UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

DISEÑO DE ALGORITMOS DE CONTROL VISUAL BASADO EN IMAGEN Y BASADO EN POSICIÓN PARA SISTEMAS ROBOTIZADOS Y VALIDACIÓN MEDIANTE SIMULACIÓN Y EXPERIMENTACIÓN REAL

AUTOR: ALBERTO GARCÍA FERNÁNDEZ

TUTORA: ALICIA ESPARZA PEIDRO

COTUTOR: LUIS IGNACIO GRACIA CALANDÍN

Curso Académico: 2017-18

Resumen:

En el presente Trabajo Fin de Máster, se ha realizado el estudio y la implementación de algoritmos de control cinemático de velocidad y de aceleración de robots, así como de servo control visual en sus dos metodologías clásicas: *Image Based Visual Servoing* (IBVS) y *Position Based Visual Servoing* (PBVS). Los algoritmos de control cinemático y de *Visual Servoing* se han implementado mediante simulación para una gran variedad de casos, desarrollando para ello una aplicación de simulación de control para brazos robóticos de hasta diez articulaciones rotativas. Por otro lado, las metodologías de control visual también se han validado mediante experimentación real en un robot industrial KUKA.

Para ello, en primer lugar, se ha realizado un estudio teórico que engloba las diferentes metodologías de control de interés para el presente trabajo, estudio plasmado en el capítulo de desarrollo teórico de la memoria descriptiva. En dicho capítulo se presentan los fundamentos matemáticos y conceptuales del control cinemático de velocidad y aceleración de robots, y se continúa después con la explicación del *Visual Servoing*, exponiendo primero sus bases generales y después su concreción en las metodologías IBVS y PBVS.

Con la adquisición de dichos conocimientos teóricos, ha sido posible desarrollar una aplicación de simulación en *Matlab* que permite un alto grado de personalización de los casos de control. Con ello, se han realizado múltiples pruebas de simulación para estudiar el desempeño de los diferentes algoritmos de control estudiados, con especial interés en el *Visual Servoing*. Todo esto se explica en el capítulo del simulador.

Por último, como se explica en el capítulo de validación experimental, se han probado los algoritmos de IBVS y PBVS para tareas de posicionado y de IBVS para *tracking* con un brazo robótico industrial KUKA, lo cual ha permitido familiarizarse con las particularidades de la implementación real.

Palabras clave: robótica, control visual, simulador, control cinemático

Resum:

Al present Treball Final de Màster, s'ha realitzat l'estudi i la implementació d'algoritmes de control cinemàtic de velocitat i d'acceleració de robots, així com de servo control visual en les seues dues metodologies clàssiques: *Image Based Visual Servoing* (IBVS) i *Position Based Visual Servoing* (PBVS). Els algoritmes de control cinemàtic i de *Visual Servoing* s'hi han implementat mitjançant simulació per a una gran varietat de casos, desenvolupant per això una aplicació de simulació de control per a braços robòtics de fins a deu articulacions rotatives. D'altra banda, les metodologies de control visual també s'han validat mitjançant experimentació real en un robot industrial KUKA.

Per això, en primer lloc, s'ha realitzat un estudi teòric que engloba les diferents metodologies de control d'interès per al present treball, estudi plasmat al capítol de desenvolupament teòric de la memòria descriptiva. En aquest capítol es presenten els fonaments matemàtics i conceptuals del control cinemàtic de velocitat i acceleració de robots, i es continua després amb l'explicació del *Visual Servoing*, exposant primer les seues bases generals i després la seua concreció a les metodologies IBVS i PBVS.

Amb l'adquisició d'aquests coneixements teòrics, ha sigut possible desenvolupar una aplicació de simulació en *Matlab* que permet un alt grau de personalització dels casos de control. Amb això, s'han realitzat múltiples proves de simulació per tal d'estudiar l'actuació dels diferents algoritmes de control estudiats, amb especial interès en el *Visual Servoing*. Tot açò s'explica al capítol del simulador.

Per últim, com s'explica al capítol de validació experimental, s'han provat els algoritmes d'IBVS i PBVS per a tasques de posicionat i d'IBVS per a *tracking* amb un braç robòtic industrial KUKA, la qual cosa ha permès familiaritzar-se amb les particularitats de la implementació real.

Paraules clau: robòtica, control visual, simulador, control cinemàtic

Abstract:

In the present Master Thesis, the study and implementation of velocity and acceleration cinematic control for robots, as well as of two classical Visual Servoing methodologies: *Image Based Visual Servoing* (IBVS) and *Position Based Visual Servoing* (PBVS), has been carried out. The cinematic control and Visual Servoing algorithms have been implemented through simulation, developing to this end a control simulation application for robotic arms with a maximum of ten rotary joints. On the other hand, Visual Servoing methodologies have also been validated by real experimentation using a KUKA industrial robot.

In order to achieve this, first of all, a theoretical study encompassing the different control methodologies of interests for this project has been carried out, which has been reflected in the theoretical development chapter of the descriptive report. In that chapter, mathematical and conceptual foundations of velocity and acceleration cinematic control for robots are presented, which is then followed by the explanation of Visual Servoing, first exposing its general basis and then its particularization in IBVS and PBVS methodologies.

With the acquisition of that theoretical knowledge, it has been possible to develop a simulation application in Matlab which allows to perform control cases with a high degree of customization. Thanks to that, many simulation tests have been done in order to study the performance of the different control algorithms which have been studied, showing a special interest in Visual Servoing. All this is explained in the simulator chapter.

Finally, as it is explained in the experimental validation chapter, IBVS and PBVS algorithms for positioning tasks and IBVS algorithm for tracking tasks have been tested using a KUKA industrial robotic arm, what has allowed getting acquainted with the particularities of real implementation.

Keywords: robotics, visual servoing, simulator, cinematic control

Documentos incluidos en el Trabajo Fin de Máster:

-Memoria descriptiva

-Presupuesto

-Anejo: Manual del simulador

Índice de la memoria descriptiva:

CAPÍTULO 1. INTRODUCCIÓN 14

1.1 Objetivos 14

1.2 Antecedentes 14

CAPÍTULO 2. DESARROLLO TEÓRICO 15

2.1 Los robots 15

2.2 Los robots industriales 15

2.3 Control cinemático de robots 16

2.3.1 Problema cinemático directo. Metodología de Denavit-Hartenberg 16

2.3.2 Relación entre velocidad del efector final y velocidades de las articulaciones. Modelo cinemático diferencial 18

2.3.2.1 Explicación conceptual. Jacobiana analítica 18

2.3.2.2 Implementación. Jacobiana geométrica 19

2.3.2.3 Aplicación a la robótica. Pseudo-inversa y configuraciones singulares 20

2.3.3 Control de velocidad 21

2.3.4 Control de aceleración 23

2.4 *Visual servoing* 25

2.4.1 Funcionamiento general del *Visual Servoing* (*eye-in-hand*) 26

2.4.1.1 Aproximación al problema mediante el caso de posicionado mediante *Free Camera* 27

2.4.1.2 *Tracking* de un *target* en movimiento 28

2.4.1.3 Implementación en robots 29

2.4.2 *Image Based Visual Servoing (IBVS)* 30

2.4.2.1 Comportamiento del IBVS 33

2.4.3 *Position Based Visual Servoing* (PBVS) 34

2.4.3.1 Comportamiento del PBVS 37

CAPÍTULO 3. SIMULADOR 39

3.1 Descripción general del simulador 39
3.2 Desarrollo del simulador 40
3.2.1 Entorno de desarrollo GUIDE 40
3.3 Secciones del simulador 43
3.3.1 Carga del robot a simular 44
3.3.2 Selección del modo de funcionamiento 51
3.3.3 Modificación manual de la posición articular del robot 52
3.3.4 Modificación de la posición articular del robot mediante trayectoria personalizada 53
3.3.5 Control cinemático de velocidad 54
3.3.6 Control cinemático de aceleración 58
3.3.7 Control IBVS *eye-in-hand* 61
3.3.8 Control PBVS *eye-in-hand* 68
3.4 Desempeño de los controles simulados 69
3.4.1 Control de velocidad 70
3.4.2 Control de aceleración 74
3.4.3 Control IBVS *eye-in-hand* 77
3.4.3.1 Posicionado 77
3.4.3.2 Seguimiento 82
3.4.4 Control PBVS eye-in-hand 88
3.4.4.1 Posicionado 88
3.4.4.2 Seguimiento 91
CAPÍTULO 4. VALIDACIÓN EXPERIMENTAL 96
4.1 Disposición del entorno de experimentación 96
4.1.1 Modificaciones realizadas sobre la configuración inicial 101
4.2 Validación experimental 102
4.2.1 Posicionado 104
4.2.1 Seguimiento mediante control IBVS 114
CAPÍTULO 5. CONCLUSIONES Y TRABAJOS FUTUROS 117
5.1 Conclusiones 117
5.2 Trabajos futuros 117
CAPÍTULO 6. BIBLIOGRAFÍA 118

