Universidad del País Vasco
Euskal Herriko Unibertsitatea

Facultad de Informática

Grado de Ingeniería Informática

▪ Proyecto Fin de Grado ▪

Ingeniería de Computadores

Control de transporte de cargas con robots aéreos autónomos.

Aitor Alday

Junio 2017

Supervisor

Manuel Graña

informatika fakultatea
facultad de informática

Resumen

Proyecto de Fin de Grado de Ingeniería Informática en la especialidad de computación. Tras un análisis de herramientas que permiten la simulación robótica, se realizan diferentes experimentos con el fin de conseguir un modelo eficiente para el transporte de carga con robots aéreos autónomos.

Índice

Índice de Figuras y Tablas

FIGURAS

1

Capitulo 1: Introducción y definiciones

1.1. Vehículos aéreos no tripulados(VANT)

Un vehículo aéreo no tripulado (VANT), UAV (Unmanned Aerial Vehicle), comúnmente denominado dron, es una aeronave que vuela sin tripulación, capaz de mantener de manera autónoma un nivel de vuelo controlado y sostenido, y propulsado por un motor de explosión, eléctrico, o de reacción.

El diseño de los VANT tiene una amplia variedad de formas, tamaños, configuraciones y características. Históricamente surgen como aviones pilotados remotamente. Existen dos variantes: los controlados desde una ubicación remota, y aquellos de vuelo autónomo a partir de planes de vuelo pre programados a través de automatización dinámica. El empleo de técnicas de control autónomo de los VANT aumenta a diario.

Existen VANT de uso tanto para investigación como comerciales, pero sus primeros usos fueron en aplicaciones militares, en este caso llamados Vehículos Aéreos de Combate No Tripulados —UCAV en sus siglas en inglés—. Los misiles de crucero no son considerados VANT, ya que aunque son vehículos no tripulados y a veces guiados remotamente, el propio vehículo del misil es un arma no reutilizable. En ese sentido, las aeronaves controladas remotamente (Aeronaves Radio controladas o Aeronaves R/C) no se consideran como VANT, al no ser sistemas autónomos que puedan operar sin intervención humana durante su funcionamiento en la misión, es decir, pueden despegar, volar y aterrizar automáticamente.

Con la progresiva popularización del uso civil de los drones sus aplicaciones varían, ampliándose el número de consumidores más allá del terreno militar. Este crecimiento tan acusado ha llevado a que emerjan cada vez más empresas para beneficiarse de este nicho de mercado, tales como Syma[Sym] o DJI[Dj].

Actualmente, los VANT militares realizan tanto misiones de reconocimiento como de ataque. Si bien se ha informado de muchos ataques de drones con éxito, también son susceptibles de provocar daños colaterales y/o identificar objetivos erróneos, como con otros tipos de arma. Los VANT también son utilizados en un pequeño pero creciente número de aplicaciones civiles, como en labores de lucha contra incendios o seguridad civil, como la vigilancia de los oleoductos. Los vehículos aéreos no tripulados suelen ser preferidos para misiones que son demasiado "aburridas, sucias o peligrosas" para los aviones tripulados. [Wi1].

1.1.1. Quadcopter

Como podemos deducir del nombre se trata de un helicóptero multirotor que se eleva y propulsa gracias a cuatro rotores.

Comúnmente los rotores están colocados en forma de cruz girando dos en el sentido de las agujas del reloj y dos en el sentido contrario. En cuanto a la dinámica de vuelo como cualquier aeronave su orientación se rige según los ángulos de navegación. Estos ángulos son tres la dirección (heading o yaw), elevación (pitch) y ángulo de alabeo (roll).

Los tres son equivalentes a tres maniobras consecutivas. Dado un sistema de tres ejes fijos en el aeroplano, llamados eje de guiñada (yaw en inglés), de cabeceo (pitch) y de alabeo (roll), existen tres rotaciones principales, normalmente llamadas igual que el eje sobre el que se producen, que permiten alcanzar el sistema del aeroplano desde el sistema de referencia. Tienen que venir dadas en ese orden y ser realizadas en ese orden, ya que el resultado final depende del orden en que se apliquen.

- Cabeceo: es una inclinación del morro del avión, o rotación respecto al eje ala-ala.
- Alabeo: rotación respecto de un eje morro-cola del avión.
- Guiñada: rotación intrínseca alrededor del eje vertical perpendicular al avión.

Son tres rotaciones intrínsecas, es decir, relativas al sistema móvil. Esto es útil por ejemplo cuando el piloto de un avión quiere describir una maniobra. [Wi2]

Esto hace que el quadrotor, en función de la inclinación, orientación o altura que desee conseguir, de más o menos impulsó a unos rotores que a otros:

-Para controlar la altitud aumenta o reduce la velocidad de todos los rotores por igual. (Figura 1.1)

-Para ajustar la guiñada aumenta la velocidad de dos de los rotores que giran en la misma dirección. (Figura 1.2)

-Para ajustar el cabeceo o el alabeo aumenta la velocidad de un rotor y disminuye la del diametralmente opuesto. (Figura1.3)

Figura 1.1 Control de altitud. [Wi4]

Figura 1.2 Control de guiñada. [Wi4]

Figura 1.3 Control de alabeo o cabeceo. [Wi4]

1.2. Modelo de transporte

El articulo “Modelling and Control for Cooperative Aerial Transportation of a Payload as a Reconfigurable Cable-Driven Parallel Robot” de Carlo Masone, Paolo Stegagno y Heinrich H. Bülthoff[Art] nos ha servido de inspiración para plantear los modelos del transporte de carga.

Por un lado habría que analizar el número y la formación del equipo de quadrotors y por otro, su estrategia de movimiento. En cuanto a la cantidad, en el artículo se propone un sistema montado por cuatro quadrotors mediante cables flexibles. Para este proyecto se hará una simplificación en cuanto a la cantidad. Se realizaran experimentos con un equipo mínimo de dos quadrotors.

En el apartado estratégico también se pueden encontrar diferentes alternativas. Por un lado se puede optar por dar libertad a cada miembro del equipo para crear un movimiento coordinado. Es decir, que todos se muevan de forma independiente buscando llegar al mismo objetivo. En este plan tendría que existir la opción que dado el caso, la presencia de un obstáculo por ejemplo, uno de ellos coja el control sobre el resto.

En contra a esta estrategia se puede implementar la conocida como follow the leader. El funcionamiento de esta se basa en que uno toma el control y el resto se limita a seguirle. En este caso también sería interesante plantear un cambio de líder para evitar colisiones. Como primera aproximación se utilizará esta estrategia de gestión del equipo.

2

Capitulo 2: Objetivos y recursos

2.1. Objetivos del proyecto

El objetivo a largo plazo del grupo de trabajo es el desarrollo de algoritmos de control de equipos de vehículos aéreos no tripulados (VANT), en especial quadrotors para el transporte de cargas. Intentamos superar las limitaciones los quadrotors individuales mediante la coordinación de un equipo de ellos.

El objetivo específico de este proyecto en este contexto es el estudio de técnicas de simulación que pueden permitir desarrollar in-silico los algoritmos de control. En este sentido se han examinado varios sistemas de simulación, entre los que hemos seleccionado V-REP para la realización de experimentos que ilustran su potencial para el estudio detallado de los algoritmos de control.

El objetivo principal de este proyecto consiste en conseguir, utilizando modelos de simulación, una estrategia de control para la coordinación de equipos de aeronaves no tripuladas capaz de transportar una carga de manera autónoma y eficiente.

Tras conseguirlo, además de lo dicho anteriormente, se conseguirán además otras metas adicionales. La primera, el dominio o unos conocimientos básicos de uno o varios simuladores. Y la segunda, conocimiento considerable de la programación de quadrotors para realizar un vuelo autónomo. Además es posible que, si el simulador lo requiere, se aprendan conceptos básicos de algún nuevo lenguaje de programación.

2.1.1. Interés/motivación personal

Desde la aparición de los drones está en mente de todos la idea de que estos se utilicen para realizar tareas autónomas, ya sea una tarea sencilla como la vigilancia, o una tarea más complicada como el transporte eficiente de una carga. Pese a que se han hecho avances en este campo, todavía queda mucho que mejorar y este proyecto proporciona las bases para una experimentación más exhaustiva. Además, se conseguirá el manejo, o unos conocimientos básicos, sobre algún simulador lo que, además de introducir en el mundo de los quadrotors, servirá de introducción al mundo de la robótica, campo muy amplio y con mucho recorrido.

2.2. Alcance

Como se ha mencionado anteriormente la idea idónea seria conseguir algo funcional en la vida real. No obstante, no se pretende llegar tan lejos. El objetivo sería conseguir un modelo el transporte de carga mediante dos quadrotors de manera eficaz pero, desde el punto de vista de la simulación.

2.4. Fases del proyecto

La realización del proyecto se divide en diferentes fases:

1. Comprensión del problema: En esta primera fase, se comprenderá el problema del transporte con quadrotors y se definirá el alcance del proyecto además de los pasos a seguir.
2. Análisis de herramientas a utilizar: El objetivo de esta fase es decidir que herramienta se utilizara para llevar a cabo los experimentos del proyecto. Para ello, primero habrá que realizar un estudio de las diferentes herramientas validas.
3. Aprendizaje y desarrollo: Una vez elegida la herramienta el siguiente trabajo será aprender las bases y empezar a realizar los diferentes experimentos planificados.
4. Memoria: La última fase se trata de redactar un documento donde quede recogido todo lo que ha rodeado este proyecto.

2.5. Material

Para la realización del proyecto será necesario un ordenador, se ha utilizado uno con sistema operativo Linux aunque cualquier otro seria igualmente valido. En cuanto al software se utilizaran matlab y V-REP.

3

Capitulo 3: Gestión

3.1. Planificación

Las tareas principales del proyecto se dividen en cuatro fases: comprensión, análisis de herramientas, aprendizaje y desarrollo y memoria. Sobre todo en la parte de los experimentos se dejará margen para posibles fallos y retrasos.

3.1.1. Fase 1: Formulación del problema

En esta primera fase se definen dos tareas principales:

1. Documentarse y comprender el problema. Donde se pretende comprender el problema del transporte de carga de los drones. Fecha inicio (1/11/2016). Fecha límite (12/12/2016).

2. Definición alcance, objetivo y tareas. Una vez formulado el problema se procederá a definir el resto de las tareas y estructura del proyecto. Fecha límite (31/12/2016).

3.1.2. Fase 2: Análisis de herramientas

En esta tarea se trata de hacer un análisis de las herramientas adecuadas para realizar el proyecto. Por lo que la fecha limite será para la fase en general y no para la tarea.

1. Analizar herramientas. Probar diferentes herramientas en busca de la más adecuada. Duración (Enero 2016).
2. Decidir cual utilizar. Tomar la decisión con las observaciones realizadas y realizando alguna más si se ve necesario. Fecha fin(12/02/2017).

3.1.3. Fase 3: Aprendizaje y desarrollo

Una vez seleccionada una herramienta se procederá al aprendizaje de la misma y después al desarrollo.

1. Aprender a utilizar la herramienta seleccionada. Familiarizarse con la forma de trabajar de la herramienta y realizar algunos tutoriales. Fecha inicio (13/02/2017). Fecha fin (Marzo 2017).
2. Planificar/decidir qué experimentos se van a realizar para llegar al experimento final de demostración del transporte de una carga. Además, decidir de cuál de las diferentes alternativas de VREP usar para programar. Límite(Primera semana de Marzo)
3. Realizar diferentes experimentos para acercarse al resultado final. Estos empezarán en febrero junto con el aprendizaje y tendrán que finalizarse en mayo. Fecha límite (31/05/2017).

