] P. Yang, R. A. Freeman, and K. M. Lynch. Distributed cooperative active sensing using consensus filters. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 405–410, April 2007.

[6] R. Olfati-Saber, J. A. Fax, and R. M. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, Jan 2007.

[7] John N. Tsitsiklis. *Problems in Decentralized Decision Making and Computation.* PhD thesis, Department of EECS, MIT, November 1984.

[8] Wei Ren and Yongcan Cao. *Distributed Coordination of Multi-agent Networks: Emergent Problems, Models, and Issues.* Springer Publishing Company, Incorporated, 2013.

[9] F. Bullo, J. Cortés, and S. Martínez. *Distributed Control of Robotic Networks*. Applied Mathematics Series. Princeton University Press, 2009. Electronically available at http://coordinationbook.info.

[10] Mehran Mesbahi and Magnus Egerstedt. *Graph Theoretic Methods in Multiagent Networks*. Princeton University Press, 2010.

[11] D. Spanos, R. Olfati-saber, and Richard Murray. Dynamic consensus on mobile networks. 01 2005.

[12] R. Olfati-Saber and J. S. Shamma. Consensus filters for sensor networks and distributed sensor fusion. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 6698–6703, Dec 2005.

[13] R. Olfati-Saber and R. M. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, 49(9):1520–1533, Sep. 2004.

[14] R. A. Freeman, P. Yang, and K. M. Lynch. Stability and convergence properties of dynamic. pages 338–343, Dec 2006.

[15] Minghui Zhu and Sonia Martínez. Discrete-time dynamic average consensus, 2009.

[16] S. S. Kia, B. Van Scoy, J. Cortes, R. A. Freeman, K. M. Lynch, and S. Martinez. Tutorial on dynamic average consensus: The problem, its applications, and the algorithms. *IEEE Control Systems Magazine*, 39(3):40–72, June 2019.

[17] William R. Scott, Gerhard Roth, and Jean-François Rivest. View planning for automated three-dimensional object reconstruction and inspection. *ACM Comput. Surv.*, 35(1):64–96, March 2003.

[18] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: an efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, Apr 2013.

[19] S. Isler, R. Sabzevari, J. Delmerico, and D. Scaramuzza. An information gain formulation for active volumetric 3d reconstruction. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3477–3484, May 2016.

[20] J. Fax and Richard Murray. Graph laplacians and stabilization of vehicle formations. *Proc. of the 15th IFAC World Congress*, Enero 2001.

[21] Solmaz Kia, Jorge Cortes, and Sonia Martínez. Dynamic average consensus under limited control authority and privacy requirements. *International Journal of Robust and Nonlinear Control*, 25, April 2014.

[22] Sonia Martínez, Francesco Bullo, Jorge Cortes, and Emilio Frazzoli. On synchronous robotic networks—part i: Models, tasks, and complexity. *Automatic Control, IEEE Transactions on*, 52:2199 – 2213, 01 2008.

[23] G. Lopez-Nicolás, M. Aranda, and Y. Mezouar. Formation of differential-drive vehicles with field-of-view constraints for enclosing a moving target. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 261–266, May 2017.

Anexos

Anexos A

Manual de usuario

En este anexo, se presentan los pasos que se han de seguir para poner en funcionamiento el sistema.

Instalación

%%%
Dependencias
Si está utilizando ROS, la mayoría de las dependencias ya deberían estar instaladas. Los paquetes fueron probados en Ubuntu 14.04 bajo ROS Indigo y Ubuntu 16.04 con ROS Kinetic. El código debe compilar con gcc 4.7+.

ig_active_reconstruction: Eigen
ig_active_reconstruction_octomap: Boost, PCL 1.7+, Eigen, Octomap
ig_active_reconstruction_ros: -
example/flying_gazebo_stereo_cam: Gazebo, stereo_image_proc

Instalación para ROS Kinetic/Ubuntu 16.04

Para instalar todas estas dependencias como paquetes de sistema, ejecute:

```
sudo apt-get install ros-kinetic-gazebo-ros-pkgs ros-kinetic-gazebo-ros-control \
                     ros-kinetic-stereo-image-proc ros-kinetic-octomap libboost-all-dev \
                     libpcl-dev libeigen3-dev
```

Todos los paquetes están escritos para ser compilados usando catkin o catkin_tools, simplemente clona este repositorio en tu espacio de trabajo de catkin:

```
cd catkin_ws/src
git clone -b kinetic https://github.com/uzh-rpg/rpg_ig_active_reconstruction.git
```

Y compila:

```
catkin_make (or catkin build)
```

%%
Running the project in MATLAB

1) Lanzar el script del Caso 1 o 2 (con n = m = 6 agentes) que se quiera simular. Para ello, acceder a la carpeta TFM Enrique Hernandez 2019, los casos con susrespectivas simulaciones están dentro de la carpeta MATLAB Dynamic Average Consensus.

Una vez se ha ejecutado el programa, las posiciones de los agentes se almacenan en los vectores v_x_k y v_y_k, y las posiciones del objeto en v_x_T_avg y v_y_T_avg.

Para ello, añadir en este script, la siguinete línea de código:

```
save variables.mat v_x_T_avg v_y_T_avg v_x_k v_y_k
```

Copiar el fichero de datos variables.mat a la carpeta MATLAB Nodo publicador dentro de MATLAB Dynamic Average Consensus.

2) Abrir de nuevo la carpeta MATLAB Dynamic Average Consensus. Aquí, abrir la carpeta MATLAB Nodo publicador, y abrir el script mainMultirobotConstrainedEnclosing.m.

En este script, cargar las variables guardadas introduciendo la siguiente línea de código:

```
load('posiciones.mat')

línea 617 for it = 1:50:size_vt

línea 619 pose_tx = v_x_T_avg(it,1);
línea 620 pose_tx = v_y_T_avg(it,1);

línea 666 pose.Position.X= v_x_k(it,i);
línea 667 pose.Position.Y= v_y_k(it,i);
```

No ejecutar el programa todavía.

Running the project in ROS

Para empezar se ha de tener en cuenta que se proporciona un archivo del modelo de Gazebo de conejo de Stanford. Por supuesto, también se puede usar cualquier otro modelo de Gazebo y ponerlo delante de la cámara estéreo de Gazebo.

Para usar nuestro modelo, copie la carpeta "bunny" (que contiene el archivo collada, la textura y la definición del modelo de Gazebo) en ~/.gazebo/models, o añada su directorio a su variable de entorno GAZEBO_MODEL_PATH con, por ejemplo:

```
export GAZEBO_MODEL_PATH=$HOME/catkin_ws/src/rpg_ig_active_reconstruction/example/flying_gazebo_stereo_cam/model:$GAZEBO_MODEL_PATH
```

Si todo está compilado, ejecute lo siguiente en cuatro terminales diferentes para iniciar el procedimiento de reconstrucción:

1) roslaunch flying_gazebo_stereo_cam multi_gazebo_stereo_cams.launch

Inicia Gazebo y carga el conejo (si lo pones en la carpeta models), inicia un modelo de Viewspace y un nodo ROS de interfaz para cada cámara, que ofrece sus servicios a otros componentes de ig_active_reconstruction.

2) roslaunch flying_gazebo_stereo_cam multi_flying_stereo_cameras.launch

Lanza los nodos que engendran las cámaras estéreo (Deberías ver dos pequeñas cajas apuntando al origen por cada nodo) en Gazebo. Esto debe iniciarse después de los nodos interfaz de las cámaras.

3) roslaunch ig_active_reconstruction_octomap octomap_world_representation.launch

Lanza un nodo ROS de representación del mundo usando Octomap como contenedor.

4) roslaunch ig_active_reconstruction_ros mover_camara_a.launch

Inicia un nodo de posicionamiento de las cámaras con una sencilla interfaz

de usuario de línea de comandos que le permite iniciar el procedimiento. Para iniciar la demostración, pulse 's' y luego 'Enter' en este terminal.
   No pulsar 's' todavía

Sincronizar MATLAB con ROS

1) Ejecutar MATLAB (script mainMultirobotConstrainedEnclosing.m en la carpeta MATLAB Dynamic Average Consensus) una vez se han seguido los pasos previos.
   Y obserbar la terminal de comandos del programa.
2) En cuanto se observe por la terminal de comandos de MATLAB que se está publicando una posición, acudir al terminal de ROS donde ejecutamos
   la orden roslaunch ig_active_reconstruction_ros mover_camara_a.launch y pulsar 's'. En Rviz se obserbará como se va construyendo el modelo
   del objeto mientras esté estás en movimiento y las cámaras lo rodean.

