UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Simulación en entornos con robots manipuladores móviles

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Carlos Martínez Navarro

Director: Enrique Jorge Bernabeu Soler

Curso académico: 2014-2015

Resumen

El objetivo del presente proyecto era realizar una plataforma que permitiera simular el comportamiento de un robot manipulador móvil en un entorno con objetos que deberá manipular y esquivar, utilizando el software VREP (`http://www.coppeliarobotics.com`) en su versión gratuita PRO EDU V3.2.1. Para ello, se ha creado el manipulador móvil a partir de dos modelos del programa y se ha implementado un controlador para el robot móvil y otro para el manipulador.

Este proyecto puede utilizarse para la prueba de distintos módulos que ofrece VREP, así como para modificar y añadir otros modelos de robots y entornos diferentes donde realizar distintas pruebas.

Palabras clave: robótica, manipulador móvil, VREP, simulación

Índice general

1. Introducción **6**

1.1. Contenido de la memoria 7

1.2. Motivación 8

2. Desarrollo del proyecto **9**

2.1. Contexto 9

2.2. Herramientas utilizadas 10

2.2.1. V-REP(*Virtual Robot Experimentation Platform*) . . . 11

2.2.1.1. Interfaz de usuario: vista principal 12

2.2.1.2. Objetos de la escena(*Scene objects*) 14

2.2.1.3. Jerarquía de escenas(*Scene hierarchy*) 15

2.2.1.4. Escritura de código: *scripts* embebidos 16

2.3. Diseño del entorno de simulación virtual 17

2.3.1. Guía de configuración de objetos del manipulador móvil 17

2.3.2. Guía de configuración de objetos de escenario 24

2.3.3. Ampliación del diseño principal 26

2.4. Implementación de control del manipulador móvil 28

2.4.1. Programación de la base móvil 28

2.4.1.1. Planteamiento inicial 28
2.4.1.2. Cálculo de trayectorias: *Path Planning module* 30
2.4.1.3. Scripts asociados 34
2.4.2. Programación del manipulador 38
2.4.2.1. Planteamiento inicial 38
2.4.2.2. Scripts asociados 39

3. Conclusiones **43**

4. Mejoras y trabajos futuros **44**

Apéndices **46**

A. Código de los *scripts* **47**

A.1. Código del *script* asociado al robot móvil principal 48
A.2. Código del *script* asociado al manipulador principal 52
A.2.1. Código del *script* asociado a la pinza del manipulador principal 55
A.3. Código del *script* asociado al robot móvil duplicado 56
A.4. Código del *script* asociado al manipulador duplicado 60
A.4.1. Código del *script* asociado a la pinza del manipulador duplicado 62

Índice de figuras

2.1. Manipuladores móviles en diferentes áreas de trabajo 10
2.2. Software de simulación robótica VREP 11
2.3. Elementos de la interfaz de usuario 13
2.4. Diferentes tipos de objetos y su representación 14
2.5. Scene Hierarchy 15
2.6. Scene Hierarchy: unión manipulador y robot móvil 18
2.7. Vista en VREP del cuadro *Scene object propierties* 19
2.8. Vista en VREP del cuadro *Scene object propierties:joint* . . . 21
2.9. Vista en VREP del cuadro *Joint Dynamic Propierties* 22
2.10. Vista en VREP con configuración de objetos realizada 24
2.11. Vista en VREP del cuadro *Object/item position/orientation* . 25
2.12. Vista en VREP con configuración de objetos realizada 26
2.13. Vista en VREP con configuración y ampliación de objetos realizada 28
2.14. Modelo cinemático configuración diferencial de robot móvil . . 29
2.15. Vistas en VREP de *Scene object propierties* 32
2.16. Vista en VREP de la escena con todos los *Paths Planning* configurados 34

2.17. Semejanza de un brazo manipulador con la anatomía humana 38

2.18. Vista en VREP durante la simulación después de toda la configuración 42

A.1. 48

A.2. 49

A.3. 50

A.4. 51

A.5. 52

A.6. 53

A.7. 54

A.8. 55

A.9. 56

A.10. 57

A.11. 58

A.12. 59

A.13. 60

A.14. 61

A.15. 62

Capítulo 1

Introducción

El objetivo del presente proyecto era realizar una plataforma que permitiera simular el comportamiento de un robot manipulador móvil. Este robot será duplicado y se creará un entorno con objetos que los dos robots deberán manipular de acuerdo a su programación.

Por un lado como robot móvil, es decir, como base de nuestro manipulador móvil, se ha elegido el robot Pioneer P3-DX de MobileRobots, del que ya existía un modelo disponible en el software utilizado para la simulación. Este robot presenta una configuración diferencial de movimiento, contiene ocho sensores de ultrasonido en la parte delantera y soporta una carga útil de 17 kg. Por otro lado, como robot manipulador se ha seleccionado el robot Mico de Kinova, también disponible su modelo en el software utilizado. Se trata de un manipulador ligero de 6 grados de libertad con una rotación ilimitada en cada eje, tiene un alcance de 70 centímetros y un peso de 5 kg. También cabe resaltar que su pinza es configurable pudiendo controlar cada uno de sus dedos.

El entorno en el que se ha desarrollado la simulación ha sido creado mediante el simulador V-REP.

1.1. Contenido de la memoria

A lo largo de este documento se describirá la realización de este proyecto. Para ello, se ha dividido la memoria en cuatro capítulos. El primero contiene dos apartados que tratan de hacer una introducción y la motivación que se ha tenido para la realización del proyecto. El segundo capítulo comprende la parte más extensa del proyecto, el desarrollo del mismo. Ésta se divide en cuatro apartados y éstos a su vez en subapartados. El primero trata de introducirnos en el contexto y el ámbito del proyecto. Después se especifican las herramientas que se han utilizado, que en su casi totalidad ha sido el software VREP. A continuación se explica como se ha diseñado y configurado todo el entorno de simulación, tanto los objetos del robot como los del escenario. Para finalizar el último apartado de este capítulo detalla toda la implementación de los controladores del robot móvil y del manipulador. El tercer capítulo tiene que ver con las conclusiones del trabajo, donde se plasma lo que primordialmente nos ha evocado nuestro estudio. El último capítulo corresponde con unas posible mejoras y trabajos que en un futuro podrían realizarse sobre este proyecto.

Para concluir se muestran al final los códigos implementados en el simulador y la reseña bibliográfica que ha servido de apoyo.

1.2. Motivación

El objeto de estudio elegido para la realización de este TFG responde a motivaciones personales, relacionadas estas con mis estudios académicos. A lo largo de toda mi formación como ingeniero informático en la universidad y sobretodo durante los dos últimos cursos, en los que escogí la especialización de Ingeniería de Computadores, he ido interesándome cada vez más en la robótica, automática e informática industrial. Gracias en parte a asignaturas que he cursado como Control por Computador o Mecatrónica, me he dado cuenta de la importancia que tiene el campo de la robótica o el campo de la automatización industrial actualmente en nuestras vidas. En la actualidad existen muchas áreas de investigación en el campo de la robótica. Concretamente el área que estudia los robots manipuladores móviles me pareció muy interesante, simplemente por el hecho de pensar que un manipulador móvil te permite sumar las ventajas de los dos robots que lo conforman: la plataforma móvil y el brazo robótico. El robot móvil permite extender el área de trabajo al manipulador, llegando a lugares y manipulando objetos que no podría hacer por si solo.

Tras haberme documentado, no encontré demasiados proyectos similares que trataran la simulación con robots manipuladores móviles. Esto también hizo que despertara mi curiosidad sobre este tipo de robots. Este proyecto podría servir como plataforma para la prueba de movimientos de robots manipuladores móviles y de su interacción con el entorno, esto es, planificando caminos, esquivando y manipulando distintas clases de objetos, etc, para facilitar la investigación y el desarrollo de los mismos.

Capítulo 2

Desarrollo del proyecto

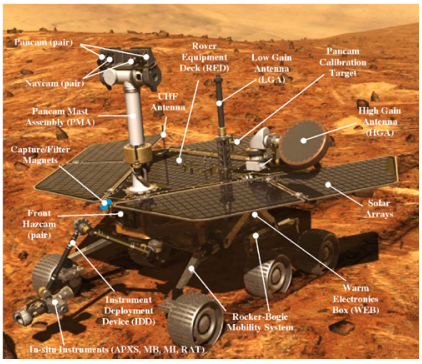
2.1. Contexto

Este proyecto se sitúa dentro de la ingeniería automática, más concretamente en la robótica. Existen dos grandes grupos en los que se divide la robótica: robots manipuladores y robots móviles. Los primeros están implantados en la industria de forma muy importante, y aunque se investiga sobre ellos en áreas muy concretas, son los robots móviles los más recientes y con la investigación mucho más activa. Cuando se combinan estos dos tipos de robots, se construyen los robots manipuladores móviles. Este sistema de manipulación móvil ofrece una doble ventaja, la movilidad que presenta la plataforma o robot móvil y las múltiples funcionalidades que proporciona el manipulador. Sin embargo, el funcionamiento de este sistema es complejo debido a los muchos grados de libertad y al medio ambiente no estructurado que realiza.

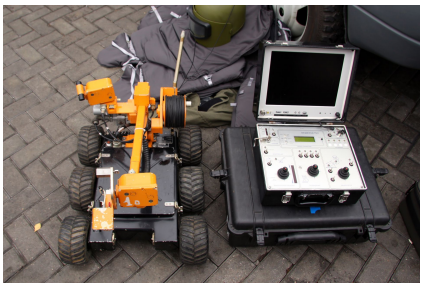
La manipulación móvil es un tema de interés en entornos de desarrollo e investigación. Los manipuladores móviles pueden ser autónomos o teleoperados y se utilizan en muchas áreas diferentes, como por ejemplo en:

- Misiones espaciales de exploración
- Operaciones militares

- Atención domiciliaria y de salud



(a) Robot Rover: misión exploración a Marte



(b) Robot Varan: utilizado para desactivar bombas

(c) Manipulador móvil G-ball de Robotnik

(d) Manipulador móvil Youbot de Kuka

Figura 2.1: Manipuladores móviles en diferentes áreas de trabajo

A pesar de la necesidad de automatización inteligente y flexible en el ámbito industrial, la aplicación e integración de los manipuladores móviles ha sido limitada. Una razón de esto es la forma tradicional en que actúan las industrias, sin asumir riesgos por la aplicación de nuevas tecnologías.

2.2. Herramientas utilizadas

La principal herramienta utilizada para la realización de este proyecto ha sido el software Coppelia Robotics V-REP(*Virtual Robot Experimentation Platform*), en su versión de licencia de estudiante gratuita V-REP Pro Edu,

que se puede descargar de la página oficial (`http://www.coppeliarobotics.com/downloads.html`).

Se han utilizado dos equipos portátiles distintos. Un equipo para la redacción del proyecto con un sistema operativo Linux; y otro equipo más completo para trabajar con el simulador, con un procesador Intel i7 y 8 GB de memoria RAM con el sistema operativo Windows 8.1. El equipo principal era el primero, pero V-REP necesitaba más hardware y por lo tanto no funcionaba lo suficientemente bien.

2.2.1. V-REP(*Virtual Robot Experimentation Platform*)

V-REP es el software ideal para lo que se pretende, que es realizar una simulación de un entorno con robots manipuladores móviles en tiempo real. Contiene gran cantidad de opciones a la hora de establecer propiedades físicas del robot, así como para su control (extensa capacidad de APIs) y visualización de información.

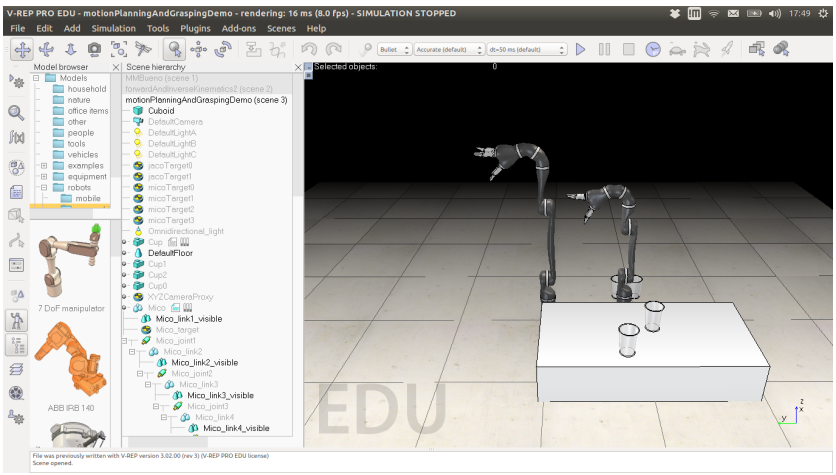


Figura 2.2: Software de simulación robótica VREP

Este software lleva un entorno de desarrollo integrado que esta basado en una arquitectura de control distribuida. Esto presenta una gran ventaja ya que cada objeto o modelo se puede controlar individualmente de seis maneras de programación distintas: scripts embebidos, plugins, API remota de

cliente, complementos (*add-ons*), nodo ROS y cliente/servidor personalizado. Además cualquiera de estas formas que se elijan, es compatible con los lenguajes de programación más extedidos actualmente, como por ejemplo C/C++, Python, Java, Matlab, etc.