Los experimentos finalmente realizados son:

5.1. Quadrotor solo. Fecha límite (12/03/2017)

5.2 Quadrotor con una cuerda. Fecha límite (12/03/2019)

5.3. Quadrotor con cuerda y carga. Fecha límite (20/03/2017 - 26/03/2017)

5.4. Quadrotor siguiendo un camino. Fecha límite (9/04/2017)

5.5 Dos quadrotors en movimiento sincronizado. Fecha límite (30/04/2017)

5.6 Dos quadrotors transportando una carga. Fecha límite (23/05/2017)

3.1.4. Fase 4: Memoria

Esta fase es la más difícil de definir con fechas dado que se prolonga durante todo el proyecto por lo que se podría decir que su fecha de inicio y fin son las mismas que las del proyecto.

1. Definir esquema e índice de la memoria. Se realizará junto con la definición de las tareas.
2. Recoger información durante el resto de las fases a modo de borrador.
3. Redactar la memoria con los datos recogidos. Fecha limite(16/06/2017)

3.2. Desviación

En las dos primeras fases todo salió como estaba planeado y no fue necesaria ninguna corrección de plazos. En la fase de desarrollo es donde se realizaron cambios de calendario.

El primer contratiempo surgía a la hora de elegir como programar en V-REP. Analizando experimentos realizados por otras personas se pudo observar que la opción mas utilizada era la de programar con ROS. Esto parecía buena idea dado que, daría muchas facilidades de cara a adaptar lo realizado para ponerlo en un robot real. Pero lo que parecía la opción lógica resulto ser la más problemática. Dicho lo anterior lo que hacía falta era instalar varios programas para poder utilizar esta opción. La instalación de esta fue bastante problemática y sin éxito.

Entonces tras mirar todas las APIs locales que ofrece VREP y que LUA se describe como un lenguaje sencillo se optó por seguir esa dirección. Esto provocó que se tuvieran que reajustar todos los plazos atrasando una semana en las restantes fechas previstas.

4

Capitulo 4: Herramientas de simulación. Selección de VREP

Básicamente se ha realizado el análisis de dos herramientas de simulación: Matlab con simulink y V-Rep.

4.1. Matlab & simulink

MATLAB (abreviatura de MATrix LABoratory, "laboratorio de matrices") es una herramienta de software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows, Mac OS X y GNU/Linux .

Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware. El paquete MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones, a saber, Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI). Además, se pueden ampliar las capacidades de MATLAB con las cajas de herramientas (toolboxes); y las de Simulink con los paquetes de bloques (blocksets). [WI3]

4.1.1. Simulink

Simulink es un entorno de diagramas de bloque para la simulación multidominio y el diseño basado en modelos. Admite el diseño y la simulación a nivel de sistema, la generación automática de código y la prueba y verificación continuas de los sistemas embebidos.

Simulink ofrece un editor gráfico, bibliotecas de bloques personalizables y solvers para modelar y simular sistemas dinámicos. Se integra con MATLAB, lo que permite incorporar algoritmos de MATLAB en los modelos y exportar los resultados de la simulación a MATLAB para llevar a cabo más análisis. [SIm1]

4.1.2. Probando matlab

Para recabar datos y poder tomar una decisión se optó por la idea de crear un quadcopter básico con matlab. Utilizando simulink mediante la unión de varios bloques se conseguía un modelo que básicamente eran dos placas y en cada uno de los cuatro extremos una hélice. En la simulación con la variación de los rotores se podía conseguir que girara a favor o en contra de las agujas del reloj.

Para crear un comportamiento más complejo había que diseñar todas las ecuaciones de movimiento a través del sistema de bloques de simulink. Aparte de eso para tener una representación más realista de un quadcopter hacía falta diseñarlo con un programa externo.

Por otro lado simulink en su panel de bloques ofrece una amplia lista de bloques para realizar infinidad de tareas. Desde bloques de objetos hasta bloques para implementar ecuaciones de movimiento.

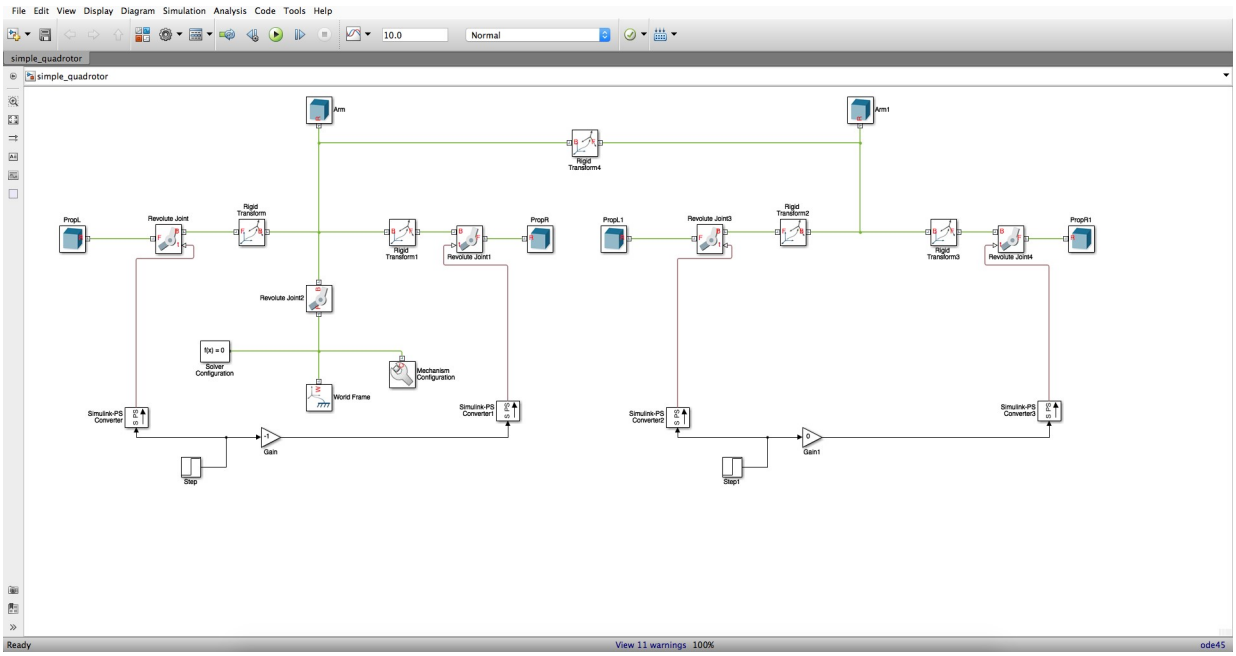

Figura 4.1 Diseño quadrotor simulink.

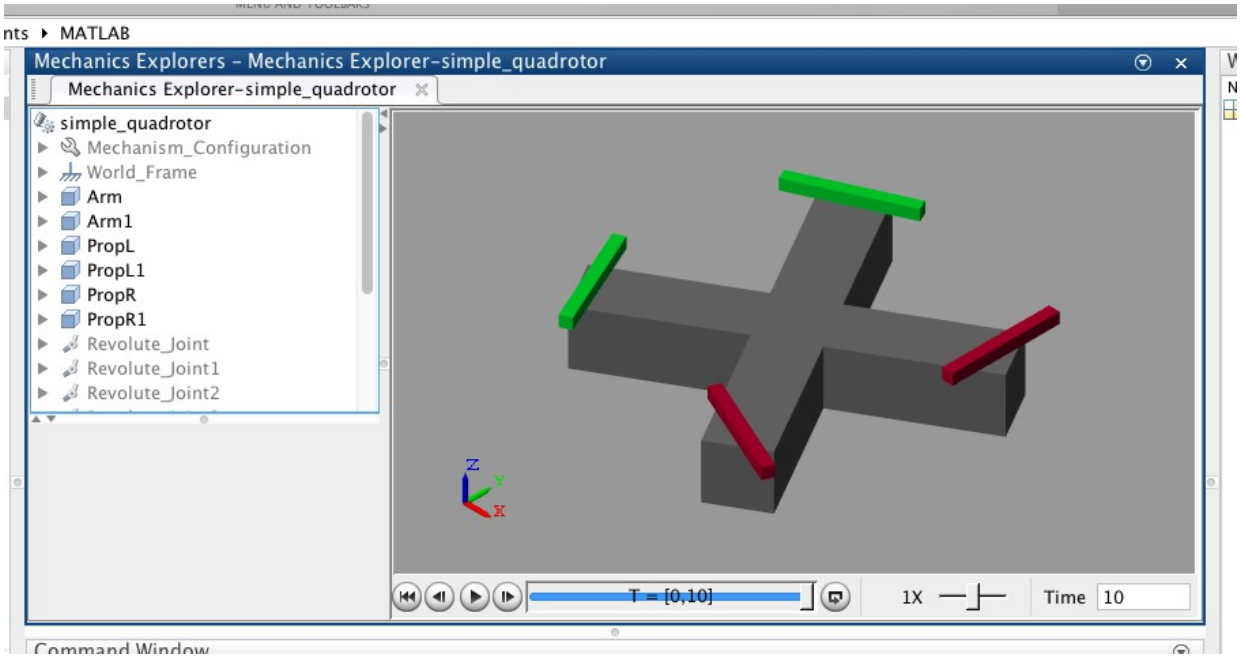

Figura 4.2 Simulación matlab.

4.2. V-Rep

Se trata de un simulador de robots, con entorno de desarrollo integrado, basado en una arquitectura de control distribuido: Cada objeto o modelo se puede controlar individualmente mediante un script embebido, un plugin, un nodo de ROS, una API de un cliente remoto o incluso mediante una solución a medida. Estas son algunas de sus características:

1. Es un software multiplataforma disponible para los tres principales sistemas operativos que permite una fácil y simple portabilidad. En un único archivo podemos guardar un modelo funcional o incluso una escena incluido el control del mismo.

2. Como se menciona anteriormente V-Rep ofrece la posibilidad de programar en cualquier lenguaje de programación mediante APIs. Cuenta con más de cien APIs remotas y más de quinientas internas.

3. Ofrece todo tipo de sensores y controladores: detección de colisiones, calculo de distancia mínima… Además cuenta con la posibilidad de montar un robot desde cero cubriendo hasta el mas pequeño detalle.

4.2.1. Probando V-REP

Para probar VREP parecía lógico hacer algo parecido a lo realizado en matlab. Pero como en este ya se encuentran modelos de robots con un comportamiento básico se decidió realizar el primero de los tutoriales del manual de VREP [Vrep]. En este se creaba un robot q rodeado de columnas trata de moverse sin chocarse con las columnas. Con este tutorial se observa cómo de forma rápida y sencilla se le puede dar movilidad y añadir sensores a un robot.

4.3. Elección

Teniendo en cuenta las características de ambos programas queda claro que cualquiera de los dos podría ser un buen candidato para la realización de este proyecto. Pero tras analizarlo un poco, se puede ver como V-REP ofrece de manera más cómoda y directa el uso de un quadrotor. Mientras que con matlab y simulink hay que construir un dron desde cero ayudándose de programas como solidworks, que permite el modelado de objetos y es compatible con simulink.

Tras probar los dos programas se puede observar como, mientras que en matlab hace falta tanto modelar el robot como programar su física, en VREP nos dan un robot ya funcional y de manera sencilla se pueden crear robots con comportamientos interesantes.