Anexos B

Ficheros de la arquitectura del sistema sobre ROS

En el siguiente anexo se muestran algunos de los ficheros *.launch* que lanzan a *Gazebo* los diferentes nodos mediante ROS. Además también se incluye el código en *C++* de algunos de los nodos de *ROS* que se han implementado en este trabajo.

El primer fichero representa el código en *XML* del nodo que se subscribe a la posición en la que se han de situar las cámaras. Estas posiciones se calculan a partir de las distintas estrategias comentadas a lo largo del documento. El segundo fichero es el *.launch* que lanzan en *Gazebo* el grupo de cámaras. En este launch se puede apreciar que todos las variables y nodos se pueden englobar en un mismo grupo o espacio de trabajo, etiquetado con un nombre, en este caso el nombre de la cámara. Para identificar el espacio al que pertenecen, se añade un prefijo con el nombre asignado al grupo de trabajo para cada elemento, sean nodos, mensajes, servicios, etc.

```
 1 <?xml version="1.0"?>
 2 <launch>
 3   <node pkg="ig_active_reconstruction_ros" type="mover_camara_a"
   name="mover_camara_a" clear_params="true" output="screen">
 4
 5     <param name="discard_visited" value="true" />
 6     <param name="max_visits" value="-1" />
 7     <param name="cost_weight" value="0" />
 8     <param name="max_calls" value="20" />
 9     <rosparam param="ig_names">[OcclusionAwareIg, UnobservedVoxelIg,
   RearSideVoxelIg, RearSideEntropyIg, ProximityCountIg,
   VasquezGomezAreaFactorIg, AverageEntropyIg]</rosparam>
10       <rosparam param="ig_weights">[0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0]</
   rosparam>
11
12   </node>
13 </launch>
```

```
<?xml version="1.0"?>
<launch>

  <arg name="first_fsc"  default="flying_stereo_cam"/>
  <arg name="second_fsc" default="flying_stereo_cam_2"/>
  <arg name="third_fsc"  default="flying_stereo_cam_3"/>
  <arg name="fourth_fsc" default="flying_stereo_cam_4"/>
  <arg name="fifth_fsc"  default="flying_stereo_cam_5"/>
  <arg name="sixth_fsc" default="flying_stereo_cam_6"/>

  <!-- <node pkg="flying_gazebo_stereo_cam" type="bunny_interface" name="bunny_interface" clear_params="true" output="screen">

    <param name="model_name_bunny" value="bunny" />
    <param name="bunny_frame_name" value="/camera/cam_pos" />
    <param name="world_frame_name_bunny" value="world" />
    <param name="sensor_in_topic_bunny" value="/camera/points2" />
    <param name="sensor_out_name_bunny" value="world/pcl_input" />

  </node> -->

  <node pkg="flying_gazebo_stereo_cam" type="robot_interface" name="robot_interface" clear_params="true" output="screen">

    <param name="model_name" value="$(arg first_fsc)" />
    <param name="camera_frame_name" value="/camera/cam_pos" />
    <param name="world_frame_name" value="world" />
    <param name="sensor_in_topic" value="/camera/points2" />
    <param name="sensor_out_name" value="world/pcl_input" />

  </node>

  <node pkg="flying_gazebo_stereo_cam" type="robot_interface_2" name="robot_interface_2" clear_params="true" output="screen">

    <param name="model_name_2" value="$(arg second_fsc)" />
    <param name="camera_frame_name_2" value="/camera_2/cam_pos" />
    <param name="world_frame_name_2" value="world" />
    <param name="sensor_in_topic_2" value="/camera_2/points2" />
    <param name="sensor_out_name_2" value="world/pcl_input" />

  </node>

    <node pkg="flying_gazebo_stereo_cam" type="robot_interface_3" name="robot_interface_3" clear_params="true" output="screen">

    <param name="model_name_3" value="$(arg third_fsc)" />
    <param name="camera_frame_name_3" value="/camera_3/cam_pos" />
    <param name="world_frame_name_3" value="world" />
    <param name="sensor_in_topic_3" value="/camera_3/points2" />
    <param name="sensor_out_name_3" value="world/pcl_input" />

  </node>
```

```
50
51   <node pkg="flying_gazebo_stereo_cam" type="robot_interface_4"
  name="robot_interface_4" clear_params="true" output="screen">
52
53     <param name="model_name_4" value="$(arg fourth_fsc)" />
54     <param name="camera_frame_name_4" value="/camera_4/cam_pos" />
55     <param name="world_frame_name_4" value="world" />
56     <param name="sensor_in_topic_4" value="/camera_4/points2" />
57     <param name="sensor_out_name_4" value="world/pcl_input" />
58
59   </node>
60
61     <node pkg="flying_gazebo_stereo_cam" type="robot_interface_5"
  name="robot_interface_5" clear_params="true" output="screen">
62
63     <param name="model_name_5" value="$(arg fifth_fsc)" />
64     <param name="camera_frame_name_5" value="/camera_5/cam_pos" />
65     <param name="world_frame_name_5" value="world" />
66     <param name="sensor_in_topic_5" value="/camera_5/points2" />
67     <param name="sensor_out_name_5" value="world/pcl_input" />
68
69   </node>
70
71   <node pkg="flying_gazebo_stereo_cam" type="robot_interface_6"
  name="robot_interface_6" clear_params="true" output="screen">
72
73     <param name="model_name_6" value="$(arg sixth_fsc)" />
74     <param name="camera_frame_name_6" value="/camera_6/cam_pos" />
75     <param name="world_frame_name_6" value="world" />
76     <param name="sensor_in_topic_6" value="/camera_6/points2" />
77     <param name="sensor_out_name_6" value="world/pcl_input" />
78
79   </node>
80
81
82   <node pkg="rviz" type="rviz" name="rviz" clear_params="true"
  output="screen" args="-d $(find flying_gazebo_stereo_cam)/config/
  bunny.rviz"/>
83 </launch>
```

```
<?xml version="1.0"?>
<launch>

  <arg name="first_fsc"  value="flying_stereo_cam"/>
  <arg name="second_fsc" value="flying_stereo_cam_2"/>
  <arg name="third_fsc"  value="flying_stereo_cam_3"/>
  <arg name="fourth_fsc" value="flying_stereo_cam_4"/>
  <arg name="fifth_fsc"  value="flying_stereo_cam_5"/>
  <arg name="sixth_fsc"  value="flying_stereo_cam_6"/>

  <!-- Start gazebo and load world -->
  <param name="/use_sim_time" value="true" />
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find flying_gazebo_stereo_cam)/model/world.sdf"/>
    <arg name="gui" value="true"/>
  </include>

  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find flying_gazebo_stereo_cam)/model/flying_stereo_cam.xacro" />

    <group ns = "camera" >

     <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
       args="-urdf -model $(arg first_fsc)
             -x 0
             -y -1
             -z 0.7
             -R 0
             -P 0
             -Y 1.570796327
             -param /robot_description"/>

     <param name ="stereo_image_proc/camera_frame" value="cam_pos" />

     <!-- stereo image processing node -->
     <include file="$(find flying_gazebo_stereo_cam)/launch/multi_stereo_processing.launch" />

    </group>

   <group ns = "camera_2" >

    <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
      args="-urdf -model $(arg second_fsc)
            -x 0
            -y -1
            -z 0.4
            -R 0
            -P 0
            -Y 1.570796327
            -param /robot_description"/>