También cuenta con otra ventaja muy importante, que es la gran cantidad de información que aporta el propio programa y su página web oficial, mediante ejemplos, tutoriales y archivos con código de control de los cuales uno se puede apoyar para configurar lo que necesite.

Aparte de lo mencionado anteriormente VREP presenta otras características importantes a destacar en el proyecto como:

- Software multiplataforma y portable para windows, Linux y MAC OS.
- Tres motores físicos diferentes para mayor velocidad en los cálculos dinámicos, como también permite simulaciones físicas e interacciones con objetos del mundo real.
- Mecanismos importantes para la simulación como la cinemática directa/inversa, la detección de colisiones, el cálculo de una distancia mínima, path/motion planning, etc.
- Integración, creación y edición de modelos y objetos.
- Simulación con sensores de proximidad y de visión.
- Personalización de la interfaz de usuario.

2.2.1.1. Interfaz de usuario: vista principal

Al iniciar VREP, se abre por defecto una escena. Las escenas son el elemento principal del programa. Éstas están formadas por subelementos llamados modelos.

Se puede navegar por el entorno mediante el ratón. Para poder rotar la escena se utiliza la barra de herramientas, que contiene además todas las funcionalidades de la simulación: propiedades físicas, manipulación de los

objetos, visualización con las diferentes cámaras, etc. A la hora de alejar y acercar la imagen también se puede utilizar la rueda del ratón.

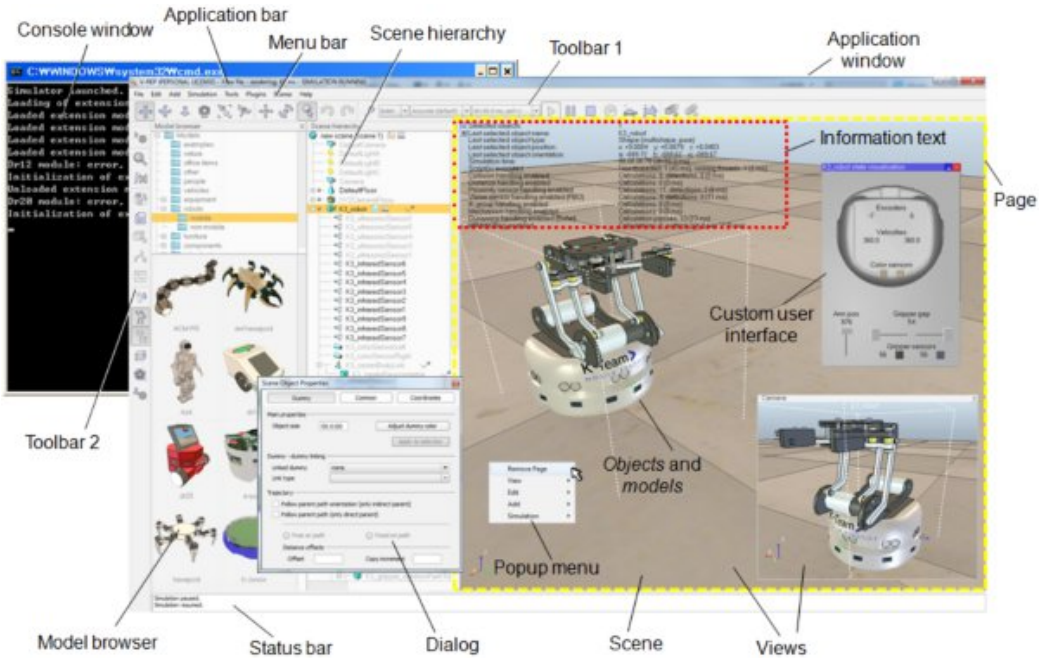


Figura 2.3: Elementos de la interfaz de usuario

Un elemento de la interfaz muy destacable es el buscador de modelos que vienen por defecto en VREP. Éste se encuentra en una ventana en la parte izquierda y están ordenados por carpetas muchos de los modelos de robots móviles, manipuladores, componentes, equipamiento, decoración, etc, de fabricantes importantes de robótica. Estos modelos se pueden modificar, como también se puede diseñar un modelo propio.

Para conocer con detalle cada elemento de la interfaz de usuario, VREP dispone de un manual de usuario, que también contiene las formas de programar, el cálculo de módulos, el funcionamiento de la simulación, tutoriales, etc.

2.2.1.2. Objetos de la escena(*Scene objects*)

Los principales elementos que se usan en VREP para crear una escena de simulación son los objetos. Estos son visibles en la jerarquía de escenas mediante un símbolo que distingue cada tipo; y en la escena de la vista principal se representan en tres dimensiones.

Hay varios tipos de objetos y cada uno tiene sus opciones de configuración. Alguna vez interesa asignar a los objetos unas propiedades especiales que les permiten interactuar con otros objetos mediante el cálculo de módulos (detección de colisiones, mínima distancia entre dos objetos, cinemática inversa, *path/motion planning*, etc)

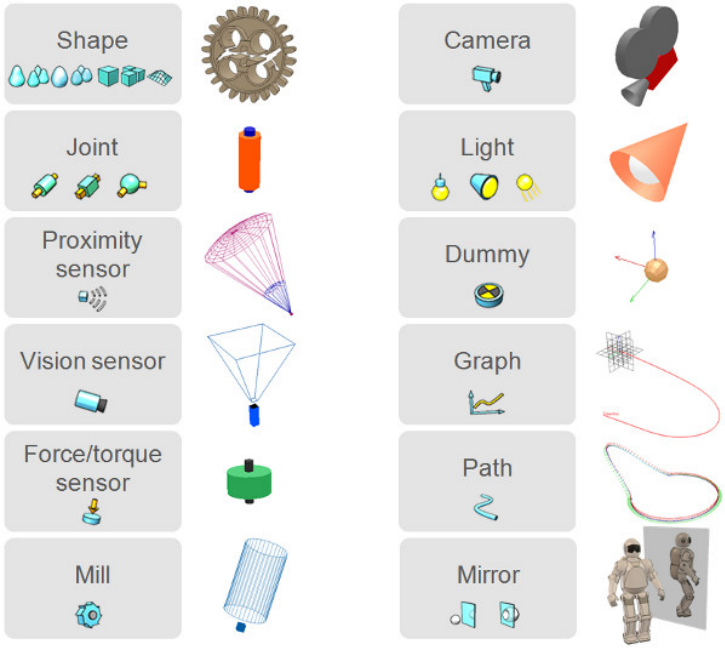


Figura 2.4: Diferentes tipos de objetos y su representación

Cada objeto tiene una posición y orientación respecto a la escena. Estas se pueden cambiar como se desee en la barra de herramientas, en la sección llamada *object position/orientation manipulation*. Otra propiedad importante de los objetos es que se pueden ensamblar a otros objetos, estableciendo así relaciones tales como el objeto A es "hijo" del objeto B. Por lo tanto,

si por ejemplo movemos el objeto B, el objeto A le sigue. Alternativamente, también se pueden desensamblar los objetos.

Las dos últimas características pertenecientes a los objetos han sido de vital importancia dominarlas para la realización del proyecto.

2.2.1.3. Jerarquía de escenas(*Scene hierarchy*)

La jerarquía de escenas aparece por defecto a lado del buscador de modelos mencionado en el apartado anterior. Ésta contiene todas las escenas abiertas por VREP. Si se hace doble click en una escena, en esta ventana se despliegan todos los objetos que componen la escena. También se puede desplegar cada elemento individual de los objetos, quedando al final una ventana en forma de árbol ordenado todo jerárquicamente.

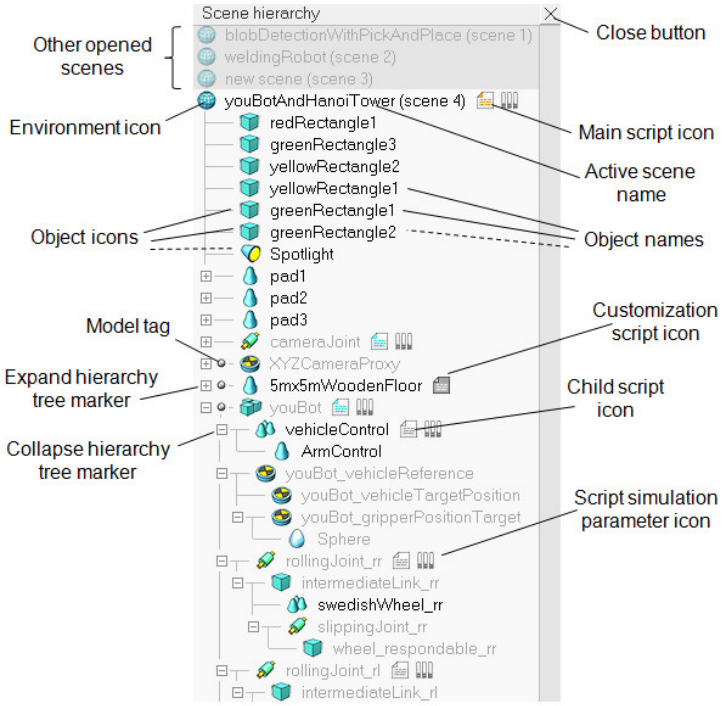


Figura 2.5: Scene Hierarchy

En esta jerarquía de escenas aparecen también todas las relaciones "padre-hijo" de los objetos de la escena. Otro detalle que podemos observar son los

scripts de control asociados al objeto que lo necesite. Estos serán explicados más adelante, pero hay que destacar que se diferencian por su color y que se pueden editar haciendo doble click sobre ellos.

La jerarquía de escenas ha sido un elemento muy importante a la hora de desarrollar el proyecto, ya que facilita mucho la estructuración y la manipulación de los objetos de nuestra escena.

2.2.1.4. Escritura de código: *scripts* embebidos

Como ya se había mencionado anteriormente, VREP soporta seis formas diferentes de programar nuestro código de control. En este proyecto se ha escogido la opción de los *scripts* embebidos por su sencillez y flexibilidad. Esta flexibilidad aparece gracias al interprete de *script* que tiene VREP integrado. El lenguaje de *scripting* que presenta VREP es LUA. Éste es un lenguaje de extensión y dado que esta basado en C, fue una gran ventaja a la hora de realizar el proyecto.

Cuando se dice que un *script* está embebido, lo está en una escena, por ejemplo un *script* que es parte de una escena será cargado y guardado con el resto de la escena.

VREP contiene varios tipos de *script*, pero en este proyecto se van a utilizar los dos más usuales: *main script* y *child script*. Los primeros controlan el bucle de simulación principal y los otros se encargan de controlar los modelos o robots.

En la implementación que veremos más adelante, editaremos los *child scripts* asociados a los objetos que tengamos que controlar. Utilizaremos dos tipos de *child scripts*: los *non-thread child scripts* y los *thread child scripts*.

Para poder abrir el editor de *scripts* haremos doble click en el icono de *script* que aparece en la jerarquía de escenas junto a algún objeto.

A lo largo del proyecto, se ha programado el control del robot manipulador móvil mediante este editor y el lenguaje LUA, apoyándose en la extensa API que presenta VREP con más de 300 funciones.

2.3. Diseño del entorno de simulación virtual

2.3.1. Guía de configuración de objetos del manipulador móvil

Para empezar a construir el entorno de la simulación se debe diseñar los modelos de robot móvil y manipulador, o bien elegir algunos de los modelos de diferentes fabricantes que tiene por defecto VREP. En este caso, como el principal objetivo del proyecto no es realizar un diseño 3D de los robots, se han seleccionado dos modelos adecuados de VREP. Los modelos utilizados ya se han detallado anteriormente en la introducción del proyecto(1).