Matemáticamente hablando matalab es una herramienta más potente y ofrece una mayor complejidad. Pero en cuanto a crear escenas y añadir diferentes sensores y objetos al robot V-REP ofrece un entorno visual muy cómodo de manejar.

Aparte de eso los scripts del propio V-REP están programados en LUA, que es un lenguaje orientado a objetos muy ligero y fácil de usar. De esta manera además de lograr los objetivos del proyecto, de forma transversal podremos aprender un nuevo lenguaje de programación que cada día va cogiendo mas fuerza.

4.4. Instalación

Para este proyecto se utilizara la versión gratuita para estudiantes. Antes de empezar ha hacer nada es necesario instalar V-REP en el equipo. En el caso de Linux es muy sencillo: se descarga una carpeta y estando dentro de ella ejecutamos un archivo sh mediante la consola. En el caso de Windows o MAC hay que seguir un proceso de instalación como con cualquier otro programa

4.5. Programación en V-REP

Tal y como se ha mencionado anteriormente V-REP ofrece una amplia variedad de aproximaciones en cuanto a la forma de programar los módulos para conseguir comportamientos personalizados:

1. Scripts embebidos: Se trata de programar todo mediante scripts utilizando LUA. Este método garantiza la compatibilidad entre diferentes instalaciones de V-REP además de ser el mas útil y utilizado de los métodos.

2. Mediante añadidos (add on): Aquí también se trata de programar en LUA pero se trata de programas que se cargan desde fuera, los cuales pueden empezar junto con la simulación o correr en paralelo.

3. Plugins: Se trata de crear plugins que implementen diferentes funcionalidades. Este método es muy usado para crear funciones customizadas de LUA por lo que suele ir en conjunto al primer método. Aun así existe la posibilidad de usarlo al margen de LUA.

4. Api remota: A través de este método se permite facilitar la conexión con otros dispositivos. Utilizando los comandos necesarios podemos controlar V-REP desde cualquier otro dispositivo o programa.

5. Nodo Ros: Podemos conectar cualquier otro robot o máquina mediante ROS (Robotic Operating Sistem) y coger el control de VREP.

6. Cliente/Servidor: En este caso tanto el cliente como el servidor tienen que trabajar en conjunto, puede que mediante un script o un plugin, usando una comunicación customizada. Es una forma de trabajar mas flexible pero a su vez requiere mas esfuerzo.

En el siguiente cuadro encontramos una comparativa, a nivel de funcionalidades, entre las diferentes formas de trabajar.

	Embedded script	Add-on	Plugin	Remote API client	ROS node	Custom client/server
Control entity is external (i.e. can be located on a robot, different machine, etc.)	No	No	No	Yes	Yes	Yes
Difficulty to implement	Easiest	Easiest	Relatively easy	Easy	Relatively difficult	Relatively difficult
Supported programming language	Lua	Lua	C/C++	C/C++, Python, Java, Matlab, Octave, Lua, Urbi	Any [1]	Any
Simulator functionality access (available API functions)	500+ functions, extendable	500+ functions, extendable	500+ functions	>100 functions, extendable	Depends on the selected ROS interface	custom implementation
The control entity can control the simulation and simulation objects (models, robots, etc.)	Yes	Yes	Yes	Yes	Yes	Yes
The control entity can start, stop, pause and step a simulation	Start, stop, pause	Start, stop, pause	Start, stop, pause, step	Start, stop, pause, step	Start, stop, pause, step	Start, stop, pause, step
The control entity can customize the simulator	Yes	Yes	Yes	No	No	No
Code execution speed	Relativ. slow [2] (fast with JiT compiler)	Relativ. slow [2] (fast with JiT compiler)	Fast	Depends on programming language	Depends on programming language	Depends on programming language
Communication lag	None	None	None	Yes, reduced [3]	Yes, reduced	Yes, can be reduced
Control entity is fully contained in a scene or model, and is highly portable	Yes	No	No	No	No	No
API mechanism	Regular API	Regular API	Regular API	Remote API	ROS	Custom communication + regular API
API can be extended	Yes, with custom Lua functions	Yes, with custom Lua functions	Yes, V-REP is open source	Yes, Remote API is open source	Yes, ROS plugin is open source	N/A
Control entity relies on	V-REP	V-REP	V-REP	Sockets + Remote API plugin	Sockets + ROS plugin + ROS framework	Custom communication + script/plugin
Synchronous operation [4]	Yes, inherent. No delays	Yes, inherent. No delays	Yes, inherent. No delays	Yes. Slower due to comm. Lag	Yes. Slower due to comm. Lag	Yes. Slower due to comm. Lag
Asynchronous operation [4]	Yes, via threaded scripts	No	No (threads available, but API access forbidden)	Yes, default operation mode	Yes, default operation mode	Yes

[1] Depends on what ROS currently supports

[2] The execution of API functions is however very fast. Additionally, there is an optional JiT (Just in Time) compiler option that can be activated

[3] Lag reduced via streaming and data partitioning modes

[4] *Synchronous* in the sense that each simulation pass runs synchronously with the control entity, i.e. simulation step by step

Figura 4.3 Comparativa de funcionalidades. [Vrep]

De un primer vistazo se puede ver como prácticamente todas son capaces de cumplir con casi todas las funciones. En el caso de la primera opción, se puede apreciar como el control no puede estar ubicado en otra maquina. Pero a su vez, permite customizar el escenario, cosa que otros métodos no. Y a diferencia del resto, todo el control se encuentra en la escena lo que le da una gran portabilidad.

4.6. Familiarizarse con el entrono

Una vez iniciado el programa este ofrece una interfaz de usuario con un amplio abanico de posibilidades, de las cuales se mencionaran las más utilizadas:

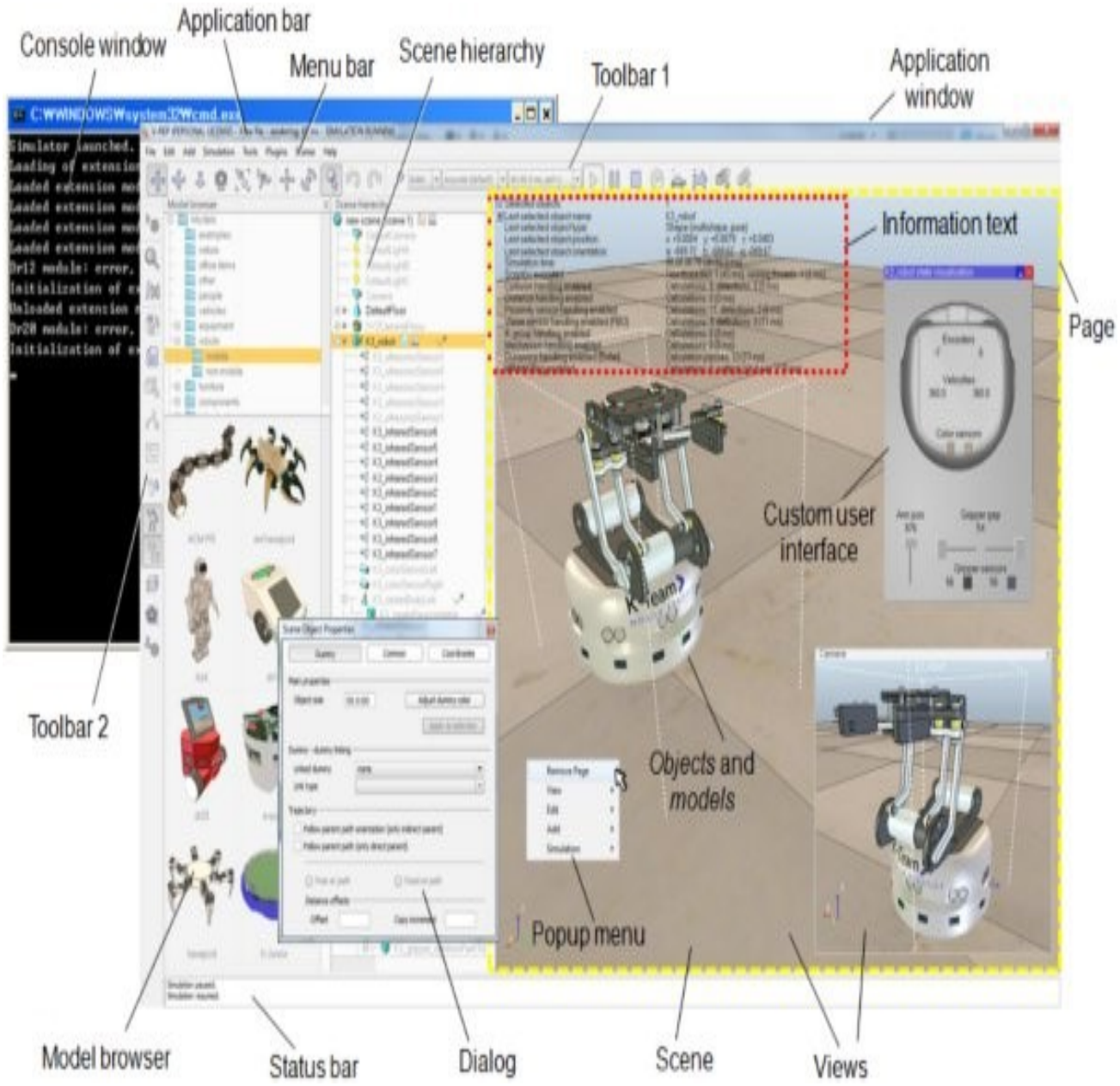

Figura 4.4 Pantalla VREP. [Vrep]

- Menu bar: Este menú permite acceder a la mayoría de las funcionalidades de V-REP aunque algunas también son accesibles de diferente forma.
- Toolbar1 y toolbar2: Ofrece las principales funcionalidades para el desarrollo del escenario como mover las cámaras, mover los objetos, iniciar la simulación…
- Model browser: Aquí se puede elegir entre una amplia variedad de modelos y una previa visualización del objeto antes de usarlo.
- Scene hierarchy: En este apartado se muestra los objetos pertenecientes a la escena además de su relación entre sí. Aparte se puede ver que objetos tienen un script y acceder a él.
- Page: Esto viene a ser la escena donde se colocan los objetos de la forma necesaria y donde se verá la simulación.

4.6.1. Entender el script por defecto

En este apartado hace falta tener en cuenta dos scripts por defecto. El primero es el que todas las escenas creadas en V-REP traen por defecto. Y el segundo, el script por defecto que traen todos los robots, y en este caso del que trae el quadrotor.

4.6.1.1 Main script

Este es el script por defecto que se carga en todas las escenas (código [C1]). En principio no es necesario hacer ningún cambio en el aunque en caso necesario se pueden realizar. A este script se la llama una única vez en cada paso de simulación, una vez en la inicialización de la simulación y otra al finalizarla. En vista de esto el script consta de tres partes principales:

- Inicialización: Esta parte solo se ejecutara cuando se inicializa la simulación. Simplemente se encarga de hacer los preparativos previos a la simulación.

- La parte regular: Este apartado se ejecuta en cada paso de simulación. En este punto del código se gestionan todas las funcionalidades del simulador. Además, hay dos funciones a tener muy en cuenta: `simLaunchThreatedChildScripts` y `simHandleChildScripts`. Estas funciones se encargan de que se ejecuten los scripts de cada uno de los robots, sin ellas no se podría usar la programación por scripts.