```

```
51     <param name ="stereo_image_proc/camera_frame" value="cam_pos" />
52
53     <!-- stereo image processing node -->
54     <include file="$(find flying_gazebo_stereo_cam)/launch/
  multi_stereo_processing.launch" />
55
56   </group>
57
58   <group ns = "camera_3" >
59
60     <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
61       args="-urdf -model $(arg third_fsc)
62             -x 0
63             -y -1
64             -z 0.1
65             -R 0
66             -P 0
67             -Y 1.570796327
68             -param /robot_description"/>
69
70     <param name ="stereo_image_proc/camera_frame" value="cam_pos" />
71
72     <!-- stereo image processing node -->
73     <include file="$(find flying_gazebo_stereo_cam)/launch/
  multi_stereo_processing.launch" />
74
75   </group>
76
77   <group ns = "camera_4" >
78
79    <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
80      args="-urdf -model $(arg fourth_fsc)
81            -x 0
82            -y 1
83            -z 0.7
84            -R 0
85            -P 0
86            -Y -1.570796327
87            -param /robot_description"/>
88
89     <param name ="stereo_image_proc/camera_frame" value="cam_pos" />
90
91     <!-- stereo image processing node -->
92     <include file="$(find flying_gazebo_stereo_cam)/launch/
  multi_stereo_processing.launch" />
93
94   </group>
95
96   <group ns = "camera_5" >
97
98     <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
99       args="-urdf -model $(arg fifth_fsc)
100            -x 0
```

```
101             -y 1
102             -z 0.4
103             -R 0
104             -P 0
105             -Y -1.570796327
106             -param /robot_description"/>
107
108     <param name ="stereo_image_proc/camera_frame" value="cam_pos" />
109
110     <!-- stereo image processing node -->
111     <include file="$(find flying_gazebo_stereo_cam)/launch/
   multi_stereo_processing.launch" />
112
113    </group>
114
115   <group ns = "camera_6" >
116
117    <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
118      args="-urdf -model $(arg sixth_fsc)
119            -x 0
120            -y 1
121            -z 0.1
122            -R 0
123            -P 0
124            -Y -1.570796327
125            -param /robot_description"/>
126
127     <param name ="stereo_image_proc/camera_frame" value="cam_pos" />
128
129     <!-- stereo image processing node -->
130     <include file="$(find flying_gazebo_stereo_cam)/launch/
   multi_stereo_processing.launch" />
131
132   </group>
133
134  <!-- Viewspace module -->
135   <include file="$(find flying_gazebo_stereo_cam)/launch/
   simple_viewspace_module.launch" />
136
137 </launch>
138
```

A continuación, se muestra su implementación en *C++* del fichero principal que ejecuta el bucle principal.

```cpp
#include <iostream>
#include <string>
#include <thread>
#include <chrono>

#include <boost/thread/thread.hpp>
#include <boost/chrono/include.hpp>

#include "geometry_msgs/Pose.h"
#include "movements/geometry_pose.h"
#include "std_msgs/String.h"

#include <ros/ros.h>
#include <ig_active_reconstruction/mover_camara_a.hpp>
#include <ig_active_reconstruction/weighted_linear_utility.hpp>
#include <ig_active_reconstruction/max_calls_termination_criteria.hpp>

#include "ig_active_reconstruction_ros/param_loader.hpp"
#include "ig_active_reconstruction_ros/robot_ros_client_ci.hpp"
#include "ig_active_reconstruction_ros/views_ros_client_ci.hpp"
#include "ig_active_reconstruction_ros/
  world_representation_ros_client_ci.hpp"
#include "ig_active_reconstruction/view.hpp"



int main(int argc, char **argv)
{
  ros::init(argc, argv, "mover_camara_a");
  ros::NodeHandle nh;

  namespace iar = ig_active_reconstruction;

  std::string gui_info = "\n\n\nBASIC VIEW PLANNER SIMPLE UI
  \n*******************************\n\n";
  std::cout<<gui_info;

  // load parameter configuration

    // ..................................................................................

  // for the view planner:
  iar::BasicViewPlanner::Config bvp_config;
  ros_tools::getParam( bvp_config.discard_visited, "discard_visited",
  false );
  ros_tools::getParam( bvp_config.max_visits, "max_visits", -1 );

  // for the utility calculator
  double cost_weight;
  ros_tools::getParam( cost_weight, "cost_weight", 1.0 );
  std::vector<std::string> ig_names;
  std::vector<double> ig_weights;
  ros_tools::getParamIfAvailableSilent( ig_names, "ig_names" );
```

```
50   ros_tools::getParamIfAvailableSilent( ig_weights, "ig_weights" );
51
52   // for the termination critera
53   unsigned int max_calls;
54   ros_tools::getParam<unsigned int, int>( max_calls, "max_calls", 20);
55
56
57
58   // only the view planner resides here
59
     // ..........................................................................
60   iar::BasicViewPlanner view_planner(bvp_config);
61
62   // for the choice of the camera
63
     // ..........................................................................
64   std::string model_name;
65 //  std::string model_name_bunny;
66   char a;
67   std::string gui_info_cam_model = "\n\n\nSeleccione un método:
   \n*******************************\n- 'S' (Start). \n\n";
68   std::cout<<gui_info_cam_model;
69   std::cin>>a;
70
71   switch (a)
72     {
73     case 'S':
74         std::cout << "\n\n\nStarting ... \n\n";
75
76         break;
77     };
78
79   /*switch (a)
80   {
81   case 'A':
82      model_name = "flying_stereo_cam";
83
84     break;
85   case 'B':
86      model_name = "flying_stereo_cam_2";
87
88     break;
89   case 'C':
90      model_name = "flying_stereo_cam_3";
91
92     break;
93   case 'D':
94      model_name = "flying_stereo_cam_4";
95
96     break;
97   case 'E':
98      model_name = "flying_stereo_cam_5";
99
```

```
100     break;
101   case 'F':
102      model_name = "flying_stereo_cam_6";
103 
104     break;
105   };*/
106 
107   ros::spinOnce();
108 //  model_name_bunny = view_planner.bunny_name;
109   model_name = view_planner.nombre_camara;
110 //  std::string gui_info_bunny_model = "\n\n\nEl objetivo a mover es:
    \n*******************************\n\n";
111 //  std::cout << gui_info_bunny_model << model_name_bunny <<" \n\n";
112   std::string gui_info_cam_model_2 = "\n\n\nLa cámara a mover es:
    \n*******************************\n\n";
113   std::cout << gui_info_cam_model_2 << model_name <<" \n\n";
114   //std::cout << gui_info_cam_model_2 << view_planner.nombre_camara <<" \n
    \n";
115 
116   // robot, viewspace module and world representation are external
117 
      // ..........................................................................
118 //  boost::shared_ptr<iar::robot::CommunicationInterface> bunny_comm =
    boost::make_shared<iar::robot::RosClientCI>(nh,model_name_bunny);
119   boost::shared_ptr<iar::robot::CommunicationInterface> robot_comm =
    boost::make_shared<iar::robot::RosClientCI>(nh,model_name);
120   boost::shared_ptr<iar::views::CommunicationInterface> views_comm =
    boost::make_shared<iar::views::RosClientCI>(nh);
121   boost::shared_ptr<iar::world_representation::CommunicationInterface>
    world_comm = boost::make_shared<iar::world_representation::RosClientCI>
    (nh);
122 
123 //  view_planner.setRobotCommUnit(bunny_comm);
124   view_planner.setRobotCommUnit(robot_comm);
125   view_planner.setViewsCommUnit(views_comm);
126   view_planner.setWorldCommUnit(world_comm);
127 
128 
129   // want to use the weighted linear utility calculator, which directly
    interacts with world and robot comms too
130 
      // ..........................................................................
131   boost::shared_ptr<iar::WeightedLinearUtility> utility_calculator =
    boost::make_shared<iar::WeightedLinearUtility>(cost_weight);
132   utility_calculator->setRobotCommUnit(robot_comm);
133   utility_calculator->setWorldCommUnit(world_comm);
134 
135   for(unsigned int i=0;i<ig_names.size() && i<ig_weights.size(); ++i)
136   {
137     std::cout<<"\nUsing information gain '"<<ig_names[i]<<"' with weight
    '"<<ig_weights[i]<<"'.";
138     utility_calculator->useInformationGain(ig_names[i],ig_weights[i]);
139   }
```

```
140
141   view_planner.setUtility(utility_calculator);
142
143
144   // using a simple max. number of calls termination critera
145
      // ...........................................................................
146   boost::shared_ptr<iar::GoalEvaluationModule> termination_criteria =
    boost::make_shared<iar::MaxCallsTerminationCriteria>(max_calls);
147
148   view_planner.setGoalEvaluationModule(termination_criteria);
149
150
151
152
153   // Simple command line user interface.
154
      // ...........................................................................
155   std::function<void()> status_readout = [&view_planner]()
156   {
157      iar::BasicViewPlanner::Status status = view_planner.status();
158
159   };
160
161   std::thread status_reading_thread(status_readout);
162
163   ROS_INFO("Basic View Planner was successfully setup. As soon as other
    modules are running, we're ready to go.");
164
165
166
167
168   view_planner.initViewSpace();
169
170   while(ros::ok())
171   {
172
173     std::string model_name;
174 //  std::string model_name_bunny;
175     //ros::spinOnce();
176     //sleep(10);
177
178 //    std::cout<<"\n\nLa nueva posicion del objetivo es: \n\n"<<
    view_planner.bunny_position;
179 //  std::cout<<"\n\nMoving...\n\n";
180 //  view_planner.movebunny();
181     std::cout<<"\n\nLa nueva posicion recibida es: \n\n"<<
    view_planner.posicion;
182     std::cout<<"\n\nMoving...\n\n";
183     view_planner.main();
184
185     /*std::string gui_info_cam_model = "\n\n\nSeleccione un método:
    \n*******************************\n- 'N' (Next View). \n\n";
```

```
186     std::cout<<gui_info_cam_model;
187     std::cin>>a;
188 
189     switch (a)
190       {
191       case 'N':
192 
193           std::cout << "\n\n\nNext View ... \n\n";
194 
195         break;
196       };*/
197 
198     ros::spinOnce();
199 //    model_name_bunny = view_planner.bunny_name;
200 //    std::cout<<"\n\nEl objetivo a mover es: \n\n"<< model_name_bunny <<
    " \n\n";
201     model_name = view_planner.nombre_camara;
202     std::cout<<"\n\nEl nombre de la cámara a mover es: \n\n"<< model_name
    << " \n\n";
203 
204 //    boost::shared_ptr<iar::robot::CommunicationInterface> bunny_comm =
    boost::make_shared<iar::robot::RosClientCI>(nh,model_name_bunny);
205     boost::shared_ptr<iar::robot::CommunicationInterface> robot_comm =
    boost::make_shared<iar::robot::RosClientCI>(nh,model_name);
206     view_planner.setRobotCommUnit(robot_comm);
207 
208   }
209 
210   status_reading_thread.join();
211 
212   return 0;
213 }
214 
215 
```

