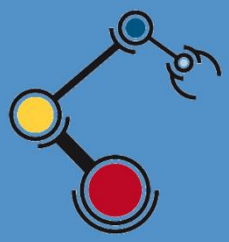
Escuela Politécnica Superior

Control de Posición y Visual de Sistemas de Manipulación Autónomos



Máster Universitario en Automática y Robótica

Trabajo Fin de Máster

Autor:
Sergio Espí Jiménez

Tutor/es:
Jorge Pomares Baeza

Junio 2017

Resumen y Objetivos

El principal objetivo de este proyecto, es implementar un sistema de control proporcional derivativo con compensación gravitacional y un sistema de control visual IBVS, (Image-Based Visual Servoing) para el guiado de distintos tipos de robots.

La implementación de los controladores se realizará en el programa Matlab para aprovechar su potencia matemática así como las facilidades de cálculo que ofrece y las toolbox existentes para el estudio de los controladores sobre robots.

Además se utilizará el programa V-REP para poder visualizar el comportamiento de los robots a los que se le apliquen los controladores estudiados en este trabajo

Estos dos programas se podrán comunicar entre sí, gracias a las posibilidades que ofrece V-REP de actuar como servidor y mantenerse a la espera de recibir comandos de un cliente, que en este caso será Matlab.

Con este sistema de simulación que se utilizará, se podrá permitir desde la simulación de brazos robóticos a vehículos omnidireccionales a los que se les aplicará los controladores implementados.

Abstract

The aim of this project is to implement a proportional derivative control system with gravity compensation and an IBVS (Image-Based Visual Servoing) visual control system to guide different kinds of robots.

The controllers' implementation will be carried out with Matlab because of its mathematical capability, its ease of calculation and its existing toolboxes for studying robot controllers.

In addition, the V-REP programme will be used to visualise the robots' behaviour.

Both programmes will communicate to each other thanks to the V-REP's capability of acting like a server, and waiting for the client's commands, in this case, Matlab.

The simulation system used in this project will make possible to simulate from robotic arms to omnidirectional vehicles, to which controllers will be implemented.

Índice de contenidos

1. Introducción 1
   1.1. Objetivos y justificación 2
   1.2. Estado del arte 2
2. Control multiarticular de robots 8
   2.1. Arquitectura del sistema 8
   2.2. Controlador Proporcional Derivativo con compensación gravitacional 11
      2.2.1. Implementación 12
   2.3. Controlador visual IBVS 21
      2.3.1. Implementación 23
3. Control de un robot omnidireccional 28
   3.1. Arquitectura del sistema 28
   3.2. Controlador visual IBVS para robot omnidireccional 30
      3.2.1. Implementación 32
4. Resultados 36
   4.1. Robot multiarticular 37
      4.1.1. Controlador PD con compensación gravitacional 37
      4.1.2. Controlador Visual IBVS 40
   4.2. Robot omnidireccional 44
      4.1.1. Desplazamiento lateral 45
      4.1.2. Desplazamiento frontal 49
      4.1.3. Desplazamiento completo 50
5. Conclusiones 53
5. Bibliografía 55

Índice de ecuaciones

Ecuación 1. Ley de control PD 11
Ecuación 2. Ley de control PD con compensación gravitacional 11
Ecuación 3. Ley de control visual basado en imagen 21
Ecuación 4. Jacobiano de la imagen 22
Ecuación 5. Matriz de interacción con n características 22
Ecuación 6. Cálculo de la pseudoinversa de la matriz 22
Ecuación 7. Jacobiano de la imagen para vehículo robótico 31
Ecuación 8. Ecuación de la velocidad del cuerpo del robot 32
Ecuación 9. Ecuación de la velocidad de cada rueda del robot 32
Ecuación 10. Distancia euclídea 37

Índice de figuras

Figura 1. Modelado del robot Mitsubishi PA10 8
Figura 2. Opciones de configuración de las articulaciones en V-REP 9
Figura 3. Herramienta Mitsubishi PA10 con cámara incorporada.....10
Figura 4. Muro con patrón que se utilizará en el controlador visual.....10
Figura 5. Diagrama de bloques del controlador PD con compensación gravitacional. (Extraído de [17]) 12
Figura 6. Modelo en Simulink para simulación del controlador.....13
Figura 7. Diagrama de comunicación de V-REP. (Extraído de Wiki de V-REP [22]) 16
Figura 8. Modo de sincronización de V-REP.....20
Figura 9. Robot KUKA YouBot.....28
Figura 10. Robot KUKA YouRobot modificado.....29
Figura 11. Rueda mecanum.....29
Figura 12. Movimiento del robot acorde a la dirección y velocidad angular de las ruedas (extraído de 19) 30
Figura 13. Esquema ejes del vehículo robótico.....31
Figura 14. Articulaciones marcadas que vienen con el vehículo robótico.....33
Figura 15. Rueda con la articulación que se utilizará como motor de la misma.....34
Figura 16. Posición de escape del Mitsubishi PA10.....37
Figura 17. Evolución de las articulaciones del Mitsubishi PA10 con Kp=10 y Kv=10.....38
Figura 18. Evolución de las articulaciones del Mitsubishi PA10 con Kp=500 y Kv=20.....39
Figura 19. Evolución de las articulaciones del Mitsubishi PA10 con Kp=200 y Kv=20.....39
Figura 20.Posición inicial y final del PA10.....41
Figura 21. Error de las distancias entre las características de la imagen de la cámara. Lambda = 0.1.....41
Figura 22. Velocidades de las 6 articulaciones del robot. Lambda = 0.1 42
Figura 23. Posición inicial y final del robot 43
Figura 24. Error de las distancias entre las características de la imagen de la cámara. Lambda = 0.1.....43
Figura 25. Velocidades de las 6 articulaciones del robot. Lambda = 0.1 44
Figura 26. Posición inicial y final del robot 45
Figura 27. Error de las distancias entre las características de la imagen de la cámara. Lambda = 0.1.....46
Figura 28. Velocidades de las 4 ruedas del robot. Lambda = 0.1.....46
Figura 29. Error de las distancias entre las características de la imagen de la cámara. Lambda = 10.....47

Figura 30. Velocidades de las 4 ruedas del robot. Lambda = 10.....47
Figura 31. Error de las distancias entre las características de la imagen de la cámara. Lambda = 0.8.....48
Figura 32. Velocidades de las 4 ruedas del robot. Lambda = 0.8.....48
Figura 33. Posición inicial de la prueba del robot enfrente. 49
Figura 34. Error de las distancias entre las características de la imagen de la cámara. Lambda = 0.8.....49
Figura 35. Velocidades de las 4 ruedas del robot. Lambda = 0.8.....50
Figura 36. Posición inicial/final difícil.....51
Figura 37. Error de las distancias entre las características de la imagen de la cámara. Lambda = 0.8.....51
Figura 38. Velocidades de las 4 ruedas del robot. Lambda = 0.8.....51
Figura 39. Error de las distancias entre las características de la imagen de la cámara maximizado. Lambda = 0.8.....52

1. Introducción

La robótica nació hace ya varias décadas con el fin de automatizar cualquier tipo de trabajo que esté relacionado con el empleo de sistemas mecánicos, electrónicos e informáticos.

Este tipo de automatizaciones, existen sobre todo en el contexto industrial, en el que el uso de robots ha permitido sustituir la actividad física del hombre en cualquier tipo de tareas repetitivas, monótonas o peligrosas, suponiendo una importante mejora en la producción de cualquier tipo de productos en las fábricas y el uso de recursos.

Además se están realizando muchos avances en temas de controladores de posición y controladores visuales para el movimiento de robots como se verá a lo largo de este trabajo.

Este trabajo se ha basado en el diseño y la implementación de un entorno de simulación simple en el que se podrán probar diferentes tipos de controladores en diferentes tipos de robots.

En la actualidad existe una gran necesidad de utilizar simuladores a la hora de programar cosas complejas junto a robots delicados que tienen un coste elevado.
Existen muchos simuladores que ayudan a simular un entorno de trabajo donde estará el robot y que se podrá probar de una manera muy básica el funcionamiento del mismo, algunos ejemplos de estos programas son RobotStudio de la empresa ABB o RobotExpert de la empresa Siemens.

El problema de estos programas, es que se utilizan para hacer pruebas relativamente simples, como hacer pruebas de entorno y alguna función simple para los robots como paletizado o cosas mecánicas son suficiente, pero si como en el caso en el que se trata en este trabajo, se requiere algo más complejo como un controlador para un robot, se necesita de otro programa en el que se pueda programar dicho control del robot.

Es evidente que no se puede poner un controlador a un robot y probarlo en el mismo sin simular previamente su comportamiento, ya que en este caso existiría un peligro muy alto de que el robot pudiese comportarse de una manera errática y poner en peligro tanto al entorno que le rodea como a sí mismo. Por ello se requiere de algún medio en el que poder comprobar si el controlador funciona correctamente en el entorno que se tiene preparado para el robot.

1.1. Objetivos y justificación

El objetivo de este trabajo es proporcionar una manera sencilla de simular diferentes robots y el comportamiento de los mismos con distintos tipos de controladores para poder hacer un estudio sobre ellos.

A lo largo de este trabajo se explicará que herramientas se han utilizado para poder simular cualquier tipo de robot y poder probar cualquier tipo de controlador de una manera sencilla y rápida. También se explicará que controladores se han implementado y se podrá comprobar los resultados que dan dichos controladores en distintos tipos de robots en el entorno de simulación. Además de cómo se ha realizado la comunicación entre el programa de simulación y el programa que ejecutará el controlador.

Se han implementado dos controladores diferentes, un controlador PD con compensación gravitacional y un controlador visual, ambos se ejecutan en el programa Matlab y se implementarán en diferentes tipos de robots, el comportamiento de estos se podrá ver en el programa de simulación VREP.
Ambos controladores se han implementado en un brazo robótico Mitsubishi PA10 y un robot móvil omnidireccional.

1.2. Estado del arte

Como se ha comentado anteriormente, en el mundo de la robótica, existen varias herramientas para poder simular entornos y la cinemática y dinámica de cualquier tipo de robot. Una de esas herramientas es ROS.

ROS, es un conjunto de bibliotecas de software y herramientas que ayudan a construir aplicaciones de robots. La mayor ventaja que tiene ROS frente al resto de programas o herramientas, es que es de código abierto y con una comunidad muy activa y dispuesta a ayudar con los problemas que se tenga en cualquier momento.

Este programa ofrece entre sus muchos paquetes para descargar, uno llamado URDF (Unified Robot Description Format), que es un formato XML con el que se puede representar modelos de cualquier robot y diferentes tipos de sensores.
Con este paquete, se puede hacer estudios sobre la cinemática y dinámica de los robots, además se puede acompañar de la herramienta visual que mantiene también ROS llamada rviz, con ella se podrá ver la simulación del robot que se programe en formato XML.

Algunos investigadores apoyan el uso de ROS para aprender sobre la cinemática y dinámica de robots en lugar de utilizar otras herramientas., como es el caso de unos investigadores de la Savannah State University [1] que destacan que se debería de utilizar ROS para aprender sobre la cinemática y dinámica de los robots en lugar de cualquier otra herramienta.

Esta herramienta tiene bastantes ventajas como por ejemplo:

- Consiste en un grupo de procesos que pueden estar en diferentes sistemas, están conectados entre ellos en una topología punto a punto, dando mucha flexibilidad a la hora de programarlo.
- Sus paquetes se pueden programar en diferentes lenguajes, como C++, Python, Octave y LISP, aunque los más extendidos y utilizados son los dos primeros.
- Y como ya se ha comentado anteriormente, es una herramienta completamente gratuita y con una comunidad muy grande y que está en crecimiento en los últimos años.

Como se puede observar, ROS es una herramienta bastante poderosa y que funciona muy bien a la hora de investigar con robots.

El problema es que no es una herramienta fácil para cualquier usuario. Desde su instalación, aunque en su página oficial vienen tutoriales de como instalarlo, siempre tiene que ser en algún sistema operativo específico, impidiendo que se pueda utilizar algunos sistemas como puede ser Windows, y en la mayoría de los casos el usuario siempre se encuentra con muchos errores y problemas a lo largo del proceso de instalación.

Y en el caso del paquete URDF, tiene varias limitaciones que son un problema:

- Para programar varios robots, se tienen que crear varios ficheros XML, ya que solo se admite un robot por fichero XML.

  Para hacer múltiples robots se tendría que utilizar xacro, que es un lenguaje de macros XML que mejora a URDF.
- Solo se pueden usar 3 estructuras, es decir, no se pueden hacer robots paralelos.
- No ofrece ninguna posibilidad de hacer links flexibles, es decir, que todos los links que aparecen en el robot tienen que ser rígidos.

Es evidente que tiene muchas desventajas y que aún necesita más desarrollo para que sea una herramienta tan potente como las que hay en el mercado actualmente.

Por ello lo que se propone en este trabajo, es evitar utilizar esto y usar otras herramientas que eviten todas estas desventajas comentadas.

En cuanto a la simulación y visualización, si se utilizan las herramientas mencionadas anteriormente, la visualización se vería a través de Gazebo. Una herramienta también libre que ofrece la comunidad de ROS para realizar simulaciones. Pero también tiene varios problemas, y si se compara este visualizador, con el software que se propone en este trabajo llamado V-REP, como hizo un investigador de la Universidad de Campinas [2], se puede observar que V-REP es mucho más intuitivo y fácil de utilizar como simulador que Gazebo. Además para Gazebo, es necesario un número muy grande de herramientas externas para ponerlo en marcha y que funcione junto con ROS.

Como se verá más adelante en este trabajo, la puesta en marcha que se propone, es mucho más sencilla y rápida a través de Matlab y V-REP que con las herramientas comentadas.

Con esas herramientas, se han desarrollado dos controladores diferentes para hacer un estudio sobre ellos.

El control de robots, es un área que se empezó a estudiar hace muchos años, pero a día de hoy sigue siendo un tema que continúa bajo estudio.

Uno de los controladores puestos a estudio en este trabajo, es el controlador proporcional derivativo (PD) con compensación gravitacional.
Este controlador como se podrá ver más en detalle en los próximos capítulos de este trabajo, es uno de los controladores más sencillos pero que consigue unos resultados muy satisfactorios a la hora de posicionar un robot en alguna posición deseada.

Este controlador apareció por primera vez en una investigación hecha en 1981 [3]. En esta investigación se propuso este método para el control de coordenadas que puede ser fácilmente implementado por un microordenador para un robot que sólo cuenta con uniones rotativas.
El controlador proporcional derivativo con compensación de gravedad, lleva en uso para muchos robots industriales desde hace mucho tiempo.

Este controlador calcula y añade al control tipo PD el par que ejerce la fuerza de gravedad utilizando parte del modelo dinámico del robot lo cual permite hacer un control mucho más preciso de las articulaciones de los robots [4].

Como se ha comentado anteriormente, la popularidad de este controlador viene de que además de que es muy sencillo a la hora de implementar, es capaz de garantizar la estabilidad asintótica en la regulación de las posiciones articulares del robot con mínimo error en el estado estacionario [4] [5].

Este controlador, es normalmente llevado a cabo por una selección de las ganancias del controlador que son siempre constantes.

En una de las investigaciones de Rafael Kelly [6], se demuestra que se puede mejorar el controlador para que con ganancias variables las articulaciones vayan más rápido a la posición deseada y la alcancen con un error menor.

A lo largo de los años, se han investigado varias mejoras para el controlador con diferentes métodos para calcular las ganancias en cada instante de tiempo, como por ejemplo utilizando lógica difusa o incluso aproximaciones con redes neuronales [7] [8] [9].

En el caso de la lógica difusa, algunos autores [10] [11] defienden que funciona con unos resultados muy satisfactorios para la regulación de las posiciones articulares en diferentes configuraciones cinemáticas. Pero sostienen que existe la posibilidad de que haya una sintonización inadecuada del modelo o que con diferentes técnicas de parametrización, conduzcan a un diseño de las particiones difusas que conduzcan al resultado de un comportamiento no deseable del controlador.
Por ello proponen mejorar el PD con compensación de gravedad con lógica difusa de intervalo de tipo 2. Este tipo de lógica fue introducida en 1975 [12], pero es realmente durante los últimos años cuando se está empezando a utilizar [13].

En cuanto al segundo controlador que se va a poner a estudio en este trabajo, es un controlador visual, por el que mediante los datos que proporcione la cámara que tenga instalado el robot manipulador o el robot móvil, se podrá controlar el movimiento del mismo.

A esta técnica, se le llama por sus siglas en inglés IBVS, Image-Based Visual Sensor. Este controlador, utiliza muchas áreas de estudio como el procesamiento de imágenes, la visión por computador, el modelado de los sistemas dinámicos de robots y la teoría de control no linear.

A pesar de tratar muchas áreas, esta técnica no tiene una implementación demasiado compleja, como ya se verá en los siguientes capítulos de este trabajo, pero que muchos

investigadores desde hace años están estudiándolo y está dando unos resultados bastante buenos.

Como es el caso de los investigadores François Chaumette y S. Hutchinson que en uno de sus artículos [12] en el que hablan de este controlador, hicieron pruebas con un brazo robótico de 6 grados de libertad con una cámara en la última articulación del robot como si fuese parte de una herramienta. Esto se ha probado en este trabajo y se podrán ver los resultados que se obtienen en los siguientes apartados.
En este artículo, hicieron varias pruebas en las que el brazo tenía que aproximarse a una imagen que contenía un cuadrado con unos círculos dentro. El controlador procesa la imagen y hace que el robot se posicione en el lugar correcto. Los resultados obtenidos son con un error muy pequeño como luego se demostrará.

Con diferentes adaptaciones de este controlador, se puede llegar a conseguir soluciones para muchos problemas que hoy en día están bajo estudio constante.
Como ejemplo de aplicación de este controlador [13], se ha llegado a hacer que un robot que tiene incorporados dos brazos robóticos, pueda coger un bolígrafo utilizando dos cámaras que proporcionan información de la escena. Una de ellas colocada en el brazo robótico y otra que está fija viendo la escena. Con estas cámaras y mediante el controlador visual, el robot logra coger el bolígrafo con unos resultados satisfactorios.

Además de utilizarse en brazos robóticos, este tipo de controlador se puede aplicar a cualquier tipo de vehículo robótico, como es el caso de UAVs. Hay muchos investigadores que han hecho pruebas de este tipo de controladores con estos robots en entornos controlados [14]. En estos entornos el UAV va a la posición deseada que está observando con la cámara que tiene incorporada.

También se ha propuesto utilizar vehículos aéreos no tripulados para misiones de detección en océanos [15] en los que pueden salir de un barco e investigar la zona de manera autónoma gracias a este tipo de controladores en los que puede ver si hay algo e ir a la zona y seguir a su objetivo cuando lo detecte.

Además si se mezclan el controlador IBVS con UAVs y técnicas de grasping, se puede llegar a conseguir simular el comportamiento de un águila o algún pájaro cuando coge algo con sus garras mientras está en vuelo [16].

Aunque en el mundo de los drones se puede observar con las investigaciones anteriores que este controlador podría ser muy útil a la hora de ayudar a llegar a un punto concreto, también se está desarrollando este controlador para robots de rescate en tierra [17].

Ya se han visto muchos tipos de robots que utilizan estos tipos de controladores, a ellos se les puede sumar los vehículos robóticos omnidireccionales. Estos son los vehículos que pueden moverse a cualquier dirección gracias a las ruedas que llevan, que les permiten ir a los lados sin necesidad de maniobrar, son las llamadas ruedas “mecanum”.

Como se ha demostrado en varios estudios realizados [18] [19], las velocidades de un vehículo robótico sin este tipo de ruedas y las ruedas de ese tipo de vehículos, están directamente relacionadas por sus velocidades y por las características del robot cómo se demostrará en los próximos apartados de este trabajo.

2. Control multiarticular de robots

A lo largo de este apartado, se explicarán los controladores que se han utilizado en este trabajo para un brazo robótico multiarticular Mitsubishi PA10.

Se han utilizado dos controladores diferentes, un controlador de posición proporcional derivativo con compensación gravitacional y un controlador visual.

2.1. Arquitectura del sistema

La simulación del sistema, se realizará en el programa V-REP. Este simulador, como se ha comentado en apartados anteriores, es muy intuitivo y bastante fácil de utilizar.

Es también muy útil a la hora de programar comportamientos en el robot ya que dispone de la posibilidad de añadirle scripts para que el robot funcione en la simulación. Y además ofrece al usuario muchos robots de diferentes clases, desde robots fijos y brazos articulares hasta robots móviles como el robot que se ha utilizado para probar el siguiente apartado de este trabajo.

Además de los robots que tiene por defecto, V-REP da la opción de importar cualquier modelado que este en formato obj, dxf, 3ds o stl. De esta manera, se agregó un modelado del PA10, como se puede observar en la parte izquierda de la figura 1.