- Restauración: Este trozo se ejecutará justo antes de que la simulación finalice. Su función es restaurar todos los valores de los objetos de la escena a su estado inicial.

4.6.1.2 Quadrotor script

Todos los robots, incluido el quadrotor, traen un script con un comportamiento básico que puede ser modificado para obtener los comportamientos deseados (código [C2]). En el caso del quadcopter el comportamiento básico consiste en seguir la esfera verde a la que llaman target (objetivo). En este script se pueden diferenciar dos partes principales: la preparación y el control.

- Preparación: En este apartado usando diferentes funciones se toma el control del robot y de los sensores que pueda tener colocados, además de inicializar las variables.

- El control: Aquí se encarga de definir tanto el empuje como la dirección del quadrotor y transmitírselo. Esta parte se podría dividir a su vez en tres partes: control vertical, horizontal y rotacional. En la parte del control vertical se encarga de calcular el empuje (thrust) necesario para llegar al objetivo. En el horizontal, controla tanto el ángulo de cabeceo como el de alabeo. Por último, el control rotacional, controla la guiñada del robot. Además de esto, por defecto, el quadcopter suelta unas partículas de agua hacia abajo como si de un sistema de riego se tratara.

4.6.2. Colocar objetos en la escena

VREP ofrece una serie de robots y demás artilugios prediseñados que se pueden utilizar de forma muy sencilla. En el apartado “model browser” podemos ver toda la variedad de objetos que ofrece. Para añadir uno a la escena basta con arrastrarlo al escenario. Si una vez colocado el objeto es necesario moverlo o rotarlo se puede realizar de dos formas: moviendo manualmente el objeto o escribiendo sus coordenadas/orientación. Para hacerlo de forma manual basta con pulsar el botón de desplazar o el de rotar y a continuación, manteniendo pulsado el objeto, se podrá reubicar. Durante la simulación es posible mover los objetos de forma manual del mismo modo. En caso de querer ponerlo en unas coordenadas u orientación específica, se puede introducir en la ventana que se abre al pulsar sobre cualquiera de los dos botones anteriores. A la hora de guardar, tanto los scripts como los objetos, se guardan todo en un único archivo con extensión ttt.

4.6.1.2 Jerarquía de objetos

Dado que se va a tratar con robots y demás artilugios es de suponer que un objeto estará compuesto de otros por lo que al objeto final se le llama colección. En V-REP para que dos o más objetos interactúen o se muevan en conjunto, además de colocarlos de tal forma que eso sea posible, hay que hacer que uno dependa de otro. Para ello hace falta ir colocándolos de forma conveniente siguiendo una estructura árbol en el apartado de la jerarquía de escena, arrastrando unos sobre otros como sea conveniente.

5

Capitulo 5: Experimentos y Resultados

El objetivo del proyecto es conseguir transportar una carga con varios robots. Para conseguirlo se plantea una estrategia divide y vencerás. La idea es plantear una serie de experimentos más sencillos para ir logrando los diferentes comportamientos necesarios y al final adaptar todo a una única escena.

5.1. Quadrotor solo

5.1.1. Descripción y preparación

Lo primero de todo es conseguir mover el robot mediante el código del script. Como ya está explicado, el comportamiento natural es seguir la bola verde (el target). Dado que basar el movimiento del robot en el objetivo parece algo lógico y se puede comprobar que el comportamiento pese a que posiblemente no sea el idóneo es eficaz, se puede usar como punto de partida. Es decir, cambiar la posición del target para mover de sitio el quadrotor.

En el script por defecto (código [C2]) se puede ver que se utiliza la función `simGetObjectHandle` en la parte de configuración para coger el control de los diferentes objetos, entre ellos el target. Lo que faltaría es poder asignarle una nueva posición. Esto se puede conseguir con la función `simSetObjectPosition` a la que hay que pasarle 3 parámetros: la referencia del objeto que se quiere mover, respecto a qué coordenadas se va a mover (las propias o las de las escena) y la nueva posición. Por lo que de esta sencilla manera se puede mover el artefacto a donde sea necesario tan solo cambiando el objetivo de sitio.

5.1.2. Resultados y pruebas

Para este experimento bastaba con moverse en un único eje. El primer intento fue desplazarlo un metro. De esta forma el quadrotor se movía demasiado rápido. Lo siguiente

fue probar con un valor más pequeño: 0,001m. Con este valor la velocidad era más adecuada aunque un poco lenta. Por último se probó con 0,003m. En este punto la velocidad ya era adecuada por lo que este sería el punto de partida para el resto de experimentos.

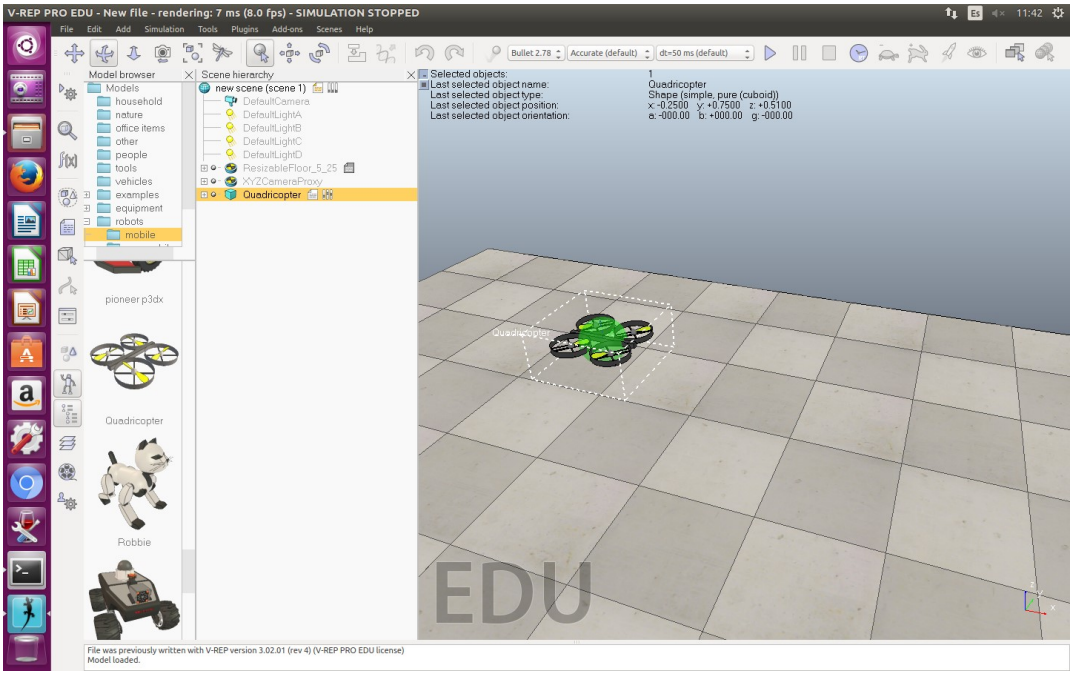

Figura 5.1 Pantalla primer experimento.

5.2. Quadrotor con cable

5.2.1. Descripción y preparación

En este segundo experimento se va a intentar probar cómo colgar la carga: si con un cable flexible o un cable rígido. Teniendo en cuenta los tipos de unión y objetos que ofrece el programa la única forma de crear una cuerda flexible es hacer una especie de cadena.

Para crear la cadena se van a usar varios trozos de tubo pequeños unidos con una articulación esférica para así imitar el movimiento de una cadena. Para ello, lo primero es ir al menú y darle “add->joint->spherical”. Después para añadir el cilindro “add->shape->cylinder”. Para tener un comportamiento lo más parecido posible a una cuerda hay que usar cilindros de un tamaño pequeño (0.001m, 0.001m, 0.002m).

Utilizando las herramientas del V-REP se coloca la articulación en un extremo del cilindro. Para que al final se comporten como un único objeto es necesario que, una vez colocados, se ponga el cilindro como hijo de la articulación, en caso contrario una vez iniciada la

simulación se separará el uno del otro. Ahora lo que hay que hacer es copiar esta estructura varias veces e ir colocando todas las parejas cilindro-articulación una detrás de la otra. Después se unirán el quadrotor y el cable con la articulación de la parte superior de la cuerda y se pondrá toda la cuerda como su hijo.

En cuanto al cable rígido el proceso es parecido. En este caso bastará con crear un cilindro alargado y fino (0.001m, 0.001m, 0.03m) y unirlo con una articulación esférica al igual que en el caso anterior.

5.2.2. Resultados y pruebas

Una vez está todo bien colocado, falta por comprobar cómo va a reaccionar al movimiento del quadrotor y a cualquier perturbación. Para lo primero basta con hacer que se mueva el quadrotor con ayuda del ratón. Lo primero que se puede observar es que el cable se mueve descontroladamente con el quadcopter en reposo. Esto se debe a las partículas que genera el quadcopter. Para quitarlas hay que acceder al panel de configuración de parámetros del script haciendo doble click en el icono junto al del script. En este panel hay que asignarle valor false al parámetro simulateParticles.

Una vez realizado el cambio, Se puede comprobar que el comportamiento es el esperado, es decir, la cuerda cuelga debajo del quadrotor deformandose de forma lógica al movimiento de este. Para lo segundo lo que se busca es provocar que la cuerda reciba un golpe. Para ello se utiliza un eje motorizado y una placa alargada. De esta forma en el escenario habrá un quadrotor con una cuerda colgando y una especie de aspa girando que la golpeara todo el tiempo. Al igual que pasaba con el vuelo del vehículo aquí también la cuerda se mueve de forma normal.

En el caso del cable rígido el resultado también es el esperado. El cable se mueve como si de un péndulo se tratase. Dado el éxito de ambos experimentos se esperara a colgar la carga para decidir cual utilizar.

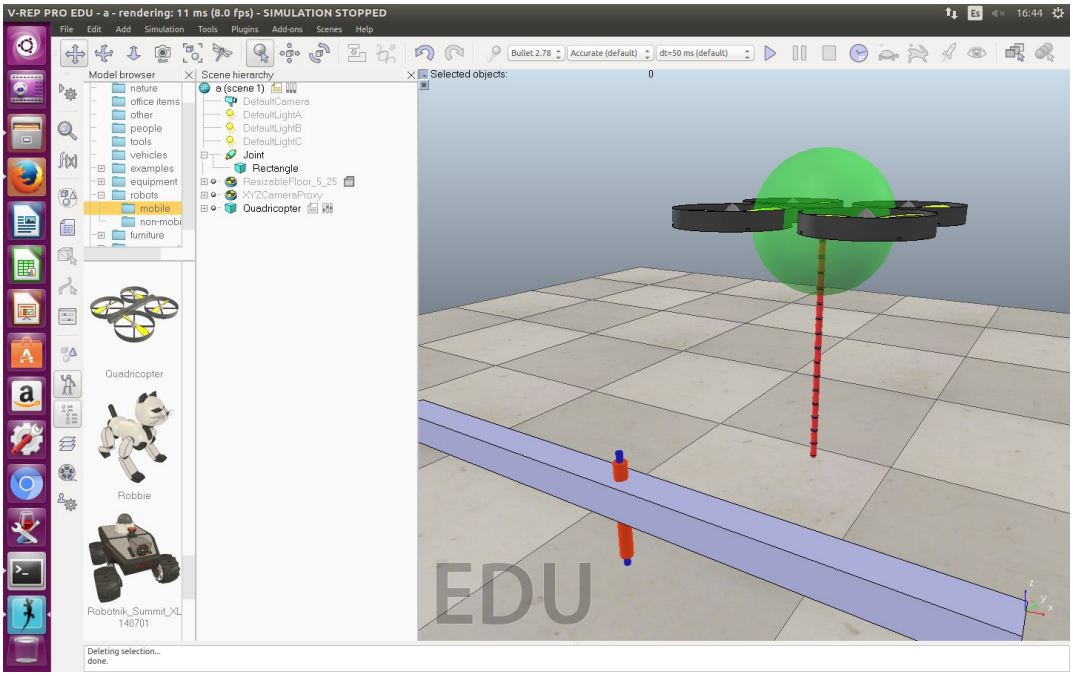

Figura 5.2 Pantalla segundo experimento con cuerda flexible.