Anexos C

Ficheros de MATLAB

En concreto, en este anexo, se muestra el código del nodo que publica las posiciones calculadas por el algoritmo de consenso en un tópico.

```
%mainMultirobotConstrainedEnclosing.m
%G. Lopez-Nicolas. University Zaragoza. 2019
%
%This implementation addresses the problem of enclosing and tracking a
%target with multiple robots. We consider a team of unicycle robots with a
%standard camera on board. The robots must maintain the desired enclosing
%formation while dealing with their nonholonomic motion constraints. The
%reference formation trajectories must also guarantee permanent visibility
%of the target by overcoming the limited field of view of the cameras.
%Several motion strategies are implemented to address this task. These are
%based on maintaining the formation shape with variable size
%(distance-based) or, alternatively, on maintaining the size of the
%formation with flexible shape (theta-based).
%
%Related paper:
%Title: Adaptive multirobot formation to enclose and track a target with
%motion and visibility constraints
%Authors: Gonzalo Lopez-Nicolas, Miguel Aranda and Youcef Mezouar

%Main program: mainMultirobotConstrainedEnclosing.m
%Required files:
% betweenPi.m
% computeBi.m
% computeDi.m
% computeThetai.m
% drawRobots.m
% generateTrajectory.m
% optimizeACd.m
% optimizeACtheta.m

%Notation: In general 'v_xxx' refers to one-dimensional vector of a
%variable xxx evolving with time. A matrix of several variables evolving
%with time would be 'm_xxx'

clear
close all

rosinit;
masterHost = 'localhost';
node_1 = robotics.ros.Node('View_Planner', masterHost);
node_2 = robotics.ros.Node('Visualizer', masterHost);

% Create a publisher and subscriber for the '/pose' topic
posePub = robotics.ros.Publisher(node_1,'/posicion','geometry_msgs/Pose');
%pause(1);
posePubmsg = rosmessage(posePub);
poseSub = robotics.ros.Subscriber(node_2,'/posicion');

% Create publishers and subscribers for the '/nombre_camara' topic
namePub = robotics.ros.Publisher(node_1,'/nombre_camara','std_msgs/String');
%pause(1);
namePubmsg = rosmessage(namePub);
nameSub = robotics.ros.Subscriber(node_2,'/nombre_camara');

% Create a publisher and subscriber for the '/target_position' topic
%pose_tPub = robotics.ros.Publisher(node_1,'/target_position','geometry_msgs/
Pose');
%pause(1);
%pose_tPubmsg = rosmessage(pose_tPub);
%pose_TSub = robotics.ros.Subscriber(node_2,'/target_position');

% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % %
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % %
%
```

```
% Create a publisher and subscriber for the '/target_velocity' topic
% vel_tPub = robotics.ros.Publisher(node_1,'/target_velocity','geometry_msgs/
Twist');
% %pause(1);
% vel_tPubmsg = rosmessage(vel_tPub);
% vel_TSub = robotics.ros.Subscriber(node_2,'/target_velocity');
%
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % %
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % %

gazebo = ExampleHelperGazeboCommunicator;
models = getSpawnedModels(gazebo)
bunny = ExampleHelperGazeboSpawnedModel('bunny',gazebo)
flying_stereo_cam = ExampleHelperGazeboSpawnedModel('flying_stereo_cam',gazebo)
flying_stereo_cam_2 = ExampleHelperGazeboSpawnedModel('flying_stereo_cam_2',gazebo)
flying_stereo_cam_3 = ExampleHelperGazeboSpawnedModel('flying_stereo_cam_3',gazebo)
flying_stereo_cam_4 = ExampleHelperGazeboSpawnedModel('flying_stereo_cam_4',gazebo)
flying_stereo_cam_5 = ExampleHelperGazeboSpawnedModel('flying_stereo_cam_5',gazebo)
flying_stereo_cam_6 = ExampleHelperGazeboSpawnedModel('flying_stereo_cam_6',gazebo)

altura= 0.1905;
%Selection of target trajectory
% selection= 'e'; %Ellipse
% selection= 's'; %Sinuisoid
% selection= '8'; %Eight
% selection= 'p'; %Spiral
 selection= 'r'; %Rectilinear
% selection= 'c'; %Circle
% selection= 'c2';%Circle small


%Selection of strategy to comply with motion and FOV constraints:
% d-based: Modify formation scale/size (distance of robots to target)
% theta-based: Modify formation shape (location angle of robots with respec to
target)
strategy= 'd';
% strategy= 'theta';


%Selection of method: Each strategy implements different alternatives
% method = 0 %Constant prescribed formation. No strategy is applied so
% constraints compliance is not guaranteed.
% method = 1 %Method that defines a constant formation in order to obey
% motion and FOV constraints. Strategy 'd' selects a constant value of d
% (v_dr), normally different than the prescribed one (d0) with the
% prescribed thetar0. Strategy 'theta' selects constant values of theta
% (m_thetar), normally different than the prescribed one (m_thetar0) with the
% prescribed scale dr0.
% method = 2 %Method that defines a variable formation in order to obey
% motion and FOV constraints. Strategy 'd' defines the evolution of d
% (v_dr) with the prescribed thetar0. Strategy 'theta' sdefines the
% evolution of theta (m_thetar) with the prescribed scale dr0.
%
 method = 0; %No strategy
% method = 1; %Design constant values (d or theta) to obey constraints
%method = 2; %Design variable values (d or theta) to obey constraints

%In method 2, the optimization is performed. You can choose the previously
%computed paramenters (optimize=0) for fast execution of the examples, or
%perform the full optimization, which may be slow (optimize=1)
optimize = 0;
% optimize = 1;
```

```
%Several figures show the results of the simulation, if animate=1 an
%animation of the formation of robots in navigation is also shown
animate = 0;
% animate = 1;


disp(['Selected trajectory: ', selection]);
disp(['Selected strategy: ', strategy, ', with method ', num2str(method), ', and
optimization = ', num2str(optimize)]);

%-------------------------------------------
%Generation of the target trajectory
delta=0.1; %Step time in each iteration
[t,v_vt,v_wt, Tmax]=generateTrajectory(delta, selection);
nt=length(t);

%Target motion computation
v_fit=cumsum(v_wt.*delta);
v_dxt=v_vt.*delta.*cos(v_fit);
v_dyt=v_vt.*delta.*sin(v_fit);

v_xt=cumsum(v_dxt);
v_yt=cumsum(v_dyt);

%-------------------------------------------
%Definition of the prescribed formation
%Number N of observing cameras (enclosing robots)
prompt = 'Number of cameras: ';
x = input(prompt);
N= x
thetar0=-pi:2*pi/N:pi-2*pi/N;

%Angle of the camera field of view (FOV), with limits (-betaMax,betaMax)
betaMax= 30*pi/180;

%-------------------------------------------
%Data preparation
if strcmp(strategy,'theta')
    m_thetar0=repmat(thetar0,nt,1);
    m_v_vt=repmat(v_vt,1,N);
    m_v_wt=repmat(v_wt,1,N);