Índice del presupuesto:

1 Justificación del presupuesto 122
2 Estudio económico 122
2.1 Costes de personal 122
2.2 Coste del material inventariable 123
2.3 Material fungible 124
3 Resumen del presupuesto 124

Índice del anejo (Manual del simulador):

Qué es *Robotic Arm Control Simulator* 127
Qué metodologías de control incluye 127
Cómo realizar una simulación 127
1-Incluir el robot en la librería 127
2-Cargar el robot en la interfaz gráfica 128
3-Seleccionar la metodología y realizar la simulación 129
Cómo utilizar cada metodología de control 130
1-Control manual 130
2-Trayectoria articular 131
3-Control de velocidad 131
4-Control de aceleración 133
5-IBVS 133
5.1-El objeto y su trayectoria 134
5.2-Posicionado y *tracking* 135
5.3-Parámetros de la simulación 135
4.4-Plano imagen de la cámara 135
6-PBVS 136

Índice de Figuras:

Figura 1-Ejemplo de brazo robot industrial 15
Figura 2-Ejemplo de notación Denavit-Hartenberg 17
Figura 3-Jacobiana analítica 18

Figura 4-Diagrama de bloques del control de velocidad 22

Figura 5-Diagrama de bloques del control de velocidad con seguimiento de trayectoria 23

Figura 6-Diagrama de bloques simplificado del control de aceleración 25

Figura 7-Configuración eye-in-hand (izq.) y configuración eye-to-hand (der.) 26

Figura 8- Proyección de un objeto sobre el plano imagen 27

Figura 9- a)Posición deseada de la cámara respecto al target de 4 puntos, b) posición de partida de la cámara respecto al target de 4 puntos, c) comparativa de la imagen de referencia y la imagen de partida 32

Figura 10- Esquema del control IBVS 33

Figura 11- Convergencia en posicionado IBVS: a) Trayectoria de los puntos del target hasta la posición de referencia siguiendo líneas prácticamente rectas, b) evolución de la velocidad de la cámara, c) desplazamiento del centro óptico de la cámara. 33

Figura 12-Desempeño al elegir la primera opción; a), b) y c) representan la misma información que en la figura 11 35

Figura 13-Desempeño al elegir la segunda opción; a), b) y c) representan la misma información que en la figura 11 35

Figura 14- Transformaciones geométricas entre los sistemas de referencia 36

Figura 15-Esquema del control PBVS 37

Figura 16-Carga de un proyecto de interfaz gráfica en GUIDE 41

Figura 17-Editor de GUIDE 41

Figura 18-Elementos descritos: a)Pulsador, b)Casillas de texto editable, c)Menú desplegable de selección múltiple, d)Barra deslizadora 43

Figura 19-Ejemplo de representación gráfica tridimensional mediante funciones de rvctools 45

Figura 20-Panel de selección del simulador para la carga del robot 46

Figura 21-Efecto de no hacer un reset del escenario antes de representar el nuevo robot tras cargar un nuevo robot 47

Figura 22-Rotación del sistema de referencia del camera-frame respecto del mundo en el simulador 48

Figura 23-Posición requerida para colocar la cámara correctamente cuando todos los ángulos de las articulaciones se anulan 49

Figura 24-Apariencia del modo manual en las primeras versiones 50

Figura 25-Librería completa de robots de la aplicación. KUKA: a) Agilus 900, b) KR5_2ARC_HW, c) KR5_arc, d) KR5_sixx_R650, e) KR5_sixx_R850, f) KR6_2, g) KR16_arc_HW, h) KR20_3, i) KR10_R900 51

Figura 26-Menú de selección del modo de funcionamiento 51

Figura 27-Ejemplo de funcionamiento del modo manual del simulador 52

Figura 28-Ejemplo de saturación de una de las articulaciones 53

Figura 29-Ejemplo de trayectoria articular personalizada 54

Figura 30-Panel de control del modo de funcionamiento de trayectoria articular 54

Figura 31-Panel de control de velocidad. Izq) tarea de posicionado, der) tarea de tracking. 55

Figura 32-Roll Pitch Yaw de acuerdo con la notación de rvctools 55

Figura 33-Posicionado con control de velocidad (referencia [0,0.7,1.0044,0,0,0]) 57

Figura 34-Tracking de una trayectoria en arco de circunferencia y orientación [0,0,0] (rpy) 58

Figura 35-Panel de control de aceleración. Izq) tarea de posicionado, der) tarea de tracking 59

Figura 36-Panel de control IBVS. Izq) Trayectoria de prueba, centro) Modo personalizado, der) Posibles leyes de control 62

Figura 37- Secuencia de tracking IBVS 63

Figura 38- Punto de vista de la cámara, tracking IBVS 63

Figura 39- Ejemplo de gráficas generadas al finalizar la simulación. Desde arriba: variables de s, error y variación del error debida al movimiento del target 64

Figura 40-Panel de control PBVS 68

Figura 41-Evolución del error de posición y la velocidad de las articulaciones del control de velocidad con referencia fija para una ganancia proporcional de 0.2 y una referencia [0, 0.7, 1.0044, 0, 0, 0]. . 71

Figura 42- Evolución del error de posición y la velocidad de las articulaciones del control de velocidad con referencia fija para una ganancia proporcional de 0.4 y una referencia [0, 0.7, 1.0044, 0, 0, 0]. . 72

Figura 43-Evolución del error de posición y la velocidad de las articulaciones del control de velocidad con referencia móvil para una ganancia proporcional de 0.5 y una referencia de trayectoria circular con altura 1 y orientación [0,0,0]. KUKA Agilus 73

Figura 44-Secuencia del control de velocidad para referencia móvil circular 73

Figura 45- Evolución del error de posición y la velocidad de las articulaciones del control de velocidad con referencia móvil para una ganancia proporcional de 0.5 y una referencia de trayectoria circular con altura 1 y orientación [0,0,0]. 74

Figura 46- Evolución del error y la velocidad de las articulaciones en control de aceleración con referencia fija [0,0.7,1.0044,0,0,0], Kp=0.5 y Kv=0.8 75

Figura 47-Secuencia demostrativa de la sobre oscilación del caso de control correspondiente a la figura 46 75

Figura 48- Evolución del error y la velocidad de las articulaciones en control de aceleración con referencia fija [0,0.7,1.0044,0,0,0], Kp=0.5 y Kv=1.5 76

Figura 49- Evolución del error de posición y la velocidad de las articulaciones del control de aceleración con referencia móvil para Kp 0.8, Kv 1.5 y una referencia de trayectoria circular con altura 1 y orientación [0,0,0]. 76

Figura 50- Desde la gráfica superior: evolución de las coordenadas de los puntos del target sobre el plano imagen, evolución del error y término del efecto del movimiento del target, que no existe en este caso por ser una referencia fija 77

Figura 51- Ídem que la figura 50, segundo caso de posicionado 78

Figura 52-Secuencia del posicionado correspondientes a los casos 1 y 2 78

Figura 53-Ídem que la figura 50, para el tercer caso de posicionado 79

Figura 54-Ídem que la figura 50, para el caso de camera retreat con convergencia 80

Figura 55- Secuencia de camera retreat con convergencia, punto de vista de la cámara 81

Figura 56-Ídem que la figura 50, para el caso límite de camera retreat con giro demandado de 180 grados 81