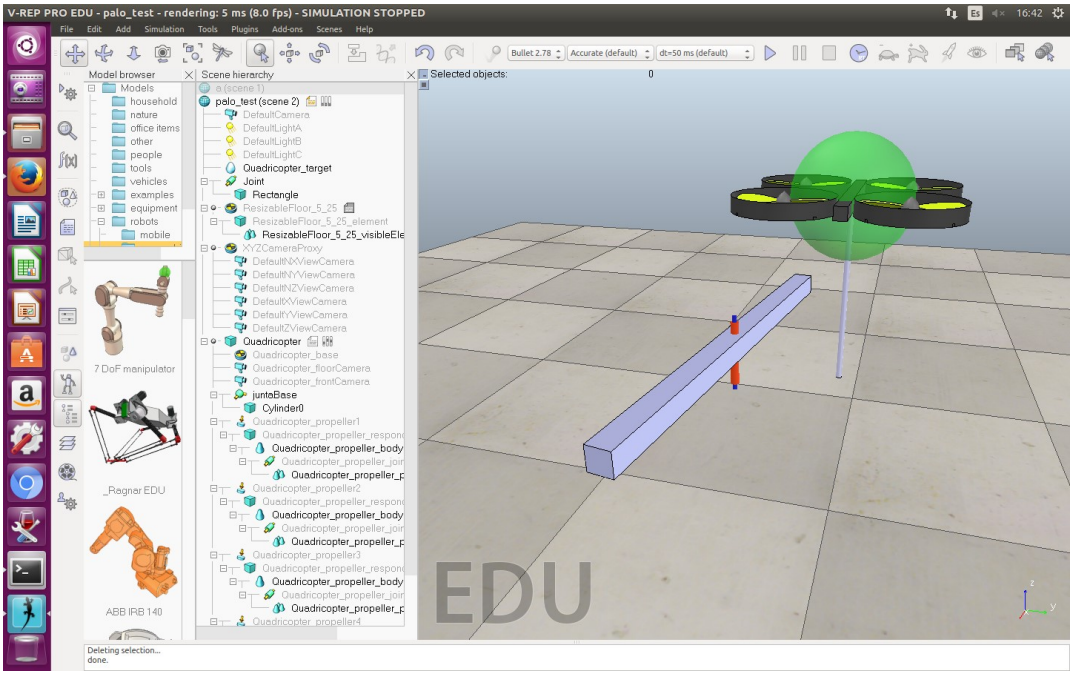

Figura 5.3 Pantalla segundo experimento con cuerda rigida.

5.3. Quadrotor con cable y carga

5.3.1. Descripción y preparación

Una vez probados los dos tipos de cables falta comprobar cual es su comportamiento cuando le colocamos una carga a cualquiera de ellos. La representación de la carga se

realizará como si fuera una caja. Para ello hay que proceder de la misma forma que para crear el cable no flexible pero en vez de seleccionar cilindro seleccionando cubo. Al igual que con el cilindro aquí también es necesario definir sus medidas (0.1m, 0.1m, 0.1m).

Para unir la caja a cualquiera de las cuerdas el procedimiento es el mismo que en el experimento anterior: una articulación esférica en la superficie de la caja y esa articulación unida al extremo libre de la cuerda, y una vez así, hacer la caja hijo del último cilindro. Si se pone la caja como hijo y después se intenta colocar, al ser parte ya de la colección, se mueve todo el quadrotor y no solo la caja.

5.3.2. Resultados y pruebas

Una vez todo colocado se puede comenzar la simulación. Como no se ha tocado ningún parámetro de la caja mas que sus medidas, aparece el primer problema: el peso. Tras iniciar la simulación el quadrotor no es capaz de alcanzar la altura objetivo debido al peso de la caja.

Para cambiar el peso de la caja, o el del cable si es necesario, hay que hacer doble click en el objeto,del cual se quiera cambiar el peso, en el panel de la jerarquía de la escena y pulsar el botón “show dynamic properties dialog”. En la venta que se abre ofrece la posibilidad de cambiar varios aspectos del objeto como la masa, el material, el momento de inercia… Para reducir la masa hay dos posibilidades: introducir manualmente un nuevo valor o darle al botón que pone “M=M/2” que lo que hace es reducir la masa a la mitad. Tras reducir el peso de la caja y del cable ocho veces sobre su valor inicial, se consigue que el robot alcance otra vez la altura del objetivo.

Para analizar el comportamiento en movimiento se utilizara el peso original y el nuevo. Con el peso original si desplazamos el objetivo a un punto lejano se aprecia como el quadrotor quiere dirigirse hacia el pero la caja no le deja. Pese a esto los dos cables no se comportan igual. Con el cable flexible se puede observar como el robot consigue acercarse más aunque el cable pierde parte de su integridad. Con el otro en cambio, la relación entre los objetos permanece intacta y el robot trata de arrastrar el cubo sin éxito. Con el peso modificado en cambio, el comportamiento de los dos es el esperado aunque cada uno con sus diferencias dadas las propiedades del cable en cuestión.

Para finalizar y tomar una decisión sobre que conexión utilizar realizaremos la misma prueba de golpeo que en el experimento anterior, y al igual que en el paso anterior se hará con diferentes masas. Cuando la caja es pesada se puede observar que, al no poder levantarla del suelo, el cable gira apoyado en la pala y con él el quadrotor. En cambio cuando es capaz de levantar la caja el golpe de la pala solo hace que el cable se mueva, cada cual a su manera.

Por lo tanto pese a que los dos cables funcionan bien cuando el peso de la caja es el óptimo para permitir el vuelo, el modelo flexible con la carga pesada falla. Esto hace que sea más lógico optar por el enlace rígido. Además, el penduleo de la carga con el flexible es más impredecible y difícil de controlar por lo que se optará por el cable rígido.

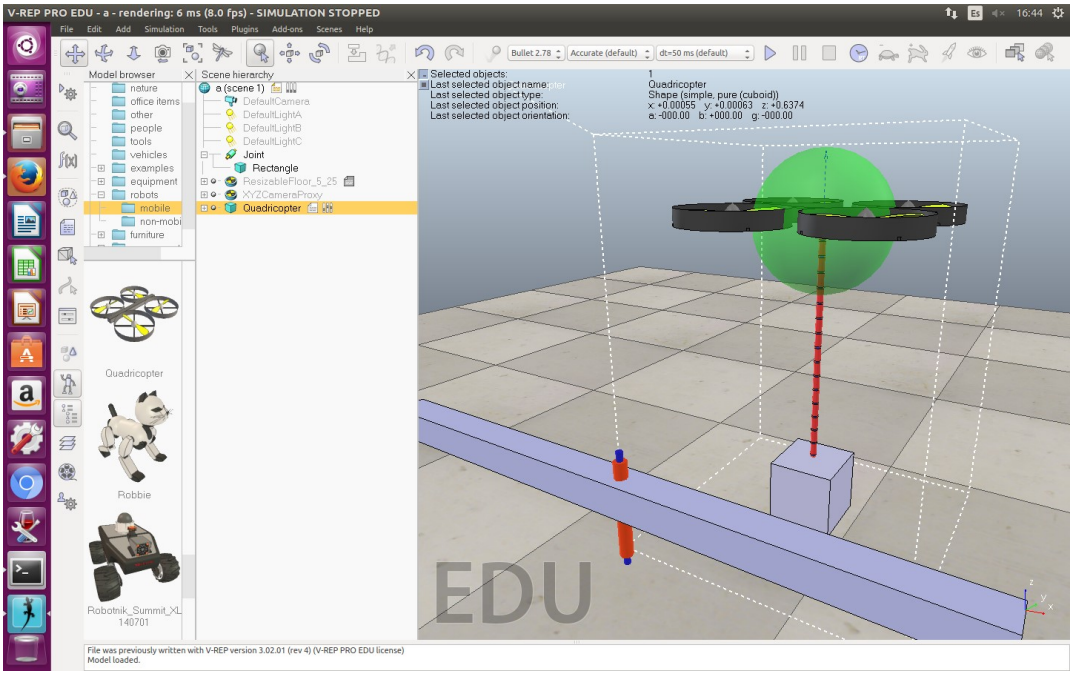

Figura 5.4 Pantalla segundo experimento con cuerda flexible y carga.

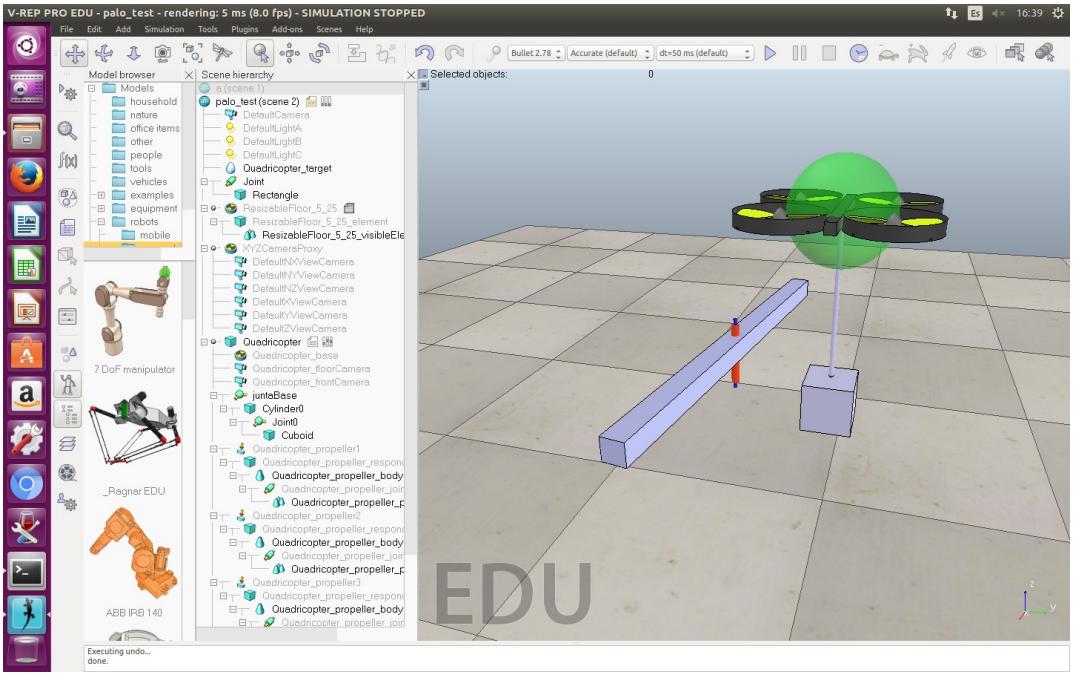

Figura 5.5 Pantalla segundo experimento con cuerda rígida y carga.