    %Prescribed formation scale (distance of robots to target)
    switch selection
        case '8' %Eight
            dr0 = 10;
        case 's' %Sinuisoid
            dr0 = 10;
        otherwise
            dr0 = 0.5;
%             dr0 = 5;
    end

    v_dr0=dr0*ones(nt,1);
    m_dr0=repmat(v_dr0,1,N);
    %%%%%%%%%%%%%%%%%%%%dmir
    xr0=dr0*cos(thetar0);
    yr0=dr0*sin(thetar0);
    m_xr0=v_dr0*cos(thetar0);
    m_yr0=v_dr0*sin(thetar0);
end

if strcmp(strategy,'d')
    v_betaMin=-betaMax*ones(nt,1);
    [m_dwc,v_dwcMint]=computeDi(delta,t,v_betaMin,betaMax,v_vt,v_wt);
    m_dwc=abs(m_dwc);
```

```
    dmin=min(min(v_dwcMint));
    dwmin=fix(dmin*100)/100;%Round off
    kLMax=1/dmin;
    v_kLMax=kLMax*ones(nt,1);
end

%-----------------------------------------
%Application of the selected strategy and method

if strcmp(strategy,'d')

    %Formation scale dr0 (distance of robots to target)
    if method==0, %d constant, without strategy.
        switch selection
            case '8' %Eight
                dr0 = 10;
            case 's' %Sinuisoid
                dr0 = 10;
            otherwise
                dr0 = 0.5;
                %dr0 = 5;
        end;

        %Check FOV constraint with initial configuration
        %[m_betari0, m_complyBmaxr0] = computeBi(delta,t,dr0*ones
(nt,1),titar0,v_vt,v_wt,betaMax);
    else %maximum constant d that obey constraints
        if dmin==Inf,
            dr0 = 0.5;
        else
            dr0= dmin ;
        end
    end

    v_dwcMin=dmin*ones(nt,1);

    v_dr0=dr0*ones(nt,1);
    xr0=dr0*cos(thetar0);
    yr0=dr0*sin(thetar0);
    m_xr0=v_dr0*cos(thetar0);
    m_yr0=v_dr0*sin(thetar0);

    v_dr=v_dr0;
    m_xr=m_xr0;
    m_yr=m_yr0;

    v_kr0=1./v_dr;
    v_kr=v_kr0;


    if method==2 %maximize variable d that obey constraints
        m_kwc=1./m_dwc;
        areakMax=sum(1/dmin-m_kwc);
        v_kLMax=1./v_dwcMin;

        switch selection %Optimization parameters for d-strategy
            case 'e' % 'e'=ellipse
                if optimize == 0
                    vA = 0.046*delta;
                    vC = 9.1770;
                else
                    vA = [0.0005,0,3];
                    vC = [1,0,20];
                end
            case 's'
                if optimize == 0
```

```
                    vA = 0.045*delta;
                    vC = 7.52;
                else
                    vA = [0.0005,0,3];
                    vC = [1,0,20];
                end
            case 'p'
                if optimize == 0
                    vA = 0.025*delta;
                    vC = 14.176;
                else
                    vA = [0.0005,0,3];
                    vC = [1,0,20];
                end
            case '8'
                if optimize == 0
                    vA = 0.026*delta;
                    vC = 12.4416;
                else
                    vA = [0.0005,0,3];
                    vC = [1,0,20];
                end
            case 'r'
                disp('For this trajectory, you can choose any d. No need to
optimize');
                return
            case 'c'
                disp('For this trajectory, constant d of method 1 is the maximum
value. No need to optimize');
                return
            case 'c2'
                disp('For this trajectory, constant d of method 1 is the maximum
value. No need to optimize');
                return
            otherwise
                if optimize == 0
                    vA = 0.0146*delta;
                    vC = 9.1770;
                else
                    vA = [0.0005,0,3];
                    vC = [1,0,20];
                end
        end

        [A,C,krCost,kr] = optimizeACd
(vA,vC,v_kLMax,dmin,N,nt,delta,t,thetar0,v_vt,v_wt,betaMax);
        areak100=100*krCost/areakMax;%Percentage of covered area with respect the
limit of the constraint (100%)

        v_dr=1./kr;
    end %if method==2

    v_kr=1./v_dr;
    m_xr=v_dr*cos(thetar0);
    m_yr=v_dr*sin(thetar0);

    %Check FOV constraint with obtained result
    [m_betari, m_complyBmaxr] = computeBi
(delta,t,v_dr,thetar0,v_vt,v_wt,betaMax,0);
end %if strategy == 'd',


if strcmp(strategy,'theta')

    if method==0, %Prescribed constant theta, without strategy.
        thetar=thetar0;
```

```
        m_xr=m_xr0;
        m_yr=m_yr0;
        m_thetar=repmat(thetar,nt,1);
    end

    %Check FOV constraint with initial configuration
    [m_betari0, m_complyBmaxr0] = computeBi
(delta,t,v_dr0,m_thetar0,v_vt,v_wt,betaMax,0);
    m_complyBmax0=repmat( sum(m_complyBmaxr0)==nt , nt,1);

    %Limits for theta to obey FOV constraints
    [m_thetarMin, m_thetarMax] = computeThetai
(betaMax,dr0,m_thetar0,m_v_vt,m_v_wt,0);

    if method==1,
        %Look for constant values of theta to obey constraints
        %It may exist solution or not
        %Escoger Tita CONSTANTE que cumpla para todo t (suponiendo que
        %exista. Por ejemplo trayectoria '0')
        m_thetarMaxT = repmat( min(m_thetarMax), nt, 1);
        m_thetarMinT = repmat( max(m_thetarMin), nt, 1);

        conditionCloser = abs(betweenPi(m_thetarMinT - m_thetar0)) < abs(betweenPi
(m_thetarMaxT - m_thetar0));
        m_thetarProx = m_thetarMinT .* conditionCloser + m_thetarMaxT .*
(~conditionCloser);

        %Do not modify those thetar0 that already obey FOV constraint
        m_thetar = m_thetar0 .* m_complyBmax0 + m_thetarProx .* (~m_complyBmax0);

        m_xr=dr0*cos(m_thetar);
        m_yr=dr0*sin(m_thetar);

    end %method==1


    if method==2
        %Find variable theta (closer as possible to thetar0) that obey constraints

        m_thetar = m_thetar0;
        m_thetarMaxT = repmat( min(m_thetarMax), nt, 1);
        m_thetarMinT = repmat( max(m_thetarMin), nt, 1);


        vA=zeros(1,N);
        vC=zeros(1,N);
        switch selection
            %Optimization parameters for theta-strategy (delta=0.1)
            case 'e' %Ellipse
                if optimize == 0
                    vA([1,4,5])=[0.0105, 0.0091, 0.015];
                    vC([1,4,5]) =[13, 15.28, 9.68];
                else
                    vAi=[0.001, 0.002, 0.02];
                    vCi=[1, 8, 17];
                end
            case 's' %Sinusoid
                if optimize == 0
                    vA = [0.0088, 0.0128, 0.0082, 0.0082, 0.0128];
                    vC = [14.44, 9.48, 14.68, 14.68, 9.48];
                else
                    vAi=[0.001, 0.002, 0.02];
                    vCi=[1, 8, 17];
                end
            case 'p' %Spiral
                if optimize == 0