La siguiente parte del proceso sería la creación en sí del robot manipulador móvil. Para ello se debe unir el manipulador al robot móvil que se ha elegido. Para poder hacer esto correctamente el robot móvil debe contener un sensor de fuerza (*force sensor*) que le permita al manipulador moverse sin ningún problema. En este caso no es necesario añadirlo, ya que el modelo lo lleva por defecto. Para terminar este proceso hay que seleccionar el sensor del robot móvil y con la tecla *control* pulsada, seleccionar el objeto base del manipulador que en este caso se llama *Mico* y apretar en la opción *assemble/disassemble* situada en la barra de herramientas superior. Un objeto base se distingue en el cuadro *Scene hierarchy* con un símbolo redondo plateado. Una vez hecho esto, el manipulador pasa a ser "hijo" del robot móvil.

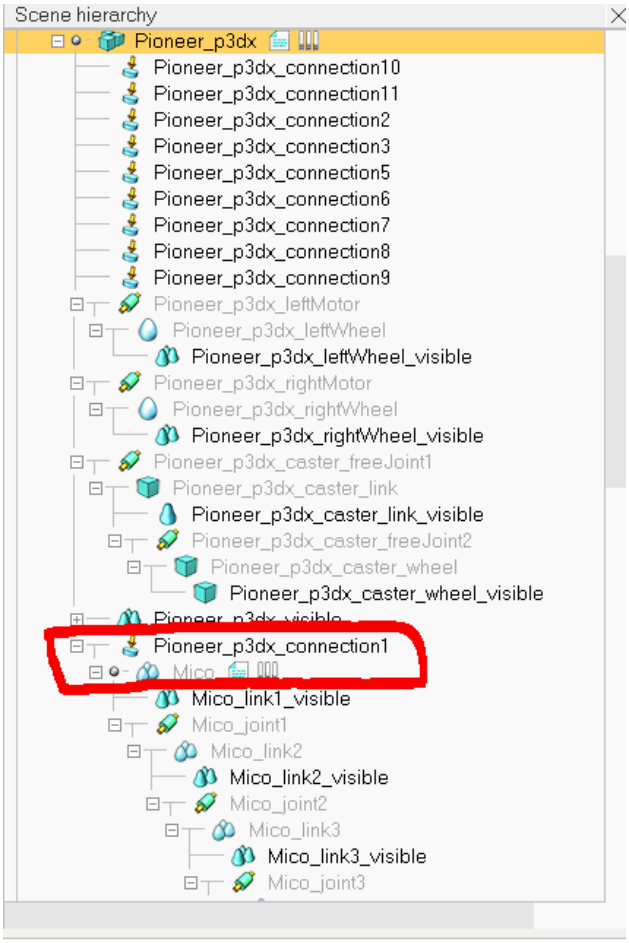


Figura 2.6: Scene Hierarchy: unión manipulador y robot móvil

Una vez se han unido los componentes del manipulador móvil, se procedería a orientar y colocar los objetos de manera que formen el robot tal y como debería ser, por comodidad y por facilidad en los pasos siguientes, usando los botones *Object/item shift* y *Object/item rotate* situados en la barra de herramientas superior.

A continuación se definiría si los objetos que se van a configurar son estáticos (*Static*) o no. Los objetos estáticos son aquellos que no pueden ser directamente actuados, si no que su posición y orientación dependen directamente de otros objetos. Los objetos que no sean estáticos, serán dinámicos y son los objetos que el motor de simulación dinámica de VREP tratará.

Tanto el manipulador como el robot móvil se definirán como no estáticos. En consecuencia para cada uno se definirán sus masas (de manera aproximada). La masa del manipulador se deja como viene por defecto, pero la del robot móvil se debe incrementar hasta cuatro veces su masa original. Esto se hace para que en la simulación el robot móvil sea lo suficientemente robusto para soportar el peso del manipulador, es decir, que en cualquier movimiento el robot móvil no vuelque o cambie a una posición errónea.

Para cambiar la masa del robot móvil (también cualquier configuración dinámica de simulación del objeto), se clica en la lista de la izquierda, *Scene hierarchy*, y sobre el icono del objeto base del robot móvil(*Pioneer_p3dx*) se hace doble click para abrir el menú *Scene object properties*. Una vez en ese menú en el subapartado *Show dynamic propierties dialog* están todas las propiedades dinámicas para el objeto, como la masa y si será estático o no.

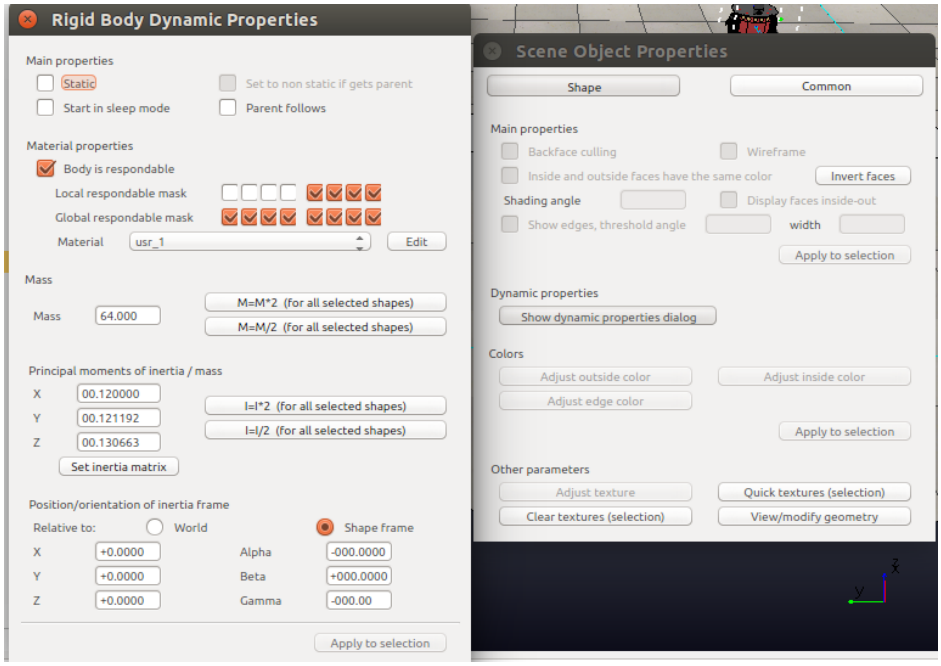


Figura 2.7: Vista en VREP del cuadro *Scene object propierties*

Llegados a este punto, el robot móvil queda configurado completamente. Ahora queda configurar el brazo robótico o manipulador y añadirle un elemento fundamental: la pinza o mano robótica (*gripper*).

Para completar el manipulador móvil se le debe añadir una pinza que le permita manipular los objetos que se han fijado para la simulación. En el menú *Model browser* que se ha utilizado anteriormente para elegir el robot móvil y el manipulador, se encuentra la carpeta *components* que contiene la subcarpeta *grippers*. En ella podremos desplegar una lista de pinzas de diferentes tipos (succión, tres dedos, dos dedos). Una de estas es la pinza que corresponde a nuestro manipulador siendo del mismo fabricante (*Mico hand*). Se trata de una pinza configurable que se compone de dos dedos que pueden ser controlados individualmente. Del mismo modo que se ha unido el manipulador a la base móvil lo haremos con la pinza al manipulador. La última articulación del manipulador lleva incorporado el sensor de fuerza necesario para ensamblar el componente. Una vez se haya ensamblado la pinza al manipulador, se creará la misma situación de parentesco que en el manipulador y el robot móvil.

Para terminar sólo quedaría configurar el manipulador. En este proyecto al no haber diseñado el propio manipulador, la configuración que viene por defecto es prácticamente la adecuada para el posterior control del mismo. Sin embargo, se explicaran algunas modificaciones y detalles importantes para entender el control del manipulador.

Lo primero, aunque es algo opcional, posicionariamos las articulaciones como queramos que comiencen en nuestra simulación. Para ello se clica en el icono de cada articulación entrando en el menú *Scene object propierties*. Una vez ahí, en el apartado *Position* se cambia la posición de la articulación a los grados que se desee.

Otra característica importante en las articulaciones es su rango de posicionamiento. En el manipulador utilizado ya viene configurado y solo la segunda y tercera articulación presentan un rango limitado del eje. Para acceder a estas propiedades, simplemente realizamos el mismo procedimiento que hemos utilizado antes, accediendo al menú *Scene object propierties*. Una vez allí, si *Position is cyclic* está marcado, quiere decir que tendrá una rotación ilimitada del eje y será el propio programa el que gestione su movimiento.

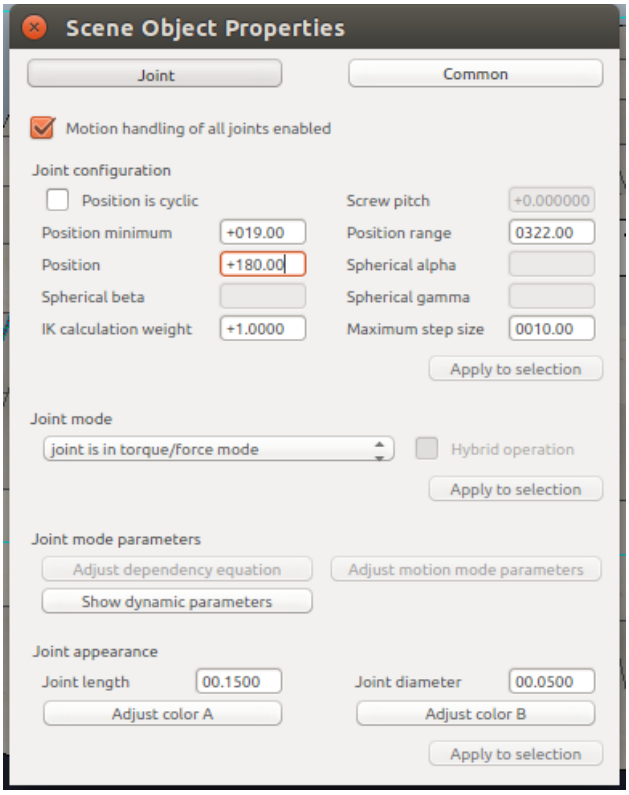


Figura 2.8: Vista en VREP del cuadro *Scene object propierties:joint*

También tenemos que elegir el tipo de control que queremos para el movimiento de cada articulación. Para ello se entra al mismo menú que las anteriores veces, clicando esta vez en *Show dynamic parameters*. Por defecto el manipulador esta diseñado con las articulaciones en modo Par/Fuerza (*torque/force mode*). En este modo las articulaciones son simuladas por los módulos dinámicos del propio simulador y soporta distintos métodos de control. Principalmente la articulación puede operar de dos modos:

- **Control de velocidad:** si en el menú que hemos indicado se encuentra marcado *Motor enabled* y la casilla *Control loop enabled* desmarcada, entonces la articulación tratará de llegar a la velocidad objetivo (*Target velocity*) deseada dado el máximo par/fuerza (*Max.torque/force*) que es capaz de ofrecer. Cuando ese valor máximo es muy alto, la velocidad de destino se alcanza instantáneamente.

- **Control de posición:** si estan marcados *Motor enabled*, *Control loop enabled* y además tenemos habilitado *Position control(PID)*, los parámetros del PID tratarán de dirigir la articulación a la posición deseada.

Para este proyecto se va a utilizar el control de posición con una simple modificación con respecto a como estaba configurado el manipulador por defecto. Esta consiste en reducir la *Upper velocity limit*, que permite limitar la velocidad de regulación a un valor máximo. Esto se hace para que en la simulación, cada movimiento de las articulaciones se realice con mayor amortiguación y precisión, ya que la velocidad del motor es menor.