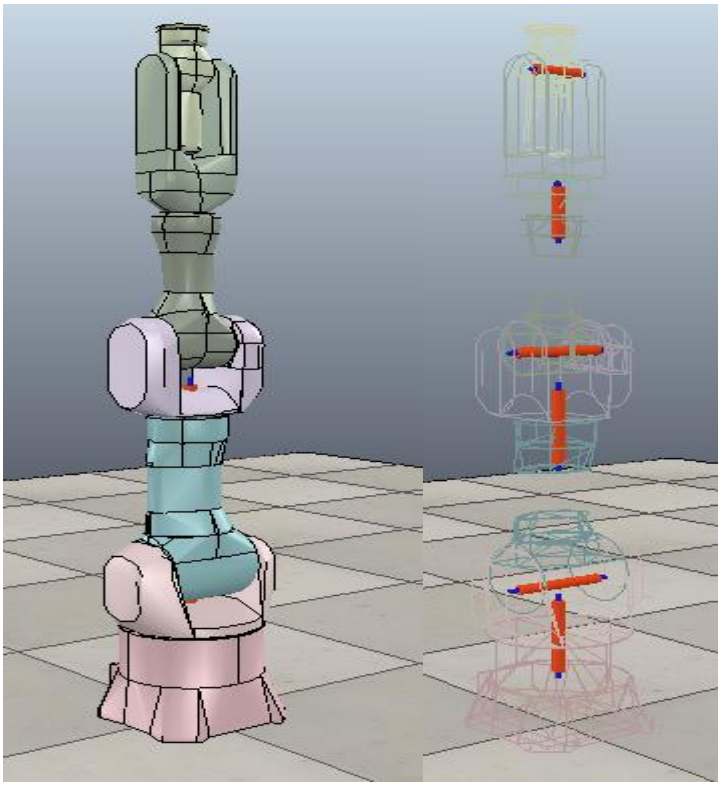


Figura 1. Modelado del robot Mitsubishi PA10

Una vez importado el modelo, se le añadieron los motores en las articulaciones como se puede observar en la parte derecha de la figura 1, en la que se puede observar que los cilindros naranjas serán las articulaciones que harán de motor para el robot.
V-REP permite cambiar muchos parámetros de los motores para que el robot se simule como si del robot real se tratase.

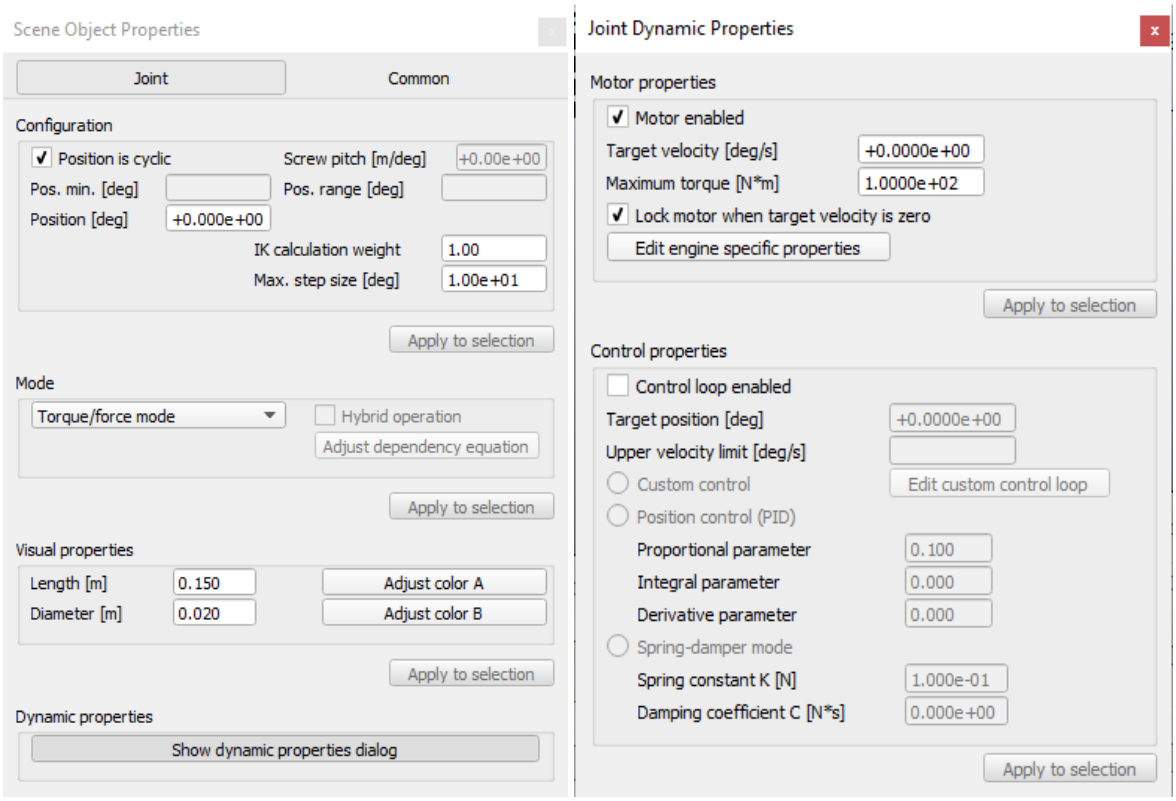


Figura 2. Opciones de configuración de las articulaciones en V-REP

Como se puede observar en la figura 2, a la articulación simulada, se le puede cambiar cualquier parámetro para hacer que se parezca realmente al motor que tiene el robot en su interior y el robot actúe como si fuese una articulación real. Se le pueden cambiar desde la longitud del tamaño de la articulación hasta el máximo torque que puede tener.

Para el caso del estudio del controlador visual, se añadió al final del robot, un cilindro cómo si fuese una herramienta que se le añadió al PA10 y en la punta de esta una cámara que se utilizará para extraer características de la imagen como se puede observar en la parte izquierda de la figura 3.

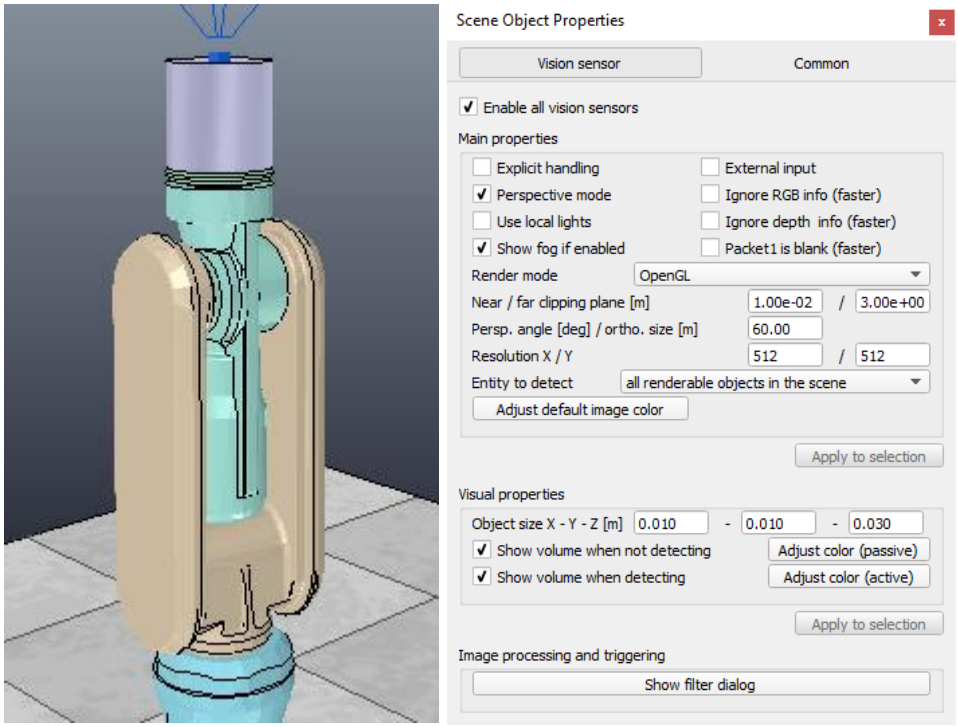


Figura 3. Herramienta Mitsubishi PA10 con cámara incorporada

A la cámara, cuando se añade en V-REP, se le pueden cambiar también los parámetros para que coincida con la que después se utilizará en el entorno real.

En este caso los parámetros escogidos se pueden ver en la parte derecha de la figura 3. No se han escogido unos parámetros que simulen una cámara muy potente, ya que en el entorno que se va a simular está bien iluminado y tiene colores básicos que no requieren de una cámara muy potente para que el algoritmo funcione correctamente como se verá más adelante.

Además de añadirle la cámara al robot, se le ha añadido a la escena un plano que hará de pared y otro plano más pequeño con el patrón que el robot utilizará para su controlador, como se puede observar en la figura 4.

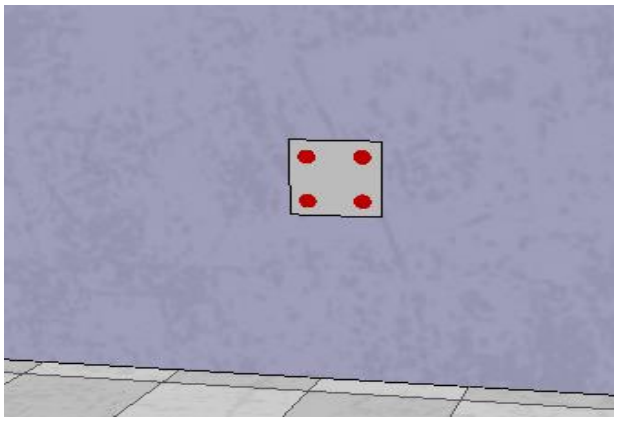


Figura 4. Muro con patrón que se utilizará en el controlador visual

Una vez explicadas las escenas que se han utilizado para el estudio de los controladores en brazos robóticos, se va a explicar los dos tipos de controladores que se ejecutarán en Matlab y se verá la simulación en el programa V-REP

En los siguientes apartados se explicará cómo se han implementado los controladores y cómo se han simulado con el brazo robótico Mitsubishi PA10.

2.2. Controlador Proporcional Derivativo con compensación gravitacional

Uno de los controladores que se han estudiado en este trabajo, ha sido el controlador PD con compensación de gravedad. Este controlador, es un controlador de posición pura que se puede realizar potencialmente mediante la técnica de control PD.

El controlador PD tiene una ley de control muy sencilla, siendo la siguiente:

$$\tau = K_p\tilde{q} + K_v\dot{\tilde{q}}$$

Ecuación 1. Ley de control PD

En ese caso, el controlador PD se utiliza para robots cuyos modelos dinámicos no poseen el vector de pares gravitacionales.

Pero en este trabajo se estudiará el controlador PD con compensación gravitacional, ya que el modelo de robot utilizado, sí que posee vector de pares gravitacionales.

Este controlador, es capaz de satisfacer el control de posición de las articulaciones del robot de n grados de libertad (DOF).

La ley de control PD con compensación de gravedad es la siguiente:

$$\tau = K_p\tilde{q} + K_v\dot{\tilde{q}} + g(q)$$

Ecuación 2. Ley de control PD con compensación gravitacional

Donde K_p y K_v son matrices definidas positivas denominadas ganancias de posición y velocidad (o derivativa), respectivamente. $\tilde{q}$ es el vector de error de la posición, es decir, es la diferencia entre la posición deseada y la actual. Y por último $\dot{\tilde{q}}$ es la velocidad articular actual.

La única diferencia de este controlador y el anterior, es en que se requiere, como se ha comentado anteriormente, del vector de pares gravitacionales $g(q)$.

Esto implica que a diferencia del controlador PD, en este sí que es necesario conocer la estructura del modelo del robot. Pero como se explicará más adelante, gracias a la toolbox de robótica para Matlab se conseguirá de una manera sencilla.

En la imagen extraída del libro escrito por R.Kelly (figura 1) se podrá ver cómo funciona el controlador con un diagrama de bloques.

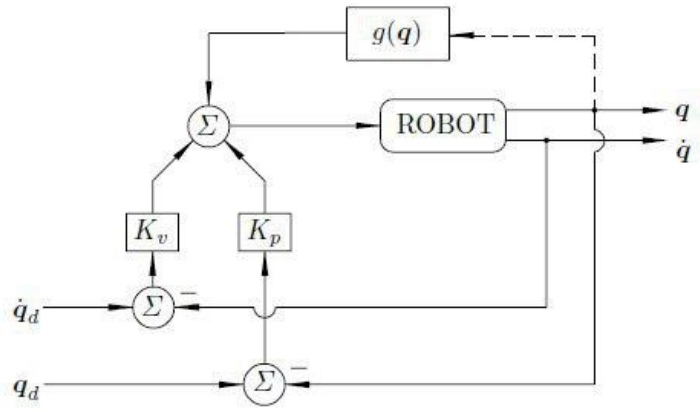


Figura 5. Diagrama de bloques del controlador PD con compensación gravitacional. (Extraído de [17])

2.2.1. Implementación

Para realizar las pruebas de este controlador, se ha implementado en el programa Matlab, que será el que se encargue del peso de las operaciones.

Matlab es una herramienta de software matemático que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio.

Como se ha podido observar en apartados anteriores de este trabajo, Matlab es actualmente uno de los programas más utilizados a la hora de hacer estudios sobre cinemática y dinámica de robots. Esto es así porque además de tener mucha potencia en cuanto a cálculo matemático, existen muchas toolbox que se le pueden añadir.

Una de ellas es la toolbox de robótica desarrollada por Peter Corke [20], la cual utilizan muchos investigadores a día de hoy y defienden que es la mejor manera para aprender estos temas de robótica, como es el caso del autor John J. Craig [21] que hace una introducción a la robótica en su libro enfocándola al control de robots y propone varios ejercicios en los que recomienda esta toolbox para solucionarlos.

Esta toolbox además es gratuita y proporciona muchas funciones que son muy potentes a la hora de realizar estudios y simulación en Matlab de brazos robóticos. Se utilizan sobre

todo para el estudio de la cinemática, dinámica y generación de trayectorias de estos brazos robóticos.

Para la implementación de este controlador, se ha implementado con scripts de Matlab y se ha utilizado Simulink.

El pseudocódigo del algoritmo utilizado para llevar a cabo la aproximación del brazo robótico a la posición deseada es el siguiente:

```
Establecer modelo del robot con la toolbox en Matlab.

Establecer la conexión con el simulador V-REP.

while (tActual < tFinal)

    • Cargar las posiciones y velocidades de las articulaciones
      del robot del simulador para su posterior procesamiento.
    • Ejecutar el controlador PD con compensación gravitacional.
    • Mandar las velocidades obtenidas del controlador al
      simulador.
Fin del while
```

El aspecto que tendrá en Simulink es el que se puede observar en la figura 6.

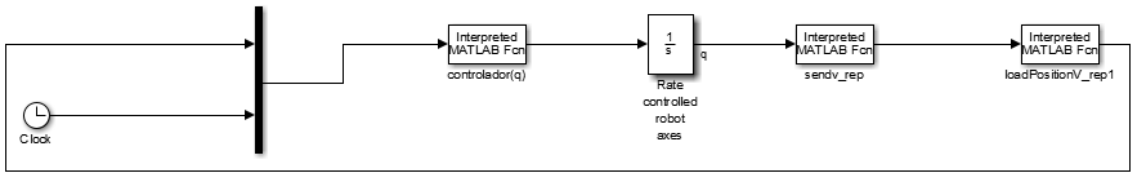


Figura 6. Modelo en Simulink para simulación del controlador

Este controlador, se aplicará al robot Mitsubishi PA10. El modelo de este robot, se cargará en Matlab haciendo uso de la toolbox comentada anteriormente.
El modelo en del robot, se carga en un objeto de Matlab proporcionándole los parámetros de Denavit – Hartenberg, como indica Peter Corke en su libro [20].

```
theta = zeros(1,6);                 % link rotation angle
D = [ 0.317 0 0 0.480 0 0.070 ];    % link offset distance
sigma = zeros(1,6);                 % joint type (0 for revolute)
offset = [ 0 -pi/2 pi/2 0 0 0 ];    % offset for theta (for zero-pose)
```

Como se puede observar en el código anterior, hay seis valores por cada parámetro de Denavit – Hartenberg, eso es porque cada valor de cada vector será el de cada articulación del PA10.

Además de estos parámetros, también se le añaden datos sobre la inercia de los tensores de las articulaciones, la viscosidad de la fricción del motor, los datos de inercia, etc.

Gracias a la toolbox, se puede modelar perfectamente el robot que queramos siempre que se conozcan los datos del mismo.

Una vez generado el objeto que haría de modelo, se crea el objeto que en la toolbox hará de robot y al que luego se le podrá mandar comandos para calcular parámetros para el controlador.

Una vez obtenido el modelado del robot, como se ha explicado en apartados anteriores, y cuando se tiene preparado el entorno de la simulación, faltará por añadirle un script a cualquier elemento que se tenga en la simulación para poder conectar Matlab con V-REP.

Con ese script, lo que se hará es decirle a V-REP que funcione como un servidor que se quedará a la espera de recibir operaciones. En este script, sólo habrá una línea de código que será la siguiente:

```
simExtRemoteApiStart(19999)
```

Con el comando anterior, se le indica a V-REP que cuando se inicie la simulación, se inicie como servidor y espere peticiones en el puerto que se le pase por parámetros.
Este servidor, cerrará el puerto y se desconectará en el momento en el que se termine la simulación.

Para utilizar Matlab como cliente para la simulación y poder hacer que se comuniquen entre ellos, hay que añadir unos ficheros que proporciona V-REP en su propia instalación. Estos ficheros harán que cuando desde Matlab se quiera mandar un comando de los que utiliza V-REP para interactuar con la escena que haya en simulación, solo haya que añadirle al comando un “vrep.” delante del mismo desde el script de Matlab.

Una vez añadidos los ficheros necesarios, hay que crear un script en Matlab para poder conectar con el simulador en el que se le añadirá lo siguiente:

```
vrep=remApi('remoteApi'); % using the prototype file (remoteApiProto.m)
vrep.simxFinish(-1); % just in case, close all opened connections
clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5);
```

Con esas líneas de código, se le está diciendo al simulador que cargue todos los posibles comandos que se le pueda enviar al simulador desde el cliente.

Luego se cierra cualquier conexión existente con ese servidor y se abre una nueva conexión a la que se le dirá la dirección IP donde esté conectado el servidor, lo que indica que si estuviese el simulador en otro ordenador que no fuese el mismo que el cliente, se podría conectar con tan solo cambiar ese parámetro.
Además de la dirección, se le pasa el puerto que está escuchando V-REP, se le indicará con el primer true que bloqué la función hasta que se conecte o hasta que pase un tiempo específico y con el segundo true se le indica que no permita al servidor atender más peticiones de conexión, y en los últimos parámetros, se le indica el tiempo de espera al intentar realizar la conexión y la frecuencia con la que se envían y reciben los paquetes de datos respectivamente.

En caso de que no se consiga realizar la conexión, V-REP devolvería un -1 y no permitiría el envío de comandos al servidor, si la conexión se completa con éxito devuelve el número de cliente que asigna el servidor al cliente.

Para poder mandarle posteriormente las velocidades que se calculen a las articulaciones, o leer sus posiciones desde el lado del cliente, en este caso desde Matlab, se debe de enviar al servidor algunos comandos para que se prepare para recibir peticiones y enviar datos al lado del cliente.

El código del lado del cliente para indicarle eso será el siguiente:

```
[res,objs]=vrep.simxGetObjects(0,vrep.sim_object_joint_type,vrep.simx_opmode_blocking);
if(res==0)
    % Subscripcion
    for i=1:6
        [returnCode(i)] = vrep.simxGetJointPosition(0,objs(i),vrep.simx_opmode_streaming);
        [returnCode(i)] = vrep.simxSetJointTargetVelocity(0,objs(i),0,vrep.simx_opmode_streaming);
    end
    disp('Subscripcion realizada');
else
    disp('Error en la lectura de los joints');
end
```

Con el comando de la primera línea del código anterior, se le pide a V-REP la identificación de las articulaciones de la escena que se esté simulando, en este caso como sólo se tendrá en la escena las articulaciones del robot, se le piden todas las articulaciones. Más adelante se explicará cómo conseguir sólo algunas articulaciones.

Como se puede observar, al final de los comandos que se envían a V-REP, siempre se le indica un modo de comunicación.

En el caso de la suscripción se le indica que se tiene que comunicar en un modo de streaming.

Se le llama suscripción porque lo que hace con ese comando es indicarle al lado del servidor que ese comando se esté ejecutando constantemente almacenándose en un buffer la información para poder recogerla o enviarla desde el lado del cliente cuando se le mande un comando de ese tipo. De esta manera en V-REP siempre están los últimos datos almacenados. Esto se ha reflejado en la figura 3, en la que se puede observar que en el lado del cliente cuando se solicita información, siempre se recoge la información del último paso de ejecución del servidor.