Figura 57-Ídem que la figura 50, para el caso de tracking de una trayectoria circular de 200 segundos de duración con el KUKA KR 5-2 ARC HW. 82

Figura 58-Ídem que la figura 50, para el caso de seguimiento de una referencia móvil de trayectoria circular recurriendo tan solo a control proporcional. Nótese el error de posición asumido. 83

Figura 59- Ídem que la figura 50, para el caso de tracking de una referencia móvil de trayectoria circular de 100 segundos 84

Figura 60- Ídem que la figura 50, para el caso de tracking de una trayectoria circular en 100 segundos, λ 0.5 y disminuyendo el periodo de muestreo a 0.1 segundos 85

Figura 61-Ídem que la figura 50, para el caso de tracking de una trayectoria de prueba de tipo “corona” en 200 segundos y λ 0.2 86

Figura 62-Ídem que la figura 50, para el caso de tracking una trayectoria personalizada de 60 segundos de duración con λ 0.2 87

Figura 63- Secuencia del tracking IBVS de una trayectoria personalizada 88

Figura 64- Para la misma posición del target y el robot Der) Vista de la cámara PBVS, izq) Vista de la cámara IBVS 88

Figura 65- Desde la gráfica superior: evolución de las características del vector s, del error de posición, movimiento de los puntos del target sobre el plano imagen y término de feedforward, para el caso de posicionado con λ 0.2 89

Figura 66-Ídem que la figura 65, para el caso de posicionado con λ de 0.5 90

Figura 67--Secuencia del posicionado PBVS que causaría fallo de camera retreat en IBVS, punto de vista de la cámara 90

Figura 68-Ídem que la figura 65, para el caso de posicionado PBVS que provocaría fallo de camera-retreat en IBVS 91

Figura 69-Ídem que la figura 65, para el caso de tracking PBVS con trayectoria circular de 200 segundos y λ 0.2 92

Figura 70- Ídem que la figura 65, para el caso de tracking de trayectoria circular de 100 segundos y λ de 0.5 93

Figura 71-Ídem que la figura 65, para el caso de tracking de trayectoria de prueba "corona" de 30 segundos y λ de 0.5 94

Figura 72-Ídem que la figura 65, para el caso de tracking de la trayectoria personalizada con λ de 0.5 95

Figura 73-Espacio de trabajo del robot KUKA Agilus 96

Figura 74-Carril lineal al que está fijado el robot KUKA 97

Figura 75-Izq) Controlador KUKA KRC4 Compact, der) KUKA smartPAD 97

Figura 76-Webcam Logitech C300 fijada solidariamente al efector final del robot KUKA Agilus 98

Figura 77-PC externo utilizado para la experimentación 99

Figura 78-Target utilizado para la experimentación real 99

Figura 79-Diagrama de bloques de la comunicación para la experimentación 100

Figura 80-KUKA Agilus en la posición deseada dentro del espacio de trabajo 101

Figura 81-Posición de partida del programa cliente UDP en el smartPAD del controlador KRC4 101

Figura 82-Selección de puntos del target en sentido horario 102

Figura 83-KUKA smartPAD, habilitación de los accionamientos en modo automático 103

Figura 84-Punto de vista de la cámara, izq) PBVS, der) IBVS 103

Figura 85-Posición final del primer experimento de posicionado IBVS. Izq) λ de 0.2, der) λ de 0.5... 105

Figura 86- Posición final del primer experimento ded posicionado PBVS. Izq) λ de 0.2, der) λ de 0.5 105

Figura 87-Evolución de s en el primer experimento de IBVS. Sup)) λ de 0.2, inf)) λ de 0.5 105

Figura 88-Evolución de los puntos del target sobre el plano imagen y de c*Mc en el primer experimento de PBVS, a) λ de 0.2, b) λ de 0.5 106

Figura 89- Posición final del segundo experimento de posicionado IBVS. Izq) λ de 0.2, der) λ de 0.5107

Figura 90- Posición final del segundo experimento de posicionado PBVS. Izq) λ de 0.2, der) λ de 0.5 107

Figura 91- Análoga a la figura 87, para el segundo experimento de IBVS 108

Figura 92- Análoga a la figura 88, para el segundo experimento de PBVS 108

Figura 93- Evolución de la velocidad de las articulaciones en el segundo experimento de IBVS, sup) λ de 0.2, inf) λ de 0.5 109

Figura 94-Posición final del tercer experimento de posicionado, izq) IBVS, der) PBVS 109

Figura 95-Evolución de variables en el tercer experimento; a) s en IBVS comparado con s*, b) puntos del target sobre el plano imagen (sup) y c*Mc (inf) 110

Figura 96-Punto de vista de la cámara, tercer experimento de PBVS 111

Figura 97- Análoga a la figura 94, para el cuarto experimento de posicionado 112

Figura 98- Análoga a la figura 95, para el cuarto experimento de posicionado 112

Figura 99-Análoga a la figura 94, para el quinto experimento de posicionado 113

Figura 100- Análoga a la figura 95, para el quinto experimento de posicionado 114

Figura 101-Secuencia del experimento de tracking IBVS 115

MEMORIA DESCRIPTIVA

CAPÍTULO 1. INTRODUCCIÓN

1.1 Objetivos:

Los objetivos principales del presente Trabajo Fin de Máster son:

- Comprender distintas metodologías de control cinemático de robots manipuladores, particularmente los dos principales esquemas de servo control visual o *visual servoing*: *Image Based Visual Servoing* (IBVS) y *Position Based Visual Servoing* (PBVS).
- Desarrollar un simulador mediante el software *Matlab* en que se implementen dichos algoritmos de control cinemático para una amplia variedad de brazos robóticos articulados y de casos de control diferentes.
- Validar los algoritmos de servo control visual en experimentación real.

Además, el presente trabajo tiene como objetivos secundarios:

- Profundizar en los conocimientos sobre simulación adquiridos en el Máster Universitario en Ingeniería Industrial (MUII) con tal de posibilitar un resultado final del simulador completo, adaptable y accesible.
- Adquirir conocimientos básicos sobre las particularidades en el control de robots manipuladores reales adicionales a los adquiridos en el MUII.

1.2 Antecedentes:

El presente trabajo se ha elaborado partiendo de los siguientes antecedentes:

- En primer lugar, los conocimientos adquiridos durante el MUII y el Grado en Ingeniería en Tecnologías Industriales (GITI) sobre algunas de las principales bases de la robótica, de los cuales concretamente se ha utilizado la metodología Denavit-Hartenberg para la resolución del problema cinemático directo. Cabe señalar, no obstante, que los conocimientos sobre control cinemático y *visual servoing* utilizados han sido adquiridos o profundizados al margen de las asignaturas impartidas en ambas titulaciones, particularmente durante una beca de colaboración realizada por parte del autor del presente trabajo.
- En segundo lugar, en el campo de la simulación aplicada a la robótica se ha partido del conjunto de herramientas de *Matlab* desarrollados por Peter Corke (*Robotics, Vision and Control toolbox*) de cara a la resolución de la cinemática básica y la simulación gráfica de robots, y los archivos gráficos utilizados son los contenidos en la *toolbox* ARTE (*A Robotics Toolbox for Education*).
- En tercer lugar, en el campo de la experimentación real, se ha contado para el presente trabajo con un robot articulado industrial, como se tratará en apartados posteriores, completamente funcional y preparado para ser controlado mediante algoritmos de servo control visual implementados sobre la arquitectura ROS (*Robotics Operating System*) gracias al trabajo de investigación y académico desarrollado en el Instituto de Diseño y Fabricación (IDF).

CAPÍTULO 2. DESARROLLO TEÓRICO

2.1 Los robots:

De acuerdo con la norma ISO 8373:2012, un robot se define como:

"*Mecanismo programable accionado sobre al menos dos ejes con un grado de autonomía, que se desplaza en su entorno, para ejecutar tareas previstas*" [1].

De modo que la robótica es precisamente la "*ciencia y práctica de diseñar, manufacturar y aplicar robots*" [1].

2.2 Los robots industriales:

Siguiendo con esta misma norma, es conveniente señalar que no todos los robots son clasificables como robots industriales, siendo la definición de robot industrial:

"*Manipulador multifuncional, reprogramable, automáticamente controlado, programable en tres o más ejes, que puede ser o bien fijado en un sitio o móvil para su utilización en aplicaciones de automatización industrial*" [1].