```

```
                    vA([1,4,5]) = [0.0113, 0.0147, 0.01];
                    vC([1,4,5]) = [4.8, 5.04, 6.2];
                else
                    vAi=[0.001, 0.002, 0.02];
                    vCi=[1, 4, 7];
                end
            case '8' %Eight
                if optimize == 0
                    vA = [0.0051, 0.0077, 0.0068, 0.0058, 0.0077];
                    vC = [16.92, 12.36, 17.08, 17.5200, 12.36];
                else
                    vAi=[0.001, 0.002, 0.01];
                    vCi=[2, 10, 20];
                end
            case 'r'
                disp('For this trajectory, you can choose theta of method 1. No
need to optimize');
                return
            case 'c'
                disp('For this trajectory, you can choose theta of method 1. No
need to optimize');
                return
            case 'c2'
                disp('For this trajectory, you can choose theta of method 1. No
need to optimize');
                return
            otherwise
                if optimize == 0
                    vA = [];
                    vC = [];
                else
                    vAi=[];
                    vCi=[];
                end
        end

        if optimize == 1
            %Different parameters can be used for each robot
            %For simplicity we use the same parameters for all of them:
            vA = repmat(vAi',1,N);
            vC = repmat(vCi',1,N);
        end

        [A,C,v_thetarCost,m_thetar]=optimizeACtheta
(vA,vC,m_complyBmax0,dr0,N,nt,delta,t,m_thetar0,v_vt,v_wt,betaMax);

        m_xr=dr0*cos(m_thetar);
        m_yr=dr0*sin(m_thetar);
    end %method==2

    %Check FOV constraint with obtained result
    [m_betari, m_complyBmax] = computeBi
(delta,t,v_dr0,m_thetar,v_vt,v_wt,betaMax,0);

end %if strategy == 'd',


%-------------------------------------------------------------------------
%Compute reference trajectories with the resultant parameters of the
%selected strategy
m_v_xt=repmat(v_xt,1,N);
m_v_yt=repmat(v_yt,1,N);
m_v_fit=repmat(v_fit,1,N);

if strcmp(strategy,'d')
    m_v_dr=repmat(v_dr,1,N);
```

```
    m_thetar0=repmat(thetar0,nt,1);
    m_v_vt=repmat(v_vt,1,N);
    m_v_wt=repmat(v_wt,1,N);
else %if strategy == 'theta',
    m_v_dr=repmat(v_dr0,1,N);
end

v_xr= cos(m_v_fit).*m_xr - sin(m_v_fit).*m_yr + m_v_xt;
v_yr= sin(m_v_fit).*m_xr + cos(m_v_fit).*m_yr + m_v_yt;

fsc= m_v_vt.*sin(m_v_fit) + m_v_dr.*cos(m_v_fit+m_thetar0).*m_v_wt;
fcs= m_v_vt.*cos(m_v_fit) - m_v_dr.*sin(m_v_fit+m_thetar0).*m_v_wt;

m_v_dvt=diff(m_v_vt./delta);
m_v_dvt=[m_v_dvt; m_v_dvt(end,:)];
m_v_dwt=diff(m_v_wt./delta);
m_v_dwt=[m_v_dwt; m_v_dwt(end,:)];

if strcmp(strategy,'d')
    m_dthetar0=zeros(nt, N); %Constant theta
else %if strategy == 'theta',
    m_thetarS=zeros(nt,N);
    for iis=1:Nv_fir(it,i) + betar0(1,i) - pi/2;
        m_thetarS(:,iis)= smooth(m_thetar(:,iis),7);
    end
    m_dthetar= diff(m_thetarS./delta);
    m_dthetar=[m_dthetar; m_dthetar(end,:)];
end

for iis=1:N
    m_v_dr(:,iis)= smooth(m_v_dr(:,iis),7);
end
m_ddr= diff(m_v_dr./delta);
m_ddr=[m_ddr; m_ddr(end,:)];

for iis=1:N
    m_ddr(:,iis)= smooth(m_ddr(:,iis),7);
end
m_dddr= diff(m_ddr./delta);
m_dddr=[m_dddr; m_dddr(end,:)];

if strcmp(strategy,'d'),
    v_dxr=fcs+m_ddr.*cos(m_v_fit+m_thetar0);
    v_dyr=fsc+ m_ddr.*sin(m_v_fit+m_thetar0);
else %if strategy == 'theta',
    v_dxr=fcs+m_ddr.*cos(m_v_fit+m_thetar) - m_v_dr.*sin(m_v_fit
+m_thetar).*m_dthetar;
    v_dyr=fsc+ m_ddr.*sin(m_v_fit+m_thetar) + m_v_dr.*cos(m_v_fit
+m_thetar).*m_dthetar;
end

v_fir=atan2( v_dyr , v_dxr);
v_fir=unwrap(v_fir);

v_vr= sqrt(v_dxr.*v_dxr+v_dyr.*v_dyr);

v_wr=diff(v_fir./delta);
v_wr=[v_wr; v_wr(end,:)];

%-------------------------------------------------------------------------
%Some computations just to plot data in the figures
%FoV angle beta in the initial reference formation
bt0=[-xr0/dr0; -yr0/dr0; zeros(1,N)];
bu0=[cos(zeros(1,N)); sin(zeros(1,N)); zeros(1,N)];
btmp0=cross(bu0,bt0);
betar0=atan2(btmp0(3,:)', diag(bu0'*bt0) )';
```

```
if strcmp(strategy,'d')
    m_betar0=repmat(betar0,nt,1);
    betar=m_betari+m_betar0;
else %if strategy == 'theta',
    %FoV angle beta in the current reference formation with time
    betad=sqrt((m_v_xt-v_xr).^2 + (m_v_yt-v_yr).^2); %==dr0
    bt=[ reshape((m_v_xt-v_xr)./betad,nt*N,1) , reshape((m_v_yt-v_yr)./
betad,nt*N,1) , zeros(nt*N,1)]';
    bu=[reshape(cos(v_fir),nt*N,1), reshape(sin(v_fir),nt*N,1), zeros(nt*N,1)]';
    btmp=cross(bu,bt);

    %To avoid memory problems:
    dbb=zeros(nt*N,1);
    for bb=1:size(bt,2),
        dbb(bb)= (bu(:,bb)'*bt(:,bb));
    end

    betar=atan2(btmp(3,:)', dbb );
    betar=reshape(betar,nt,N);
    betar=unwrap(betar);
    betar=betweenPi(betar);

    m_betar0=repmat(betar0,nt,1);
    beta= betar - m_betar0;
    beta=betweenPi(beta);
end

%--------------------------------------------------------------------------
%Main loop of the formation evolution

%Initial formation (1 unit behind the reference)
x=v_xr(1,:) - 1;
y=v_yr(2,:);
fi=v_fir(1,:);

%Arrays to save data
v_x=zeros(nt,N);
v_y=zeros(nt,N);
v_fi=zeros(nt,N);

v_xe=zeros(nt,N);
v_ye=zeros(nt,N);
v_fie=zeros(nt,N);
v_roe=zeros(nt,N);
v_alfe=zeros(nt,N);
v_chie=zeros(nt,N);

v_v=zeros(nt,N);
v_w=zeros(nt,N);

for it=1:nt,

    %Current robot locations
    v_x(it,:)=x;
    v_y(it,:)=y;
    v_fi(it,:)=fi;


    %Tracking error relative to reference trajectory
    %In Cartesian coordinates
    xetr= x-v_xr(it,:);
    yetr= y-v_yr(it,:);
    xe= cos(v_fir(it,:)) .* xetr + sin(v_fir(it,:)) .* yetr ;
    ye=-sin(v_fir(it,:)) .* xetr + cos(v_fir(it,:)) .* yetr ;
    %In polar coordinates
```

```
    chie=atan2(ye,xe) ;
    roe= sqrt(xe.^2 + ye.^2);
    chir=atan2(v_yr(it,:),v_xr(it,:));
    fie=fi-v_fir(it,:);
    alfe=fie-chie -pi;


    %Tracking control to be implemented
    %v= ... ;
    %w= ... ;

    %Instead, we consider perfect tracking and the reference velocities
    %(v_vr, v_wr) are applied to the robots:
    v=v_vr(it,:);
    w=v_wr(it,:);

    %Motion of the robots
    fi=fi + w*delta;
    Dy= zeros(1,N);
    Dx= v*delta;
    nz=find(w ~= 0);
    if ~isempty(nz),
        L= v(nz)./w(nz); %Curvature radius (radians)
        Dy(nz)= L -L.*cos(w(nz)*delta);
        Dx(nz)= L.*sin(w(nz)*delta);
    end
    x= x +Dx .*cos(fi) -Dy .*sin(fi);
    y= y +Dx .*sin(fi) +Dy .*cos(fi);


    %Store data
    v_v(it,:)=v;
    v_w(it,:)=w;
    v_xe(it,:)=xe;
    v_ye(it,:)=ye;
    v_fie(it,:)=fie;
    v_roe(it,:)=roe;
    v_alfe(it,:)=alfe;
    v_chie(it,:)=chie;