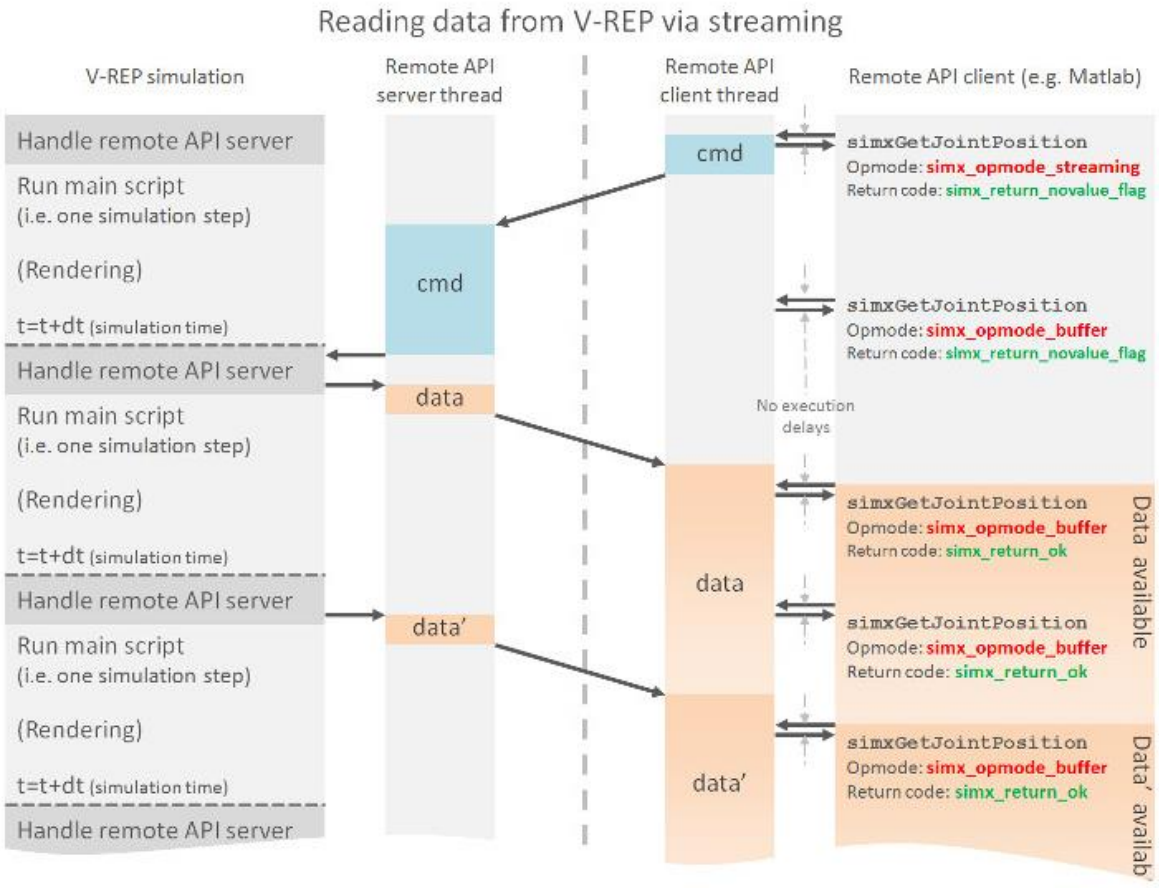


Figura 7. Diagrama de comunicación de V-REP. (Extraído de Wiki de V-REP [22])

Una vez que se ha suscrito a los comandos de recibir las posiciones y enviar las velocidades a las articulaciones, se puede empezar con el modelo de Simulink que se ha mostrado en la figura 2.

En la primera iteración de ese modelo de Simulink, se le pasarán al controlador velocidades y posiciones 0, es decir, que hasta que no se recoja ninguna posición del simulador, no se empezará a ejecutar el controlador y todas las velocidades enviadas serán 0.

El código del controlador PD con compensación gravitacional que se ha utilizado en este trabajo, es el siguiente:

```
function y = SCAcontrol(u,PA10)

% Posicion deseada que alcanzarán las articulaciones
qd = [0 0.523599 0 pi/2 0 -1.047198];

% Posicion articular actual pasada por parametros al controlador
q = u(1:6)';

% Velocidad articular actual pasada por parametros al controlador
qv = u(7:12)';

% Error de posicion calculada en cada instante de tiempo
Q = qd - q;

% Ganancias del controlador
Kp = 200;
Kv = 20;

% Carga gravitacional de las articulaciones
g = PA10.gravload(q);

% tau
tau = Kp*Q - Kv*qv + g;

% Aceleracion articular en respuesta a los pares tau
y = PA10.accel(q, qv, tau);
```

Como se puede observar en el código de arriba, el controlador es muy sencillo gracias a la toolbox de robótica que se ha explicado anteriormente.

Al controlador, le llegarán por parámetros, el modelo del robot que se va a utilizar, en este caso el modelo del PA10 que se ha cargado anteriormente, y un vector de 12 posiciones en las que tendrá en los 6 primeros valores, las posiciones articulares que serán recogidas de V-REP como se comentará después y en las 6 últimas, las últimas velocidades que se han calculado y enviado al robot. Mientras que el cliente no consiga enviar las velocidades o leer las articulaciones, este vector, será siempre 0 para que el robot no se mueva mientras no se le proporcionen al controlador los datos reales extraídos de V-REP.

En este controlador, también se le indicarán sus ganancias, que son los parámetros que se cambiarán para ajustar el controlador y que el robot alcance la posición deseada de una manera más correcta, esta parte se explicará más en detalle y se podrá ver cómo actúan las ganancias en el controlador en el apartado de resultados de este trabajo.

En los últimos tres pasos del controlador, se puede ver cómo se carga en la variable g la carga gravitacional de las articulaciones, esto se hace de una manera sencilla gracias a la

toolbox que ya lo proporciona automáticamente con la información del modelo que se le ha cargado en los pasos anteriores.
De la misma manera, las aceleraciones de las articulaciones que se utilizarán para pasarle las velocidades al robot, también se consiguen a través de esta toolbox con un simple comando en el que se le pasa las posiciones y las velocidades actuales del robot y el torque, que es lo que se calcula con la ley de control.

Una vez se ha obtenido las aceleraciones articulares, hay que pasarlas a velocidades, ya que V-REP sólo admite comandos en los que se le envíen velocidades, no aceleraciones. Para ello se utiliza el integrador de Simulink, para conseguir las velocidades de una manera muy sencilla, ya que lo que hace es integrar con respecto al tiempo las aceleraciones que se le pasen, esto actúa como una caja negra para el usuario, devolviendo las velocidades sin necesidad de que el usuario tenga que implementar ese paso.

Una vez obtenidas las velocidades que se le tienen que mandar al robot, se utiliza el siguiente código para enviarlas al simulador.

```
function vel = sendV_REP(u,objs)
    if(vrep.simxGetConnectionId(clientID)~=-1)
        % Envia las velocidades a los joints
        vrep.simxPauseCommunication(clientID,1);
        disp('Enviando velocidades a los joints');
        for i=1:6
            [returnCode(i)] = vrep.simxSetJointTargetVelocity(clientID,objs(i),u(i),vrep.simx_opmode_oneshot);
        end
        vrep.simxPauseCommunication(clientID,0);
        if(returnCode ~=0)
            disp('No se han enviado las velocidades a los joints');
            vel = zeros(1,6);
        else
            disp('Se han enviado las velocidades a los joints');
            vel = u;
        end
    end
```

A la función anterior, como se puede observar, se le pasan por parámetros los identificadores de las articulaciones de la escena que se ha visto anteriormente cómo se han conseguido y el vector de velocidades que se han calculado en el paso anterior con la ley de control de este controlador y después integrando su resultado.

El primer comando que se le envía en la función de arriba, es un comando que permite parar temporalmente el hilo de comunicaciones para enviar datos a V-REP.
Esto se hace en el caso de querer enviar muchos datos a la vez y que sean evaluados en el mismo paso de ejecución del simulador, como es el caso del envío de velocidades para las articulaciones de un robot. Por ello se para la comunicación, y esto se realiza indicándole

en el comando un 1, para indicarle que vuelva a correr de una manera normal la comunicación, se le envía un 0 con el mismo comando.

Y como se ha comentado anteriormente, en el caso de que el envío de las velocidades al servidor desde el cliente falle, se pondrá el vector de las velocidades a 0.

Una vez enviadas las velocidades satisfactoriamente, se tienen que cargar de nuevo las posiciones de las articulaciones para que se continúe con el algoritmo.

La recogida de información de las posiciones de las articulaciones, se realizará con el código siguiente:

```
function vectorPosVel = loadPositionsVrepSimulink(u,objs)
    q = zeros(1,6);
    returnCode = zeros(1,6);

    if(vrep.simxGetConnectionId(clientID)~=-1)
        % Saca la posicion de los joints
        for i=1:6
            [returnCode(i),q(i)] = vrep.simxGetJointPosition(0,objs(i),vrep.simx_opmode_buffer);
        end
        if(size(u)==1)
            vectorPosVel = zeros(1,12);
        elseif(returnCode(1:6) == 0)
            vectorPosVel = q;
            vectorPosVel(7:12) = u(1:6);
        else
            vectorPosVel = zeros(1,12);
        end
    end
```

Como se puede observar con el código de arriba, la función recibirá por parámetros, tanto los identificadores de las articulaciones, como un vector en el que estarán las velocidades que se acaban de calcular en el controlador y se han enviado al robot.

Esta función lo que realizará es leer de V-REP las posiciones que tiene el robot en ese momento y las añadirá en los primeros 6 elementos del vector resultante y en los 6 últimos elementos del vector añadirá las velocidades que ha recibido por parámetros para que, como se puede observar en la figura 2, se pase al controlador este vector para que pueda continuar con el algoritmo.

Todos los comandos de V-REP devuelven un código de confirmación en el cual, si el servidor consigue realizar el comando enviado, devuelve un 0. Por ello hay comprobaciones de que se hayan recogido todos los valores de las articulaciones correctamente.

En el caso de que la lectura de las posiciones articulares diese un error y no devolviese el código de confirmación, le pasaría al controlador un vector de 0 como se ha comentado anteriormente.

Es posible que en algunos ordenadores al no ser muy potentes, la simulación de V-REP y el procesamiento de Matlab, hagan que ralentice demasiado la lectura de los datos del servidor en el cliente haciendo que ocurran comportamientos que no deberían de producirse.

Para solucionar este tipo de problemas, V-REP proporciona un modo de sincronización para hacer que esto no ocurra. Desde el cliente, se le puede indicar al servidor que se ponga en el modo de sincronización que ofrece V-REP para estos casos, esto se realiza con la línea de código siguiente:

```
simxSynchronous(clientID, true)
```

Con el comando anterior, V-REP se mantiene a la espera de que el cliente le envíe un disparador para dar el siguiente paso de simulación, esto se realiza desde el cliente con el comando siguiente:

```
simxSynchronousTrigger(clientID)
```

Para entender mejor cómo es el comportamiento de estos comandos, se puede observar la figura 8.

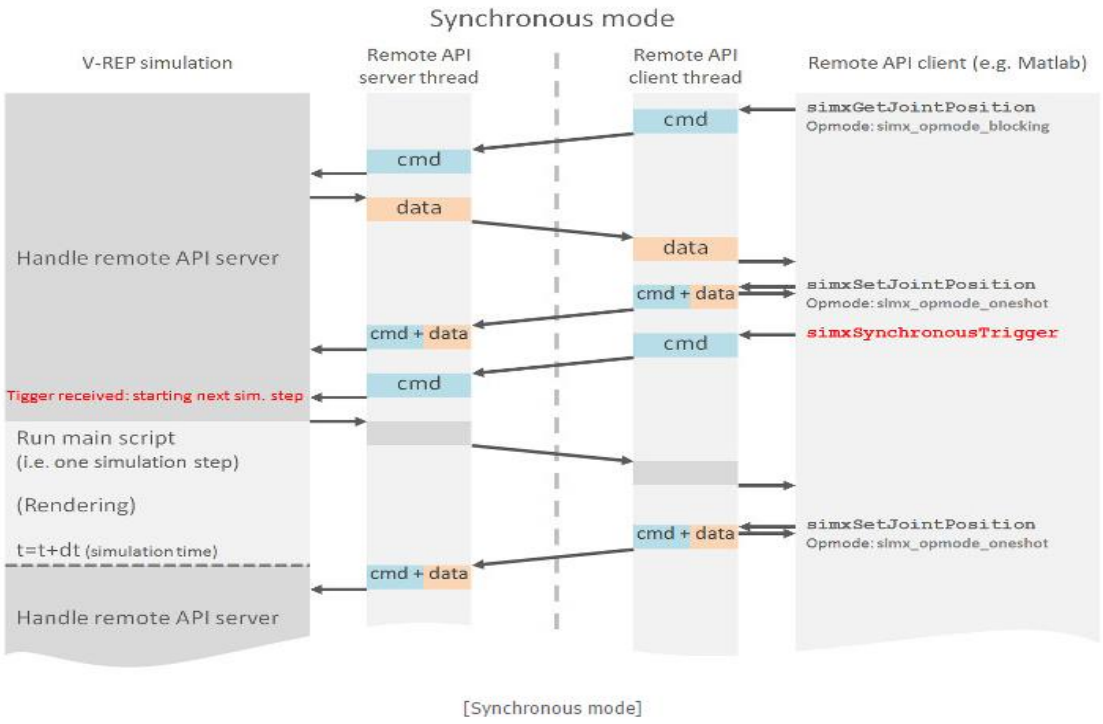


Figura 8. Modo de sincronización de V-REP

En la figura 8 se puede observar que en caso de que desde el lado del cliente se envíen dos comandos de lectura de datos al servidor, sin que el cliente le haya dicho que avance el paso de la simulación entre medias de ambas peticiones, el cliente recibirá los mismos datos.

2.3. Controlador visual IBVS

El siguiente controlador que se ha estudiado en este trabajo, es el controlador visual IBVS, que sus siglas en inglés es Image-Based Visual Servoing.

Este controlador se basa en ir disminuyendo el error de posición de los objetivos en las imágenes que se van adquiriendo de la cámara traduciendo ese error en movimiento para las articulaciones de los robots.

Este controlador, como se podrá ver en los resultados, irá ajustando la velocidad de las articulaciones de los robots conforme se vaya alejando o acercando de los objetivos que tenga marcados en la imagen.

La ley de control que utiliza este controlador es la que se puede apreciar en la ecuación 3

$$v^c = -\lambda \cdot {J_f}^+ \cdot (s - s_d)$$

Ecuación 3. Ley de control visual basado en imagen

Esta ley de control a priori puede parecer compleja, pero cuando se desglosa no es difícil de entender ni de implementar.

El resultado que se obtiene de la ley de control, son las velocidades que se le pasarán a las articulaciones del brazo robótico. Por lo que en este caso no se necesitará integrar como en el caso del controlador PD que lo que se obtenían eran las aceleraciones.

El primer parámetro que se puede apreciar en la ecuación 3, es lambda (λ), que en este caso será la ganancia a ajustar para que el brazo robótico alcance la posición deseada de una manera suave y correcta, el estudio de la ganancia en este controlador se podrá ver en el apartado de resultados de este trabajo.

Cuando se habla en esta ecuación de $\boldsymbol{J_f}^+$, se trata de la pseudoinversa del jacobiano de la imagen o matriz de interacción. Esta matriz describe la relación de movimiento entre las características de la imagen y la cámara en el espacio 3D en términos de sus velocidades.

Para una característica de la imagen, el valor de la matriz será la matriz que se puede observar en la ecuación 4.

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = J_f \cdot \begin{bmatrix}\dot{x}_{t0}^C\\ \dot{y}_{t0}^C\\ \dot{z}_{t0}^C\\ \dot{\alpha}_{t0}^C\\ \dot{\beta}_{t0}^C\\ \dot{\gamma}_{t0}^C\end{bmatrix} \Rightarrow \begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = \begin{bmatrix}-\frac{1}{z_0^C} & 0 & \frac{x}{z_0^C} & x\cdot y & -(1+x^2) & y\\ 0 & -\frac{1}{z_0^C} & \frac{y}{z_0^C} & 1+y^2 & -x\cdot y & -x\end{bmatrix} \cdot \begin{bmatrix}\dot{x}_{t0}^C\\ \dot{y}_{t0}^C\\ \dot{z}_{t0}^C\\ \dot{\alpha}_{t0}^C\\ \dot{\beta}_{t0}^C\\ \dot{\gamma}_{t0}^C\end{bmatrix}$$

Ecuación 4. Jacobiano de la imagen

En la ecuación 4, la x y la y, son las coordenadas de una característica que se ha extraído de la imagen en metros, en el siguiente apartado se explicará cómo se han extraído las características de la imagen para utilizarlas en el controlador.

El valor de z_0^C, es la profundidad o distancia de la cámara al objeto del cual se extraerán las características. Y por último, el vector columna que está multiplicando a la matriz, representa la velocidad de movimiento de los tres ejes de la cámara, que son los tres primeros valores y las velocidades angulares que son los tres últimos.

En caso de que en la imagen se extraigan n características, se obtendrá la matriz de interacción correspondiente como se puede observar en la ecuación 5.

$$J_f = \begin{bmatrix}J_{f1}(p_1)\\ J_{f1}(p_1)\\ \vdots\\ J_{fn}(p_n)\end{bmatrix}$$

Ecuación 5. Matriz de interacción con n características

Una vez obtenida la matriz anterior, se hace la pseudoinversa de la misma, que correspondería con la ecuación 6, aunque cómo se verá en el apartado de implementación, gracias a Matlab, simplifica mucho el problema ya que se hará con una función que proporciona el propio software.

$$J^+ = J^T \cdot (J \cdot J^T)^{-1}$$

Ecuación 6. Cálculo de la pseudoinversa de la matriz

Una vez calculada la pseudoinversa, lo último que queda por aclarar de la ecuación 3, es que la $\mathbf{s}$ es el vector de características de la imagen y $\mathbf{s_d}$ es el vector de las características deseadas de la imagen.

Con todos los datos obtenidos y una vez calculada la ecuación 3, se obtienen las velocidades a aplicar al brazo robótico.

2.3.1. Implementación

En el caso de este controlador, la implementación no se ha realizado a partir de Simulink como en el caso del controlador PD con compensación gravitacional, ya que en este caso devuelve directamente las velocidades que se le pasarán a las articulaciones del brazo robótico Mitsubishi PA10 y no hay que realizar ningún cálculo muy complejo.

El algoritmo en pseudocódigo que se ha utilizado para simular este controlador es el siguiente:

```
Establecer la conexión con el simulador V-REP.

Extraer la información de la imagen de la posición del brazo robótico deseada

Colocar el robot en la posición inicial

while (tActual < tFinal)

    • Extraer la información de la imagen que está viendo la cámara.
    • Segmentar la imagen extraída de la cámara.
    • Ejecutar el controlador visual con la información segmentada y los parámetros intrínsecos de la cámara.
    • Enviar las velocidades que se han calculado a las articulaciones.

Fin del while
```

Como se puede observar en el pseudocódigo anterior, para este controlador, no se utilizará la toolbox de robótica de Peter Corke como en el controlador anterior, ya que no se utiliza el modelo del robot. En este caso, sólo se utiliza las imágenes de la cámara y las características de las imágenes que se extraen de ellas.

Para extraer la información de la imagen de la cámara en el simulador, lo primero que se tiene que hacer, es extraer el identificador numérico que almacena V-REP de la cámara, esto se realiza con el siguiente comando:

```
vrep.simxGetObjectHandle(clientID,'camera',vrep.simx_opmode_oneshot_wait)
```

En el comando anterior, por parámetros se le envía, el identificador del cliente, como a todos los comandos de V-REP, el nombre del objeto que tiene en el simulador, en este caso se le ha dado el nombre de 'camera'. Y por último otro comando que le indica al simulador que espere y no permita más comunicaciones con el servidor hasta que se devuelva un valor o un error en caso de que se le acabe el tiempo de espera que tenga establecido.

Cuando se ha obtenido el valor del identificador de la cámara, se deberá extraer las características de la imagen de la posición a la que se quiere que el robot llegue.

Para extraer una imagen de la cámara en V-REP, se utiliza también un comando desde Matlab que será el siguiente:

```
vrep.simxGetVisionSensorImage2(clientID, camera, 0, vrep.simx_opmode_oneshot_wait)
```

Con este comando se obtendrá un array de datos cuyos valores irán de 0 a 255, que será la imagen que se obtenga de la cámara de la escena. Por parámetros, el 0 que se le envía, indica que se almacene la imagen con el formato RGB y no con escala de grises.

Una vez se haya obtenido la imagen de la posición deseada, se tiene que segmentar la misma. Para este proceso se ha hecho una segmentación por forma, debido a que en la escena no existe ningún círculo y la imagen que se utilizará como patrón de búsqueda es la que se puede observar en la figura 4 que contiene cuatro círculos.