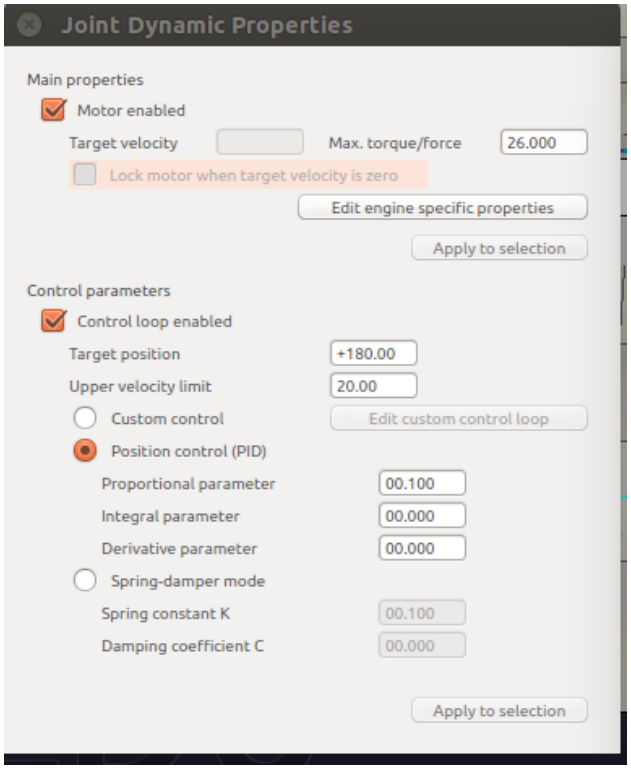


Figura 2.9: Vista en VREP del cuadro *Joint Dynamic Propierties*

Ahora solamente faltaría configurar la pinza de nuestro manipulador, pero se ha decidido que las propiedades que lleva por defecto son las necesarias para una correcta manipulación.

Lo único que falta para que el manipulador móvil esté correctamente configurado, sería marcar en *Object special propierties* del menú *Scene Object Propierties / Common* las casillas *Collidable*, *Renderable*, *Detectable* y *Measurable*. Es un paso importante, ya que si no se marcaran el objeto no sería detectado por los demás objetos del entorno, aparte de no poder correr en la simulación desapareciendo del espacio de trabajo al no considerarse un "objeto físico". Tampoco podríamos usar los modulos de cálculo de distancia mínima, detección de colisiones,etc, que nos ofrece VREP.

El paso más importante para el correcto funcionamiento de la simulación, viene dado por las relaciones de parentesco que se establezcan entre los distintos objetos que componen el robot. Si no se hace correctamente este paso, se provocarían anomalías en el funcionamiento de la simulacción puesto que no se movería como si fuera un solo ente.

Las relaciones de parentesco pues, implican una relación de dependencia de unos componentes a otros. Aunque en este proyecto, al utilizar un manipulador y un robot móvil ya diseñados, estas relaciones ya estaban implícitas. Pero a la hora de unir el manipulador y la base móvil si que hay que fijarse que la relación de "padres" e "hijos" sea la correcta.

Para explicarlo de otra manera, la relación de parentesco influye en la dependencia e independencia de los movimientos de unas articulaciones con respecto a otras. Por ejemplo, que la pinza del manipulador se mueva no afecta nada al resto de articulaciones. Sin embargo, si la primera articulación (base del manipulador) rota, el brazo entero rotará al estar éste unido a la base.

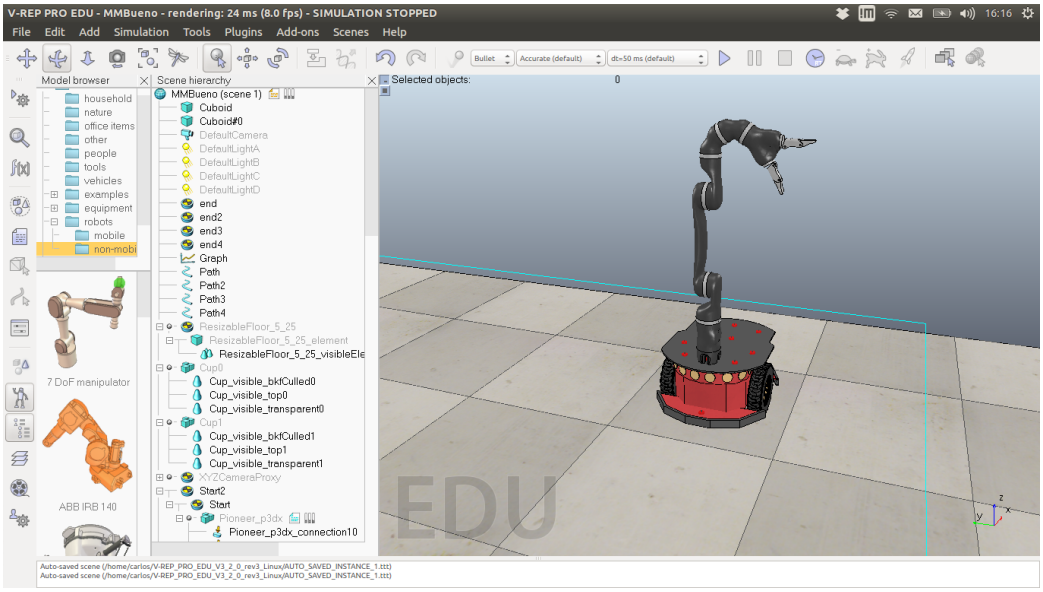


Figura 2.10: Vista en VREP con configuración de objetos realizada

2.3.2. Guía de configuración de objetos de escenario

Para terminar de configurar el entorno de simulación virtual con el manipulador móvil que se ha creado anteriormente, necesitamos diseñar un escenario de simulación con diferentes objetos que proporciona VREP, los cuales nos permitirán que el robot interactue y muestre su funcionalidad.

La primera parte sería orientar y colocar el manipulador móvil como queramos en el escenario. En nuestro caso el robot se situará centrado en el suelo (*ResizableFloor*) del escenario, mientras que estará orientado exactamente con los siguientes ángulos: $\alpha = 0^o, \beta = 0^o, \gamma = 180^o$.

Para poder modificar la orientación y la posición en plano de cualquier objeto, debemos seleccionarlo y apretar a los botones *Object/item shift* para la posición, u *Object/item rotate* para la orientación. Estos se encuentran en la barra de herramientas superior y se puede cambiar el valor exacto de cada eje de coordenadas, o bien arrastrando con el botón izquierdo del ratón el objeto.

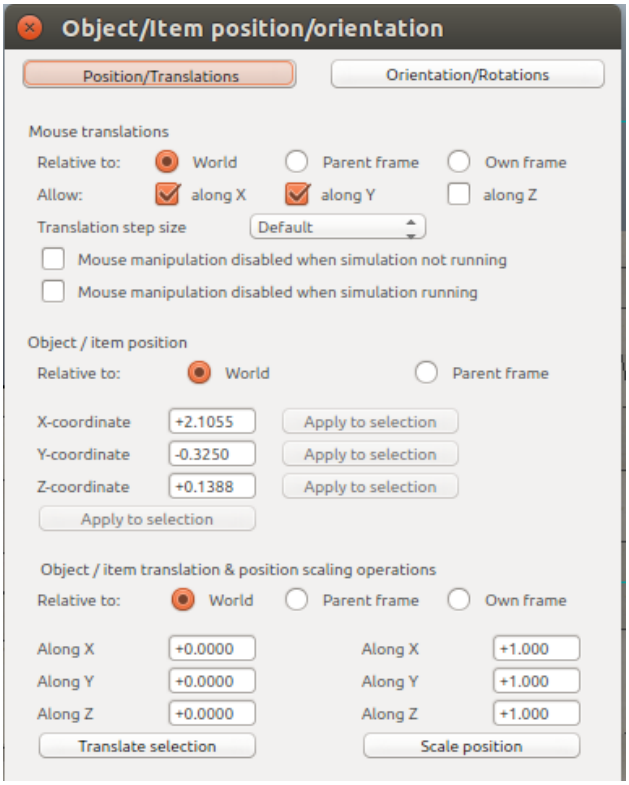


Figura 2.11: Vista en VREP del cuadro *Object/item position/orientation*

El siguiente paso sería crear los objetos para nuestra escena de simulación. Se ha utilizado lo que en VREP se denomina *Primitive shape*, más concretamente dos cubos (*Cuboid*) para simular las "mesas" o superfícies donde se situarán los objetos que nuestro robot debe manipular. Para crear este tipo de formas primitivas, simplemente clicamos con el botón derecho en la escena y elegimos del desplegable *Add >Primitive Shape >Cuboid*. Después podemos escalar y posicionar el objeto como ya hemos explicado anteriormente.

Para finalizar hay que elegir algún tipo de objeto para la manipulación de nuestro robot. VREP ofrece múltiples posibilidades, incluso se puede diseñar o importar. Aquí se ha elegido un objeto sencillo pero que a la vez requiere cierta precisión a la hora de manipularlo: un vaso. De nuevo se ha seleccionado uno de los modelos que ofrece VREP. Vamos al menú *Model browser* y dentro de la carpeta *household* encontramos nuestro objeto *Cup*. Colocare-

mos dos vasos encima de la mesa más centrada en determinadas posiciones establecidas para su posterior manipulación.

Otro detalle muy importante a tener en cuenta en los objetos que se acaban de crear, es marcar todas las opciones especiales (*Collidable*, *Renderable*, etc) que habíamos marcado anteriormente en nuestro robot, para que en cualquier cálculo de control el programa los considere como objetos.

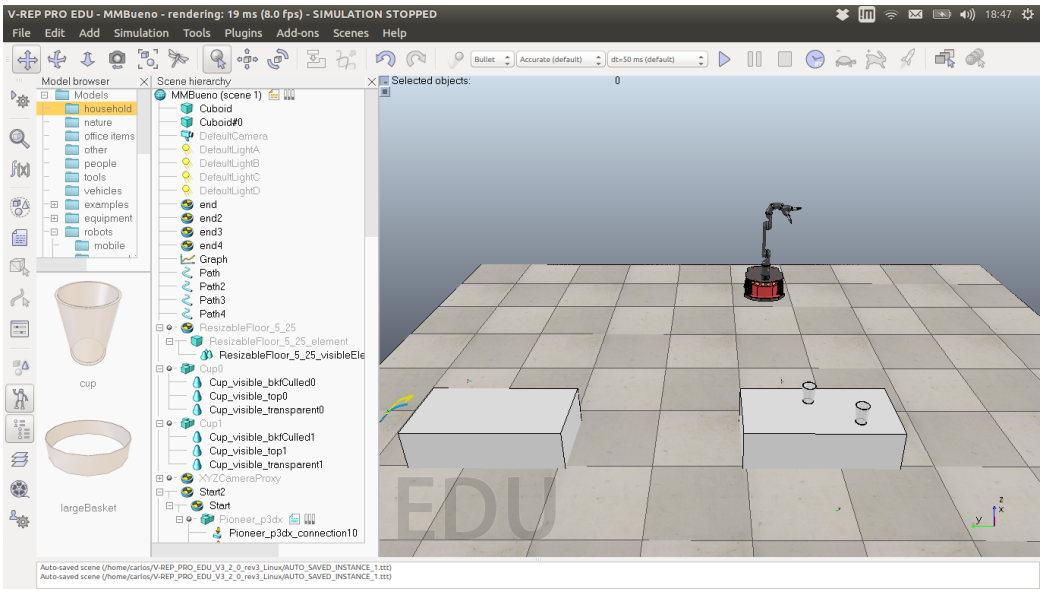


Figura 2.12: Vista en VREP con configuración de objetos realizada

2.3.3. Ampliación del diseño principal

Hasta ahora hemos creado un entorno de simulación virtual con un robot manipulador móvil, el cual hemos configurado para su posterior control. También se han configurado los objetos restantes para simular la escena.

Profundizando más, nuestro robot desempeñará una simple tarea, que consistirá en llegar al objeto destino (vaso), agarrarlo y transportarlo a una posición objetivo.

Dada la simplicidad de esta simulación, se ha pensado realizar una especie de ampliación de la idea principal. Esta consiste en duplicar el robot que

ya hemos creado y configurado, diseñando así un entorno de simulación un poco más complejo, donde los dos robots realicen sus correspondientes tareas simultáneamente.

La simulación que se ha hecho en el proyecto con los dos robots se podría analizar como un espacio (primera mitad de la escena) en el que solo el robot situado ahí se encargaría de la manipulación de objetos. La otra mitad la ocuparía el segundo manipulador que también sería el único que se encargaría de recorrerla. Si por ejemplo se quiere pasar el vaso de una mitad a otra, éste debería ser manipulado por los dos robots que interactuarían entre ellos. Esto es un posible análisis, pero también se puede ver como una simple escena que representa el concepto básico de la manipulación móvil.

Duplicar el robot en VREP es muy sencillo, únicamente se selecciona el objeto base y "padre" del manipulador móvil en *Scene Hierarchy* y utilizamos la conocida combinación de teclas de copiar/pegar "Contol+C" y "Control+V". Con esto VREP crea automáticamente una copia del robot, pero añadiendole al nombre del propio y al de todos los objetos que lo componen un sufijo que lo distinga del original. Por ejemplo, duplicamos el objeto *Cuboid1* y el nombre de la copia que origina VREP es *Cuboid1#0*. Por lo tanto, cuando tengamos que acceder a un objeto a la hora de programarlo, podremos distinguir la copia del original.