Dentro de esta clasificación destacan por su extensión y variedad de aplicaciones los robots seriales, caracterizados por estar constituidos por "*una secuencia de uniones y articulaciones que comienza en una base y termina en un efector final*" [2] conocidos como brazos robóticos, particularmente aquellos que son robots articulados, que pueden definirse como aquellos robots "*cuyo brazo tiene al menos tres ejes rotativos*" (IFR, 2016) [3].

Figura 1-Ejemplo de brazo robot industrial

2.3 Control cinemático de robots [4] [5]:

El control cinemático de robots consiste esencialmente en determinar las trayectorias a seguir por las articulaciones del robot para lograr seguir una referencia cumpliendo una serie de especificaciones.

Para hacer posible el control cinemático, es necesario conocer la relación entre el movimiento de las articulaciones y el movimiento del efector final del robot.

Por un lado, el problema cinemático directo consiste en la obtención de la posición y orientación del efector final del robot a partir de la configuración de sus articulaciones. Este problema puede resolverse de forma sistemática gracias al método Denavit-Hartenberg, que se explicará sucintamente más adelante, y que permite obtener un modelo de la cinemática directa del robot de forma independiente a su configuración.

Por otro lado, el problema cinemático inverso consiste en la obtención de la configuración de las articulaciones del robot a partir de la posición y orientación de su efector final. Este problema no puede ser resuelto de forma sistemática, dado que la solución a este problema no es única (una misma posición y orientación puede obtenerse con diferentes valores de las coordenadas articulares) y la forma de obtener las ecuaciones depende de la configuración de cada robot en particular.

Si bien existen métodos iterativos para obtener la solución al problema cinemático inverso, son costosos computacionalmente, lo cual dificulta el control a tiempo real, su convergencia no está garantizada y sigue existiendo el problema de la no unicidad de la solución.

Como alternativa a la cinemática inversa, es posible, como se tratará en detalle más adelante, relacionar la velocidad o aceleración del efector final y las de las articulaciones del robot, estableciendo un control de velocidad o de aceleración que tan solo requiera resolver el problema cinemático directo y corregir el error entre la posición y la referencia sin necesidad de obtener las ecuaciones de la cinemática inversa.

2.3.1 Problema cinemático directo. Metodología de Denavit-Hartenberg [5]:

La relación entre dos puntos del robot viene dada por la relación entre los sistemas de referencia asociados a ellos. Situando sistemas de referencia en cada articulación del robot, en su base y en el efector final, es entonces posible relacionar la posición y orientación de cada uno de esos puntos del robot entre sí a cada instante.

Una matriz de transformación homogénea tiene la forma:

$$T = \begin{pmatrix} Matriz\ de\ Rotación_{3x3} & Vector\ de\ traslación_{3x1} \\ Vector\ de\ perspectiva_{1x3} & Término\ de\ escalado_{1x1} \end{pmatrix}$$

Cuando se trata tan solo de rotaciones y traslaciones, como es el caso, el vector de perspectiva se anula y el término de escalado toma el valor unidad.

La metodología Denavit-Hartenberg plantea un esquema sistemático de asignación de los sistemas de referencia a cada articulación del robot y de definición de parámetros geométricos del robot, de tal manera que la matriz de transformación que relaciona cada articulación con la siguiente tiene un modelo matemático único dependiente de dichos parámetros, precisamente conocidos como parámetros de Denavit-Hartenberg.

Existe una relación estrecha entre los parámetros de Denavit-Hartenberg y la manera de asignar los sistemas de referencia, dado que el punto de partida de la metodología es concebir la transformación entre un sistema de referencia (i-1) y el siguiente (i) como una combinación de cuatro transformaciones:

1-Una rotación de ángulo θ_i (también referido a menudo como q_i) alrededor del eje Z_{i-1} y medido desde x_{i-1} hasta x_i. El parámetro q será el variable cuando la articulación sea rotativa.

2-Una traslación de una distancia d_i a lo largo del eje Z_{i-1}. El parámetro d será variable cuando la articulación sea prismática. En caso de articulación prismática, es común encontrar esta distancia indicada también como q_i.

3-Una traslación de una distancia a_i a lo largo del eje X_i. Este parámetro siempre será fijo.

4-Una rotación de ángulo α_i alrededor del eje X_i, medido desde Z_{i-1} hasta Z_i. Este parámetro siempre será fijo.

Cabe señalar que, de acuerdo a la notación Denavit-Hartenberg, se numeran los eslabones desde 0 y las articulaciones desde 1. Desde i=0 hasta i=n-1 los sistemas de referencia $\{S_i\}$ son solidarios al eslabón i, y se asignan según la articulación i+1. Así, los parámetros de Denavit-Hartenberg iniciarán la cuenta desde 1 y los sistemas de referencia la iniciarán desde 0, siendo i=0 la base e i=n el extremo del robot.

De manera coherente a este planteamiento, es necesario asignar los sistemas de referencia de tal manera que el eje Z_i sea el eje de rotación o traslación de la articulación i+1 (según sea rotativa o prismática), el eje X_i en la línea perpendicular común a Z_{i-1} y Z_i y el eje Y_i en último lugar para asegurar que el sistema de referencia es tri-rectangular dextrógiro.

El origen de coordenadas de cada uno de estos sistemas se asignará de la siguiente manera: en el caso i=0 (la base) el origen está en cualquier punto del eje Z_0, y desde i=1 hasta i=n-1, el origen está en la articulación i+1 si $Z_i||Z_{i-1}$ o bien en el mismo punto que el origen de i-1 si Z_i intersecta con Z_{i-1}.

En cuanto al sistema de referencia del efector final, este debe tener su eje Z paralelo al de la articulación anterior y los ejes X e Y situados con el mismo criterio que en el caso de las articulaciones.

A modo de ejemplo, se muestra en la figura 2 una asignación de sistemas de referencia de acuerdo a la notación Denavit-Hartenberg:

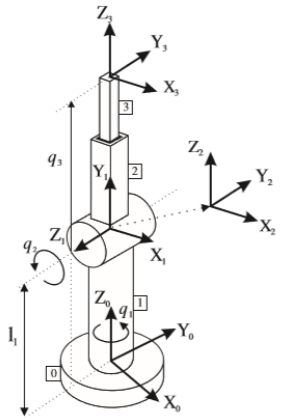


Figura 2 [5]-Ejemplo de notación Denavit-Hartenberg

Así, la matriz de Denavit-Hartenberg que relaciona el sistema i-1 con el sistema i es:

$$A_i^{i-1} = \begin{bmatrix} \cos(q_i) & -\cos(\alpha_i)\cdot\sin(q_i) & \sin(\alpha_i)\cdot\sin(q_i) & a_i\cdot\cos(q_i) \\ \sin(q_i) & \cos(\alpha_i)\cdot\cos(q_i) & -\sin(\alpha_i)\cdot\cos(q_i) & a_i\cdot\sin(q_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Con lo que la matriz Denavit-Hartenberg que da la relación entre la base y el efector final del robot es:

$$bMe = A_n^0 = A_1^0 \cdot \dots A_i^{i-1} \dots \cdot A_n^{n-1}$$

Y en caso de que la base del robot tenga un sistema de referencia diferente al sistema de referencia global (mundo) o haya algún elemento rígido a continuación del efector final del robot (como es el caso de una cámara, que se tratará más adelante), tan solo habrá que multiplicar por la izquierda o por la derecha con una matriz de transformación homogénea constante.

2.3.2 Relación entre velocidad del efector final y velocidades de las articulaciones. Modelo cinemático diferencial:

Como se ha indicado anteriormente, es posible conocer la relación entre el movimiento del elemento terminal del robot y su movimiento articular; esta relación es conocida como "modelo cinemático diferencial", y viene dada por la matriz Jacobiana.