end %Main loop
%--------------------------------------------------------------------------


%for it=1:10:nt,
   for it=1:500:2000

   pose_tx= v_xt(it,1);
   pose_ty= v_yt(it,1);
   setState(bunny, 'position', [pose_tx pose_ty 0]);
   [bunny_position, bunny_orientation, bunny_velocity] = getState(bunny)
   %pause(1);

     for i=1:N,
      pose= rosmessage('geometry_msgs/Pose');

      ang_rad = v_fir(it,i) + betar0(1,i) - pi/2;
      eulZYX = [ang_rad 0 -pi/2];
      qZYX = eul2quat(eulZYX);
      orientation = qZYX;

%       ang_rad_gazebo = v_fir(it,i) + betar0(1,i);
%       eulZYX_gazebo = [ang_rad_gazebo 0 0];
%       qZYX_gazebo = eul2quat(eulZYX_gazebo);
%       orientation_gazebo = qZYX_gazebo;
```

```
    if (i==1)
        str='A';
        %setState(flying_stereo_cam, 'position', [v_xr(it,i) v_yr(it,i) altura],
'orientation', [orientation_gazebo(2) orientation_gazebo(3) orientation_gazebo(4)
orientation_gazebo(1)]);
    else if (i==2)
           str='B';
          %setState(flying_stereo_cam_2, 'position', [v_xr(it,i) v_yr(it,i)
altura], 'orientation', [orientation_gazebo(2) orientation_gazebo(3)
orientation_gazebo(4) orientation_gazebo(1)]);
         else if (i==3)
                str='C';
                %setState(flying_stereo_cam_3, 'position', [v_xr(it,i) v_yr(it,i)
altura], 'orientation', [orientation_gazebo(2) orientation_gazebo(3)
orientation_gazebo(4) orientation_gazebo(1)]);
              else if (i==4)
                     str='D';
                     %setState(flying_stereo_cam_4, 'position', [v_xr(it,i) v_yr
(it,i) altura], 'orientation', [orientation_gazebo(2) orientation_gazebo(3)
orientation_gazebo(4) orientation_gazebo(1)]);
                   else if (i==5)
                          str='E';
                          %setState(flying_stereo_cam_5, 'position', [v_xr(it,i)
v_yr(it,i) altura], 'orientation', [orientation_gazebo(2) orientation_gazebo(3)
orientation_gazebo(4) orientation_gazebo(1)]);
                        else if (i==6)
                            str='F';
                            %setState(flying_stereo_cam_6, 'position', [v_xr(it,i)
v_yr(it,i) altura], 'orientation', [orientation_gazebo(2) orientation_gazebo(3)
orientation_gazebo(4) orientation_gazebo(1)]);
                                end
                            end
                        end
                    end
            end
    end

      pose.Position.X= v_xr(it,i);
      pose.Position.Y= v_yr(it,i);
      pose.Position.Z= altura;

      pose.Orientation.W= orientation(1);
      pose.Orientation.X= orientation(2);
      pose.Orientation.Y= orientation(3);
      pose.Orientation.Z= orientation(4);
      pose_vector(i)= pose;


      posePubmsg = pose_vector(i);
      send(posePub,posePubmsg);
      pos = [pose_vector(i).Position.X pose_vector(i).Position.Y pose_vector
(i).Position.Z]
      ori = [pose_vector(i).Orientation.X pose_vector(i).Orientation.Y pose_vector
(i).Orientation.Z pose_vector(i).Orientation.W]
      namePubmsg.Data = str
      send(namePub,namePubmsg);
%
%
       pause(10);

     end

     %pause(5);

   end
%end
```

```
rosshutdown;
%--------------------------------------------------------------------------
%%%%%%%%%%% %%%%%%%%%%% %%%%%%%%%%% %%%%%%%%%% %%%%%%%%%% %%%%%%%%%%
% FIGURES % % FIGURES % % FIGURES % % FIGURES % % FIGURES % % FIGURES %
%%%%%%%%%%% %%%%%%%%%%% %%%%%%%%%%% %%%%%%%%%% %%%%%%%%%% %%%%%%%%%%
%--------------------------------------------------------------------------


%--------------------------------------------------------------------------
%--------------------------------------------------------------------------
scrsz = get(0,'ScreenSize');
figure('position',[51 51 scrsz(3)-100 scrsz(4)-150]);
%--------------------------------------------------------------------------
subplot(2,3,1);
plot(t,v_xt,'r','LineWidth',3);
hold;
plot(t,v_xr,'k','LineWidth',1);
plot(t,v_x,'b','LineWidth',1);
xlabel('time');
legend('target','robots');

title('x-coordinate');
%--------------------------------------------------------------------------
subplot(2,3,2);
plot(t,v_yt,'r','LineWidth',3);
hold;
plot(t,v_yr,'k','LineWidth',1);
plot(t,v_y,'b','LineWidth',1);
xlabel('time');
title('y-coordinate');
%--------------------------------------------------------------------------
subplot(2,3,3);
plot(t,v_fit*180/pi,'r','LineWidth',3);
hold;
plot(t,v_fir*180/pi,'k','LineWidth',1);
plot(t,v_fi*180/pi,'b','LineWidth',1);
xlabel('time');
title('phi-coordinate');
%--------------------------------------------------------------------------
subplot(2,3,4);
plot(t,v_vt,'r','LineWidth',3);
hold;
plot(t,v_vr,'k','LineWidth',1);
plot(t,v_v,'b','LineWidth',1);
xlabel('time');
title('Linear velocity: v');
%--------------------------------------------------------------------------
subplot(2,3,5);
plot(t,v_wt*180/pi,'r','LineWidth',3);
hold;
plot(t,v_wr*180/pi,'k','LineWidth',1);
plot(t,v_w*180/pi,'b','LineWidth',1);
xlabel('time');
title('Angular velocity: w');
%--------------------------------------------------------------------------
subplot(2,3,6);
plot(v_xt,v_yt,'r','LineWidth',3);
hold;
plot(v_xr,v_yr,'k','LineWidth',1);
plot(v_x,v_y,'b','LineWidth',1);

axis equal
title('Top view: x-y');
```

```
%--------------------------------------------------------------------------
%--------------------------------------------------------------------------
% %Tracking result
% figure
% plot(v_xt,v_yt,'r.');
% hold;
% drawRobots([v_xt,v_yt,v_fit], 'r-', 1);
%
% plot(v_xr,v_yr,'k.');
% plot(v_x,v_y,'bo-');
%
% np=5; %Plot only every np steps
% itr=[1:round(Tmax/np/delta):round(Tmax/delta)];
%
% %Polygon of the reference formation
% plot([v_xr(itr,:),v_xr(itr,1)]',[v_yr(itr,:),v_yr(itr,1)]','k-');
% plot([v_xr(itr,1),v_xt(itr)]',[v_yr(itr,1),v_yt(itr)]','k-');
%
% %Polygon of the current formation
% plot([v_x(itr,:),v_x(itr,1)]',[v_y(itr,:),v_y(itr,1)]','b-');
% plot([v_x(itr,1),v_xt(itr)]',[v_y(itr,1),v_yt(itr)]','b-');
%
% %Trajectory of one robot ii of the reference formation
% ii=1;
% drawRobots([v_xr(itr,1),v_yr(itr,1),v_fir(itr,1)], 'k-', 1);
% %Trajectory of one robot ii of the formation
% drawRobots([v_x(itr,1),v_y(itr,1),v_fi(itr,1)], 'b-', 1);
% axis equal
% title('Tracking of the reference formation. Top view (x-y)');

%--------------------------------------------------------------------------
%--------------------------------------------------------------------------
% scrsz = get(0,'ScreenSize');
% figure('position',[51 51 scrsz(3)-100 scrsz(4)-150]);