Para realizar dicha segmentación por forma se ha utilizado el siguiente código que se irá explicando paso a paso:

```
function uv_seg = miSegmentacion_forma(uv_mat)
    % Imagen a gris
    imGray = rgb2gray(uv_mat);
    % Binarizacion de la imagen
    imgd = imGray>=180;
    [cols,rows] = size(imgd);
    % Se invierten los colores a la imagen
    for i=1:cols
        for j=1:rows
            if imgd(i,j)==0
                imgd(i,j) = 1;
            else
                imgd(i,j)= 0;
            end
        end
    end
```

Lo primero, gracias a las funciones que proporciona ya Matlab, con una sola línea, se le indica que pase la imagen que se obtiene de la escena a escala de grises. Lo siguiente es binarizar la misma para poder luego utilizar la imagen en las funciones siguientes.

```
[regiones,num_regiones] = bwlabel(imgd);
objetos = regionprops(regiones,'all');
pos = 1;
uv_seg = 0;
for i=1:num_regiones
    aux = (4*objetos(i).Area*pi)/objetos(i).Perimeter^2;
    % Todo lo que no sea circulo se quita de la imagen
    if aux > 0.92 && objetos(i).Eccentricity < 0.2 && objetos(i).Eccentricity >= 0.01
          for j=1:cols
              for k=1:rows
                  if regiones(j,k)==i
                      imgd(j,k)=0;
                  end
              end
          end
        uv_seg(1,pos) = objetos(i).Centroid(1);
        uv_seg(2,pos) = objetos(i).Centroid(2);
        pos = pos + 1;
    end
end
```

En el trozo anterior de código, con la función bwlabel, Matlab ya segmenta todas las figuras que existen en la imagen, proporcionando en la variable num_regiones, la cantidad de objetos que existen en la imagen y en la variable regiones, se almacenan con diferentes identificadores los objetos segmentados.

Con la función regionprops, se obtienen muchos datos que calcula el software por si solo con la información de la imagen segmentada, como la excentricidad, el área, el perímetro o el centroide, que son los parámetros que se utilizarán para buscar los círculos en la imagen.

Para el caso de los círculos, se hacen dos comprobaciones diferentes, una de ellas es un cálculo que cuanto más cerca esté de 1, querrá decir que es más parecido a un círculo. Además se comprueba la excentricidad, ya que este valor si se acerca a 0 quiere decir que es más círculo y si se acerca a 1 es más óvalo.

Una vez se comprueba que ese objeto segmentado de la imagen es un círculo, se almacena como características de la imagen el centroide del círculo.

Esa característica en x y en y, serán utilizadas después para realizar la diferencia de las características que vaya consiguiendo, en tiempo de simulación, la cámara y aplicar ese valor al controlador.

Una vez se han obtenido las características de la imagen final, se almacenan estás características y se tiene que poner el robot en la posición inicia, y como es evidente, la cámara del robot tiene que poder ver la imagen del patrón para poder hacer los cálculos del controlador.

Cuando empieza el algoritmo, lo primero que se hace es extraer las características de la imagen que está observando la cámara de la misma manera que se acaba de explicar.

Lo siguiente es extraer las características de profundidad en metros desde la cámara a la imagen.
En el caso de V-REP, no hay ningún comando que obtenga esa información en metros, pero sí que se puede extraer la posición de la cámara y dónde está el patrón colocado, por lo que se puede utilizar esa diferencia de posiciones como solución en el simulador para este problema. Esto se realiza con el siguiente código.

```
function features_seg = loadDepthCameraNew(uv_seg, clientID, camera, vrep)
    [returnCode,positionCamera]=vrep.simxGetObjectPosition(clientID,camera,-1, vrep.simx_opmode_streaming);
    [returnCode,positionCamera]=vrep.simxGetObjectPosition(clientID,camera,-1, vrep.simx_opmode_buffer);
    [returnCode, picture] = vrep.simxGetObjectHandle(0,'Plane',vrep.simx_opmode_oneshot_wait);
    [returnCode,positionPict]=vrep.simxGetObjectPosition(clientID,picture,-1, vrep.simx_opmode_streaming);
    [returnCode,positionPict]=vrep.simxGetObjectPosition(clientID,picture,-1, vrep.simx_opmode_buffer);

    res = positionPict(1)-positionCamera(1);
```

En el código anterior, se puede observar como para conseguir las posiciones de la cámara y del patrón, se tiene que realizar la suscripción antes de recoger los datos al igual que se tenía que hacer en el control PD para la obtención de la posición de las articulaciones.

Con los datos anteriores se puede calcular la ley de control que se hará con el siguiente código.

```
function cvc = ContrLaw(uv,uv_final,z,Fu,Fv,u0,v0,lambda)

    n = size(uv,2);

    s = pixel2coord(uv,Fu,Fv,u0,v0);
    s_final = pixel2coord(uv_final,Fu,Fv,u0,v0);

    J = zeros(2*n,6);

    for i=1:n
        x = s(i*2-1);
        y = s(i*2);
        J(i*2-1,:) = [-1/z(i) 0     x/z(i)   x*y -(1+x^2) y];
        J(i*2,:)   = [    0 -1/z(i) y/z(i) 1+y^2  -x*y  -x];
    end

    Ji = pinv(J);


    cvc = -lambda * Ji * (s - s_final);
```

En el código anterior, se puede ver que por parámetro se le pasan varias cosas. uv y uv_final, serán las características que se han extraído de la imagen actual y la deseada respectivamente.

El valor de z, será el valor de profundidad que se ha calculado con la función anterior. Fu, Fv serán el ratio entre la focal y el tamaño del pixel y u0 y v0, el centro óptico de la cámara. Estos parámetros, son los parámetros intrínsecos de la cámara.

Y por último, se le pasará λ, que será la ganancia del controlador, que ya se verá en el apartado de resultados de este trabajo cómo afecta el cambio de ganancias al controlador.

En el controlador, lo primero que se hará es pasar los píxeles a metros, ya que las características vienen dadas en pixeles que serían los centros de los círculos en este caso, pero el controlador trabaja con metros. Este paso se hace de la siguiente forma:

```
function s = pixel2coord(uv,Fu,Fv,u0,v0)

n = size(uv,2);

u = uv(1,:);
v = uv(2,:);

x = (u - u0) / Fu;
y = (v - v0) / Fv;
s = reshape([x; y], 2*n,1);
```

Para extraer los metros de la característica, como se puede observar en el código anterior, se resta cada coordenada del pixel con el centro óptico de la cámara y se divide entre el ratio entre la focal y el tamaño del pixel. De esta manera se obtendrá el valor del pixel de la imagen en metros.

Volviendo al controlador, una vez que se han obtenido las características en metros, se realiza la matriz jacobiana, que no es más que realizar la matriz que se ha visto en la teoría de este controlador pero con los valores reales que se han obtenido de la imagen.

Y para calcular la pseudoinversa, gracias a Matlab, con pasarle la jacobiana a la función pinv, devuelve el resultado sin tener que realizar los cálculos.

Una vez hechos los pasos anteriores, se aplica la ley de control vista en este apartado y se obtendrán los valores de las velocidades de las articulaciones que se pasarán al robot de la misma manera que se enviaban las velocidades en el apartado del controlador PD.

3. Control de un robot omnidireccional

Además del estudio de los controladores anteriores para brazos articulares, también se ha realizado el estudio del controlador visual para un vehículo robótico omnidireccional.

A lo largo de este apartado se explicará que tipo de robot es y cómo se ha adaptado el controlador visual visto en el capítulo anterior para que funcione sobre este tipo de robots.

3.1. Arquitectura del sistema

Al igual que en las simulaciones anteriores, para este caso se utilizarán los mismos programas, tanto Matlab para los cálculos necesarios, como V-REP para realizar la simulación.

El robot utilizado para el estudio de este caso, es un robot que V-REP tiene almacenado en su biblioteca de robots. Este vehículo robótico omnidireccional es el KUKA YouBot.

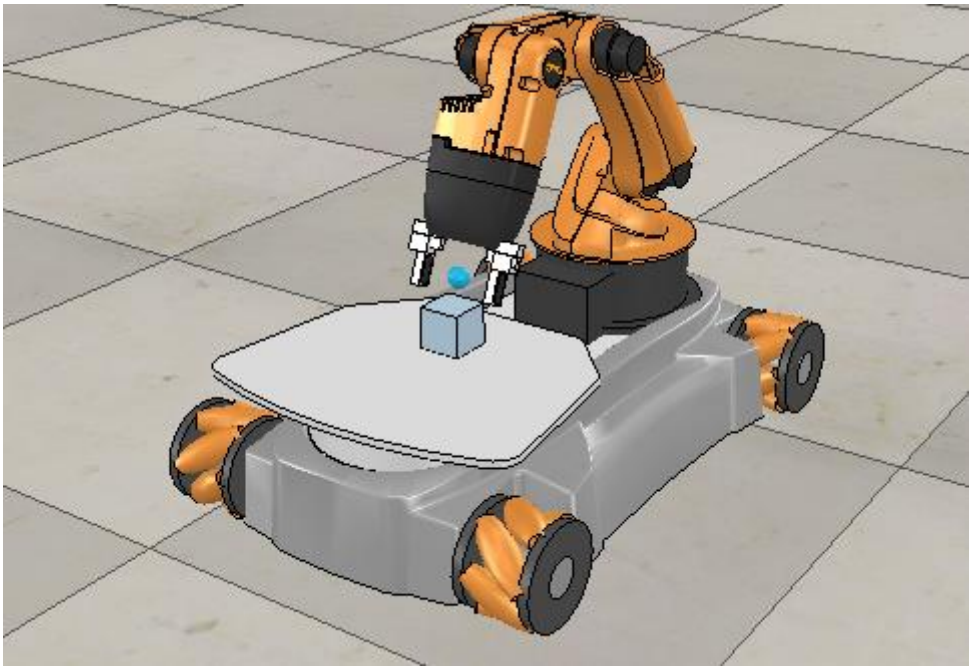


Figura 9. Robot KUKA YouBot

Este robot, como se puede observar en la figura 9, está compuesto de una base rectangular, 4 ruedas que harán que el robot se pueda mover en cualquier dirección sin realizar maniobras y un brazo robótico incorporado.

Para adecuarlo al estudio que se va a realizar en este trabajo, se le han hecho algunos cambios que se pueden observar en la figura 10.

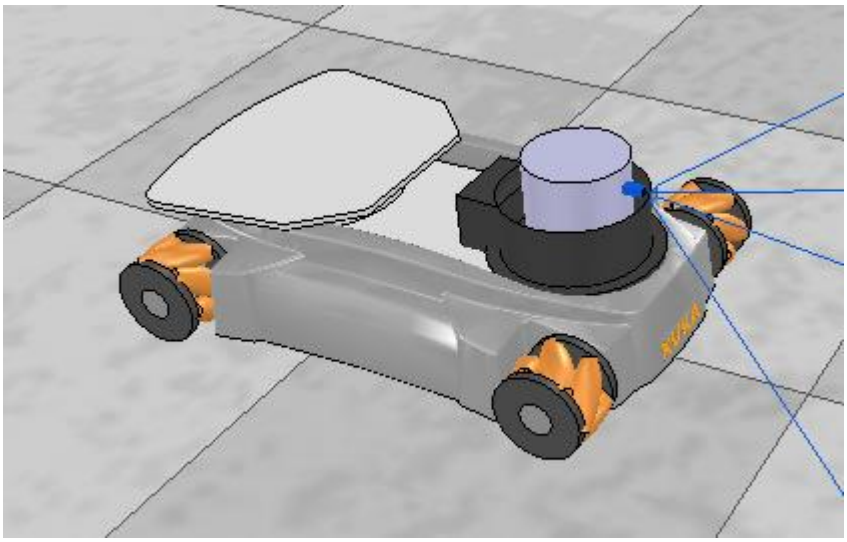


Figura 10. Robot KUKA YouRobot modificado

Se le ha eliminado al robot el brazo robótico, ya que para este estudio no se va a utilizar y lo único que hacía era aportarle peso al robot. Y se le ha añadido un cilindro que simula una especie de cabeza con una cámara por la que podrá ver el vehículo e ir calculando las velocidades como se verá más adelante.

Este robot se ha seleccionado principalmente porque la principal característica del mismo son las ruedas que tiene. Estas ruedas, son las ruedas llamadas “mecanum” que se pueden visualizar en la figura 11.

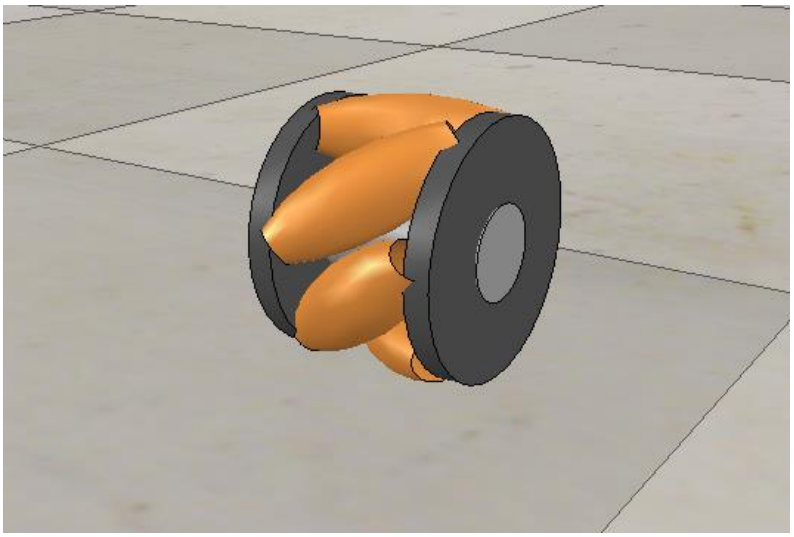


Figura 11. Rueda mecanum

Este tipo de ruedas, se caracteriza por permitir al vehículo robótico, desplazarse en cualquier dirección sin necesidad de tener que maniobrar para ello, permitiendo a este ser mucho más rápido y haciéndole que se pueda desplazar por zonas que con otro tipo de ruedas no podría hacerlo por tener giros muy cerrados.

La manera que tienen de mover las ruedas para conseguir el desplazamiento del robot son las que se pueden apreciar en la figura 12.

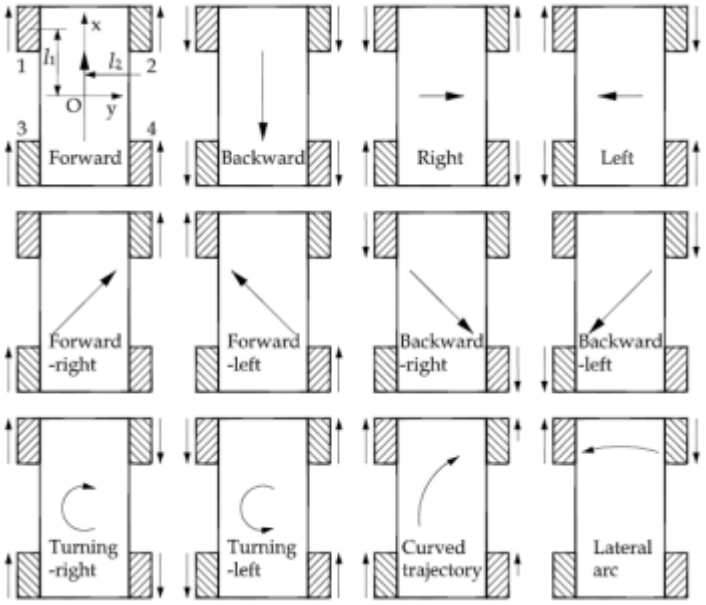


Figura 12. Movimiento del robot acorde a la dirección y velocidad angular de las ruedas (extraído de 19)

El entorno que se utilizará para el estudio de este controlador, será el mismo que para el robot multiarticular, con la diferencia de que al vehículo robótico se le podrá poner a más distancia o en zonas a los laterales del patrón gracias a su libertad de movimiento.

3.2. Controlador visual IBVS para robot omnidireccional

Para este robot, se utilizará una variante del controlador que se ha comentado para el robot Mitsubishi PA10. La base del controlador será la misma y se regirá por la misma ley de control expuesta en la ecuación 3.

La diferencia se basa en que la jacobiana de la imagen calculada para el controlador, no será la misma. Como se comentó en el apartado anterior, la matriz se multiplica por un vector columna que representa la velocidad de los tres ejes de la cámara y los ángulos de giro.

En este caso, el vehículo robótico solo se desplazará en el plano xy y solo rotará en el eje z. De esta manera, para una característica de la imagen, el valor de la matriz quedaría como se puede observar en la ecuación 7.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -\frac{1}{z_0^C} & 0 & y \\ 0 & -\frac{1}{z_0^C} & -x \end{bmatrix} \cdot \begin{bmatrix} \dot{x}_{t0}^C \\ \dot{y}_{t0}^C \\ \dot{\gamma}_{t0}^C \end{bmatrix}$$

Ecuación 7. Jacobiano de la imagen para vehículo robótico

De esta manera, se puede apreciar que el cambio es simplemente quitar columnas a la jacobiana y multiplicarla por las velocidades en las que se desplaza el robot y la velocidad de giro que realiza el mismo.

Estos robots omnidireccionales, se mueven de una manera distinta a la que se movería un vehículo robótico con unas ruedas normales, ya que como se puede ver en la figura 11, dependen de cómo se mueven sus ruedas y a que velocidades para moverse hacia una dirección u otra.

El controlador, como se ha visto y se verá más en detalle en la implementación, devuelve 3 valores en un vector, los dos primeros valores serán la velocidad en el eje x y en el eje y respectivamente y el tercer valor será la velocidad angular del robot, para entenderse mejor se puede observar la figura 13.

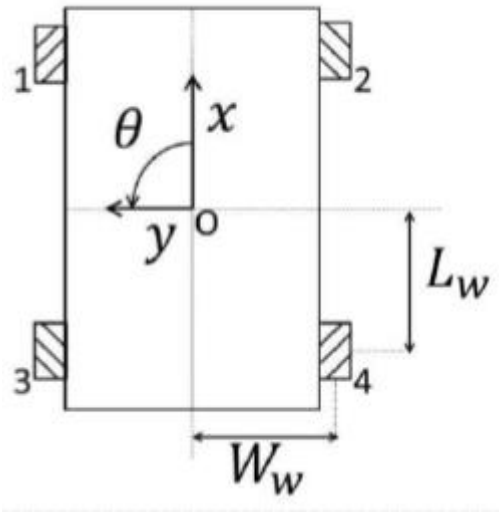


Figura 13. Esquema ejes del vehículo robótico

Como ya se ha visto, dependiendo de las velocidades que se le apliquen a cada rueda individualmente, el robot se moverá un una dirección distinta sin necesidad de girar las ruedas, ya que estas son fijas y siempre miran a la misma dirección.

Si se considera un instante de tiempo para el robot, se puede decir que la velocidad de su cuerpo se puede calcular con la ecuación 8.

$$\begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} = \frac{R}{4} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ -\frac{1}{W_w + L_w} & \frac{1}{W_w + L_w} & -\frac{1}{W_w + L_w} & \frac{1}{W_w + L_w} \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$

Ecuación 8. Ecuación de la velocidad del cuerpo del robot

Donde la R es el radio de la rueda. El parámetro $\mathbf{w_i}$ es la velocidad angular de cada rueda del vehículo, y W_w y L_w son las distancias desde el centro del cuerpo del robot hasta los ejes de las ruedas como se puede apreciar en la figura 13.

Como lo que en este caso se sabe es la velocidad en x y en y del robot y la velocidad de giro que tiene por el controlador utilizado, lo que se quiere obtener es la velocidad que se le tiene que aplicar a cada rueda. Por lo tanto, lo que se tiene que hacer es despejar estos valores en la ecuación 8. La ecuación quedaría como la que se puede apreciar en la figura 9 en la que ya se obtienen las velocidades de cada rueda.

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \frac{1}{R} \cdot \begin{bmatrix} 1 & 1 & -(W_w + L_w) \\ 1 & -1 & W_w + L_w \\ 1 & -1 & -(W_w + L_w) \\ 1 & 1 & W_w + L_w \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix}$$

Ecuación 9. Ecuación de la velocidad de cada rueda del robot

3.2.1. Implementación

La implementación en este caso es muy parecida al aplicarle el controlador visual a un brazo multiarticular. El algoritmo en pseudocódigo es el que se puede ver a continuación:

```
Establecer la conexión con el simulador V-REP.

Extraer la información de la imagen de la posición del vehículo
robótico deseada

Colocar el robot en la posición inicial

Almacenar los identificadores de los motores de las ruedas

while (tActual < tFinal)
```

- Extraer la información de la imagen que está viendo la cámara.
- Segmentar la imagen extraída de la cámara.
- Ejecutar el controlador visual con la información segmentada y los parámetros intrínsecos de la cámara.
- Transformar las velocidades obtenidas en el controlador a las velocidades individuales de cada rueda

- Enviar las velocidades que se han calculado a las articulaciones.

Fin del while

Como se puede observar, la única variación que habrá en la implementación para el vehículo robótico con respecto al brazo multiarticular, será almacenar los identificadores específicos de las articulaciones que harán de motores para las ruedas, que el controlador se calculará con una jacobiana diferente y que las velocidades obtenidas en el controlador, se tendrán que transformar.

En este caso, el vehículo robótico como se puede observar en la figura 14, tiene varias articulaciones en el simulador.