2.3.2.1 Explicación conceptual. Jacobiana analítica [5]:

Si bien para la implementación del modelo cinemático diferencial se utiliza la Jacobiana geométrica obtenida por cálculo numérico, es interesante para entender conceptualmente el modelo cinemático diferencial tratar la Jacobiana analítica. La relación expresada por la Jacobiana analítica puede observarse en la figura 1:

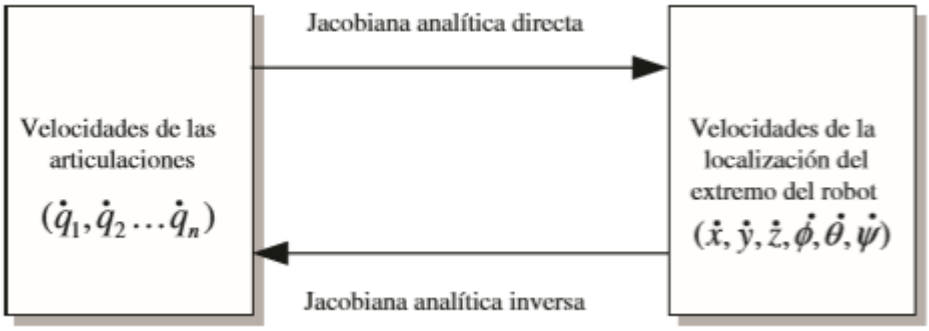


Figura 3 [5]-Jacobiana analítica

La Jacobiana analítica se obtiene derivando el modelo cinemático directo que relaciona cada variable de posición y orientación del elemento terminal del robot con cada articulación, tal y como se muestra a continuación:

$$x = f_x(q_1, q_2, \dots, q_n)$$

$$\dot{x} = \sum_{1}^{n} \frac{\partial f_x}{\partial q_i} \dot{q}_i$$

Generalizando este procedimiento para el resto de variables de posición y orientación del elemento terminal y expresando el resultado de forma matricial, se obtiene la relación entre las velocidades del extremo del robot y las velocidades articulares (en este caso se ha utilizado "q" para referenciar la variable de cada articulación, independientemente de si es un ángulo o una distancia):

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{bmatrix} \frac{\partial f_x}{\partial q_1} & \cdots & \frac{\partial f_x}{\partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_\psi}{\partial q_1} & \cdots & \frac{\partial f_\psi}{\partial q_n} \end{bmatrix} \begin{pmatrix} \dot{q_1} \\ \dot{q_2} \\ \cdots \\ \cdots \\ \dot{q_n} \end{pmatrix}$$

Donde la matriz Jacobiana analítica es precisamente la matriz de derivadas parciales de las funciones del modelo cinemático directo con respecto a las variables de cada una de las articulaciones.

El problema que presenta esta Jacobiana analítica es que, precisamente por ser una derivada analítica, no es eficiente implementarla computacionalmente. Por ello, se recurre a la obtención numérica de la Jacobiana geométrica.

2.3.2.2 Implementación. Jacobiana geométrica [5]:

La Jacobiana geométrica expresa la relación entre las velocidades articulares y las velocidades lineales y angulares del elemento terminal del robot. Analíticamente, la Jacobiana geométrica se calcula a partir de la matriz de Denavit-Hartenberg (matriz DH, de ahora en adelante) entre la base del robot y su extremo, con un resultado igual al de la Jacobiana analítica en los elementos de posición y diferente, aunque sencillo de relacionar con el caso analítico, en los elementos de orientación [6]. No obstante, dado que la Jacobiana geométrica obtenida por métodos analíticos presenta la misma problemática para su implementación que la analítica, se pasa directamente a explicar su obtención mediante métodos numéricos.

Partiendo de las matrices DH ${}^{i-1}A_i$ que constituyen el modelo cinemático directo del robot, es posible obtener la matriz Jacobiana geométrica no solo en términos analíticos, sino también para cada instante concreto.

Con dichas matrices, gracias al método de propagación de las velocidades, las columnas de la Jacobiana geométrica J pueden ser obtenidas tal y como se expone a continuación:

A partir de lo expuesto en el apartado 2.3.1 del presente trabajo (Problema cinemático directo. Metodología de Denavit-Hartenberg) es sencillo comprobar que, dada la matriz DH desde la base hasta el eslabón i, ${}^{0}A_i$, la expresión de los vectores de los tres ejes de coordenadas del sistema $\{S_i\}$ expresados con respecto al sistema de la base $\{S_0\}$ viene dada por las tres columnas del bloque de rotación de la matriz DH. Así, los ejes X, Y y Z del sistema i respecto de la base serán las columnas primera, segunda y tercera del bloque de rotación de la matriz ${}^{0}A_i$, respectivamente, es decir, los vectores $\vec{n}$, $\vec{o}$ y $\vec{a}$ de la matriz tal y como se muestra a continuación:

$$A_i^0 = \begin{pmatrix} \vec{n} & \vec{o} & \vec{a} & \vec{p} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Para caracterizar la rotación y traslación que aporta cada articulación al efector final del robot, es posible utilizar, tal y como se verá más adelante, el eje que caracteriza a cada articulación, es decir, el eje Z (recordando que según la notación DH, el eje Z_i se corresponde con la articulación i+1).

En el caso de las articulaciones de rotación, las de interés para el presente trabajo, la rotación respecto del eje de la articulación (q_i entorno a Z_{i-1}) se aporta íntegramente al elemento terminal, de manera que tan solo será necesario expresar la rotación q_i con respecto al sistema de referencia de la base. O expresado de forma más simple, la velocidad angular que aporta cada articulación es:

$$\overrightarrow{\omega_\iota} = \overrightarrow{z_{\iota-1}} \cdot \dot{q}_\iota$$

Y la velocidad lineal en el elemento terminal se calculará a partir del efecto de dicha rotación teniendo en cuenta la distancia que separa el punto donde tiene lugar la rotación (articulación i) del elemento terminal. Este segmento es el valor del vector $\vec{p}$ de la matriz DH ${}^{i-1}A_n$. Por tanto, la velocidad lineal aportada por cada articulación es:

$$\overrightarrow{v_\iota} = \overrightarrow{z_{\iota-1}} \times \overrightarrow{p_n^{\iota-1}} \cdot \dot{q}_\iota$$

En el caso de una articulación prismática, es la traslación lo que se aporta directamente al elemento terminal, y no se aporta rotación alguna, así que se cumple:

$$\overrightarrow{v_\iota} = \overrightarrow{z_{\iota-1}} \cdot \dot{q}_\iota$$

$$\overrightarrow{\omega_\iota} = \vec{0}$$

De modo que cada columna de la matriz Jacobiana geométrica calculada numéricamente es:

$$J_i = \begin{bmatrix} \overrightarrow{z_{\iota-1}} \times \overrightarrow{p_n^{\iota-1}} \\ \overrightarrow{z_{\iota-1}} \end{bmatrix}$$ en una articulación "i" rotativa

$$J_i = \begin{bmatrix} \overrightarrow{z_{\iota-1}} \\ \vec{0} \end{bmatrix}$$ en una articulación "i" prismática

Y la matriz Jacobiana geométrica resulta:

$$J = [J_1\, J_2 \ldots J_n]$$

2.3.2.3 Aplicación a la robótica. Pseudo-inversa y configuraciones singulares:

Hasta ahora, se ha resuelto de nuevo el problema cinemático directo de forma diferencial, pero en robótica, como se ha comentado anteriormente, resulta de interés conocer qué configuración de las articulaciones es necesaria para conseguir una cierta posición y orientación del efector final.

Para ello, es necesario invertir la matriz Jacobiana. La inversa de la matriz J solo se puede obtener si esta es cuadrada, es decir, si tiene 6 articulaciones, caso en que será 6x6. No obstante, el problema puede resolverse de forma general recurriendo a la pseudo-inversa.

Particularmente en el presente trabajo se recurre a la pseudo-inversa de Moore-Penrose, dado que es una pseudo-inversa que engloba a todas las demás inversas generalizadas, tiene solución única y minimiza por mínimos cuadrados la norma euclídea del error de aproximación en caso de no ser J una matriz invertible [7].