%--------------------------------------------------------------------------
% subplot(2,3,1);
% plot(t,betar*180/pi,'LineWidth',1);
% hold;
% plot([ones(1,N); Tmax*ones(1,N)],[betar0; betar0]*180/pi,'k:','LineWidth',3);
% plot(itr*delta,zeros(1,length(itr)),'*');
% title('betar - target angle in the camera wrt fir');
% %-------------------------------------------------------------------------
% subplot(2,3,2);
%
% if strcmp(strategy,'d')
%     plot(t, m_betari*180/pi,'LineWidth',1);
% else %if strategy == 'theta',
%     plot(t, beta*180/pi,'LineWidth',1);
% end
%
% hold;
% plot([1 Tmax],[betaMax, betaMax]*180/pi,'k:','LineWidth',3);
% plot([1 Tmax],-[betaMax, betaMax]*180/pi,'k:','LineWidth',3);
% plot(itr*delta,zeros(1,length(itr)),'*');
%
% title('beta - target angle in the FOV camera reference');

%--------------------------------------------------------------------------
%--------------------------------------------------------------------------
figure
plot(t,m_betari*180/pi,'LineWidth',3);
hold;
plot([1 Tmax],[betaMax, betaMax]*180/pi,'k:','LineWidth',3);
plot([1 Tmax],-[betaMax, betaMax]*180/pi,'k:','LineWidth',3);
xlabel('time');
ylabel('FOV [degrees]');
```

```
title('Evolution of the target projection in the cameras FOV between (-betaMax,
betaMax)');

%-------------------------------------------------------------------------
%-------------------------------------------------------------------------
if strcmp(strategy,'d')
    figure
    plot(t,v_kLMax,'r','LineWidth',3);
    hold;
    plot(t,1./m_dwc,'b','LineWidth',3);
    plot(t,v_kr,'k','LineWidth',3);

    legend('kL','kwc','kr');
    xlabel('time');
    ylabel('Inverse formation radius');
    title('Evolution of the inverse formation radius (kappa=1/d)');
else %strategy == 'theta',
    figure
    plot(t,1./v_dr0,'k','LineWidth',3);

    legend('kr0');
    xlabel('time');
    ylabel('Inverse formation radius');
    title('Evolution of the PRESCRIBED inverse formation radius 1/dr0');
end

%-------------------------------------------------------------------------
%-------------------------------------------------------------------------
if strcmp(strategy,'theta')
    figure
    plot(0,0,'--k','LineWidth',3);%For show common legend
    hold;
    plot(0, 0,':k','LineWidth',3);%For show common legend
    plot(0,0,'k','LineWidth',1);%For show common legend

    ax=gca;%To control the colors in recen versions of matlab
    plot(t, m_thetarMax*180/pi,'--','LineWidth',3);
    ax.ColorOrderIndex = 1;
    plot(t, m_thetarMin*180/pi,':','LineWidth',3);
    ax.ColorOrderIndex = 1;
    plot(t, betweenPi(m_thetar)*180/pi,'LineWidth',1);
    %plot(t, (m_thetar)*180/pi,'LineWidth',1);

    legend('thetaMax', 'thetaMin','thetar');

    xlabel('time');
    ylabel('theta [degrees]');
    title('Evolution of angle theta of each robot in the formation');
else %strategy == 'd',
    figure
    plot(0,0,'k','LineWidth',3);%For legend
    hold;
    plot(t, betweenPi(m_thetar0)*180/pi,'LineWidth',3);

    legend('thetar0');
    xlabel('time');
    ylabel('theta [degrees]');
    title('Evolution of the PRESCRIBED angle theta of each robot in the
formation');
end %if strategy == 'theta',

%-------------------------------------------------------------------------
%-------------------------------------------------------------------------
figure
plot(v_xt,v_yt,'r','LineWidth',3);
hold;
```

```
plot(v_xr,v_yr,'k','LineWidth',1);

np=5; %Plot only every np steps
itr=1:round(Tmax/np/delta):round(Tmax/delta);

%Polygon of the reference formation
plot([v_xr(itr,:),v_xr(itr,1)]',[v_yr(itr,:),v_yr(itr,1)]','k-');
plot([v_xr(itr,1),v_xt(itr)]',[v_yr(itr,1),v_yt(itr)]','k-');

if strcmp(strategy,'theta')
    pipi=(-pi:0.01:pi)';
    for iitr=itr,
        %Draw a circle along the formation
        xcircf= dr0*cos(pipi) + v_xt(iitr);
        ycircf= dr0*sin(pipi) + v_yt(iitr);
        plot(xcircf', ycircf',':k','LineWidth',1);
    end
end

colors=get(gca,'ColorOrder');
if N>size(colors,1),
    colors=repmat(colors, ceil(N/size(colors,1) ) ,1);
end
iic=1;

%Draw FoV
for ii=1:N, %FoV de todos los robots
    drawRobots([v_xr(itr,ii),v_yr(itr,ii),v_fir(itr,ii)], 'k-', 1);
    drawRobots([v_xt(itr),v_yt(itr),v_fit(itr)], 'r-', 1);
    plot(v_xt(itr),v_yt(itr), '*r');

    colori=colors(iic,:);
    iic=iic+1;

    for iitr=itr,
        betari= v_fir(iitr,ii)+betar0(1,ii)+betaMax : -betaMax/10 : v_fir(iitr,ii)
+betar0(1,ii)-betaMax ;
        xcirc= m_v_dr(iitr,ii)*cos(betari) + v_xr(iitr,ii);
        ycirc= m_v_dr(iitr,ii)*sin(betari) + v_yr(iitr,ii);
        plot([v_xr(iitr,ii),xcirc,v_xr(iitr,ii)]', [v_yr(iitr,ii),ycirc,v_yr
(iitr,ii)]','-','color',colori,'LineWidth',2);
    end
end
axis equal
title('Evolution of the reference formation. Top view (x-y)');

%--------------------------------------------------------------------------
%--------------------------------------------------------------------------
if animate == 1

    figure
    plot(v_xt,v_yt,'r','LineWidth',3);
    hold;
    plot(v_xr,v_yr,'k','LineWidth',1);

    np=5; %Plot only every np steps
    itr=1:round(Tmax/np/delta):round(Tmax/delta);

    %Polygon of the reference formation
    plot([v_xr(itr,:),v_xr(itr,1)]',[v_yr(itr,:),v_yr(itr,1)]','k-');
    plot([v_xr(itr,1),v_xt(itr)]',[v_yr(itr,1),v_yt(itr)]','k-');

    colors=get(gca,'ColorOrder');
    if N>size(colors,1),
        colors=repmat(colors, ceil(N/size(colors,1) ) ,1);
    end
```

```
    %FoV
    for iitr=1:10:nt,
        cla
        plot(v_xt,v_yt,'r','LineWidth',3);
        plot(v_xr,v_yr,'k','LineWidth',1);

        plot([v_xr(iitr,:),v_xr(iitr,1)]',[v_yr(iitr,:),v_yr(iitr,1)]','k-');
        plot([v_xr(iitr,1),v_xt(iitr)]',[v_yr(iitr,1),v_yt(iitr)]','k-');
        iic=1;
        for ii=1:N, %Draw FoV of all the robots
            colori=colors(iic,:);
            iic=iic+1;
            betari= v_fir(iitr,ii)+betar0(1,ii)+betaMax : -betaMax/10 : v_fir
(iitr,ii)+betar0(1,ii)-betaMax ;
            xcirc= m_v_dr(iitr,ii)*cos(betari) + v_xr(iitr,ii);
            ycirc= m_v_dr(iitr,ii)*sin(betari) + v_yr(iitr,ii);

            plot([v_xr(iitr,ii),xcirc,v_xr(iitr,ii)]', [v_yr(iitr,ii),ycirc,v_yr
(iitr,ii)]','-','color',colori,'LineWidth',2);

            drawRobots([v_xr(iitr,ii),v_yr(iitr,ii),v_fir(iitr,ii)], 'k-', 1);
        end
        drawRobots([v_xt(iitr),v_yt(iitr),v_fit(iitr)], 'r-', 1);
        plot(v_xt(iitr),v_yt(iitr), '*r');

        axis equal
        pause(0.05)
        title('Evolution of the reference formation. Top view (x-y)');
    end

end %if animate == 1

load handel;
sound(y,Fs);
```