Por último, debemos tener en cuenta cual es el momento más conveniente para realizar la copia del robot. En este caso, una vez hayamos configurado nuestro robot por completo podremos duplicarlo. Después ya pasaremos a programar el control de cada uno.

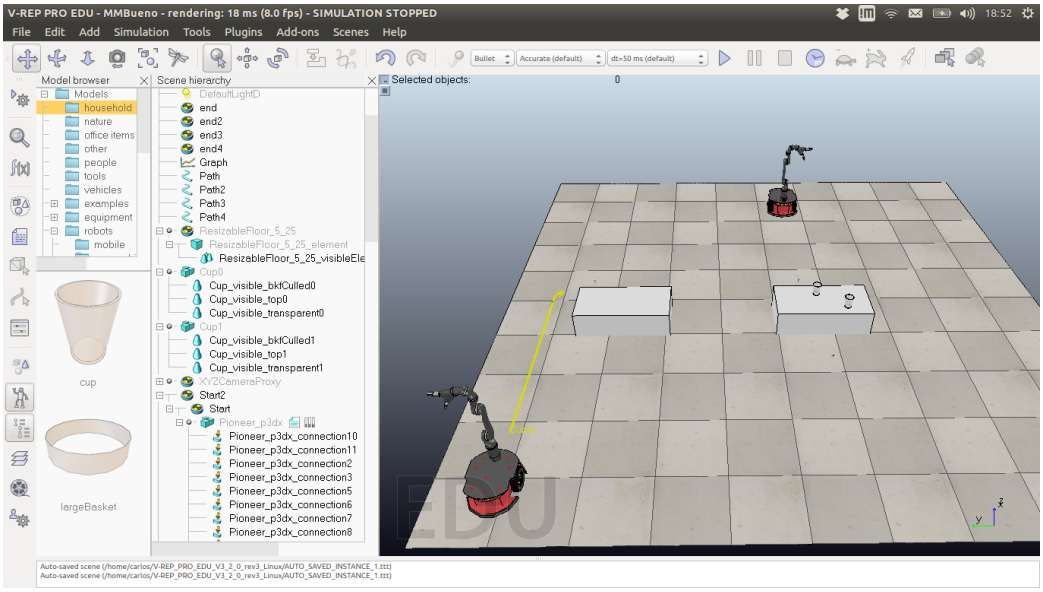


Figura 2.13: Vista en VREP con configuración y ampliación de objetos realizada

2.4. Implementación de control del manipulador móvil

Una vez configurado el manipulador móvil y toda la escena necesaria para la simulación con todo detalle dentro de VREP, faltaría la programación de nuestro robot para otorgarle acciones de control que simulen su movimiento real. Primero se programará el robot móvil, con su recorrido hacia las posiciones donde el manipulador consecutivamente podrá manipular el objeto. En este último también habrá que controlar cada movimiento de sus articulaciones.

2.4.1. Programación de la base móvil

2.4.1.1. Planteamiento inicial

El primer paso que se da en el control del robot móvil que se ha utilizado como base del manipulador móvil, es conocer detalladamente la configuración del mismo, esto es, analizar como están distribuidos los principales elementos que lo componen: ruedas, motores, sensores. En relación a las ruedas, la configuración que tiene nuestro robot es la diferencial.

La configuración diferencial se presenta como la más sencilla de todas las utilizadas típicamente en robótica móvil. Consta de dos ruedas diametralmente opuestas en un eje perpendicular a la dirección del robot. Cada una de ellas irá dotada de un motor, de forma que los giros se realizan dándoles diferentes velocidades. En VREP los motores de las dos ruedas se visualizan en *Scene Hierarchy* unidos al objeto base del robot como *Pioneer_p3dx_leftMotor* y *Pioneer_p3dx_rightMotor*.

Con dos ruedas es imposible mantener al robot horizontalmente, ya que se producen cabeceos al cambiar de dirección. Para resolver este problema se añaden lo que se conoce como ruedas "locas". Estas ruedas no llevan asociadas ningún motor, giran libremente según la velocidad del robot. Ésta también se ve representada en nuestra escena de VREP como *Pioneer_p3dx_caster_link_visible*.

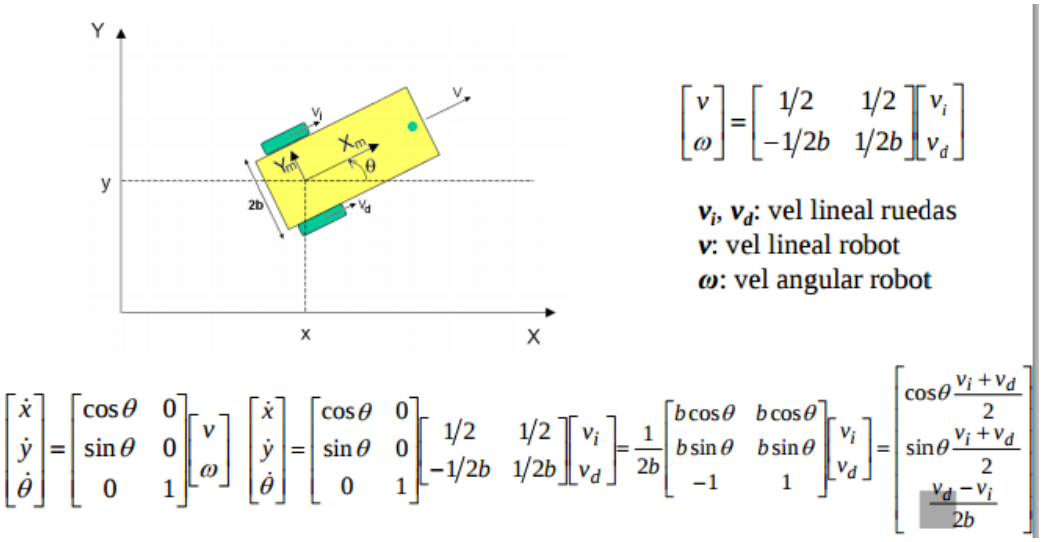


Figura 2.14: Modelo cinemático configuración diferencial de robot móvil

El modelo cinemático servirá como apoyo para la implementación del script de control para otorgar a nuestro robot móvil la orientación, posición y velocidades adecuadas.

Por otro lado nuestro robot debería ser capaz de esquivar objetos si es necesario y por supuesto de generar una trayectoria hacia una posición objetivo en la escena. El modelo escogido incorpora en su diseño en la parte frontal sensores de ultrasonidos que nos solucionarian nuestros problemas. Sin embargo VREP proporciona un módulo que permite calcular o planificar trayectorias a partir de un camino preestablecido (*path*) llamado *Path Planning*. Por la facilidad que nos da para el objetivo del proyecto, éste es el método elegido.

2.4.1.2. Cálculo de trayectorias: *Path Planning module*

Como ya se ha comentado anteriormente VREP ofrece potentes funcionalidades de cálculo o como lo llaman *calculation modules*. Estos módulos no encapsulan directamente a un objeto de escena, si no que trabajan con objetos de cálculo definidos por el usuario. Éstos son diferentes que los de escena, pero se relacionan indirectamente operando en ellos. Un detalle im-

portante que se tendrá en cuenta sobre los objetos de cálculo es que al realizar una operación de copiar/pegar en un objeto de escena, también se duplica el objeto de cálculo que iba asociado.

El módulo que se empleará es el de planificación de trayectorias o *Path Planning*. Permite realizar cálculos de planificación de caminos para objetos de dos a seis dimensiones. Pero la función que interesa de este módulo y se va hacer uso de ella es la planificación para robots no-holonómicos. Decimos que un robot es holonómico si es capaz de modificar su dirección instantáneamente (masa nula) y sin necesidad de rotar previamente.

Para trabajar con *Path Planning* se toman varios valores o parámetros de entrada (objetos):

- **Una entidad robot**: representa el dispositivo que debe evitar los obtáculos, siendo nuestro manipulador móvil.
- ***Start dummy***: objeto especial que representa la configuración inicial del robot. Para crearlo clicamos con el botón derecho en la escena y seleccionamos *Add >dummy*. Tenemos que asegurarnos que la posición y orientación coinciden con el centro del robot. Dentro de *Scene object propierties/Dummy* se marcará la casilla *Follow parent path orientation(only indirect parent)*. Con esto el *dummy* no está directamente relacionado con el camino, y sigue únicamente la orientacion de los puntos del camino.
- ***Goal dummy***: representa la configuración deseada del robot. Los algoritmos de planificación se encargan de mover el *Start dummy* hacia este objetivo evitando colisiones y obtáculos. Lo situamos con una posición y orientación óptimas para que el manipulador pueda realizar sus movimientos sin problemas. Se dejará sin marcar la casilla comentada en *Start dummy*, ya que este objeto no tendrá que seguir al camino.
- **Una entidad obstáculo**: representan los obtáculos que hay que evitar.
- **Camino o *Path***: actúa como un contenedor para una trayectoria calculada por el *Path Planning*. Para crear este objeto de cálculo seguiremos el mismo procedimiento que en los *dummies*: *Add >Path >Segment*

Type. Las propiedades que vienen por defecto al crearlo son las necesarias, añadiendo las tres casillas de *Scene object propierties/ path/ Main propierties*, que nos ayudarán con la visualización del *path* en la escena.

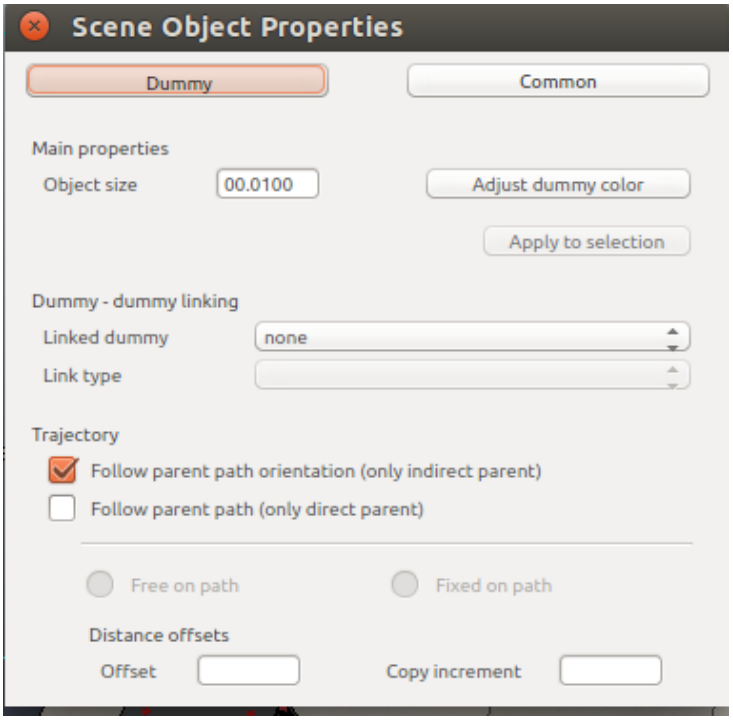
En cuanto a las relaciones de parentesco de estos objetos en *Scene hierarchy*, los *paths* y *end dummies* se sitúan al mismo nivel que la entidad robot, descendiendo de la propia escena. Sin embargo, los *start dummies* pasarán a ser "padres" de los dos robots.

Como hay cuatro trayectorias, dos por cada robot, el *start dummy* del primer camino será "hijo" del *start dummy* del segundo camino que tiene que realizar ese robot. También será "hijo" el tercero del cuarto, que pertenecerán a las trayectorias del robot duplicado.

Con los objetos de entrada creados, sólo faltaría construir los propios *Path Planning*. Para crear un *Path Planning* tenemos que clicar en la opción *calculation module propierties* que encontramos en VREP en la barra de la parte izquierda. Una vez en el menú, seleccionamos la pestaña de *Path Planning* y añadimos los objetos de entrada creados anteriormente. Primero pulsaremos *Add new object* donde aparecerá un submenú, en el cual indicaremos *non-holonomic* y el *start dummy* correspondiente. Se nos creará un *Path-PlanningTask* que podremos modificar su nombre como queramos.

A continuación procederemos a rellenar cada campo con su objeto correspondiente. Por ejemplo en *Associated objects* indicaremos la configuración objetivo (*end dummy*) y el *path*. En el subapartado *Illegal configuration check* configuramos nuestro robot para que no se le permita colisionar con ningún objeto en toda la escena.