Siendo J^+ la matriz pseudo-inversa de J y T_s el periodo de muestreo, y k un instante tal que $t=T_s \cdot k$, se cumple:

$$\vec{q}(k) = J^+(k) \cdot \begin{bmatrix} \vec{v}(k) \\ \vec{\omega}(k) \end{bmatrix} \quad \{2.3.2.1\}$$

$$\vec{q}(k) = \dot{\vec{q}}(k) \cdot T_s + \vec{q}(k-1) \quad \{2.3.2.2\}$$

Lo que significa que es posible calcular a cada instante el valor de la configuración de las articulaciones a partir de la velocidad del elemento terminal del robot.

La necesidad de integrar la velocidad numéricamente para obtener la posición se justifica teniendo en cuenta que muchos robots industriales, particularmente el robot KUKA utilizado en el apartado de experimentación real del presente Trabajo Fin de Máster, tan solo cuentan con la posibilidad de controlar su posición angular, no permiten introducir como variable manipulada la velocidad o la aceleración.

Por otro lado, una cuestión a tener en cuenta a la hora de aplicar la matriz Jacobiana para resolver la cinemática diferencial es la de las configuraciones singulares: para posiciones que quedan fuera del alcance del robot o configuraciones con dos o más ejes superpuestos, el Jacobiano del robot se anula, lo que significa que una velocidad del elemento terminal muy pequeña implicaría movimientos de las articulaciones infinitamente rápidos, lo cual es inasumible para el control [5].

2.3.3 Control de velocidad:

El control de velocidad proporcional deriva directamente de lo expuesto en los apartados previos, y es, como se tratará más adelante, la base para las leyes de control en *Visual Servoing*.

Se trata de un control proporcional para corregir el error de posición (e) respecto a una referencia de posición y orientación (p_{ref}) indicada por el usuario. Ello implica que la velocidad deseada del elemento terminal del robot será proporcional al error:

$$v = \dot{p} = K_p \cdot e \quad \{2.3.3.1\}$$

Siendo K_p una ganancia proporcional escalar positiva.

Dicha ley de control de la velocidad garantiza que el error disminuye de forma exponencial, dado que el elemento terminal se moverá para corregir el error de posición más rápidamente cuanto mayor sea el error, cumpliéndose:

$$\dot{e} = -v \;\rightarrow\; \dot{e} = -K_p \cdot e \;\rightarrow\; \dot{e} + K_p \cdot e = 0, \qquad EDO\ de\ primer\ orden \rightarrow$$

$$\rightarrow\; e = \exp(-K_p \cdot t)$$

Además, como se deduce de las ecuaciones anteriores, cuanto mayor sea la ganancia proporcional, más agresiva será la corrección del error y más rápida la convergencia.

Para construir la ley de control completa para una referencia invariante en el tiempo, es necesario incluir la ecuación {2.3.3.1} en la ecuación del apartado anterior, {2.3.2.1}, quedando:

$$\dot{q}(k) = J^{+}(k) \cdot \left(K_p \cdot e(k)\right) = J^{+}(k) \cdot \left(K_p \cdot \left(p_{ref} - p(k)\right)\right) \quad \{2.3.3.2\}$$

Y la posición de las articulaciones en el instante siguiente (k+1) vendría dada por la aplicación de la expresión {2.3.2.2} del apartado anterior.

En la figura 4, que se muestra a continuación, se observa el funcionamiento de este control de velocidad para una referencia constante de posición y orientación del elemento terminal del robot de forma más clara.

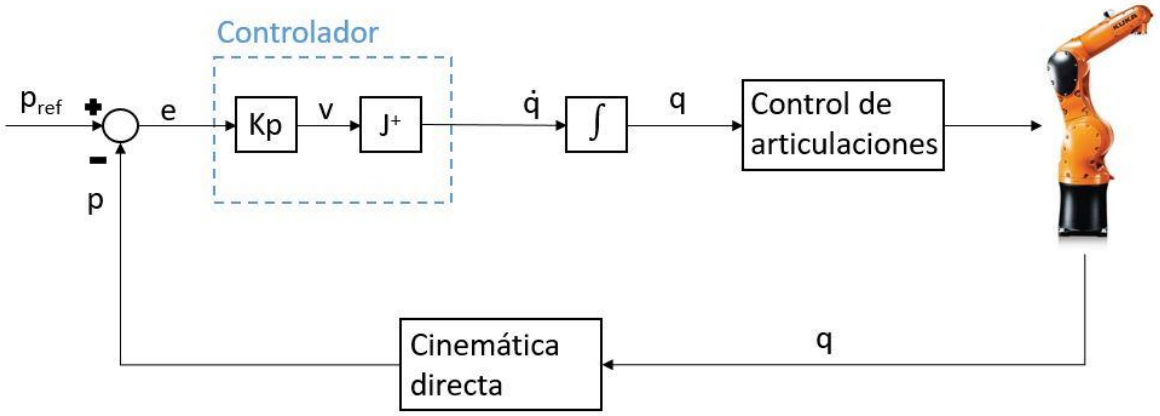


Figura 4-Diagrama de bloques del control de velocidad

Esta ley de control permite el posicionado del efector final del robot en un punto fijo del espacio, pero no es exacto para el seguimiento de trayectorias, cuando ésta es una aplicación de interés tanto si el seguimiento de una determinada trayectoria es el objetivo en sí mismo, como si el objetivo es llegar a un determinado punto del espacio, pero se requiere llegar a ese punto siguiendo una determinada trayectoria (por ejemplo, rectilínea).

Para el caso en que la referencia indicada por el usuario no es constante en el tiempo, basta con ampliar la ley de control incorporando la velocidad con la que varía la referencia en cada instante como elemento de pre alimentación o *feedforward*. Añadiendo la derivada de la referencia con respecto al tiempo $\dot{p_{ref}}$ a la expresión de la ley de control {2.3.3.2}, la ley de control con *tracking* de una trayectoria resulta:

$$\dot{q}(k) = J^{+}(k) \cdot \left(K_p \cdot e(k)\right) = J^{+}(k) \cdot \left(\dot{p_{ref}} + K_p \cdot \left(p_{ref}(k) - p(k)\right)\right) \quad \{2.3.3.3\}$$

De esta manera, el controlador del robot tan solo tiene que encargarse de corregir el error de posición dando por descontado el desplazamiento de la referencia.

Dado que se trata de trayectorias de referencia definidas por el usuario, $\dot{p_{ref}}$ es conocida, y puede obtenerse para cada instante de forma iterativa con la siguiente expresión:

$$\dot{p_{ref}} = \frac{p_{ref}(k+1) - p_{ref}(k)}{T_s} \quad \{2.3.3.4\}$$

El diagrama de bloques del control resultante es el mostrado en la figura 5, a continuación:

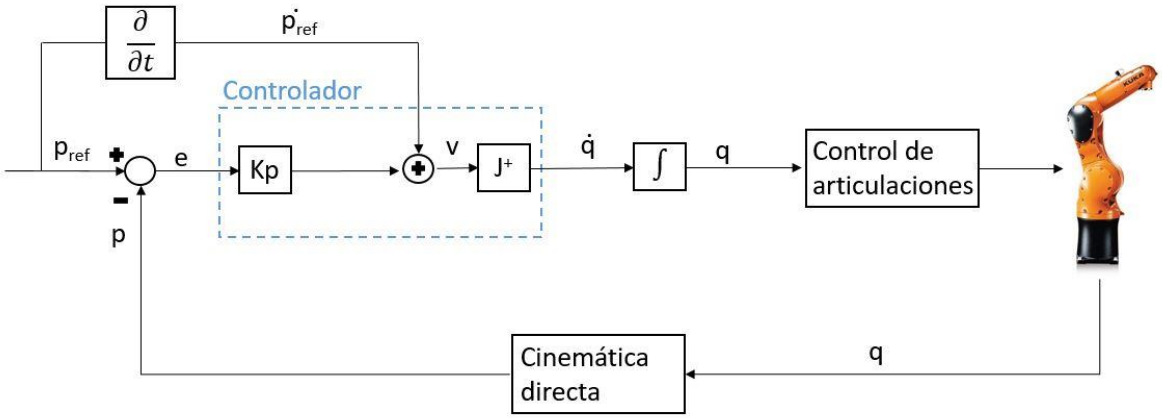


Figura 5-Diagrama de bloques del control de velocidad con seguimiento de trayectoria

2.3.4 Control de aceleración:

Con un razonamiento similar al seguido en los apartados anteriores, puede lograrse un control que posicione el elemento terminal en una determinada posición y orientación fija o bien siga una referencia móvil indicada por el usuario, pero controlando la aceleración del elemento terminal, de tal manera que se eviten sobreoscilaciones y movimientos bruscos.