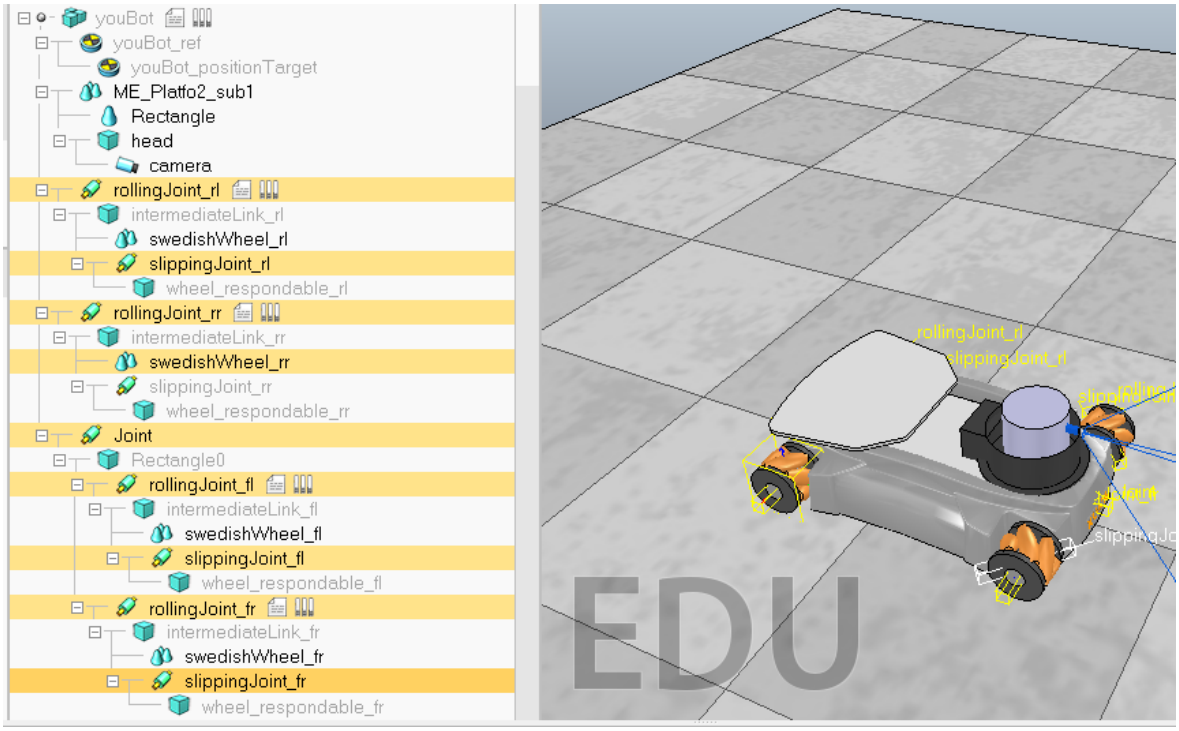


Figura 14. Articulaciones marcadas que vienen con el vehículo robótico

Pero en el caso de este estudio, se necesitan sólo las articulaciones que simularían el motor de la rueda, esas son las que se pueden apreciar en la figura 15 marcado en blanco.

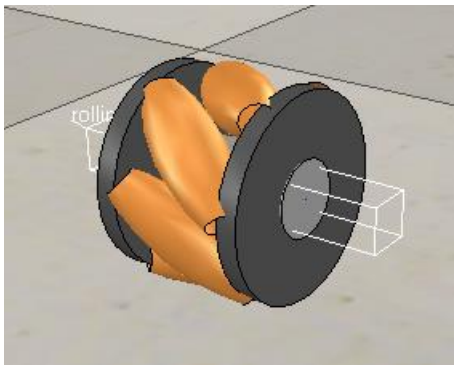


Figura 15. Rueda con la articulación que se utilizará como motor de la misma

Para recoger sólo esas articulaciones y no todas las que hay como era el caso del robot multiarticular, se tienen que seleccionar con el siguiente código:

```
[returnCode, fr] = vrep.simxGetObjectHandle(0,'rollingJoint_fr',vrep.simx_opmode_oneshot_wait);
[returnCode, rr] = vrep.simxGetObjectHandle(0,'rollingJoint_rr',vrep.simx_opmode_oneshot_wait);
[returnCode, rl] = vrep.simxGetObjectHandle(0,'rollingJoint_rl',vrep.simx_opmode_oneshot_wait);
[returnCode, fl] = vrep.simxGetObjectHandle(0,'rollingJoint_fl',vrep.simx_opmode_oneshot_wait);
```

De esta manera, se están almacenando los identificadores de esas articulaciones para cuando se calculen las velocidades se les envíen sólo a estos identificadores.

En el caso de la jacobiana, que también tiene diferencias con respecto al robot anterior. El código para obtenerla ha sido el siguiente:

```
J = zeros(2*n,3);

for i=1:n
    x = s(i*2-1);
    y = s(i*2);

    J(i*2-1,:) = [-1/z(i) 0 y];
    J(i*2,:)   = [   0 -1/z(i) -x];
end
```

Cómo se puede observar, se han quitado las columnas de las velocidades que serían 0, ya que el vehículo no se movería en el eje z y solo giraría sobre ese eje como se ha explicado anteriormente.
Por ello la matriz será de 2*n filas y de 3 columnas. Siendo n la cantidad de características que se hayan extraído de la imagen.

Y lo último diferente que tiene este tipo de robots, es el cálculo de las velocidades.

```
function cvc = Transform_velocities(cvc)

Rw = 0.100;

lw = 0.2355;
ww = 0.15023;

aux = cvc;

vx = cvc(1,1);
vy = cvc(2,1);
theta = cvc(3,1);

matrix = [1 1 -(lw+ww);
          1 -1 lw+ww;
          1 -1 -(lw+ww);
          1 1 lw+ww];

cvc = (1/Rw)*matrix*[vx;vy;theta];
```

Como se puede observar en el código anterior, sólo se le pasa un parámetro a esta función. Este parámetro es un vector de tres componentes que son la velocidad en el eje x, la velocidad en el eje y y la velocidad angular, que habrán sido calculadas por la ley de control de la ecuación 3.

Los parámetros Rw, lw y ww, como se han comentado anteriormente son, el radio de las ruedas, la longitud desde el centro hasta el eje de la rueda y la anchura desde el centro al eje de la rueda respectivamente. En este caso se han utilizado las especificaciones que proporcionaba la página de KUKA sobre este robot.

Una vez almacenados los datos, se hace la matriz 3x4 que se ha comentado en la ecuación 9 y se obtendrá un vector columna que serán las 4 velocidades que se les tendrán que enviar a las ruedas del vehículo robótico, correspondiendo el primer valor a la rueda delantera izquierda, el segundo a la delantera derecha, el tercero a la rueda trasera izquierda y el último a la trasera derecha.

4. Resultados

A lo largo de este apartado, se podrán a estudio ambos controladores en los diferentes tipos de robots explicados en los apartados anteriores.

De esta manera, se podrá observar cómo funcionan las ganancias en cada caso y explicar que sucede con los robots cuando se cambian esos parámetros, indicando también que clase de controlador podría funcionar mejor en qué tipo de robot.

Para estudiar el comportamiento de estos robots con los diferentes controladores explicados anteriormente, se han utilizado dos gráficas, la primera de ellas, que se ha realizado la distancia entre las características de la imagen extraídas por la cámara en cada instante de tiempo y las características de la imagen en la posición final con respecto al tiempo.

Y en la otra gráfica, se podrá observar la evolución de la velocidad de las ruedas en cada instante de tiempo.

Para realizar la primera de las gráficas, se ha utilizado el siguiente código:

```
tam = size(uv_complete);
tam = tam(1)/2;

for i=1:tam
    u = uv_complete(i*2-1,:);
    v = uv_complete(i*2,:);
    complete = [u;v];

    res(i,1) = sqrt((complete(1,1)-uv_final(1,1))^2+(complete(2,1)-uv_final(2,1)));
    res(i,2) = sqrt((complete(1,2)-uv_final(1,2))^2+(complete(2,2)-uv_final(2,2)));
    res(i,3) = sqrt((complete(1,3)-uv_final(1,3))^2+(complete(2,3)-uv_final(2,3)));
    res(i,4) = sqrt((complete(1,4)-uv_final(1,4))^2+(complete(2,4)-uv_final(2,4)));

end

plot(timeTotalVector,res);
%plot(timeTotalVector,cvcTotalAfterTrans);
```

En el código anterior, se puede observar que es un script de Matlab muy simple. Se utilizan los parámetros que ha ido capturando el robot durante su desplazamiento. uv_complete es el historial de las características que se han ido segmentando con la cámara durante el recorrido que ha hecho el robot, uv_final son las características de la segmentación de la imagen final del desplazamiento. Y por último, t es todos los instantes de tiempo de la simulación.

Para calcular las distancias de las características se ha utilizado la distancia euclídea, que corresponde a la ecuación 10.

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Ecuación 10. Distancia euclídea

4.1. Robot multiarticular

Como se ha explicado en apartados anteriores, al robot Mitsubishi PA10, se le han aplicado dos tipos diferentes de controladores, uno de posición y otro visual.

En este apartado, se comprobará cómo funcionan los controladores y las ganancias en este tipo de robots y se podrá comparar que controlador es mejor para los robots multiarticulares y en qué situaciones.

4.1.1. Controlador PD con compensación gravitacional

Para probar cómo funciona este controlador en un robot multiarticular, se ha realizado una traslación entre dos posiciones del Mitsubishi PA10. El robot saldrá desde la posición home que es la posición inicial del robot, que se puede observar en la figura 1, hasta llegar a la llamada posición de escape del robot que se puede ver en la figura 16.

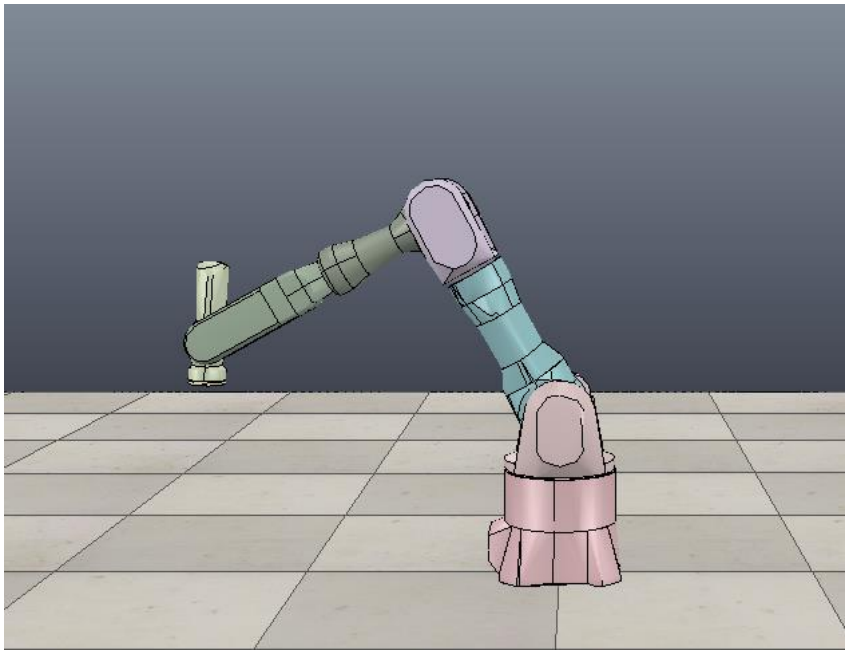


Figura 16. Posición de escape del Mitsubishi PA10

La posición de escape, tendrá los siguientes ángulos en sus articulaciones:

$$s_1 = 0, s_2 = 30, e_1 = 90, e_2 = 0, w_1 = 60, w_2 = 0$$

Para entender cómo funcionan las ganancias en este controlador, se han hecho diferentes simulaciones para observar los resultados y poder estudiarlos a lo largo de este apartado.
El tiempo de cada simulación en las diferentes pruebas, ha sido de tres segundos, ya que es un tiempo razonable para que el robot alcance la posición deseada.

Para empezar con el estudio de las ganancias, se le aplicó una ganancia en la que tanto la ganancia Kp, ganancia de las posiciones, como la ganancia Kv, ganancia de las velocidades, eran de un valor bajo e iguales, con un valor de 10.

En la figura 17, se puede observar cómo el robot no alcanza la posición en el tiempo estimado, las articulaciones se acercan muy lentamente y con una oscilación muy débil, por ello impide que el robot alcance la posición de escape rápidamente. Aunque si se dejase más tiempo la simulación, el robot conseguiría poner la posición deseada. Pero en este estudio se busca que el controlador actúe sobre las articulaciones de manera que haga que lleguen a las posiciones marcadas más rápido.

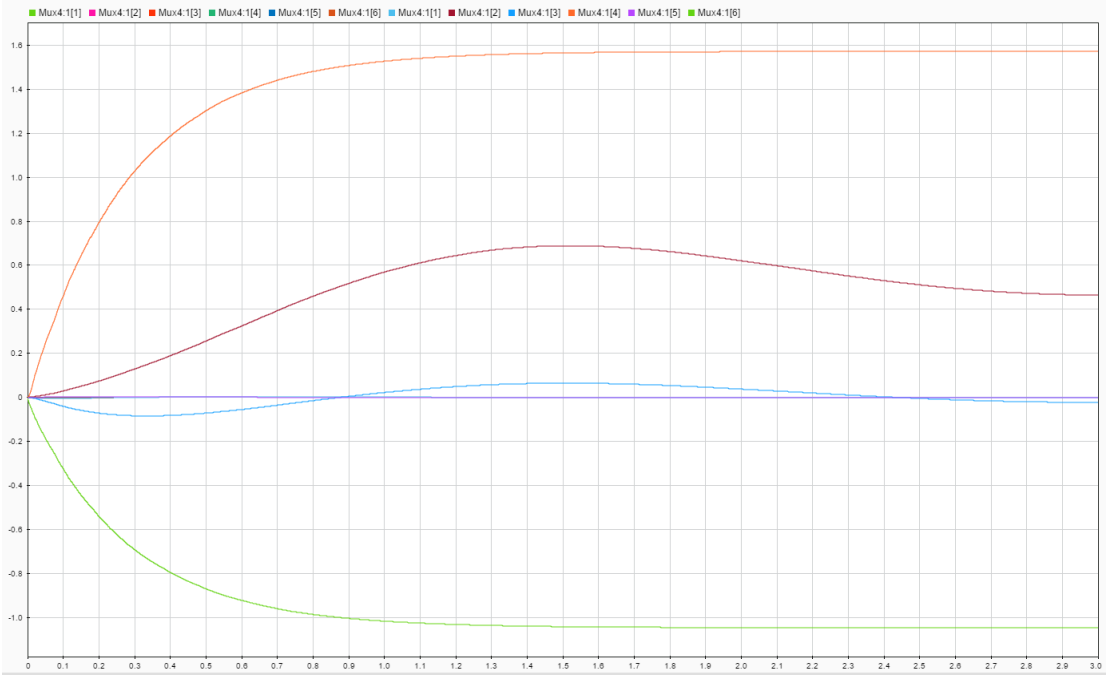


Figura 17. Evolución de las articulaciones del Mitsubishi PA10 con Kp=10 y Kv=10

La siguiente prueba del estudio, fue proporcionarle al robot una ganancia de Kp = 500 y Kv = 200. Con estas ganancias se puede observar en la figura 18, que el robot consigue alcanzar la posición en el tiempo estipulado de la simulación, pero con un riesgo para sí mismo y para el entorno que le rodea elevado, ya que algunas de las articulaciones sobrepasan mucho las posiciones deseadas y además tienen mucha oscilación, sobre todo

la articulación 4 que en pocos milisegundos hace que esta articulación se ponga a más de 110 grados cuando se buscan 90.

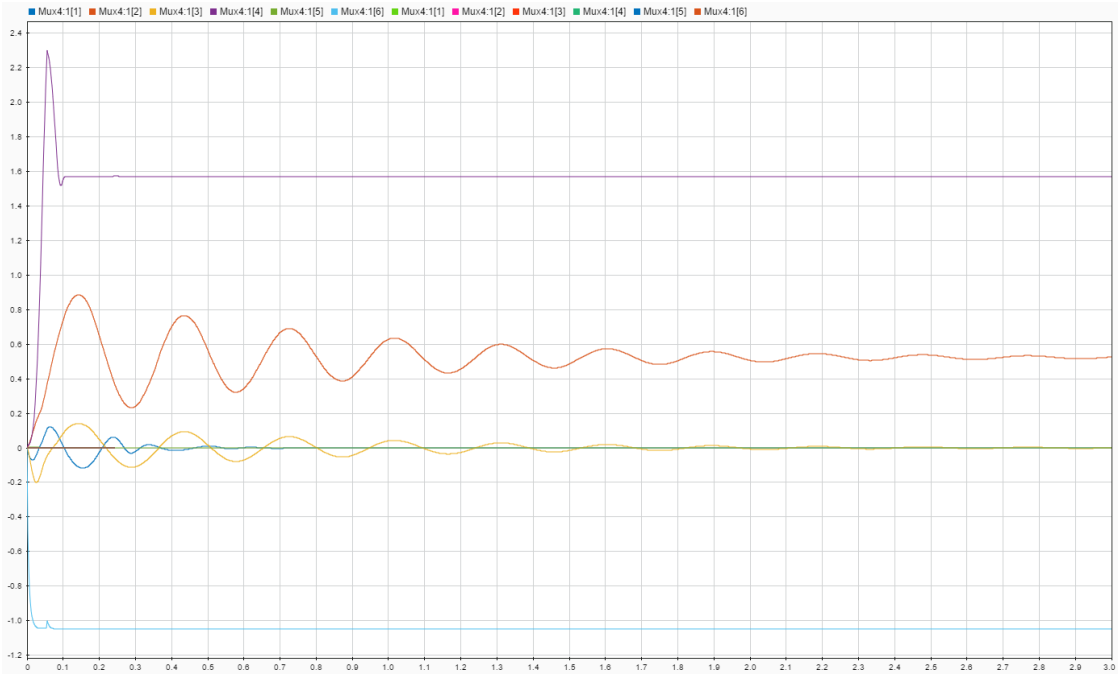


Figura 18. Evolución de las articulaciones del Mitsubishi PA10 con Kp=500 y Kv=20

Por último se le aplicó a la ganancia de las posiciones, Kp, un valor de 200 y a la ganancia de las velocidades, Kv, un valor de 20. Con estas ganancias, como se puede observar en la figura 19, los resultados son bastante buenos.

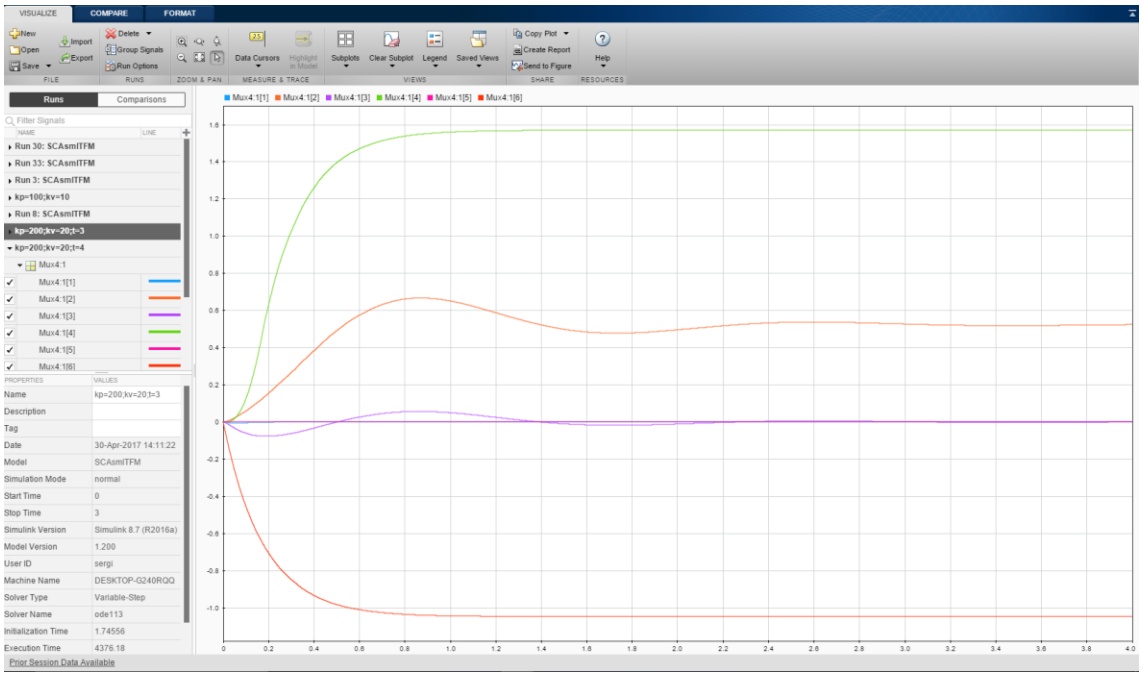


Figura 19. Evolución de las articulaciones del Mitsubishi PA10 con Kp=200 y Kv=20

Se puede observar que a partir del primer segundo y medio, todas las articulaciones prácticamente han llegado a las posiciones deseadas.
La que más oscila es la articulación 2. Esto es debido a que es la articulación que más peso mantiene y más esfuerzo necesita para llegar a la posición al tener que soportar la tensión que ejercen las demás articulaciones sobre ella.