Un último paso importante, es el de ajustar el área en la que el módulo *Path Planning* es capaz de cálcular la trayectoria. Para ello abrimos el submenú *Adjust search parameters* y modificamos el rango de búsqueda variando las coordenadas hasta que encontremos la deseada. La casilla *Visualize the search area* es de gran ayuda a la hora de concretar el área. Las dos últimas casillas son opcionales, pero también se ha ido jugando con ellas para facilitar la configuración.



(a) Dummy

(b) Path

Figura 2.15: Vistas en VREP de *Scene object propierties*

Una vez completado todo esto, clicaríamos en *Compute path* y si no ha habido ningún problema se genera la trayectoria deseada. Si ha habido algún error, si tenemos marcada la casilla *If no path was found, use a partial path*, representará una parte del *path*.

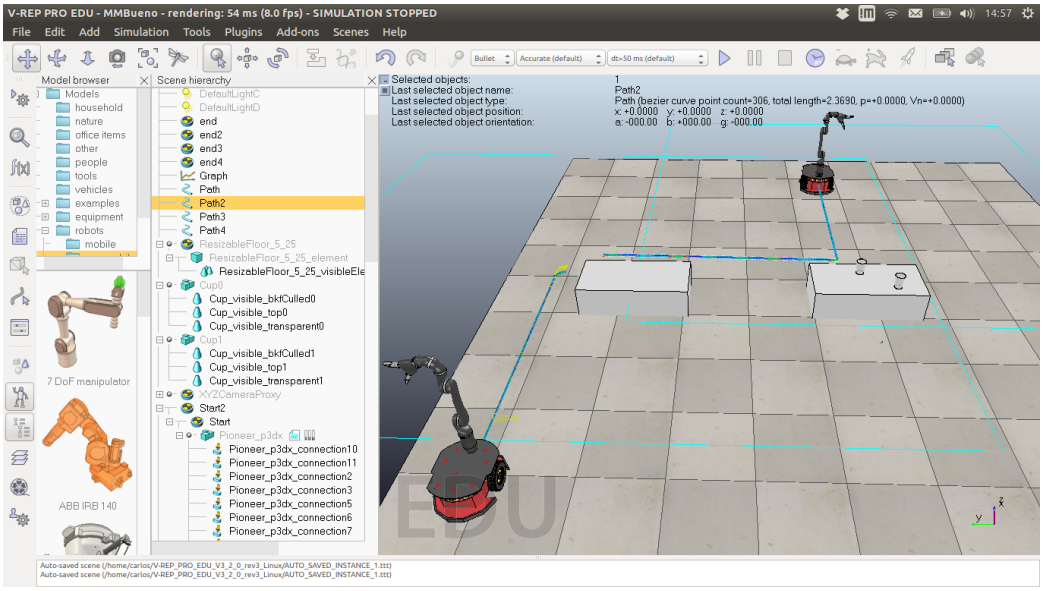


Figura 2.16: Vista en VREP de la escena con todos los *Paths Planning* configurados

2.4.1.3. Scripts asociados

Una vez configurados todos los objetos y el módulo necesario para la programación del robot móvil, falta asociar al objeto base el código para el funcionamiento del robot. Para conseguirlo hay que usar los *scripts* de los que disponemos en VREP.

El botón para consultar los *scripts* aparece en la barra de herramientas izquierda del programa, con un icono similar a una hoja de papel. Una vez se accede al menú se puede consultar la información de los *scripts* existentes y se pueden crear nuevos.

Como ya hemos comentado brevemente en el apartado 2.2.1.4, existen dos tipos de *scripts* los *Main Scripts* y los *Child Scripts*, dentro de estos últimos

se diferencian en *Child Scripts (threaded)* y *Child Scripts (non-threaded)*. El *Main Script* se recomienda no utilizarlo (ni modificar el ya existente por defecto) porque define los valores por defecto de la simulación y el arranque de los *Child Script*.

Para el caso que nos ocupa, usaremos un *Child Script (threaded)*. El motivo es que no se producirán bloqueos al llamar a ciertas funciones (sobretodo para el *Path Planning*), además de permitir enviar señales para establecer una buena comunicación tanto de la base móvil con el manipulador como del robot principal al duplicado. El modelo de robot móvil que cogimos de VREP ya lleva un *script* implementado por defecto, pero es un simple ejemplo de navegación en el que se esquivan obstáculos utilizando el algoritmo de Braitenberg. Para este proyecto se ha modificado por completo implementando un código propio.

Las funciones fundamentales utilizadas en el *script* son:

- ***simGetObjectHandle("nombre_objeto")*:** Devuelve el *handle* correspondiente a un determinado objeto.
- ***simGetPathPlanningHandle("nombre_PathPlanning")*:** devuelve el *handle* de un objeto *Path Planning*.
- ***simSearchPath("handle_PathPlanning", "máximo_tiempo_búsqueda")*:** se encarga de buscar un *path* completo en un tiempo máximo preestablecido. Si devuelve un 2 significa que ha encontrado el *path*, si es un 1 ha encontrado parte de él y si devuelve un 0 no lo ha encontrado.
- ***simGetObjectPosition("handle_objeto", "-1")*:** devuelve la posición de un determinado objeto. El segundo parámetro indica que sea la posición absoluta.
- ***simGetPositionOnPath("handle_Path", "distancia_relativa")*:** Devuelve una posición absoluta interpolada de un punto a lo largo del *path*. El segundo argumento acepta valores entre 0 y 1. Si es 0 significa que está comenzando el *path* y si es 1 que ha finalizado.

- ***simSetObjectPosition("handle_objeto", "-1", "posición")*:** Fija la posición del tercer parámetro (coordenadas x,y,z) a un objeto.
- ***simSetJointTargetVelocity("handle_objeto", "velocidad_objetivo")*:** fija la velocidad objetivo en una articulación (motor). Este comando se usa porque las dos articulaciones de nuestro robot móvil son de tipo par/fuerza, además de estar activado el motor y el control por velocidad y no por posición.

Aparte de estas funciones, se han utilizado otras dos que han permitido la comunicación entre el manipulador, la base móvil y el segundo robot duplicado para tener así una buena sincronización en la simulación. Son las siguientes:

- ***simWaitForSignal("nombre_señal")*:** bloquea y espera a una señal con el nombre que se le indica en el parámetro. Cuando la recibe devuelve un valor entero.
- ***simSetIntegerSignal("nombre_señal", "valor_señal")*:** se podría decir que es la función análoga a la anterior. Ésta crea una señal con un valor entero (segundo parámetro) y su nombre (primer parámetro).

Primero de todo, se obtendría el *handle* de todos los objetos con los que vamos a trabajar: los dos motores, el *Path Planning*, el robot, el *path* y el *start dummy*. También se definen dos variables importantes para el *Path Planning*, una establecerá la posición a lo largo del *path* en referencia al robot, la otra servirá para calcular la distancia que hay desde el robot al punto del *path*. A continuación se crea un bucle que será el encargado de que nuestro robot siga el *path* calculado hasta que las velocidades lineales y angulares de éste sean 0, es decir, hasta que lo detengamos porque hemos llegado a la posición objetivo.

Dentro del bucle es donde se realizará el movimiento del robot siguiendo el *path* calculado. Primero se obtienen las posiciones absolutas del robot y

de un punto dentro del *path*. Se determina también la posición de comienzo utilizando el *handle* del *start dummy*. Seguidamente se utilizan tres funciones distintas que nos ofrece VREP para poder trabajar con matrices. Se obtiene la matriz transformada de posición absoluta del robot, que se multiplica por el vector de posición en el *path* obtenido anteriormente, creando así una posición relativa de nuestro robot en el *path*.

Una vez hecho esto, ayudándose de la librería matemática (*math*) de LUA, por una parte se calcula la distancia entre el robot y el punto en el *path*; y por otra parte el ángulo de rotación (*phi*) de nuestro robot móvil.

La última parte consiste en aplicar la cinemática inversa de la configuración diferencial del robot (2.4.1.1). Para ello se define una velocidad lineal constante y una velocidad angular constante por el ángulo de rotación. Si el punto que recorre el *path* (*pos_on_path*) es menor que 1 significa que aun no hemos alcanzado el final y por lo tanto las velocidades del robot tienen que seguir según se han definido. En cambio, si este punto es igual a 1 se ha finalizado el *path*, debemos detener el robot dándoles un valor 0 a las velocidades. También enviaremos una señal con un valor entero y un nombre específico que será recibida por el manipulador, advirtiéndole que tiene que empezar su tarea. A continuación se calculan las velocidades angulares definitivas de las dos ruedas. Para ello se necesitan dos datos importantes extraídos del menú de propiedades de VREP, que son la distancia de separación de las ruedas (d) y el radio de la rueda (r_w). Una vez calculadas las velocidades, se aplican a cada uno de los motores del robot con la función *simSetJointTargetVelocity*.

Lo único que faltaría sería incrementar los puntos de posición en el *path*, que actúa como un "contador" del bucle principal.

Después de todos estos pasos, ya estaría el robot móvil definido al completo para ser simulado bajo cualquier condición. Ahora sólo habría que esperar una señal del manipulador, la que daría paso a la siguiente trayectoria del robot móvil. Ésta seguirá el mismo procedimiento que la primera pero con su correspondiente *Path Planning* y sus respectivos objetos.

Para el robot duplicado también se deberán crear sus dos *Path Planning*

correspondientes y configurar todos los objetos de la misma forma que se ha hecho a lo largo de esta sección.

2.4.2. Programación del manipulador

2.4.2.1. Planteamiento inicial

Para realizar una buena simulación de manipulación móvil, podemos plantearnos que nuestro manipulador debería manipular los objetos creados (vasos) con una precisión muy similar a como lo haría un humano, ya que existe una gran semejanza entre un brazo manipulador con uno de anatomía humana.

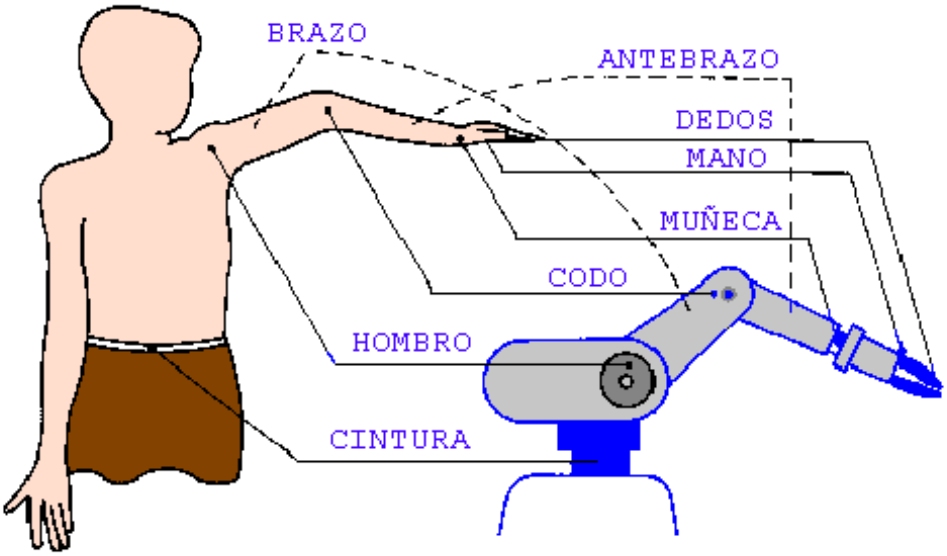


Figura 2.17: Semejanza de un brazo manipulador con la anatomía humana

El primer paso para poder controlar un manipulador o robot industrial es configurar todos sus elementos estructurales, es decir, sus eslabones, articulaciones y actuador final. Este paso ya se ha realizado anteriormente en la sección 2.3.1.

El modelo de manipulador que se ha elegido de VREP (*Mico*)consta de

seis articulaciones, incluyendo la base del manipulador (*Mico_joint1*) y el elemento terminal (*Mico_joint6*) al que se le une la pinza (*MicoHand*). Si cada articulación provee al robot de un grado de libertad se puede confirmar que este manipulador tiene seis grados de libertad.

Lo que nos interesa y vamos a implementar para que nuestro manipulador pueda simularse sin ningún problema compenetrandose totalmente con su base móvil, es lo que se conoce como controlador. Como su propio nombre indica es el que regula cada uno de los movimientos del manipulador, las acciones, cálculo y procesado de información.