Para ello, en primer lugar, es necesario obtener la relación entre las aceleraciones de las articulaciones del robot y la aceleración de su efector final, derivando la relación entre velocidades:

$$v = J \cdot \dot{q} \;\rightarrow\; a = J \cdot \ddot{q} + \dot{J} \cdot \dot{q} \qquad \{2.3.4.1\}$$

Donde "a" es la aceleración del elemento terminal del robot, que, en el caso más general del control de aceleración, incluyendo la pre alimentación o *feedforward* de la aceleración de la referencia, se define como:

$$a = \ddot{p} = \ddot{p_{ref}} + K_v \cdot \dot{e} + K_p \cdot e \qquad \{2.3.4.2\}$$

$$donde\ \ \dot{e} = e_v = \dot{p_{ref}} - \dot{p}$$

Es decir, el controlador mueve el elemento terminal tratando de corregir el error con una ganancia proporcional (K_p) a la vez que intenta corregir la diferencia de velocidad con respecto a la referencia con otro término proporcional (K_v) y se pre alimenta la aceleración de la referencia ($\ddot{p_{ref}}$).

En este caso, la evolución del error viene dada por una Ecuación Diferencial Ordinaria de segundo orden, como se muestra en la expresión a continuación, despreciando el término de la aceleración de la referencia para facilitar su lectura:

$$\ddot{e} + K_v \cdot \dot{e} + K_p \cdot e = 0, \quad EDO\ de\ 2º\ orden$$

Al ser introducidos por el usuario, los datos referidos a la trayectoria de la referencia son conocidos, mientras que la velocidad del elemento terminal del robot $\dot{p}$ es el resultado de la aceleración adquirida en el instante anterior y, por tanto, se mantiene constante hasta el instante siguiente. Así, la velocidad

de la referencia se puede obtener numéricamente como en la expresión {2.3.3.4} y el resto de variables también pueden calcularse a cada instante de forma iterativa, según las siguientes expresiones:

$$\ddot{p_{ref}} = \frac{\dot{p_{ref}}(k+1) - \dot{p_{ref}}(k)}{T_s}$$

$$\dot{p} = \frac{p(k) - p(k-1)}{T_s}$$

Para construir la ley de control de aceleración, hay que despejar $\ddot{q}$ en la ecuación {2.3.4.1} y sustituir $\ddot{p}$ según la expresión {2.3.4.2}:

$$\ddot{q}(k) = J^{+} \cdot \left(\ddot{p}(k) - \dot{J} \cdot \dot{q}(k-1)\right) \rightarrow$$

$$\rightarrow \ddot{q}(k) = J^{+} \cdot \left(\ddot{p_{ref}}(k) + K_v \cdot e_v + K_p \cdot e - \dot{J} \cdot \dot{q}(k-1)\right) \quad \{2.3.4.3\}$$

Nótese que en la aceleración de las articulaciones en un instante K ($\ddot{q}(k)$), influye la velocidad de las articulaciones en el instante anterior ($\dot{q}(k-1)$).

Tal y como ocurría en el caso del control de velocidad, es necesario integrar el resultado del control, en este caso dos veces, para obtener la posición articular de referencia para el control de articulaciones del robot:

$$\dot{q}(k) = \dot{q}(k-1) + \ddot{q}(k) \cdot T_s$$

$$q(k) = q(k-1) + \dot{q}(k-1) \cdot T_s + \frac{1}{2}{T_s}^2 \cdot \ddot{q}(k) \quad \{2.3.4.4\}$$

Nótese que, en este caso, a diferencia de la integración en el control de velocidad (expresión {2.3.3.2}), el objetivo es realizar un cierto incremento en las articulaciones que se corresponda con la aceleración deseada en las articulaciones según el control; es decir, la velocidad de las articulaciones ya no es la acción de control, sino una consecuencia de la acción de control. Por ello, se utiliza para el cálculo de la referencia de q la velocidad articular en el instante anterior ($\dot{q}(k-1)$), porque dicha velocidad tiene un efecto sobre la posición actual, igual que lo tiene la posición articular de partida, mientras que la velocidad provocada por la aceleración resultante del control ($\dot{q}(k)$) se dejará sentir en el instante siguiente.

El conjunto del control puede observarse de forma simplificada en el siguiente diagrama de bloques, en la figura 6:

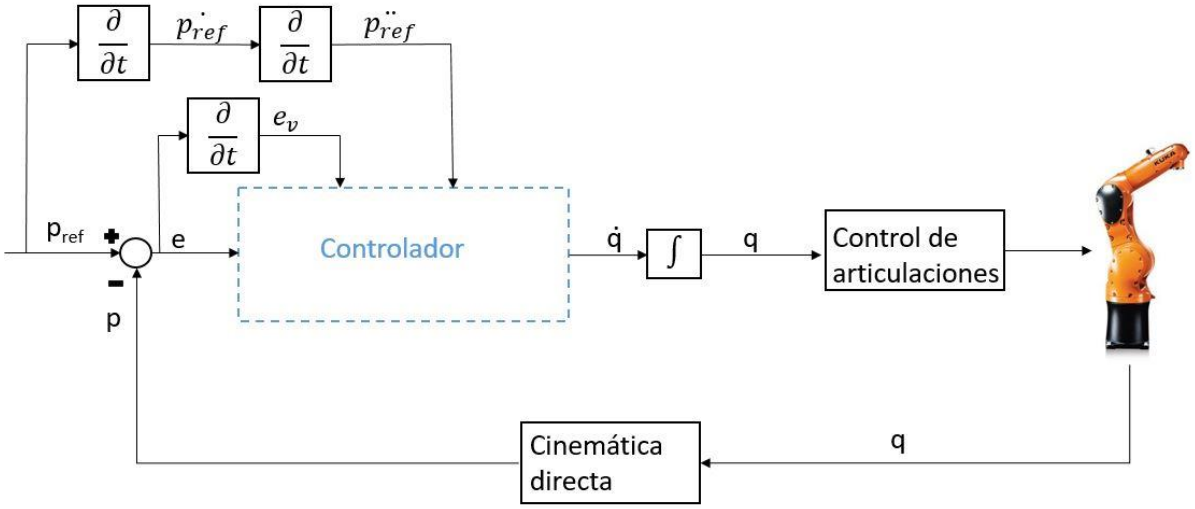


Figura 6-Diagrama de bloques simplificado del control de aceleración

Si bien el control de aceleración puede utilizarse también para posicionar referencias fijas, es necesario tener en consideración que, dado que en dicho caso $\dot{p}_{ref} = 0$, el control proporcional de velocidad tenderá a reducir la velocidad del elemento terminal para corregir el error de velocidad, lo cual da lugar a un desempeño de la tarea de posicionado menos fluido que en el control de velocidad visto anteriormente.

2.4 *Visual servoing:*

El *Visual Servoing* o servo control visual se define como "el uso de datos de visión artificial para controlar el movimiento de un robot" [8].

Para el *Visual Servoing* se recurre comúnmente a un control cinemático de velocidad muy similar al ya explicado en el apartado 2.3.3, con la salvedad de que en dicho control *Visual Servoing* es posicionar o hacer un seguimiento de un objeto de interés o *target* fijo o móvil, obteniendo para ello la información necesaria sobre el *target* a través de la toma de imágenes mediante un sistema y técnicas de visión artificial.

Dentro de la disciplina del *Visual Servoing* existen dos clasificaciones de interés para el presente trabajo:

- Según la configuración de la cámara empleada: [9]
    - Eye-in-hand: La cámara se encuentra conectada al efector final del robot y se mueve solidaria a éste, observando la posición relativa del *target* respecto al robot.
    - Eye-to-hand: La cámara se encuentra fija en algún punto del mundo, enfocando tanto al efector final del robot como al *target*, trabajando así con las posiciones absolutas de ambos respecto de un sistema de coordenadas mundo.