5.4. Quadrotor siguiendo un camino

5.4.1. Descripción y preparación

El objetivo de este experimento es controlar el quadrotor más allá del simple movimiento de un punto A a un punto B sin tener en cuenta el entorno. Por lo que seguirá un camino pintado en el suelo, realizando un vuelo autónomo ayudado por tres sensores de visión.

Para colocar los sensores simplemente hay que seleccionar "add->vision sensor-> orthographic" (existe la posibilidad de poner una cámara con proyección perspectiva pero para este experimento no se necesita). Utilizando las herramientas del programa se coloca justo en la base del robot pero sobresaliendo un poco para que no detecte el robot sino el suelo. Una vez hecho esto se ponen otros dos sensores idénticos, uno en cada lado, y se ponen los tres como hijos del quadrotor.

Para crear el camino el proceso es parecido al de añadir la cámara pero en este caso es "add->path", lo que abrirá un panel para la creación del mismo. Este permite dibujarlo con el ratón para darle cualquier forma. Una vez conformes con lo dibujado, se le da a que cierre el circuito. En caso de verlo necesario se pueden suavizar las curvas utilizando parámetros de interpolación desde el mismo panel.

Una vez en este punto hay que cambiar el script para que en vez de solo seguir al objetivo siga la línea marcada en el suelo. Para ello se añade esto al código (Código [C3]).

Como el objetivo es simplemente familiarizarnos con el movimiento autónomo del quadrotor, el algoritmo es sencillo:

```
1. Si veo la línea -> avanzar
2. Sino :
3.          Retroceder el ultimo avance
4.          Si el sensor izquierdo no ve línea -> rotar en el
   sentido de las agujas del reloj
5.          Sino -> girar en el sentido de las agujas del reloj
```

5.4.2. Resultados y pruebas

La cantidad de avance será la analizada en el primer experimento. Teniendo en cuenta que la corrección se hace retrocediendo, se usara la misma cantidad.

La primera prueba se hizo con un único sensor. En ese caso en vez de girar se desplazaba hacia la izquierda, ya que se daba la vuelta al circuito en sentido contrario a las agujas del reloj. De esta forma no se conseguía dar una vuelta completa por lo que se opto por poner más sensores. Con tres y girando el quadrotor se lograba el objetivo. En cuanto al giro, se reajusta la orientación del quadrotor con la función `simSetObjectOrientation`. Para ello hay que pasarle los mismos parámetros que para desplazar un objeto. En cuanto a la cantidad de giro el valor utilizado es el mismo que el del avance 0,003.

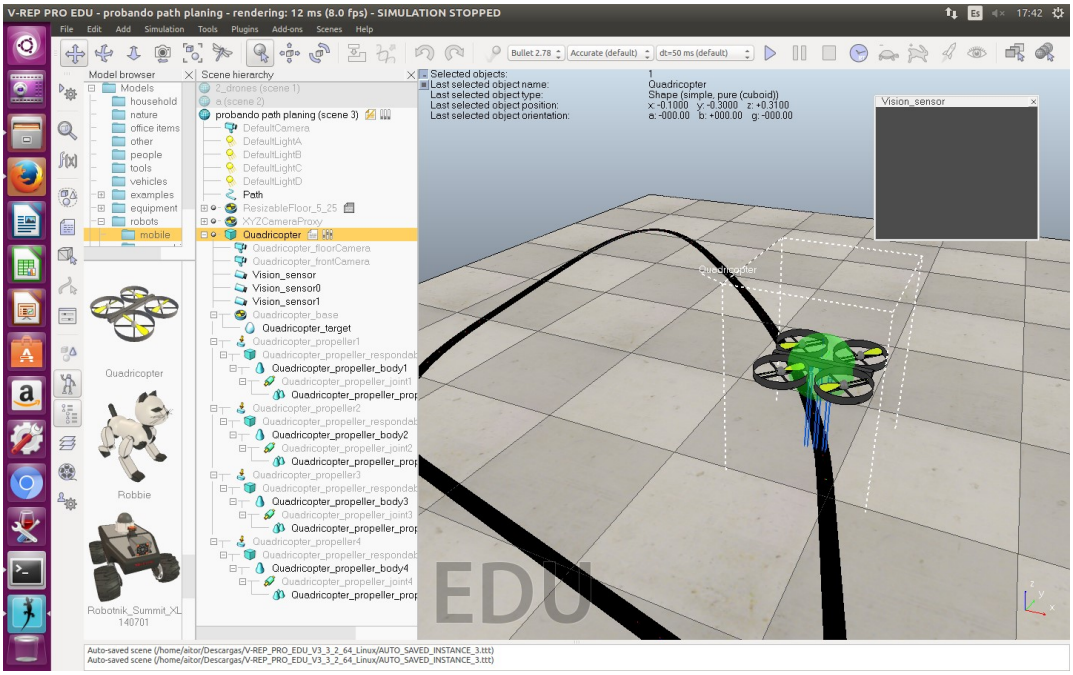

Figura 5.6 Pantalla quadrotor siguiendo camino.

5.5. Dos quadrotors sincronizados

5.5.1. Descripción y preparación

En este experimento lo que se pretende es conseguir mover dos qudrotors de forma sincronizada. Para lograrlo se va a implementar un comportamiento “follow the leader”, es decir, uno de los dos se moverá de forma independiente, será al que habrá que dar las órdenes, y l otro se limitará a seguirlo. Para ello el quadrotor perseguidor irá equipado con un sensor de visión perspectiva, el cual se coloca de la misma forma que los ortográficos utilizados previamente, pero en este caso habrá que hacer algunos ajustes. Utilizando las herramientas del programa se coloca un robot detrás del otro y después haciendo doble click en el sensor se configurara tanto la resolución (256x256) como el “far plane” y “near plane” (2m, 0.01m).

Lo que falta por hacer es conseguir que el sensor perciba al otro robot. Para esta tarea necesitaremos aplicarle varios filtros al sensor. Esto se hace volviendo al panel anterior y

pulsando “show filter dialog”. Una vez ahí, en la lista de filtros hay que seleccionar “selective color on work image” y “blob detection on work image”. El primero permite filtrar la imagen en base a un color. Haciendo doble click sobre el filtro se puede elegir el color y la desviación permitida. El otro lo que hace es detectar la cantidad de objetos que ve y devuelve información útil sobre ellos, como el tamaño o la posición. Lo que se pretende hacer es configurar el filtro de color de tal forma que solo filtre un único color y pintar el quadrotor líder de ese color, para así detectarlo con facilidad.

Para cambiar de color el quadrotor lo primero es hacer doble click sobre el y darle al botón “adjust color” y se elige el mismo que en el paso anterior. Con esto habrá cambiado solo el color del eje central por lo que, faltaría desplegar los complementos del quadrotror y hacer click sobre los cuatro motores (proppeller) y repetir este proceso. Ya con todo listo el siguiente paso es cambiar el script del quadcopter perseguidor. Al igual que para seguir el camino en este caso también se utilizaran los datos de la cámara para cambiar el objetivo de sitio y que este se mueva hacia donde esté el líder. Para recoger los datos de los filtros lo primero es coger el control del sensor como en los anteriores experimentos. Después, hace falta recoger los datos que nos devuelve el filtro de detección de formas, para ello se utiliza la función `simReadVisionSensor`. Como parámetros hace falta pasarle el sensor sobre el que se quieran recoger datos. Como resultado devuelve mínimo dos variables, un número que indica si ha habido algún error o no y después varios arrays de valores por cada filtro que se haya aplicado. Por lo que en este caso devolverá dos. La información necesaria estará en el segundo array ya que primero se aplica el filtro de color y luego el de detección. Dentro de este array habrá que fijarse solo en la información de las posiciones cinco y tres. Estas indican la ubicación de lo que detecta en el eje “x” de la proyección y el tamaño de lo que detecta respectivamente.

Estos datos van a ser utilizados para saber si el robot que va delante se ha desplazado lateralmente, si se aleja o acerca. Dependiendo de la orientación de los robots habrá que utilizar un dato para reubicar el objetivo en el eje “x” y el otro en el “y” o viceversa. La idea para la reubicación del objetivo es mantenerlo siempre en el centro de la cámara y con el mismo tamaño por lo que habrá que hacer una corrección proporcional de la posición actual, es decir, cuanto más lejos este de la posición deseada o mas diferencia de tamaño haya, que la velocidad de corrección sea más rápida. Para ello se cogerán los valores deseados en estático y después se utilizaran para el control. La obtención de los valores se hará usando la consola como ayuda. En el caso de la posición en el eje “x” no hará falta dado que la imagen resultante del sensor es un cuadro de 1x1 por lo que lo ideal sería tenerlo siempre con un valor de 0,5.

Para obtener el dato del tamaño óptimo se va utilizar una consola auxiliar la cual se puede visualizar usando el comando `simAuxiliaryCosoleOpen`, al que se le pasan como parámetros el nombre de la ventana y el numero de líneas a visualizar. Después para visualizar datos en ella se utilizara el comando `simAuxiliaryConsolePrint`, al que se le pasarán como parámetros la consola y el texto con sus respectivos atributos. Una vez así

bastará con ejecutar la simulación colocar los robots a una distancia adecuada y apuntar el dato. Ya con estos datos solo quedaría añadir el control al código del robot perseguidor.

5.5.2. Resultados y pruebas

Tras hacer las primeras pruebas se puede comprobar que una vez el quadrotor líder está en movimiento la cámara no siempre lo detecta como un solo objeto lo cual hace que el algoritmo de control falle. Para corregir este error se volverán a poner los rotores del color original dejando solamente pintado el eje central del robot. De esta forma se puede observar como el robot perseguidor es capaz de seguir perfectamente al robot líder.

Corregido ese comportamiento se observa que cuando el perseguidor pierde de vista al líder este se vuelve loco. Para corregirlo se crean unas variables auxiliares para que cuando lo pierda de vista calcule la corrección co la última posición.

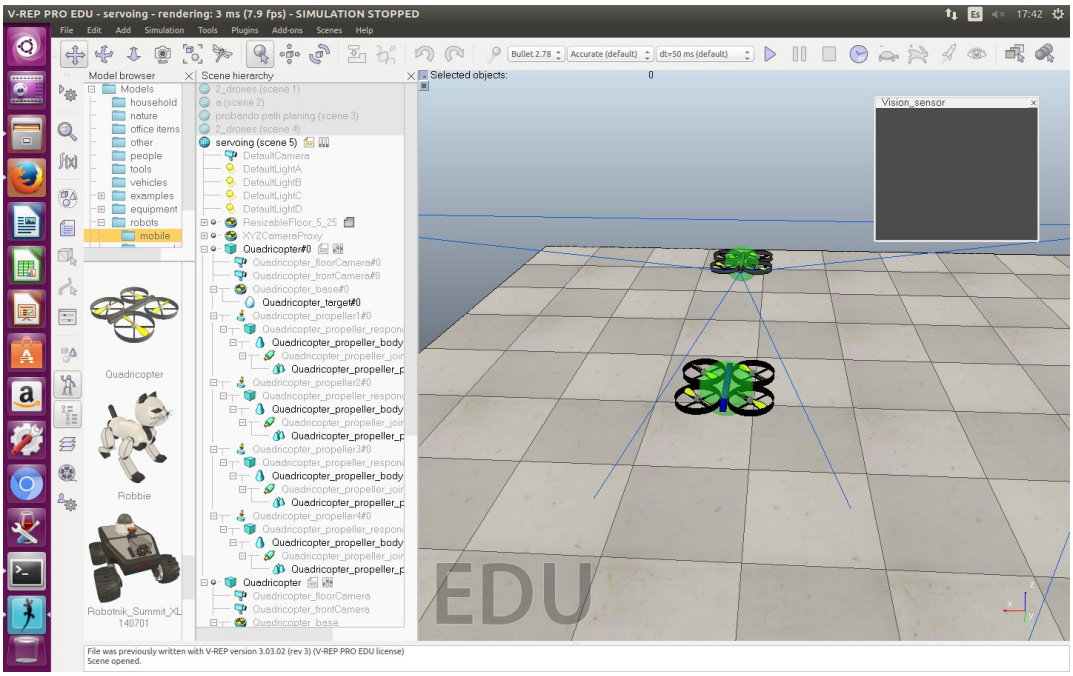

Figura 5.7 Pantalla dos quadrotors sincronizados

5.6. Dos quadrotors con cable y carga

5.6.1. Descripción y preparación

El objetivo de este experimento es unir todos los anteriores, es decir, construir un escenario en el que dos robots sean capaces de transportar una carga colgada con cables rígidos alrededor de un circuito de forma independiente basándose solo en las lecturas de sus sensores.