Existen varios grados de control que son función del tipo de parámetros que se regulan:

- **de posición:** el controlador interviene únicamente en el control de la posición del elemento terminal.
- cinemático: en este caso el control se realiza sobre la posición y la velocidad.
- dinámico: además de regular la velocidad y la posición, controla las propiedades dinámicas del manipulador y de los elementos asociados a él.
- adaptativo: engloba todas las regulaciones anteriores y, además, se ocupa de controlar la variación de las características del manipulador al variar la posición.

VREP nos ofrece la posibilidad de implementar todos estos tipos de control para nuestro manipulador, por ejemplo con los módulos de cálculo de cinemática directa/inversa o mediante *Motion Planning*. Pero en este caso se ha decantado por un control de posición más sencillo, en el cual tendremos que otorgar a cada articulación el ángulo correcto para llevar al elemento terminal a la posición deseada.

2.4.2.2. Scripts asociados

Para realizar la implementación del controlador de nuestro manipulador, se seguirá el mismo procedimiento que con el robot móvil.

Por defecto, como sucedía con la base móvil, también vienen implementados *scripts* del modelo *Mico*. En general se han podido conservar muy poco código, por lo tanto también se implementará un código propio de nuevo. En el caso del manipulador haremos uso de los dos tipos de *scripts* de VREP, el *Child Script (threaded)* asociado al objeto con la relación de parentesco más alta (*Mico*) y el *Child Script (non-threaded)* para la mano robótica o pinza (*MicoHand*). Este último es del que más código ya implementado por el fabricante aprovecharemos.

Para el manipulador también necesitamos conocer ciertas funciones imprescindibles para la implementación, aunque el código en sí es más simple que el robot móvil. Algunos comandos como *simGetObjectHandle* o *simWaitForSignal*, ya se han explicado anteriormente (2.4.1.3). Pero aparecen dos funciones nuevas muy importantes a la hora de mover el manipulador:

- ***simSetJointTargetPosition("nombre_handle", "posición_objetivo")*:** establece una posición objetivo determinada (segundo parámetro) sobre una articulación que opera en modo par/fuerza y el control de posición activado. Esta función permitirá cambiar de posición las articulaciones para dejar a la pinza en una posición en la que pueda agarrar el objeto.
- ***simWait("tiempo_mínimo")*:** es una operación bloqueante, ya que el hilo que se esta procesando en ese momento en la simulación hace una espera del tiempo indicado. Se utiliza para esperar a que una o varias articulaciones terminen de realizar su movimiento y a partir de ahí comenzar la siguiente acción.

El primer paso para la implementación es comprobar el rango de las articulaciones, realizar "pruebas de error" para diferentes movimientos cada una de las articulaciones y de varias articulaciones simultáneamente.

Primero se obtienen y se almacenan los *handles* de las seis articulaciones en la variable *jh*. También se inicializa *iniJ* que contiene una posible configuración inicial de la posición de cada articulación. La posición intrínseca que le pasamos como parámetro a la función *simSetJointTargetPosition* se expresa en radianes por defecto. Para esto hay que utilizar la biblioteca matemática para el número π.

Una vez hemos acabado la inicialización de las variables necesarias, esperaremos la señal producida por el robot móvil al llegar al final de su trayectoria (*simWaitForSignal("turnoMM")*). A partir de aquí comenzarán los movimientos del manipulador:

- Primer movimiento: consiste en llevar al brazo a un punto óptimo para poder manipular el vaso. Para ello se guardan todas las posiciones de las articulaciones en una variable (*desiredJ*), luego mediante un bucle *for* se recorre esa variable fijando cada posición a su respectiva articulación.
- Segundo movimiento: la finalidad es colocar el elemento terminal en el objetivo para que al cerrar la mano el vaso no resbale o caiga. Para cerrar la pinza, primero se espera unos segundos para que todas las articulaciones alcancen su posición, después se crea una señal entera que será recibida en el script asociado a la mano. Luego esperamos 1,25 segundos, que es el tiempo aproximado que tarda la pinza en abrirse o cerrarse.
- Tercer movimiento: después de haber cogido el primer vaso y haber enviado una señal al robot móvil para que inicie la próxima trayectoria, esperamos una nueva señal de la base para que el manipulador pueda volver a trabajar. Una vez recibida, se coloca el brazo con el mismo procedimiento que el primer movimiento a poca altura sobre la superfície (Cubo) que se dejará el vaso. Este movimiento es un poco más lento que los anteriores, por eso aumentaremos el retardo.
- Cuarto movimiento: es el movimiento definitivo en el que se incluyen tres movimientos. El primero posiciona dos articulaciones para centrar el brazo, el segundo mueve la segunda articulación hacia abajo para

amortiguar el movimiento que deja el vaso en la superfície. Por último se abre la mano creando la misma señal que en el cierre pero con un valor entero distinto. De manera opcional se lleva el brazo a la configuración inicial.

En el último movimiento antes de abrir la mano, hace falta crear una señal que la reciba el segundo manipulador móvil. Una vez la haya recibido, realizará una serie de movimientos muy similares a los del primer robot, manipulando el mismo vaso.

Lo único que faltaría es el control de la pinza (*gripper*) o mano robótica. El *script* asociado a este objeto es del tipo *Child Script (non-threaded)*. Éstos se dividen en varias zonas de código, una de inicialización (*sim_childscriptcall_initialization*) y otra de actuación donde reside la parte principal de control (*sim_childscriptcall_actuation*).

En la primera se inicializan los *handles* de los dos motores de la pinza y una variable (*cv*) con la velocidad constante con la que éstos funcionaran.

En la segunda zona, se guarda en una variable (*close*) el valor entero (0 ó 1) de la señal creada en el otro *script* antes de abrir y cerrar la pinza. Posteriormente se comprueba con una condición que si el valor es mayor que 0 se cerrará la pinza, sino se abrirá. Para controlar los motores de la pinza utilizaremos la misma función que para los motores del robot móvil (*simSetJointTargetVelocity*).

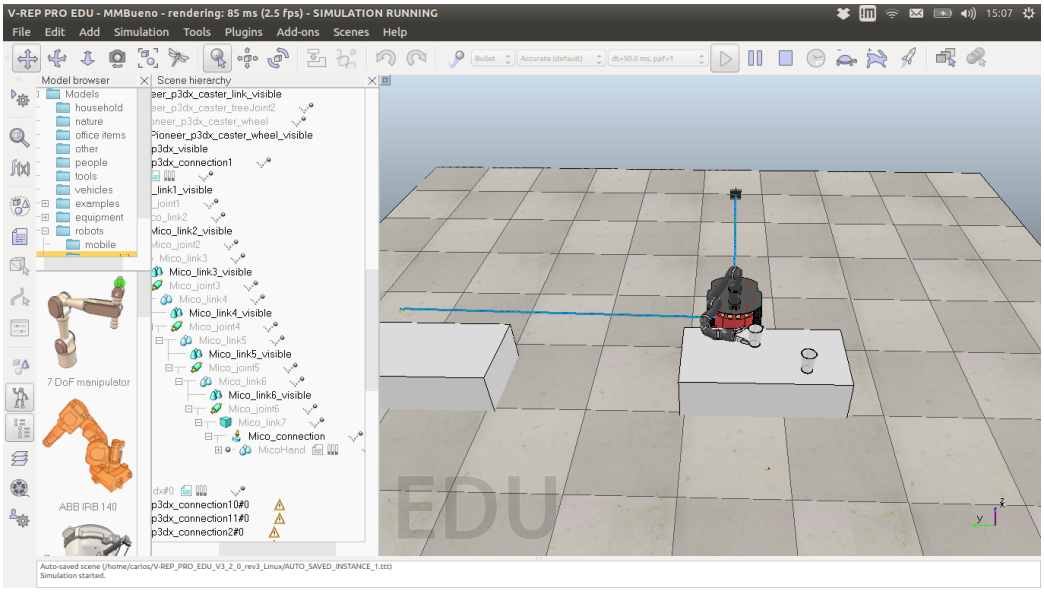


Figura 2.18: Vista en VREP durante la simulación después de toda la configuración

Capítulo 3

Conclusiones

Al comienzo de esta memoria se establecieron unos objetivos que se han conseguido cumplir. En primer lugar, se ha conseguido realizar un entorno de simulación virtual donde se muestra un ejemplo básico de manipulación móvil, permitiendo la posibilidad de inclusión de más módulos y características que nos ofrece el programa VREP con diferentes diseños y modelos. Esto significa que no solo se ha llegado a una solución donde los dos manipuladores móviles son funcionales, sino que además se pueden incluir más técnicas de control o incluso modificaciones de los anteriores de una forma sencilla. Puesto que la manipulación móvil actualmente es un área de investigación en la robótica, este proyecto favorece el diseño y la ceación, ya que otra persona puede empezar a trabajar desde el punto en que el autor ha terminado este proyecto, valorando el intercambio de ideas y la compartición de las mismas.

Personalmente, el proyecto me ha ayudado a familiarizarme con este tipo de simuladores y aprender conceptos de robótica hasta ahora desconocidos. Esto conlleva una motivación extra para futuros estudios de esta rama de informática industrial y automatización.

Finalmente, se ha implementado una solución básica para el control del brazo, pero con una buena funcionalidad en una simulación.

Capítulo 4

Mejoras y trabajos futuros

A lo largo de la realización de este proyecto, han surgido diferentes inconvenientes que se podrían solventar llevando a cabo ciertas mejoras del modelo propuesto en esta memoria. Estas exceden por muchos motivos los objetivos principales que se establecieron al iniciar este proyecto, por lo que las englobo en el marco de trabajos futuros. Podemos destacar las siguientes:

- Diseño propio del manipulador móvil:
  El manipulador móvil utilizado en este proyecto es una composición entre dos modelos ya diseñados por sus respectivos fabricantes que ofrece VREP. Podría haberse diseñado un modelo para el robot móvil y uno para el manipulador, ya que VREP también presenta esta opción de diseño.
- Mejor controlador para el manipulador:
  El controlador que regula los movimientos de nuestro manipulador se ha implementado con un tipo de control simple de posición. Existen otras alternativas de control mejoradas como la cinemática directa/inversa, o también un módulo que ofrece VREP llamado *Motion Planning*.
- Impresión y montaje del manipulador:
  El reciente *boom* de las impresoras 3D también está muy involucrado en el campo de la robótica. Sabiendo esto podemos diseñar nuestro

manipulador, imprimirlo en 3D y realizar su correspondiente montaje. Después nos ayudaríamos de una placa Arduino para el control de servos.

Apéndices

Apéndice A

Código de los *scripts*

A.1. Código del *script* asociado al robot móvil principal

```
simSetThreadSwitchTiming(2)
simDelegateChildScriptExecution()

-- Put some initialization code here:

motorLeft=simGetObjectHandle("Pioneer_p3dx_leftMotor")
motorRight=simGetObjectHandle("Pioneer_p3dx_rightMotor")

path_plan_handle=simGetPathPlanningHandle('PathPlanningTask')
robot_handle=simGetObjectHandle('Pioneer_p3dx')
path_handle=simGetObjectHandle('Path')
planstate=simSearchPath(path_plan_handle,5)
pos_on_path=0
dist=0

start_dummy_handle=simGetObjectHandle('Start')

while (v_lin~=0 and v_ang~=0) do

    rob_pos=simGetObjectPosition(robot_handle,-1)
    path_pos=simGetPositionOnPath(path_handle,pos_on_path)

    simSetObjectPosition(start_dummy_handle,-1,path_pos)

    m=simGetObjectMatrix(robot_handle,-1)
    m=simGetInvertedMatrix(m)

    path_pos=simMultiplyVector(m,path_pos)

    dist=math.sqrt( (path_pos[1]*path_pos[1])+(path_pos[2]*path_pos[2]))
    phi=math.atan2(path_pos[2],path_pos[1])
```


Figura A.1:

```
    if pos_on_path<1 then
        v_lin=0.1
        v_ang=0.7*phi
    else
        v_lin=0
        v_ang=0
        simSetIntegerSignal('turnoMM',1)
    end

    d=0.24  --wheels separation
    v_r=(v_lin+d*v_ang)
    v_l=(v_lin-d*v_ang)

    r_w=0.0975 --wheel radius
    vang_right=v_r/r_w
    vang_left=v_l/r_w

    simSetJointTargetVelocity(motorRight,vang_right)
    simSetJointTargetVelocity(motorLeft,vang_left)

    if (dist<0.1) then
        pos_on_path=pos_on_path+0.01
    end


    simWait(0.025,true)


end
```