Con este estudio del control proporcional derivativo con compensación gravitacional, se puede demostrar que al aplicarle una ganancia mayor a las posiciones, el robot intenta llegar a la posición más rápido, pero con una oscilación mayor. Y al aplicarle una mayor ganancia de las velocidades, se observa que la velocidad a la que el robot alcanza la posición es más lenta pero no oscila tanto.

Como se ha demostrado en este estudio, el objetivo es encontrar el punto de equilibrio entre ambas ganancias para que el robot se adecúe a cada situación.

4.1.2. Controlador Visual IBVS

Como se ha explicado en los apartados anteriores, además de probar el control posicional visto en el apartado anterior, se ha probado el controlador visual para el robot Mitsubishi PA10.

En este apartado se estudiarán dos casos en los que se podrá ver las ventajas y desventajas que ofrece este controlador frente al anterior.

Para empezar el estudio, se ha colocado el robot como se puede observar a la derecha de la figura 20. Esta posición será la que tendrá que alcanzar el robot. El robot tiene todas las articulaciones en el estado home del robot, es decir, todas con 0 grados de giro menos la última, que tiene 90 grados para que el robot pueda ver el patrón colocado en el muro.

La posición inicial se puede observar en la parte izquierda de la figura 20.
En este caso se le ha dado a la primera articulación rotacional 15 grados para que se le quede el patrón a la derecha y tenga que girar para colocarse en la posición correcta.

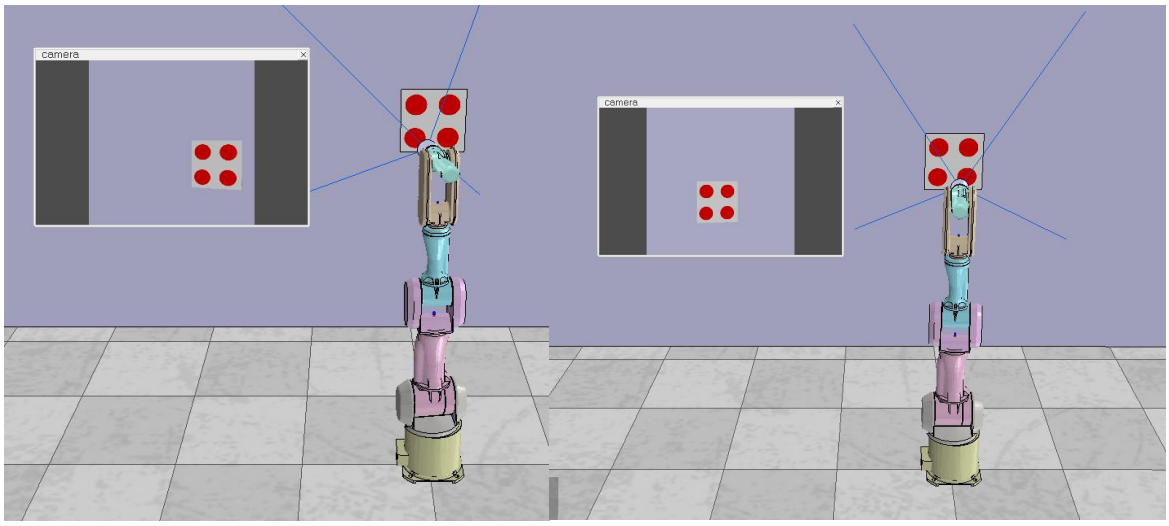


Figura 20.Posición inicial y final del PA10

Para comprobar el error de las características del patrón con respecto al tiempo, se han utilizado las gráficas que se han explicado al principio de este capítulo.

En la figura 21 se puede observar como el error de las características de la imagen propuesta como patrón, disminuye muy rápido en los primeros segundos, de manera que antes de los primeros 5 segundos, el error ya está muy bajo.

El problema de este controlador visual, es que para que el robot consiga estar exactamente en la posición deseada, tarda bastante, ya que la manera de funcionar de este controlador es ajustar con una velocidad muy elevada cuando las distancias de las características son lejanas con respecto a las deseadas y mucho más lentas conforme se aproxime a la posición final.

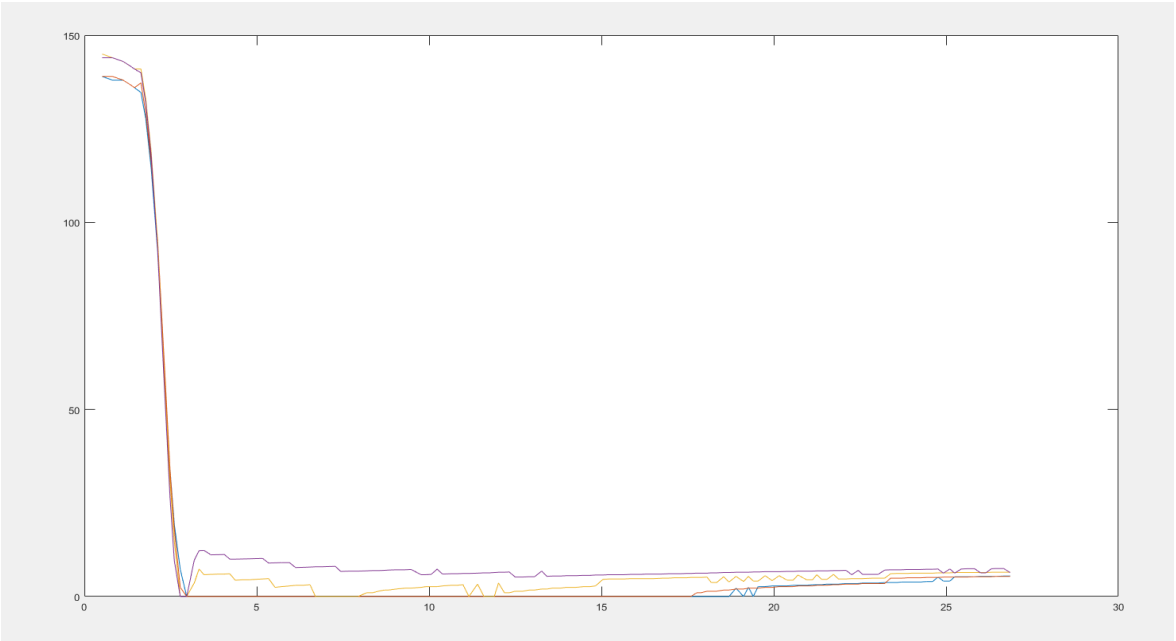


Figura 21. Error de las distancias entre las características de la imagen de la cámara. Lambda = 0.1

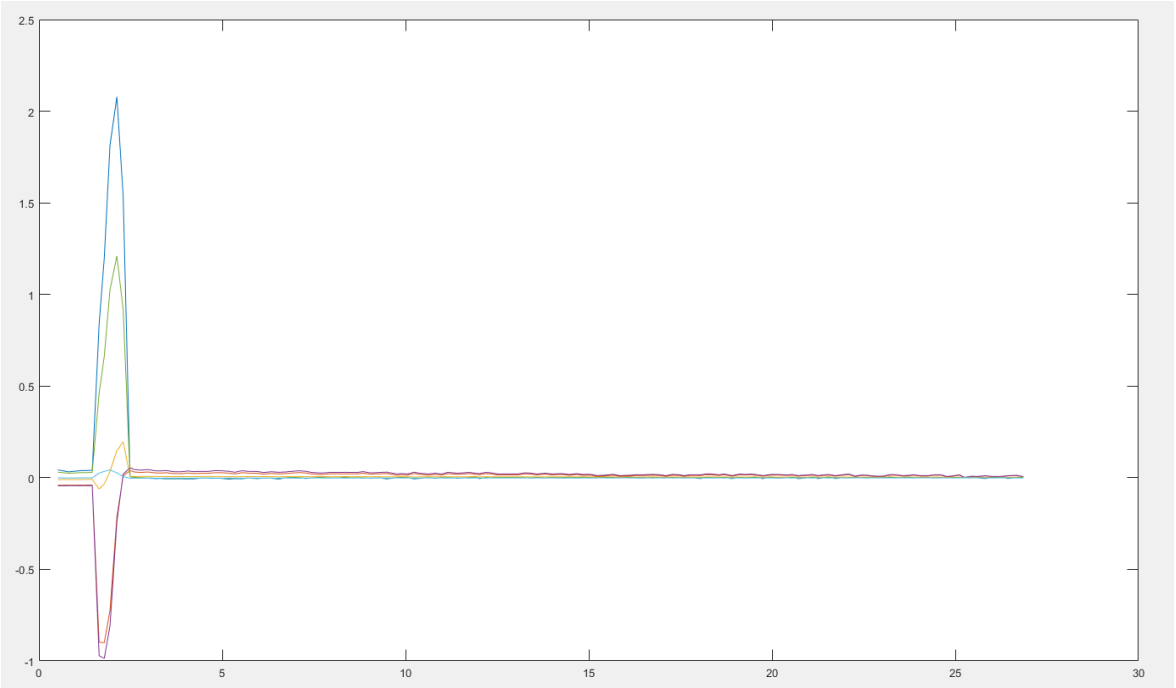


Figura 22. Velocidades de las 6 articulaciones del robot. Lambda = 0.1

Con la figura 22, se puede comprobar que las velocidades funcionan de esa manera. Al principio hay picos muy elevados de velocidades pero conforme se acerca a la posición final, las velocidades que se le proporcionan a las articulaciones son muy cercanas a 0.

Una vez comprobado el funcionamiento del controlador con el desplazamiento lateral del robot, se hizo la prueba de un desplazamiento más completo como se puede observar en la figura 23.

A la izquierda de esta figura, se puede ver la posición inicial del robot en la que se le ha dado a la articulación 4 una inclinación de 15 grados y a la articulación 6, que es la articulación que controla le herramienta en la que se ha colocado la cámara para poder controlar la escena, 90 grados.

Y se quiere que el robot alcance, la posición que se puede observar a la derecha de la figura 23. En la que el robot tiene la misma posición que la posición inicial de la prueba del desplazamiento anterior.
Es decir, que el robot debe de retirar la inclinación que se le ha proporcionado y girar hacia la izquierda para dejar el patrón en la posición que se puede observar en esa figura.

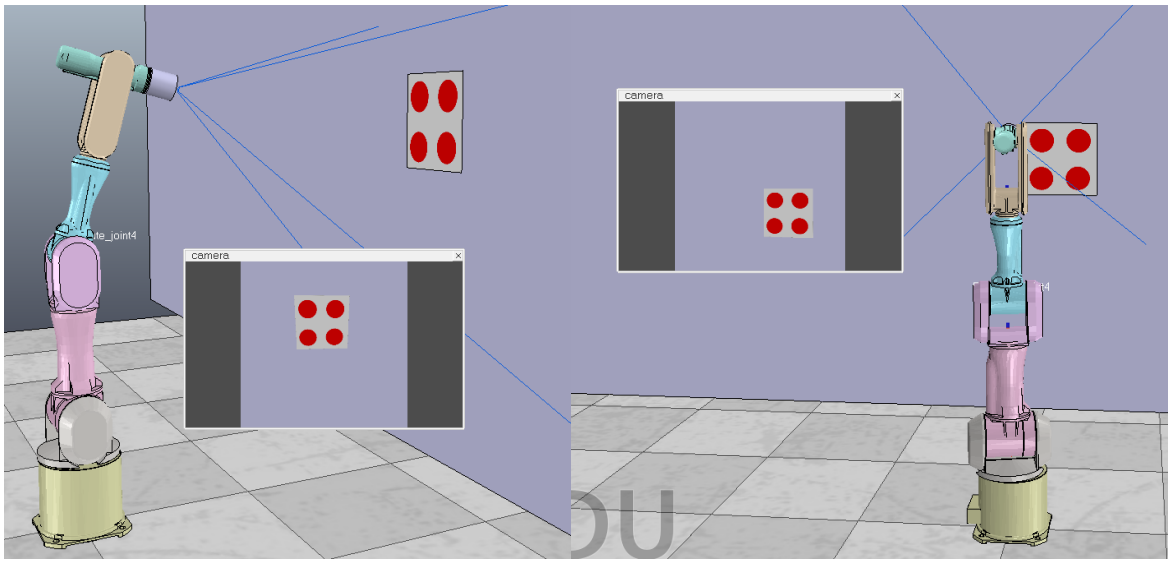


Figura 23. Posición inicial y final del robot

En la figura 24, se puede observar cómo el error disminuye muy rápido al principio como en el caso anterior y luego va más despacio hasta estabilizarse en la posición final.

Se ha comprobado en la simulación, que el robot realiza el movimiento de inclinación primero y rota muy poco y cuando casi tiene completo ese movimiento, realiza el de rotación para aproximar las características al patrón final.

Esto se puede comprobar en la figura 24 y 25 ya que en el segundo 22 aproximadamente de la simulación, el robot vuelve a tener un pico de velocidades para terminar de ajustar su posición con la rotación de la que se está hablando.

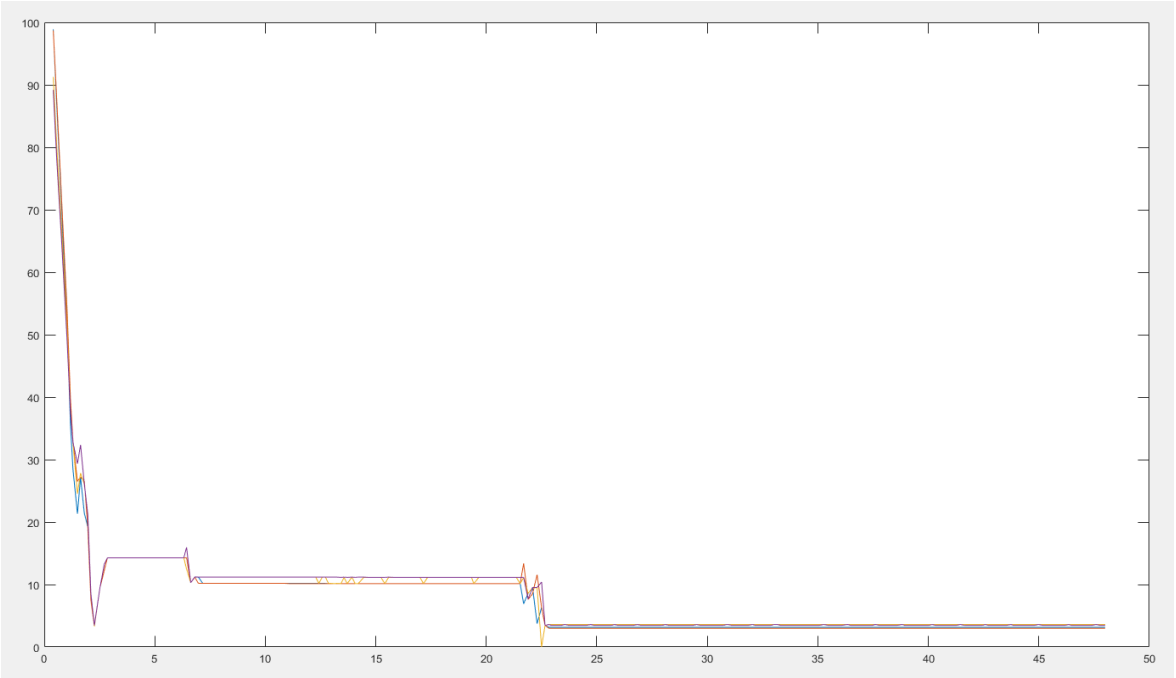


Figura 24. Error de las distancias entre las características de la imagen de la cámara. Lambda = 0.1

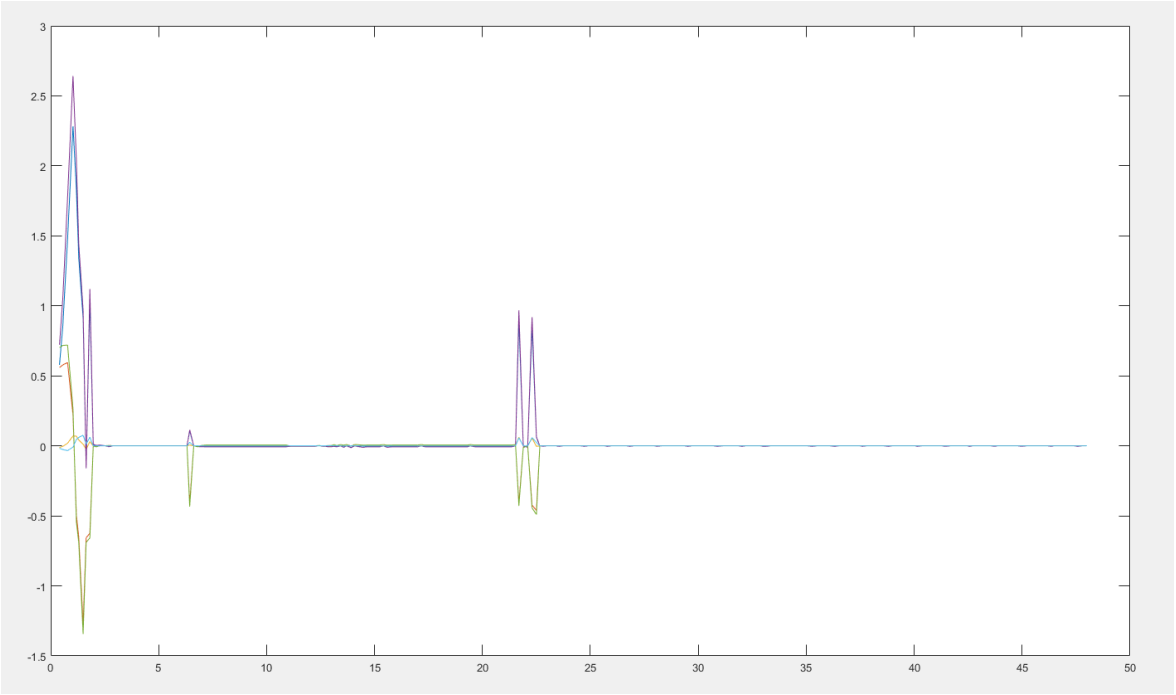


Figura 25. Velocidades de las 6 articulaciones del robot. Lambda = 0.1

Este controlador, funciona bastante bien para este tipo de robots.

En comparación con el controlador de posición propuesto anteriormente, el robot es capaz de llegar a una posición determinada, pero en el caso del controlador PD con compensación gravitacional, siempre se tienen en cuenta la posición de cada articulación, y al final de la trayectoria, el robot siempre va a estar en la posición articular que el usuario decida.

En cambio, en este controlador, los grados de rotación de las articulaciones del robot, no tienen que ser siempre las mismas, ya que se busca la aproximación al patrón sin importar de qué manera lo haga, siempre que sea rápido y eficiente.

4.2. Robot omnidireccional

Para el estudio de este robot, se han realizado tres desplazamientos diferentes. Para poder realizar un estudio de cómo se mueve este tipo de robots.

Los desplazamientos son, el lateral en el que se podrá ver que no es necesario que haga ningún tipo de maniobra para moverse hacia algún lado, un movimiento frontal en el que se desplazará solamente hacia adelante y por último un desplazamiento mucho más completo que incluirá los dos anteriores.

4.1.1. Desplazamiento lateral

La primera prueba que se hará para el estudio de este tipo de robot, es un desplazamiento lateral, ya que es lo que hace especial a los robots omnidireccionales.

Se comprobará que no requiere de maniobras complicadas para desplazarse lateralmente.

La posición final del robot será la que se puede observar en la parte derecah de la figura 26 y la posición inicial de la simulación se puede comprobar en la parte izquierda de la figura 26 en la que está desplazado el patrón hacia la derecha.

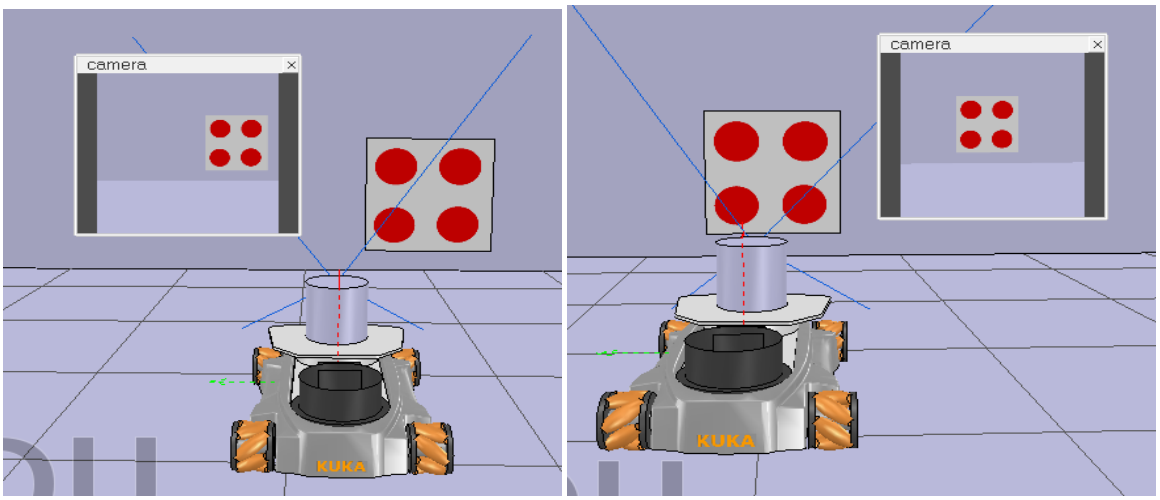


Figura 26. Posición inicial y final del robot

El robot en esta simulación se tendrá que desplazar hacia la derecha para que se coloque en la posición final que se ha almacenado.