Para ello habrá que añadir al escenario dos quadrotors, un path y una carga. En este caso para facilitar la unión entre los quadrotors y la carga se le dará forma rectangular en vez de cubica. La forma de realizar la unión es la misma que en el experimento anterior. Dado que el objetivo es que los dos quadrotors lleven la caja, habrá que conectar los objetos siguiendo esta estructura:

Dron1->cable1->carga->cable2->dron2

Siendo el dron1 el padre de todos los objetos. En cuanto al equipamiento del robot delantero, el líder, llevará tres sensores ortográficos para controlar la línea y su eje pintado de azul. Mientras que el segundo, el perseguidor, llevará un sensor en su parte delantera para controlar la posición del líder. Una vez se pone en marcha utilizando la configuración creada en los anteriores experimentos, se puede percibir el primer problema: el algoritmo de seguimiento del robot es óptimo para un espacio sin obstáculos, en el que los quadrotors se moven en línea recta. ¿Pero qué pasa si se encuentran con un obstáculo? O aplicado al escenario ¿Qué pasa si quiere realizar una curva?

Por ello hace falta que si el robot delantero gira el segundo también lo haga. Dado que la cámara lo recoge todo en dos dimensiones es difícil predecir cuando el robot delantero está realizando un giro. Pero tras consultar las funciones que ofrece V-REP se encuentra una que permite comunicar un script con otro, por lo que, el primer robot puede comunicarle al segundo cuanto tiene que girar y cuando. Para ello habrá que modificar tanto el script de seguimiento de línea como el de seguimiento de robot.

En el primero el cambio es sencillo: utilizar el comando `simSetFloatSignal` para mandar la información, al que se le pasa como parámetros el nombre del dato y el dato. Para conseguir un correcto funcionamiento del segundo robot cuando el primero gire le mandara cuanto tiene que girar y en los momentos de seguir recto, le mandará que el giro es cero. En el según lo que hay que hacer es recoger esta información. Lo primero será usar el comando `simGetFloatsignal` al que se le pasa como parámetro el nombre del parámetro que se espera recibir. Una vez recibido el dato lo primero será girar el robot y luego corregir la posición.

5.6.2. Resultados y pruebas

Una vez solucionado el problema de giro se percibe otro problema. Cuando el líder gira el motor tapa el eje por lo que el perseguidor le pierde la pista. Para solucionarlo la idea es que mientras se gira el perseguidor no se preocupe de corregir la posición en los ejes verticales. El primero de los quadrotors o avanza o gira por lo tanto no es necesario que el segundo se preocupe de corregir su posición hasta que este esté avanzando en línea recta. Código final de los dos quadrotors(código [C3] y [C4])

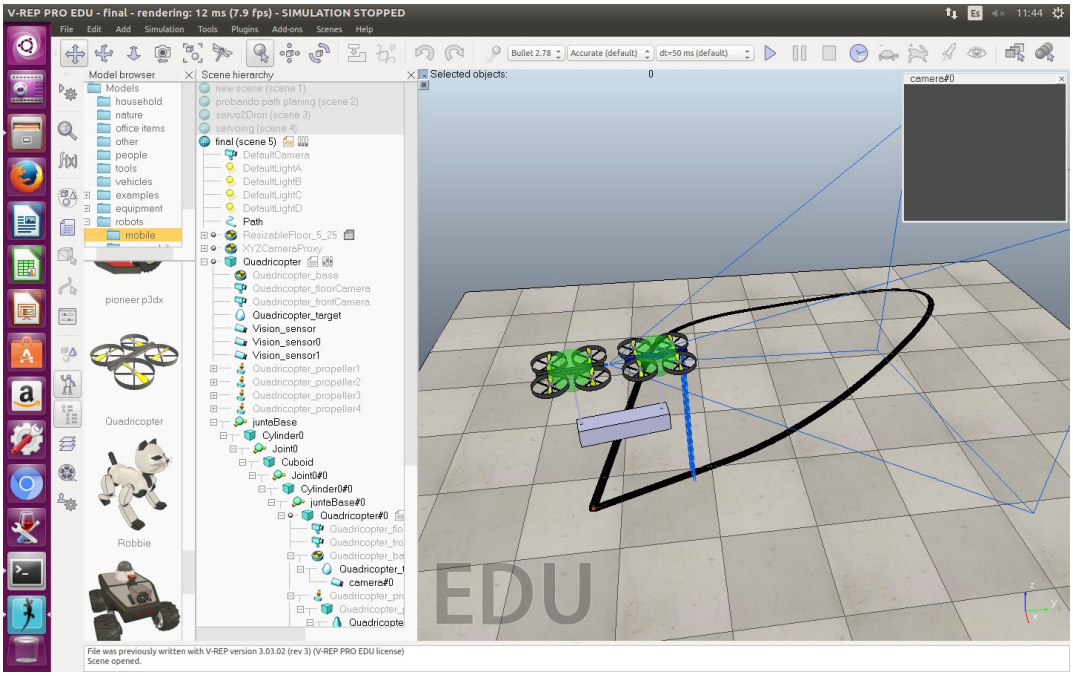

Figura 5.8 Pantalla experimento final.

6

Capitulo 6: Conclusiones y trabajo futuro

Se ha comprobado que con una configuración básica se puede lograr de manera simple que dos quadcopters transporten un objeto a lo largo de un camino marcado. Aún se podrían hacer varios y muy interesantes experimentos como continuación de este proyecto, puesto que sienta las bases para trabajos futuros.

Analizando el último experimento se ha observado que, si V-REP ofrece la posibilidad de comunica los dos scripts, sería posible que el segundo quadrotor se moviera exactamente igual que el primero. Además esto nos ahorraría el tener que utilizar el sensor de visión. Pero esto haría totalmente dependiente al segundo quadrotor. En caso de querer llevar el experimento a la vida real y optar por una estrategia en la que todos tengan más libertad o al menos capacidad de decisión, para evitar accidentes por ejemplo, habría que equiparle un sensor. De esta manera en caso de llegar a ese punto ya se ha conseguido experiencia en el manejo de sensores.

En la vida real, salvo volando a mucha altura, es muy posible que se encuentren con obstáculos en el camino marcado. Siendo eso así un siguiente paso sería añadir al experimento final un algoritmo, ya sea uno propio o alguno ya existente, para evitar las colisiones.

Otro aspecto en que profundizar podría ser el estudio del comportamiento de la carga. En este trabajo no hemos intentado controlar el estado o la posición de la carga, pero si esto se llevará a la vida real, sería importante tener en cuenta el impacto que van a tener en la caja los movimientos realizados por los robots. En el caso de transportar bloque sólidos, como un trozo de madera o un yunque, no sería necesario. Pero si se usa para enviar paquetes, en los cuales pueda ir algo frágil, habría que tener en cuenta que no se puede romper.

Además de esto, bien añadiendo más robots o bien manteniendo solo dos, sería interesante analizar si la filosofía follow de leader es la más eficaz o si habría que dar más libertad al quadrotor esclavo. De esta forma a la hora de hacer maniobras complejas con giros o para pasar por zonas estrechas, se podría ir cambiando la formación de los quadrotors para conseguir un movimiento más eficaz.

Y el paso más importante que quedaría por hacer, teniendo en cuenta el principal argumento de las anteriores propuestas, sería el hecho de llevar esta simulación a la vida real. Todo en el mundo avanza y se hace con el objetivo de facilitarnos la vida por lo que, robots que transportan cosas de forma autónoma sería un gran avance de cara a esa comodidad constante que tanto gusta al ser humano.

7

Capitulo 7: Referencias

- [Art] Articulo “Modelling and Control for Cooperative Aerial Transportation of a Payload as a Reconfigurable Cable-Driven Parallel Robot” de Carlo Masone, Paolo Stegagno y Heinrich H. Bülthoff.

https://drive.google.com/file/d/0B17jceAMNr9dNTZmNTNJVDlSVHM/view?usp=sharing

- [Wi1] VANT Wikipedia:

https://es.wikipedia.org/wiki/Veh%C3%ADculo_a%C3%A9reo_no_tripulado

- [Wi2] Ángulos de navegación:

https://es.wikipedia.org/wiki/%C3%81ngulos_de_navegaci%C3%B3n

- [Wi3] Matlab:

https://es.wikipedia.org/wiki/MATLAB

- [Wi4] Mecanica quadcopter:

https://en.wikipedia.org/wiki/Quadcopter

- [Sim1] Simulink:

https://es.mathworks.com/products/simulink.html

- [Vrep] Opciones de programación

http://www.coppeliarobotics.com/helpFiles/

- [Sym] SYMA:

http://www.symatoys.com/

- [Dj] DJI

http://www.dji.com/es

Bibliografía

[1] Tutorial path 1. https://www.youtube.com/watch?v=OfpB87pRoUk

[2] Tutorial path 2. https://www.youtube.com/watch?v=OfpB87pRoUk

[3] Manual de VREP https://www.youtube.com/watch?v=OfpB87pRoUk

[4] Tutorial servoing. https://www.youtube.com/watch?v=kOjQRYmeX_o

Anexo A: Código

[C1] Código escena:

```
1. -- This is the main script. The main script is not supposed to
   be modified,
2. -- unless there is a very good reason to do it.
3. -- The main script is called at each simulation pass. Without
   main script,
4. -- there is no real simulation (child scripts are not called
   either in that case).
5. -- A main script marked as "default" (this is the default
   case) will use the
6. -- content of following file: system/dltmscpt.txt. This allows
   your old simulation
7. -- scenes to be automatically also using newer features,
   without explicitely coding
8. -- them. If you modify the main script, it will be marked as
   "customized", and you
9. -- won't benefit of that automatic forward compatibility
   mechanism.
10.
11.      -- DO NOT WRITE CODE OUTSIDE OF THE if-then-end SECTIONS
   BELOW!! (unless the code is a function definition)
12.
13.      -- Initialization part (executed just once, at
   simulation start) ---------
14.      if (sim_call_type==sim_mainscriptcall_initialization) th
   en
15.          simOpenModule(sim_handle_all)
16.          simHandleGraph(sim_handle_all_except_explicit,0)
17.      end
18.      ----------------------------------------------------------
   ------------------
19.
20.      -- Regular part (executed at each simulation step) -----
   ------------------
21.      if (sim_call_type==sim_mainscriptcall_regular) then
22.          -- "Actuation"-part --
23.          simResumeThreads(sim_scriptthreadresume_default)
24.
   simResumeThreads(sim_scriptthreadresume_actuation_first)
25.          simLaunchThreadedChildScripts()
26.          simHandleChildScripts(sim_childscriptcall_actuation)
27.
   simResumeThreads(sim_scriptthreadresume_actuation_last)
28.
   simHandleCustomizationScripts(sim_customizationscriptcall_simu
   lationactuation)
29.          simHandleModule(sim_handle_all,false)
30.
   simHandleJoint(sim_handle_all_except_explicit,simGetSimulation
   TimeStep()) -- DEPRECATED
```

```
31.
   simHandlePath(sim_handle_all_except_explicit,simGetSimulationT
   imeStep()) -- DEPRECATED
32.          simHandleMechanism(sim_handle_all_except_explicit)
33.          simHandleIkGroup(sim_handle_all_except_explicit)
34.          simHandleDynamics(simGetSimulationTimeStep())
35.          simHandleVarious()
36.          simHandleMill(sim_handle_all_except_explicit)
37.
38.          -- "Sensing"-part --
39.          simHandleCollision(sim_handle_all_except_explicit)
40.          simHandleDistance(sim_handle_all_except_explicit)
41.
   simHandleProximitySensor(sim_handle_all_except_explicit)
42.
   simHandleVisionSensor(sim_handle_all_except_explicit)
43.
   simResumeThreads(sim_scriptthreadresume_sensing_first)
44.          simHandleChildScripts(sim_childscriptcall_sensing)
45.
   simResumeThreads(sim_scriptthreadresume_sensing_last)
46.
   simHandleCustomizationScripts(sim_customizationscriptcall_simu
   lationsensing)
47.          simHandleModule(sim_handle_all,true)
48.
   simResumeThreads(sim_scriptthreadresume_allnotyetresumed)
49.
   simHandleGraph(sim_handle_all_except_explicit,simGetSimulation
   Time()+simGetSimulationTimeStep())
50.      end
51.      ------------------------------------------------------------
   ------------------
52.
53.      -- Clean-up part (executed just once, before simulation
   ends) ------------
54.      if (sim_call_type==sim_mainscriptcall_cleanup) then
55.          simResetMilling(sim_handle_all)
56.          simResetMill(sim_handle_all_except_explicit)
57.          simResetCollision(sim_handle_all_except_explicit)
58.          simResetDistance(sim_handle_all_except_explicit)
59.
   simResetProximitySensor(sim_handle_all_except_explicit)
60.          simResetVisionSensor(sim_handle_all_except_explicit)
61.          simCloseModule(sim_handle_all)
62.      end
63.      ------------------------------------------------------------
   ------------------
64.
65.      -- By default threaded child scripts switch back to the
   main thread after 2 ms. The main
66.      -- thread switches back to a threaded child script at
   one of above's "simResumeThreads"
67.      -- location
```

[C2] Código por defecto del quadrotor:

```
1. if (sim_call_type==sim_childscriptcall_initialization) then
2.     -- Make sure we have version 2.4.13 or above (the
   particles are not supported otherwise)
3.     v=simGetInt32Parameter(sim_intparam_program_version)
4.     if (v<20413) then
5.         simDisplayDialog('Warning','The propeller model is
   only fully supported from V-REP version 2.4.13 and
   above.&&nThis simulation will not run as
   expected!',sim_dlgstyle_ok,false,'',nil,{0.8,0,0,0,0,0})
6.     end
7.
8.     -- Detatch the manipulation sphere:
9.     targetObj=simGetObjectHandle('Quadricopter_target')
10.          simSetObjectParent(targetObj,-1,true)
11.
12.          -- This control algo was quickly written and is
   dirty and not optimal. It just serves as a SIMPLE example
13.
14.          d=simGetObjectHandle('Quadricopter_base')
15.
16.
   particlesAreVisible=simGetScriptSimulationParameter(sim_handle
   _self,'particlesAreVisible')
17.
   simSetScriptSimulationParameter(sim_handle_tree,'particlesAreV
   isible',tostring(particlesAreVisible))
18.
   simulateParticles=simGetScriptSimulationParameter(sim_handle_s
   elf,'simulateParticles')
19.
   simSetScriptSimulationParameter(sim_handle_tree,'simulateParti
   cles',tostring(simulateParticles))
20.
21.          propellerScripts={-1,-1,-1,-1}
22.          for i=1,4,1 do
23.
   propellerScripts[i]=simGetScriptHandle('Quadricopter_propeller
   _respondable'..i)
24.          end
25.
   heli=simGetObjectAssociatedWithScript(sim_handle_self)
26.
27.          particlesTargetVelocities={0,0,0,0}
28.
29.          pParam=2
30.          iParam=0
31.          dParam=0
32.          vParam=-2
33.
34.          cumul=0
```

```
35.          lastE=0
36.          pAlphaE=0
37.          pBetaE=0
38.          psp2=0
39.          psp1=0
40.
41.          prevEuler=0
42.
43.
44.
   fakeShadow=simGetScriptSimulationParameter(sim_handle_self,'fa
   keShadow')
45.          if (fakeShadow) then
46.
   shadowCont=simAddDrawingObject(sim_drawing_discpoints+sim_draw
   ing_cyclic+sim_drawing_25percenttransparency+sim_drawing_50per
   centtransparency+sim_drawing_itemsizes,0.2,0,-1,1)
47.          end
48.
49.          -- Prepare 2 floating views with the camera views:
50.
   floorCam=simGetObjectHandle('Quadricopter_floorCamera')
51.
   frontCam=simGetObjectHandle('Quadricopter_frontCamera')
52.          floorView=simFloatingViewAdd(0.9,0.9,0.2,0.2,0)
53.          frontView=simFloatingViewAdd(0.7,0.9,0.2,0.2,0)
54.          simAdjustView(floorView,floorCam,64)
55.          simAdjustView(frontView,frontCam,64)
56.      end
57.
58.      if (sim_call_type==sim_childscriptcall_cleanup) then
59.          simRemoveDrawingObject(shadowCont)
60.          simFloatingViewRemove(floorView)
61.          simFloatingViewRemove(frontView)
62.      end
63.
64.      if (sim_call_type==sim_childscriptcall_actuation) then
65.          s=simGetObjectSizeFactor(d)
66.
67.          pos=simGetObjectPosition(d,-1)
68.          if (fakeShadow) then
69.              itemData={pos[1],pos[2],0.002,0,0,1,0.2*s}
70.              simAddDrawingObjectItem(shadowCont,itemData)
71.          end
72.
73.          -- Vertical control:
74.          targetPos=simGetObjectPosition(targetObj,-1)
75.          pos=simGetObjectPosition(d,-1)
76.          l=simGetVelocity(heli)
77.          e=(targetPos[3]-pos[3])
78.          cumul=cumul+e
79.          pv=pParam*e
80.          thrust=5.335+pv+iParam*cumul+dParam*(e-
   lastE)+l[3]*vParam
81.          lastE=e
82.
83.          -- Horizontal control:
84.          sp=simGetObjectPosition(targetObj,d)
85.          m=simGetObjectMatrix(d,-1)
86.          vx={1,0,0}
```

```
87.          vx=simMultiplyVector(m,vx)
88.          vy={0,1,0}
89.          vy=simMultiplyVector(m,vy)
90.          alphaE=(vy[3]-m[12])
91.          alphaCorr=0.25*alphaE+2.1*(alphaE-pAlphaE)
92.          betaE=(vx[3]-m[12])
93.          betaCorr=-0.25*betaE-2.1*(betaE-pBetaE)
94.          pAlphaE=alphaE
95.          pBetaE=betaE
96.          alphaCorr=alphaCorr+sp[2]*0.005+1*(sp[2]-psp2)
97.          betaCorr=betaCorr-sp[1]*0.005-1*(sp[1]-psp1)
98.          psp2=sp[2]
99.          psp1=sp[1]
100.
101.         -- Rotational control:
102.         euler=simGetObjectOrientation(d,targetObj)
103.         rotCorr=euler[3]*0.1+2*(euler[3]-prevEuler)
104.         prevEuler=euler[3]
105.
106.         -- Decide of the motor velocities:
107.         particlesTargetVelocities[1]=thrust*(1-
    alphaCorr+betaCorr+rotCorr)
108.         particlesTargetVelocities[2]=thrust*(1-alphaCorr-
    betaCorr-rotCorr)
109.         particlesTargetVelocities[3]=thrust*(1+alphaCorr-
    betaCorr+rotCorr)
110.
    particlesTargetVelocities[4]=thrust*(1+alphaCorr+betaCorr-
    rotCorr)
111.
112.         -- Send the desired motor velocities to the 4
    rotors:
113.         for i=1,4,1 do
114.
    simSetScriptSimulationParameter(propellerScripts[i],'particleV
    elocity',particlesTargetVelocities[i])
115.         end
116.     end
```

[C3] Código final seguimiento línea:

```
1.          result, data=simReadVisionSensor(sensor)
2.          result0, data0=simReadVisionSensor(sensor0)
3.          result1, data1=simReadVisionSensor(sensor1)
4.          if(data[11]<0.2)then
5.          position={-0.003,0,0}
6.          orientation={0,0,0}
7.          simSetFloatSignal("orienZ",0)
8.          else
9.          position={0,0,0}
10.              if(data1[11]>=0.2)then
11.                      orientation={0,0,0.003}
12.                      simSetFloatSignal("orienZ",0.003)
13.              else
14.                      orientation={0,0,-0.003}
15.                      simSetFloatSignal("orienZ",-0.003)
16.              end
```

```
17.             end
18.
   simSetObjectPosition(targetObj,targetObj,position)
19.
   simSetObjectOrientation(targetObj,targetObj,orientation)
```

[C4] Código final quadrotor perseguidor:

```
1.          result,pack1,pack2=simReadVisionSensor(camera)
2.
3.          if(pack2[5]~=nil)then xval=pack2[5]
4.          else
5.          xval=prevxval
6.          end
7.          if(pack2[3]~=nil)then size=pack2[3]
8.          else
9.          size=prevsize
10.             end
11.
12.             orienZ=simGetFloatSignal("orienZ")
13.             orientation={0,0,orienZ}
14.
   simSetObjectOrientation(targetObj,targetObj,orientation)
15.             posy=0.1*(xval-0.5)
16.             posx=0.5*(size-0.009)
17.
18.             position={posx,posy,0}
19.
     if(orienZ==0)then simSetObjectPosition(targetObj,targetObj,p
   osition)end
20.
21.
22.             --previous values
23.             prevxval=xval
24.             prevsize=size
```

Anexo B: Videos

1. https://drive.google.com/file/d/0B8Fsy95oLgZIYW41Q3ZnUVgyN0k/view?usp=sharing
2. https://drive.google.com/file/d/0B8Fsy95oLgZIb3BsZnNyWmFKaDA/view?usp=sharing
3. https://drive.google.com/file/d/0B8Fsy95oLgZISEVGNFhZVTcxdGc/view?usp=sharing
4. https://drive.google.com/file/d/0B8Fsy95oLgZIUEZWWVk4MVVoNU0/view?usp=sharing
5. https://drive.google.com/file/d/0B8Fsy95oLgZIcUR2Z0NHMVNPM2c/view?usp=sharing
6. https://drive.google.com/file/d/0B8Fsy95oLgZIZjlJRW9vcW1HQUU/view?usp=sharing