Figura A.2:

```
simWaitForSignal('turnoRM')

path_plan_handle2=simGetPathPlanningHandle('PathPlanningTask2')
path_handle2=simGetObjectHandle('Path2')
planstate2=simSearchPath(path_plan_handle2,5)
pos_on_path2=0
dist2=0

start_dummy_handle2=simGetObjectHandle('Start2')

while (v_lin2~=0 and v_ang2~=0) do

    rob_pos=simGetObjectPosition(robot_handle,-1)

    path_pos=simGetPositionOnPath(path_handle2,pos_on_path2)

    simSetObjectPosition(start_dummy_handle2,-1,path_pos)

    m=simGetObjectMatrix(robot_handle,-1)
    m=simGetInvertedMatrix(m)

    path_pos=simMultiplyVector(m,path_pos)


    dist2=math.sqrt( (path_pos[1]*path_pos[1])+(path_pos[2]*path_pos[2]))
    phi=math.atan2(path_pos[2],path_pos[1])
```

Figura A.3:

```
    if pos_on_path2<1 then
        v_lin2=0.1
        v_ang2=0.7*phi
    else
        v_lin2=0
        v_ang2=0
        simSetIntegerSignal('turno2MM',1)
    end

    d=0.24  --wheels separation
    v_r=(v_lin2+d*v_ang2)
    v_l=(v_lin2-d*v_ang2)

    r_w=0.0975 --wheel radius
    vang_right=v_r/r_w
    vang_left=v_l/r_w

    simSetJointTargetVelocity(motorRight,vang_right)
    simSetJointTargetVelocity(motorLeft,vang_left)

    if (dist2<0.1) then
        pos_on_path2=pos_on_path2+0.01
    end


    simWait(0.025,true)

end
```

Figura A.4:

A.2. Código del *script* asociado al manipulador principal

```
simSetThreadSwitchTiming(2)

jh={-1,-1,-1,-1,-1,-1}
for i=1,6,1 do
    jh[i]=simGetObjectHandle('Mico_joint'..i)
end

iniJ={math.pi,math.pi,math.pi,math.pi,math.pi,270*(math.pi/180)}

simWaitForSignal('turnoMM')

--Primer movimiento
jt1=207*(math.pi/180)
jt2=285*(math.pi/180)
jt3=175*(math.pi/180)
jt4=195*(math.pi/180)
jt5=180*(math.pi/180)
jt6=30*(math.pi/180)
desiredJ={jt1,jt2,jt3,jt4,jt5,jt6}
for i=1,6,1 do
    simSetJointTargetPosition(jh[7-i],desiredJ[7-i])
end

simWait(6)

--Segundo movimiento
jt1=199*(math.pi/180)
desiredJ={jt1,jt2,jt3,jt4,jt5,jt6}
for i=1,6,1 do
    simSetJointTargetPosition(jh[7-i],desiredJ[7-i])
end

simWait(1.5)
-- Cerrar la mano
simSetIntegerSignal("hand2",1)
simWait(1.25)
```

Figura A.5:

```
for i=1,6,1 do
    simSetJointTargetPosition(jh[7-i],iniJ[7-i])
end

simSetIntegerSignal("turnoRM",1)

simWaitForSignal('turno2MM')

--Tercer movimiento
jt1=70*(math.pi/180)
jt2=242*(math.pi/180)
jt3=34*(math.pi/180)
jt4=29*(math.pi/180)
jt5=134*(math.pi/180)
jt6=41*(math.pi/180)
desiredJ={jt1,jt2,jt3,jt4,jt5,jt6}
for i=1,6,1 do
    simSetJointTargetPosition(jh[7-i],desiredJ[7-i])
end

simWait(3.25)
```

Figura A.6:

```
--Cuarto movimiento

jt2=252*(math.pi/180)
jt3=68*(math.pi/180)
desiredJ={jt1,jt2,jt3,jt4,jt5,jt6}
for i=1,6,1 do
    simSetJointTargetPosition(jh[7-i],desiredJ[7-i])
end
simWait(3.5)

simSetJointTargetPosition(jh[2],265*(math.pi/180))

simSetIntegerSignal("otherMM",1)
simWait(1)

--Abrir la mano
simSetIntegerSignal("hand2",0)
simWait(1.25)

simSetJointTargetPosition(jh[3],15*(math.pi/180))
simWait(2.5)


iniJ={0,math.pi,math.pi,math.pi,math.pi,270*(math.pi/180)}
for i=1,6,1 do
    simSetJointTargetPosition(jh[7-i],iniJ[7-i])
end
```

Figura A.7:

A.2.1. Código del *script* asociado a la pinza del manipulador principal

```
if (sim_call_type==sim_childscriptcall_initialization) then
    j0=simGetObjectHandle("MicoHand_fingers12_motor1")
    j1=simGetObjectHandle("MicoHand_fingers12_motor2")
    cv=0.04
end

if (sim_call_type==sim_childscriptcall_cleanup) then

end

if (sim_call_type==sim_childscriptcall_actuation) then
    close=simGetIntegerSignal("hand2")
    if not close then
        close=0
    end
    if (close>0) then
        simSetJointTargetVelocity(j0,-cv)
        simSetJointTargetVelocity(j1,-cv)
    else
        simSetJointTargetVelocity(j0,cv)
        simSetJointTargetVelocity(j1,cv)
    end
end
```

Figura A.8:

A.3. Código del *script* asociado al robot móvil duplicado

```
simSetThreadSwitchTiming(2)
simDelegateChildScriptExecution()
-- Put some initialization code here:
motorLeft=simGetObjectHandle("Pioneer_p3dx_leftMotor#0")
motorRight=simGetObjectHandle("Pioneer_p3dx_rightMotor#0")

]while val~=1 do
    simSetJointTargetVelocity(motorRight,0)
    simSetJointTargetVelocity(motorLeft,0)
    val=simGetIntegerSignal("otherMM")
end

path_plan_handle=simGetPathPlanningHandle('PathPlanningTask3#')
robot_handle=simGetObjectHandle('Pioneer_p3dx#0')
path_handle=simGetObjectHandle('Path3#')
planstate=simSearchPath(path_plan_handle,5)
pos_on_path=0
dist=0

start_dummy_handle=simGetObjectHandle('Start3#')

]while (v_lin~=0 and v_ang~=0) do

    rob_pos=simGetObjectPosition(robot_handle,-1)
    path_pos=simGetPositionOnPath(path_handle,pos_on_path)

    simSetObjectPosition(start_dummy_handle,-1,path_pos)

    m=simGetObjectMatrix(robot_handle,-1)
    m=simGetInvertedMatrix(m)

    path_pos=simMultiplyVector(m,path_pos)

    dist=math.sqrt( (path_pos[1]*path_pos[1])+(path_pos[2]*path_pos[2]))
    phi=math.atan2(path_pos[2],path_pos[1])
```

Figura A.9:

```
    if pos_on_path<1 then
        v_lin=0.1
        v_ang=0.7*phi
    else
        v_lin=0
        v_ang=0
        simSetIntegerSignal('turnoMM2',1)
    end

    d=0.24  --wheels separation
    v_r=(v_lin+d*v_ang)
    v_l=(v_lin-d*v_ang)

    r_w=0.0975 --wheel radius
    vang_right=v_r/r_w
    vang_left=v_l/r_w

    simSetJointTargetVelocity(motorRight,vang_right)
    simSetJointTargetVelocity(motorLeft,vang_left)

    if (dist<0.1) then
        pos_on_path=pos_on_path+0.01
    end


    simWait(0.025,true)


end
```

Figura A.10:

```
simWaitForSignal("turno2RM2")

path_plan_handle=simGetPathPlanningHandle('PathPlanningTask4#')
path_handle=simGetObjectHandle('Path4#')
planstate=simSearchPath(path_plan_handle,5)
pos_on_path=0
dist=0

start_dummy_handle=simGetObjectHandle('Start4#')

while (v_lin2~=0 and v_ang2~=0) do

    rob_pos=simGetObjectPosition(robot_handle,-1)
    path_pos=simGetPositionOnPath(path_handle,pos_on_path)

    simSetObjectPosition(start_dummy_handle,-1,path_pos)

    m=simGetObjectMatrix(robot_handle,-1)
    m=simGetInvertedMatrix(m)

    path_pos=simMultiplyVector(m,path_pos)

    dist=math.sqrt( (path_pos[1]*path_pos[1])+(path_pos[2]*path_pos[2]))
    phi=math.atan2(path_pos[2],path_pos[1])
```

Figura A.11:

```
    if pos_on_path<1 then
        v_lin2=0.1
        v_ang2=0.7*phi
    else
        v_lin2=0
        v_ang2=0
        simSetIntegerSignal('turno2MM2',1)
    end

    d=0.24  --wheels separation
    v_r=(v_lin2+d*v_ang2)
    v_l=(v_lin2-d*v_ang2)

    r_w=0.0975 --wheel radius
    vang_right=v_r/r_w
    vang_left=v_l/r_w

    simSetJointTargetVelocity(motorRight,vang_right)
    simSetJointTargetVelocity(motorLeft,vang_left)

    if (dist<0.1) then
        pos_on_path=pos_on_path+0.01
    end
    simWait(0.025,true)
end
```

Figura A.12:

A.4. Código del *script* asociado al manipulador duplicado

```
simSetThreadSwitchTiming(2) -- Default timing for automatic thread switching

jh={-1,-1,-1,-1,-1,-1}
for i=1,6,1 do
    jh[i]=simGetObjectHandle('Mico_joint'..i)
end

iniJ={math.pi,math.pi,math.pi,math.pi,math.pi,270*(math.pi/180)}

simWaitForSignal('turnoMM2')

--Primer movimiento
jt1=270*(math.pi/180)
jt2=206*(math.pi/180)
jt3=80.5*(math.pi/180)
jt4=-9*(math.pi/180)
jt5=-37*(math.pi/180)
jt6=177*(math.pi/180)
desiredJ={jt1,jt2,jt3,jt4,jt5,jt6}
for i=1,6,1 do
    simSetJointTargetPosition(jh[7-i],desiredJ[7-i])
end

simWait(6)
```

Figura A.13:

```
--Segundo movimiento
jt2=215*(math.pi/180)
jt3=86*(math.pi/180)
desiredJ={jt1,jt2,jt3,jt4,jt5,jt6}
for i=1,6,1 do
    simSetJointTargetPosition(jh[7-i],desiredJ[7-i])
end

simWait(1.5)
-- Cerrar la mano
simSetIntegerSignal("hand3",1)
simWait(1.25)

jt2=190*(math.pi/180)
desiredJ={jt1,jt2,jt3,jt4,jt5,jt6}
for i=1,6,1 do
    simSetJointTargetPosition(jh[7-i],desiredJ[7-i])
end

simSetIntegerSignal("turno2RM2",1)

simWaitForSignal('turno2MM2')
```

Figura A.14:

A.4.1. Código del *script* asociado a la pinza del manipulador duplicado

```
if (sim_call_type==sim_childscriptcall_initialization) then
    j0=simGetObjectHandle("MicoHand_fingers12_motor1")
    j1=simGetObjectHandle("MicoHand_fingers12_motor2")
    cv=0.04
end

if (sim_call_type==sim_childscriptcall_cleanup) then

end

if (sim_call_type==sim_childscriptcall_actuation) then
    close=simGetIntegerSignal("hand3")
    if not close then
        close=0
    end
    if (close>0) then
        simSetJointTargetVelocity(j0,-cv)
        simSetJointTargetVelocity(j1,-cv)
    else
        simSetJointTargetVelocity(j0,cv)
        simSetJointTargetVelocity(j1,cv)
    end
end
```

Figura A.15:

Bibliografía

[1] **Tutorial Coppelia Robotics:** tutorial de la empresa desarrolladora de VREP donde se puede consultar cualquier información relativa a su programa.
`http://www.coppeliarobotics.com/helpFiles/`

[2] **MobileRobots:** página oficial del modelo de robot móvil utilizado.
`http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx`

[3] **Foro Coppelia Robotics:** foro oficial de VREP para consultar cualquier tipo de duda relacionada con el programa.
`http://www.forum.coppeliarobotics.com/`

[4] **Robots Industriales:** pequeño documento introductorio a la manipulación industrial.
`http://platea.pntic.mec.es/vgonzale/cyr_0708/archivos/_15/Tema_5.4.htm`

[5] Valera, A., *Tema 4: control de robots móviles.* Mecatrónica