Para ello, se ha probado con varios tipos de ganancias para comprobar en que afecta al controlador ese parámetro.

En la primera prueba, se le ha proporcionado al controlador una ganancia de 0.1. Como se puede observar en la figura 27, el robot prácticamente alcanza la posición, el tiempo que tarda en llegar a una zona muy estable es de 300 segundos aproximadamente.

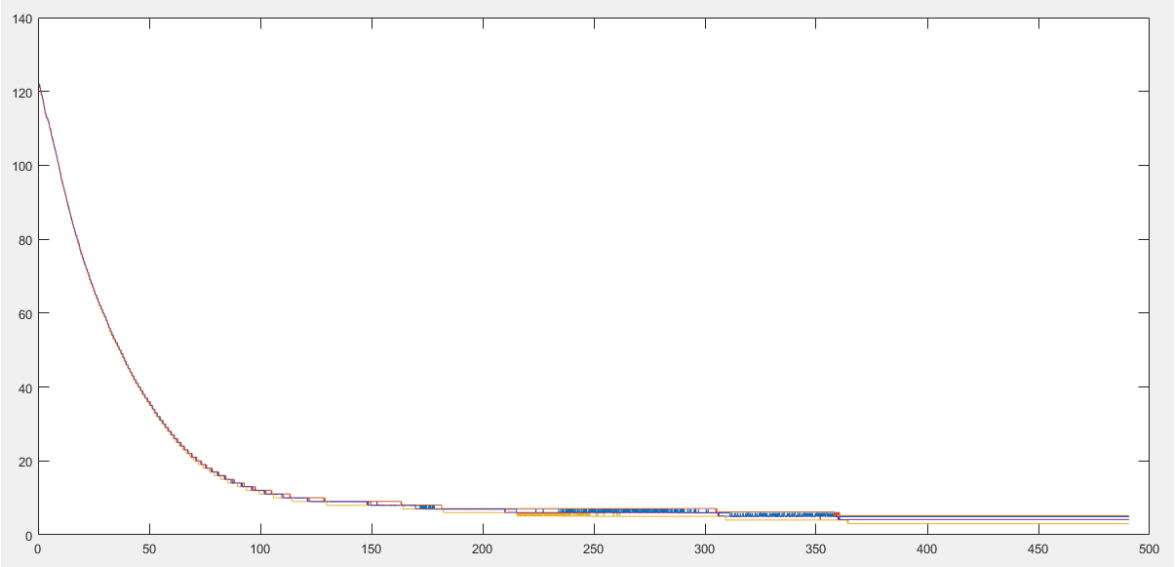


Figura 27. Error de las distancias entre las características de la imagen de la cámara. Lambda = 0.1

Es evidente que la causa de tardar tanto en llegar a la posición final, es porque las velocidades que saca el controlador son muy bajas como se puede observar en la figura 28.

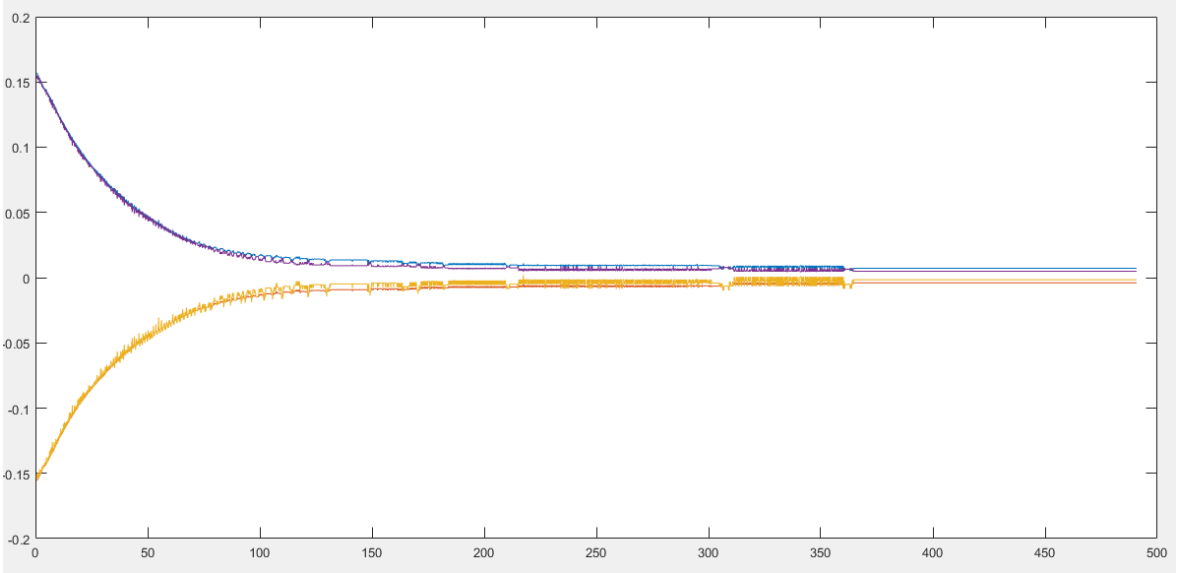


Figura 28. Velocidades de las 4 ruedas del robot. Lambda = 0.1

Para aumentar la velocidad del robot para que llegue a la posición deseada en menor tiempo, habrá que aumentar la ganancia. En la figura 29 y 30, se puede observar que ocurre cuando se le pone al controlador una ganancia muy elevada.

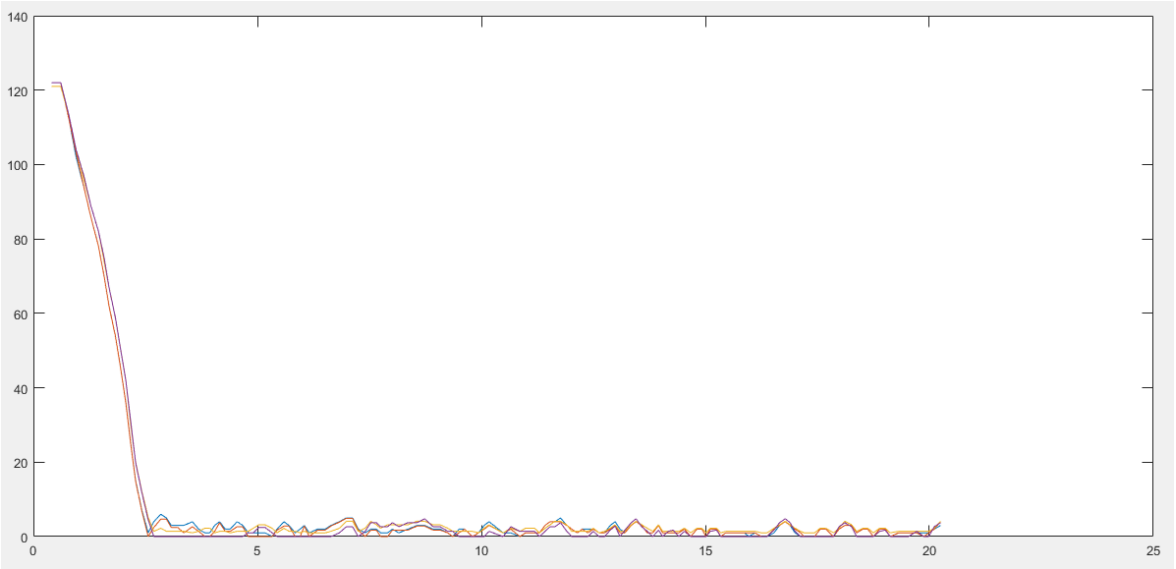


Figura 29. Error de las distancias entre las características de la imagen de la cámara. Lambda = 10

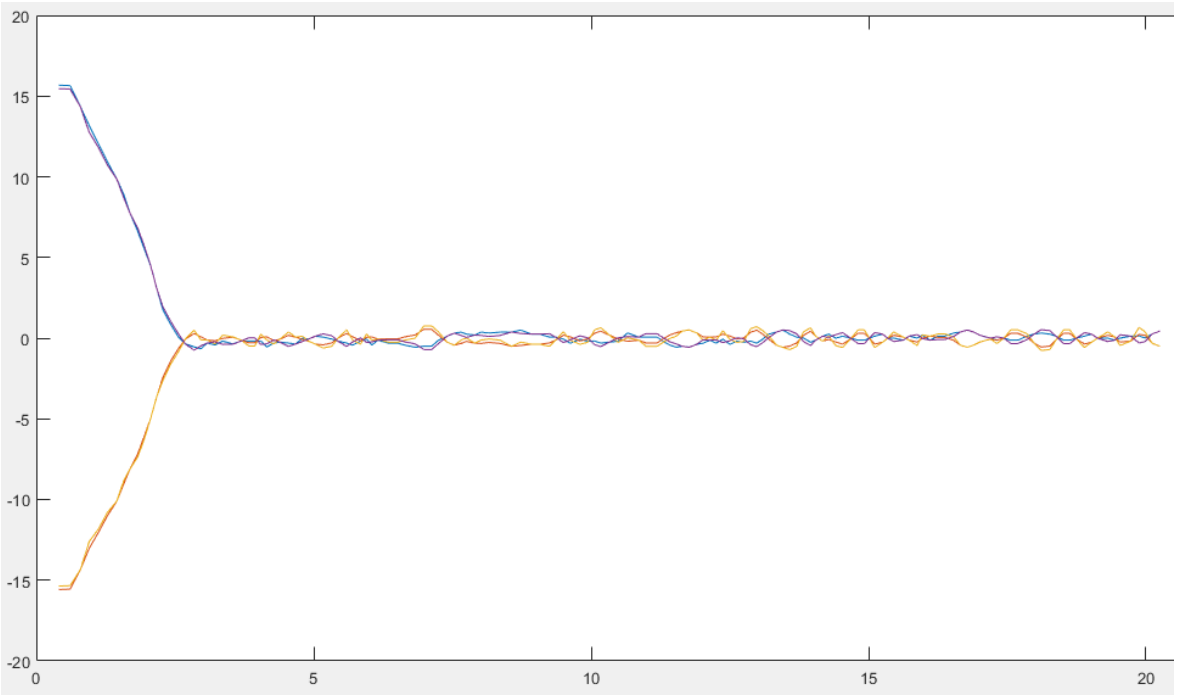


Figura 30. Velocidades de las 4 ruedas del robot. Lambda = 10

En la figura 30, se puede comprobar que las velocidades de las ruedas no consiguen estabilizarse impidiendo que el robot esté quieto y las distancias entre las características de las imágenes siempre estén aproximándose y alejándose.

Después de varias pruebas, se consiguió una ganancia bastante buena que tardaba relativamente poco en llegar a una posición cercana a la deseada y estabilizarse en esa posición. La ganancia utilizada en las figuras 31 y 32, es de 0.8.

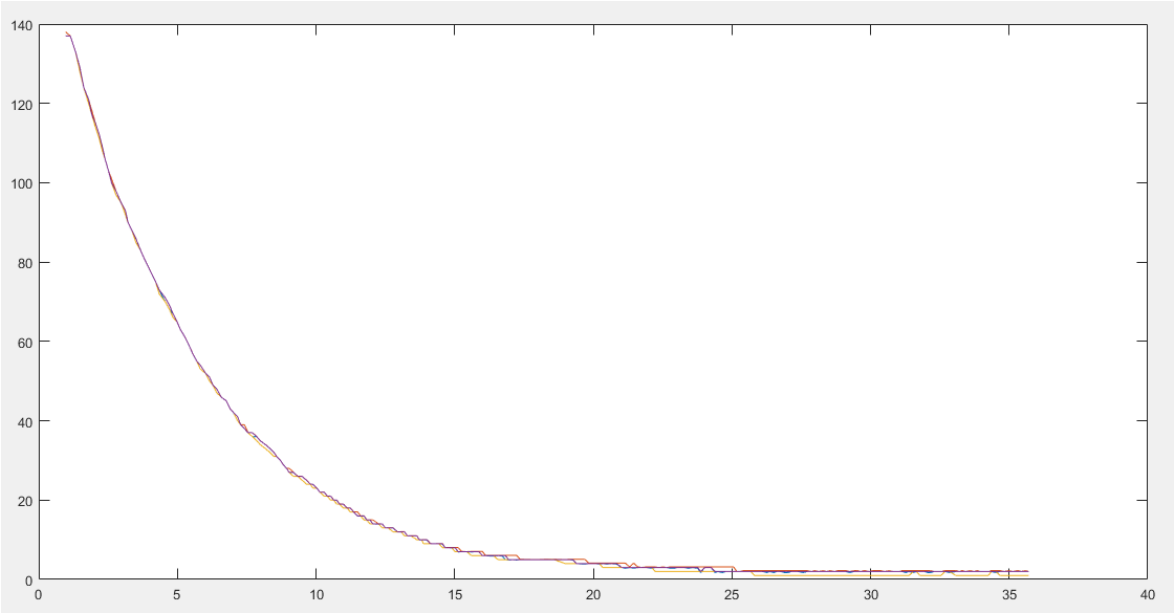


Figura 31. Error de las distancias entre las características de la imagen de la cámara. Lambda = 0.8

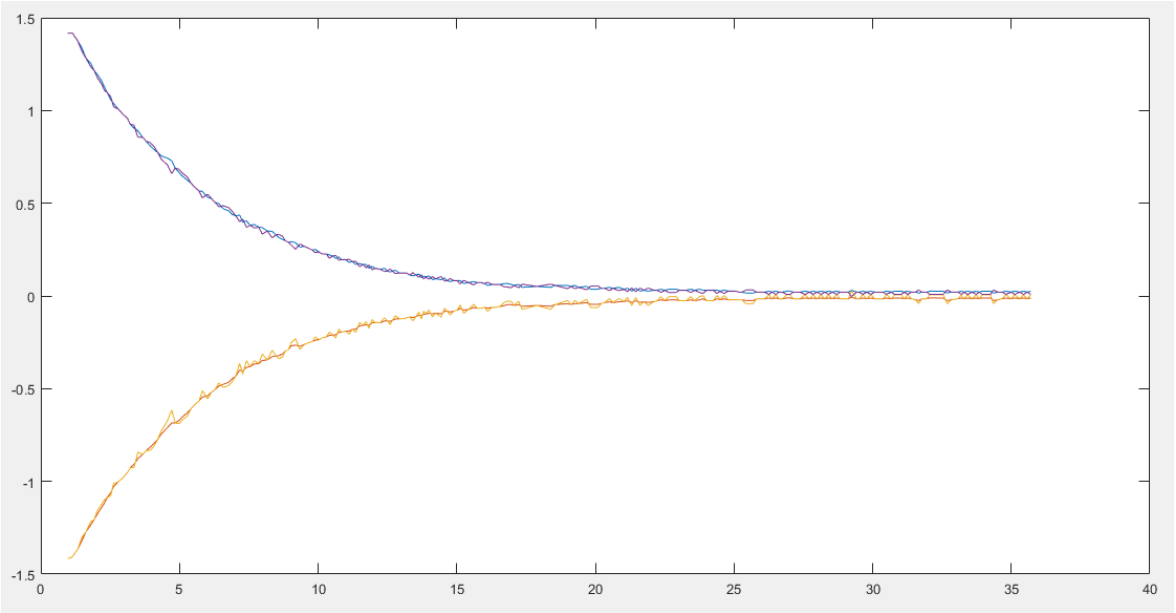


Figura 32. Velocidades de las 4 ruedas del robot. Lambda = 0.8

En las figuras anteriores, se puede observar como las velocidades de las ruedas se siguen moviendo pero es un movimiento prácticamente nulo, ya que las distancias entre las características es después de unos 15 segundos son también cercanas a 0.

Con este pequeño estudio de las ganancias para este controlador en el robot omnidireccional, se puede decir que cuanto más elevada es la ganancia, el robot va mucho más rápido a la posición, pero no consigue estabilizarse, por ello hay que buscar el equilibrio o utilizar algún tipo de mejora más inteligente para este controlador, para que la ganancia sea más elevada al principio para la aproximación inicial y cambie para la aproximación final haciendo movimientos más lentos.

4.1.2. Desplazamiento frontal

En este caso, el desplazamiento se hará hacia adelante. La posición final será la misma que la vista en la figura 26 y la inicial la que se puede ver en la figura 33.

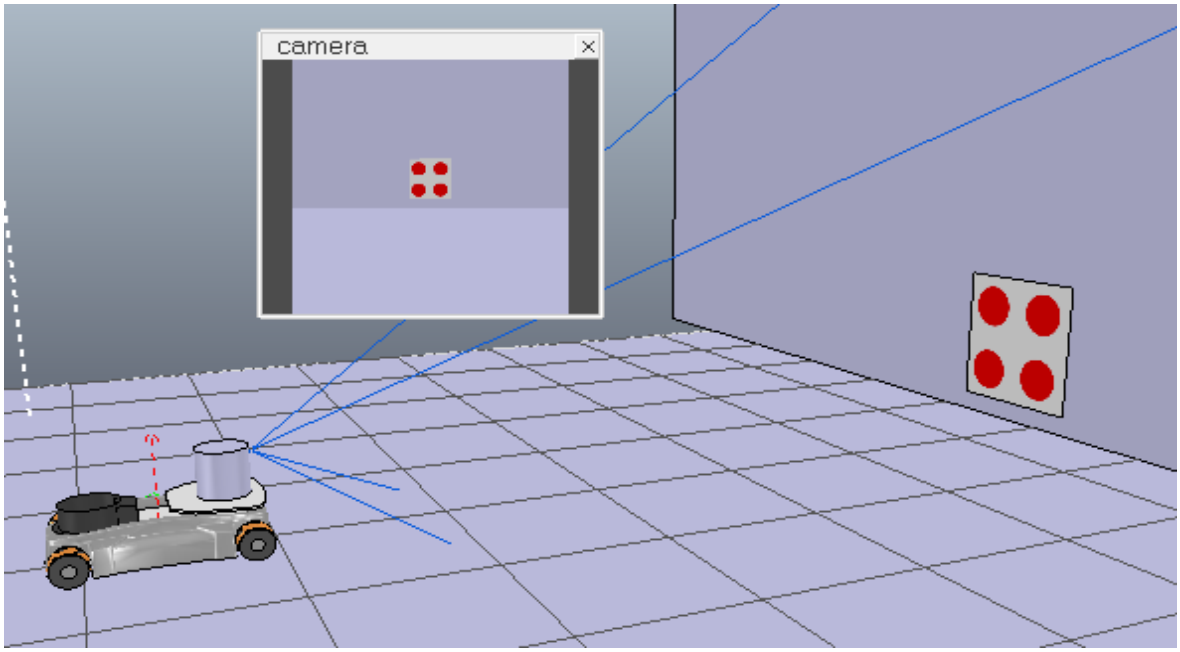


Figura 33. Posición inicial de la prueba del robot enfrente.

Para el caso de este estudio, se le ha proporcionado al controlador una ganancia de 0.8 también, ya que como se ha visto en el caso anterior, era de los mejores resultados que proporcionaba.

Los resultados que devuelve el estudio al ir de enfrente se pueden ver en las figuras 34 y 35.

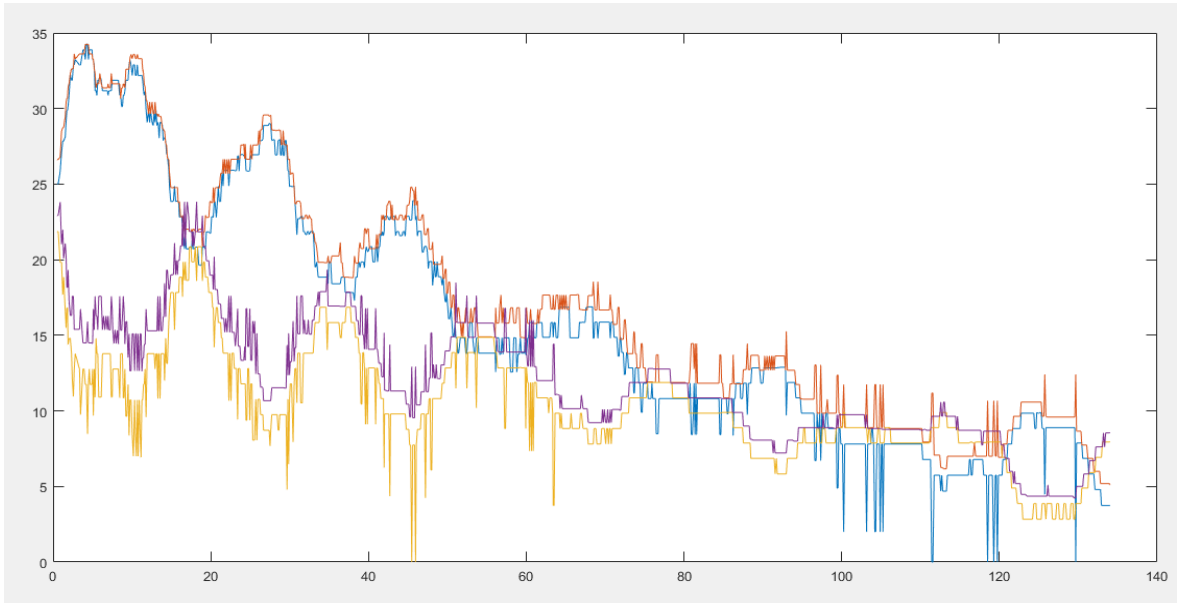


Figura 34. Error de las distancias entre las características de la imagen de la cámara. Lambda = 0.8

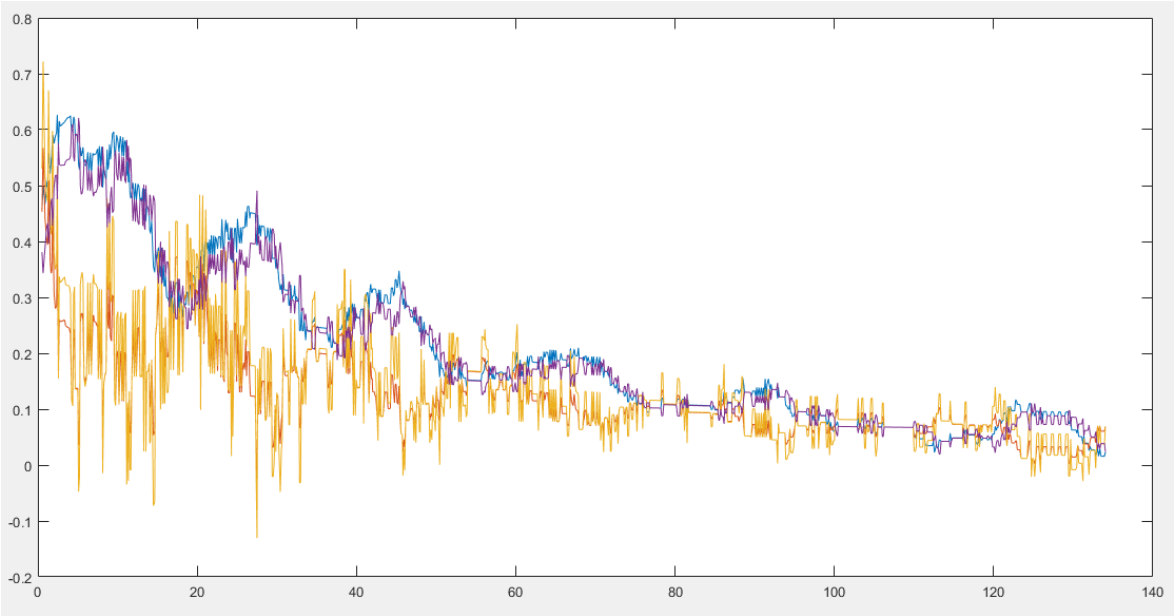


Figura 35. Velocidades de las 4 ruedas del robot. Lambda = 0.8

Se puede observar que en la figura 34, las distancias entre las características se van aproximando a 0, es decir, que el robot se aproxima a la situación final, pero se está aproximando con un movimiento no muy natural. En lugar de ir de frente como era lo esperado, el robot se está moviendo con un ligero movimiento hacia los laterales.

Esto se puede apreciar mejor en la figura 34 en la que se puede comprobar como las velocidades que van recibiendo las ruedas no son tan fluidas como en el caso del desplazamiento lateral, aunque se van acercando a 0 porque se aproxima al patrón, pero el movimiento que realiza este robot con el controlador visual que se le ha proporcionado, no es el movimiento fluido y rectilíneo que se esperaba.

4.1.3. Desplazamiento completo

En este caso, se ha querido hacer un desplazamiento más completo que en los casos anteriores para comprobar de qué manera se aproximaba el robot a su objetivo.

En la parte derecha de la figura 36, se puede observar que la posición final, está como en los estudios anteriores, pero en esta ocasión se ha girado un poco para que el movimiento sea más completo. Y la posición inicial que se puede ver en la parte izquierda de la misma figura, está mucho más alejado que en los casos anteriores. También se le ha desplazado hacia la izquierda y se ha girado en el sentido contrario al que tenía que llegar.

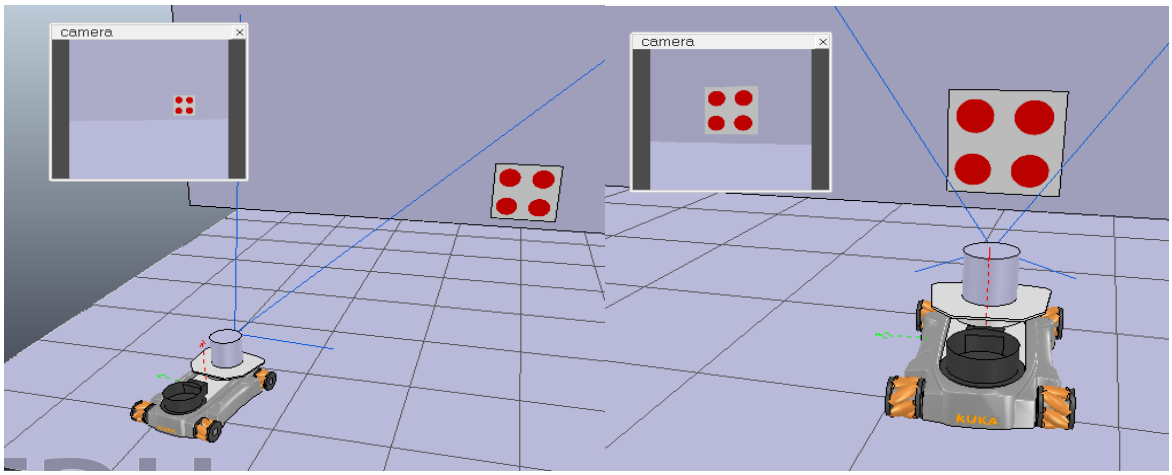


Figura 36. Posición inicial/final difícil

En este caso también se le ha puesto una ganancia al controlador de 0.8 porque es la ganancia que mejores resultados ha obtenido en el estudio de los desplazamientos anteriores, los resultados se pueden observar en las figuras 37, y 38.

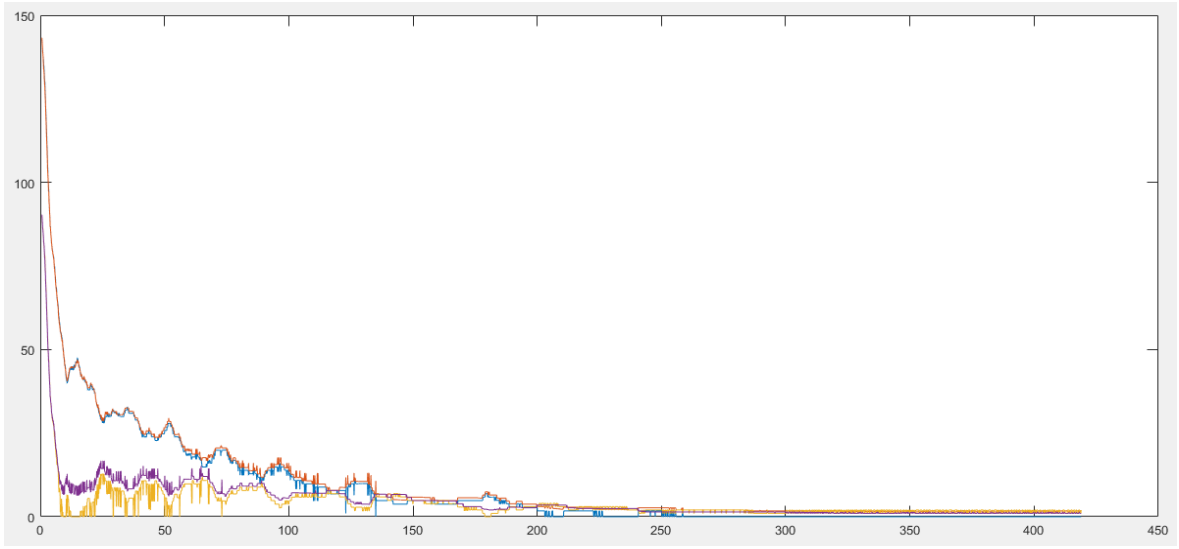


Figura 37. Error de las distancias entre las características de la imagen de la cámara. Lambda = 0.8

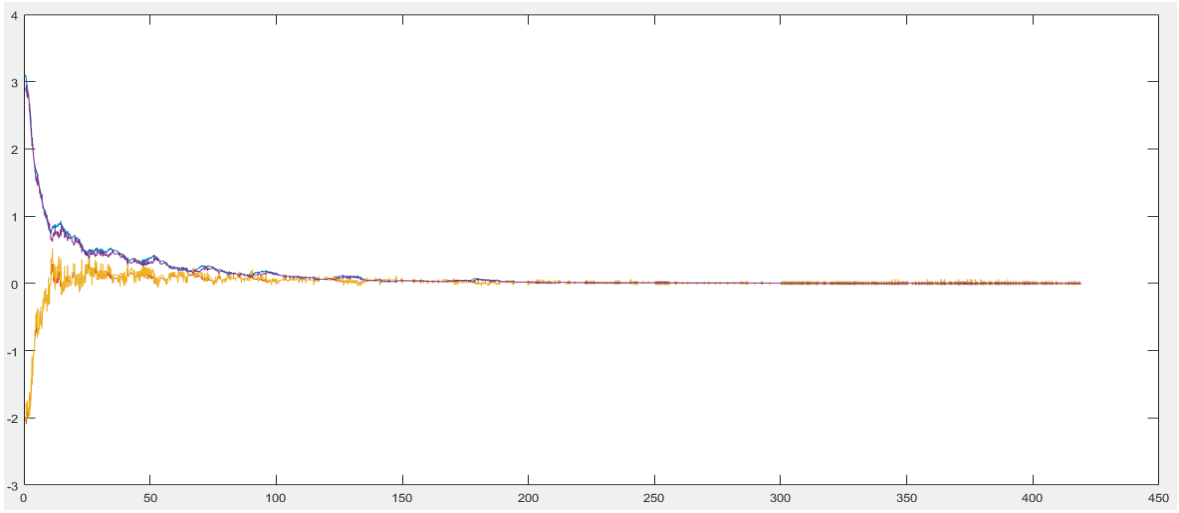


Figura 38. Velocidades de las 4 ruedas del robot. Lambda = 0.8

En las figuras anteriores, se puede observar que como era de esperar por los estudios de los desplazamientos realizados anteriormente, el robot tiene en los primeros 30 segundos una aceleración muy elevada en las ruedas, como se puede observar en la figura 38, pero conforme el robot se aproxima a la posición final, se frena mucho su velocidad para aproximarse con más suavidad. En este caso, no ocurre como en el estudio del desplazamiento frontal, en el que el robot tenía una aproximación muy mala. En este caso al mezclar todos los movimientos, el robot tiene un acercamiento bastante suave y fluido.

Además, como se puede observar en el zoom realizado de las distancias entre las características de las imágenes en la figura 39, el error en algunos puntos es de 0 pixeles de diferencia y en otros de 1 o 2 pixeles. Por lo que se puede decir que la aproximación con este controlador es bastante buena para movimientos complejos.

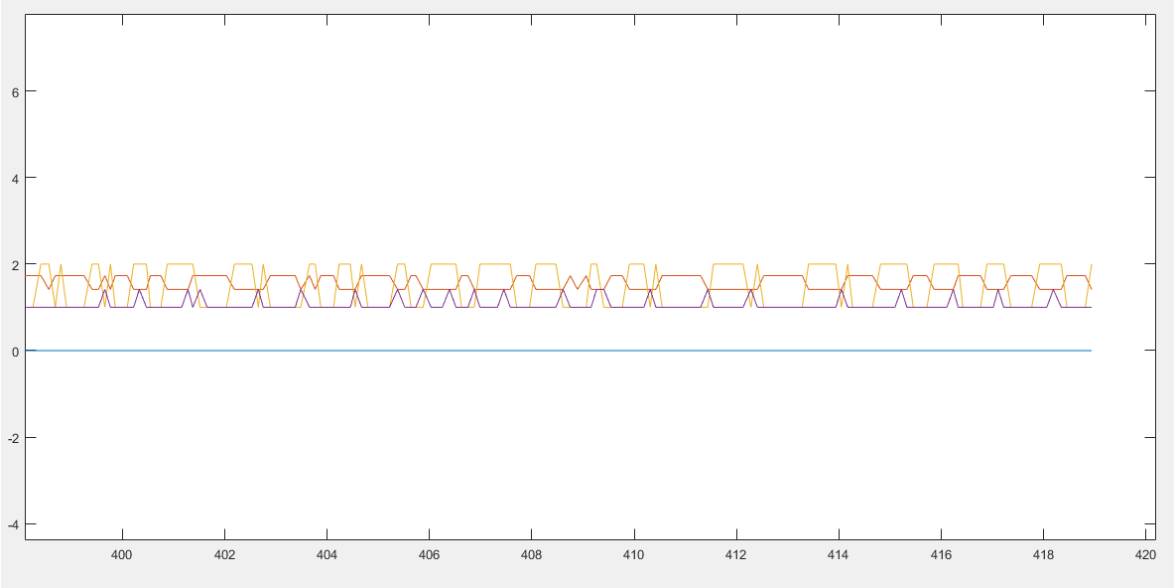


Figura 39. Error de las distancias entre las características de la imagen de la cámara maximizado. Lambda = 0.8

5. Conclusiones

A lo largo de este trabajo, se ha podido comprobar que para aplicar algún controlador a cualquier tipo de robot, es necesario tener una simulación previa del mismo para que no pueda sufrir ningún tipo de daños, ni el robot, ni el entorno que lo rodea. Debido a que con un mal ajuste del controlador que se le ponga al robot, podría moverse de manera completamente errática y desplazarse hacia cualquier posición inesperada.

Se ha demostrado que Matlab y V-REP funcionan perfectamente a la hora de hacer simulaciones sobre estos temas. Sobre todo gracias a la toolbox de robótica creada por Peter Corke, la cual permite crear un modelo de cualquier tipo de robot, y a que en V-REP, se le pueden añadir cualquier tipo de modelado de cualquier objeto para poder recrear exactamente el entorno real.

Para la comunicación entre estos dos programas, como se ha visto, es mucho más fácil y rápida que con otro tipo de programas que requieren de un conocimiento muy profundo de las aplicaciones que se utilizan.

Además gracias a V-REP, se pueden incorporar cualquier tipo de robots mientras se tenga el modelado de los mismos para poder probarlos, que actualmente los mismos fabricantes, por lo general, dejan sus modelados ya hechos de sus robots.

También se ha comprobado la eficiencia del controlador PD con compensación gravitacional a la hora de poner todas las articulaciones de un brazo robótico en una posición deseada. Este tipo de controladores son muy útiles en el mundo de la ingeniería industrial en la que se puede necesitar conocer en todo momento la posición del brazo robótico, ya que se podría causar daños al entorno en caso de no controlar todas sus articulaciones.

En el caso del controlador visual para los brazos multiarticulares, es imposible para el usuario controlar el movimiento de las articulaciones de estos brazos robóticos, ya que el controlador no se basa en la posición de ellas, si no en la imagen que está viendo el robot con la cámara incorporada. Pero como se ha podido ver, en caso de no importar cómo lleguen las articulaciones a la posición final, este controlador aproxima perfectamente al patrón indicado.

Y en el caso de los robots omnidireccionales, con los resultados de las pruebas que se han realizado en este trabajo, se puede observar que es capaz de seguir cualquier patrón que se

le ponga y seguir su objetivo siempre que no lo pierda de vista, corrigiendo siempre su posición en caso de que el patrón haga algún desplazamiento, haciendo este tipo de controladores muy buenos para el caso de querer realizar algún tipo de tracking.

Para futuros trabajos, se podrían incorporar estos controladores al robot KUKA YouBot sin modificar, el cual tenía un brazo robótico incorporado de 6 grados de libertad y una herramienta de tipo pinza incorporada en el mismo.
Se podría utilizar el controlador visual del robot omnidireccional para llega a algún tipo de objetivo y utilizar el controlador de posición para el brazo robótico o incluso utilizar también un control visual para coger algún tipo de pieza con este.

El controlador visual implementado durante este trabajo, se ha probado en un entorno simulado de un vehículo robótico que se desplaza a una posición, pero este tipo de controladores, ya que en su aproximación final al objetivo reduce tanto su velocidad para poder llegar correctamente y sin demasiada oscilación, como se han podido ver con las gráficas resultantes de este controlador, se podría utilizar para cualquier tipo de aproximación peligrosa como desactivación de explosivos, que pone en riesgo vidas humanas, o incluso algún tipo de aproximación espacial en la que se le aplique algún tipo de patrón para que busque la posición ideal para el acercamiento de estructuras sin dañar las mismas.

5. Bibliografía

[1] Dr. Asad Yousuf, Mr. William Lehman, Dr. Mohamad A. Mustafa, Dr. Mir M Hayder (2015). *Introducing Kinematics with Robot Operating System (ROS)*

[2] Lucas Nogueira, *Comparative Analysis Between Gazebo and V-REP Robotic Simulators*

[3] Takegaki M., Arimoto S. (1981). *A new feedback method for dynamic control of manipulators, Transactions ASME, Journal of Dynamic Systems, Measurement and Control,* Vol. 103, pp. 119-125.

[4] Kelly, R., Santibáñez, V. (2003). *Control de movimiento de robots manipuladores*.

[5] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Dynamics and Control.* Pág 239-241. *Segunda edición.*

[6] Rafael Kelly, Rodolfo Haber, Rodolfo E. Haber-Guerra & Fernando Reyes. (2013) *Lyapunov Stable Control of Robot Manipulators: A Fuzzy Self-Tuning Procedure.*

[7] Tan, C.C. and A.Y. Zomaya. (December 1993) *Neural-network-based direct adaptive controller design for nonlinear manipulators, In Proc. of the 32nd Conference on Decision and Control, San Antonio, TX.*, pp. 3231-3235.

[8] Malki, H. Li and G. Chen. (November 1994) *New design and stability analysis of fuzzy Proportional-Derivative control systems, IEEE Transactions on Fuzzy Systems*, Vol. 2, No.4, pp. 245-254.

[9] Xu, Y., J.M. Hollerbach and D. Ma. (February 1995). *A nonlinear PD controller for force and contact transient control, IEEE Control Systems*, Vol. 15, No. 1, pp. 15-21.

[10] Xinyu Du, Hao Ying. (2010). *Derivation and Analysis of the Analytical Structures of the Interval Type-2 Fuzzy-PI and PD Controllers*.

[11] Dongrui Wu, Jerry M. Mendel. (2014). *Designing Practical Interval Type-2 Fuzzy Logic Systems Made Simple.*

[12] François Chaumette, S. Hutchinson. (2009). *Visual servo control, Part I: Basic approaches*.

[13] Renliw Fleurmond, Viviane Cadenat. *Multi-cameras Visual Servoing to Performa Coordinated Task using a Dual Arm Robot*.

[14] Ciro Potena, Daniele Nardi and Alberto Pretto. *Target Aware Optimal Visual Navigation for UAVs.*

[15] Gurdeep Singh, Amir Anvar. *Investigating feasibility of target detection by visual servoing using UAV for oceanic applications.*

[16] Justin Thomas, Giuseppe Loianno, Koushil Sreenath, and Vijay Kumar. *Toward Image Based Visual Servoing for Aerial Grasping and Perching.*

[17] R. Kelly, V.Santibáñez y A.Loria. *Control of Roboy Manipulators in Joint Space*

[18] M.K.S.H.Maldeniya1, R.C.Madurawe , L.B.H.T.Thilakasiri , T.M.S.Thennakoon , R.M.T.P. Rajakaruna. *Remote Controlled 4wd Omni Directional Robot Using Mecanum Wheels*.

[19] Ioan Doroftei, Victor Grosu and Veaceslav Spinu. *Omnidirectional Mobile Robot – Design and Implementation.*

[20] Peter Corke. *Robotics, Vision and Control*

[21] John J. Craig. *Introduction to Robotics. Mechanics and Control*

[22] http://www.coppeliarobotics.com/helpFiles/en/remoteApiModusOperandi.htm