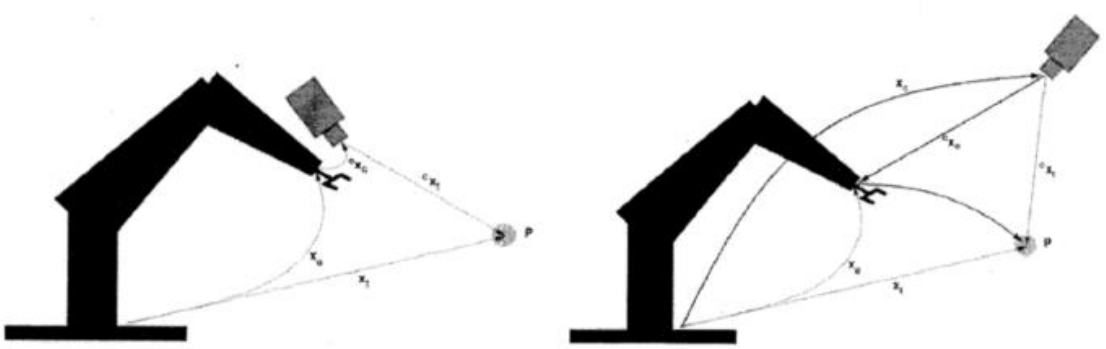


Figura 7 [9]-Configuración eye-in-hand (izq.) y configuración eye-to-hand (der.)

- Según la metodología empleada para obtener las características visuales de interés (sobre este tema se volverá más adelante) para calcular el error a corregir por el controlador, se encuentran dos categorías clásicas: [8]
    - *Image Based Visual Servoing (IBVS)*: Servo control visual basado en imágenes, donde las características con las que se trabaja están disponibles sobre la imagen.
    - *Position Based Visual Servoing (PBVS)*: Servo control visual basado en posiciones, donde las características utilizadas para calcular el error son posiciones estimadas a partir de la imagen.

Además, tanto en la configuración de la cámara como en la metodología empleada existen enfoques más complejos (diversas metodologías *Hybrid Visual Servoing*, combinación de varias cámaras) que derivan de la clasificación planteada anteriormente, pero que escapan al ámbito del presente Trabajo Fin de Máster [8].

2.4.1 Funcionamiento general del *Visual Servoing* (*eye-in-hand*): [8]

El objetivo de los algoritmos de control de *Visual Servoing* es reducir el error dado por:

$$e = s - s^* \quad \{2.4.1\}$$

Donde *s* es un vector de características visuales (características obtenidas a partir de la imagen) y *s** es la referencia. Nótese que, a diferencia de los controles tratados en el apartado 2.3, en este caso el error se calcula como señal medida menos referencia, y no como referencia menos señal medida. Esto afecta, como se verá más adelante, al criterio de signos de la ley de control.

Para obtener *s* es necesario partir de:

- Un conjunto de medidas de la imagen, como en el caso de este trabajo, coordenadas de puntos de interés del *target*.
- Un conjunto de parámetros que aportan información adicional sobre el sistema, como puedan ser los parámetros intrínsecos de la cámara.

Independientemente de cómo se determine finalmente *s*, que es lo que define la metodología de *Visual Servoing* que se está utilizando (IBVS, PBVS u otros), puede obtenerse la ley de control expresada en términos comunes a los diferentes métodos.

Cabe señalar que en el presente trabajo se han desarrollado los controles para el caso *eye-in-hand*, lo cual afecta también al desarrollo teórico de aquí en adelante.

2.4.1.1 Aproximación al problema mediante el caso de posicionado mediante *Free Camera*: [8]

Con tal de hacer una primera aproximación al control de velocidad *Visual Servoing*, es de interés partir desde un caso ficticio pero genérico en el que se trata cómo se controlaría el movimiento de una cámara capaz de moverse por sí sola con 6 grados de libertad (todos los giros y desplazamientos), conocido como modelo *free camera*, para después añadir la restricción impuesta por el robot y convertir dicho movimiento de la cámara libre en el movimiento deseado de una cámara situada en el extremo del robot, a conseguir mediante el movimiento de sus articulaciones.

Además, inicialmente se realizarán los desarrollos suponiendo que la tarea objetivo del control es el posicionado, es decir, el seguimiento de una referencia invariante en el tiempo.

De forma intuitiva es fácil concluir que existe una relación entre el movimiento de la cámara y el vector de características visuales, puesto que la imagen es una proyección bidimensional del mundo tridimensional sobre el plano de la cámara:

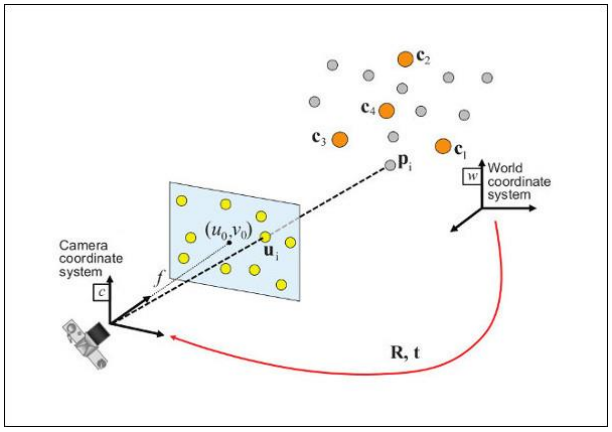


Figura 8 [10]- Proyección de un objeto sobre el plano imagen

Tratando este problema de forma sistemática, se halla que dadas la velocidad de la cámara (lineal y angular) v_c y L_s la llamada *matriz de interacción* o *Jacobiano de propiedades*, la relación entre la variación del vector de características visuales y el movimiento de la cámara es:

$$\dot{s} = L_s \cdot v_c \quad \{2.4.1.1\}$$

La matriz de interacción se calcula de una u otra forma dependiendo de la metodología de *Visual Servoing* utilizada, como se tratará en detalle en posteriores apartados. En los casos de aplicación reales, no puede ser obtenida de forma exacta, de modo que debe ser estimada (ella o su pseudo-inversa). La estimación de la matriz de interacción no es objeto del presente trabajo académico puesto que su implementación se ha realizado en el caso de simulación, caso ideal en que sí puede ser conocida con exactitud, si bien el desarrollo de dicha estimación puede encontrarse en la fuente principal de este apartado del presente trabajo, el capítulo sobre *Visual Servoing* del *Springer Handbook of Robotics* [8].

Tal y como se define la relación entre el error *e* y el vector de características *s* (relación {2.4.1}), se comprueba que la relación del movimiento de la cámara con la variación del error es la misma:

$$\dot{e} = L_s \cdot v_c \quad \{2.4.1.2\}$$

A partir de esta última relación, es posible definir un control de la velocidad de la cámara proporcional al error, de forma análoga a la ecuación {2.3.3.1}, calculando la inversa generalizada de Moore-Penrose de L_s para hacerlo posible:

$$v_c = -\lambda \cdot L_s^+ \cdot e = -\lambda \cdot L_s^+ \cdot (s - s^*) \quad \{2.4.1.3\}$$

Donde λ es la ganancia proporcional análoga a K_p de la expresión {2.3.3.1}. Nótese que el signo negativo difiere de la expresión {2.3.3.1} y se debe a la inversión del criterio de signo del error tal y como ya se ha comentado.

Con este control sigue siendo válido el razonamiento realizado en el apartado 2.3.3 (control de velocidad), por el cual se concluye que el error disminuirá de forma exponencial con el control proporcional de la velocidad.

2.4.1.2 *Tracking* de un *target* en movimiento: [8]

En esencia, el problema del *tracking* de un cuerpo en movimiento puede resolverse análogamente al caso del control de velocidad del apartado 2.3.3, incorporando un término de pre alimentación o *feedforward* de la variación del error.

Pasando al caso concreto de interés en este apartado, es necesario incorporar a la variación del error, recogida en la expresión {2.4.1.2} para tareas de posicionado, un término que introduzca la variación del error debida al movimiento del *target*, término que en este caso es $\frac{\partial e}{\partial t}$:

$$\dot{e} = L_s \cdot v_c + \frac{\partial e}{\partial t} \quad \{2.4.1.4\}$$

Nótese que, a diferencia del control de velocidad del apartado 2.3.3, en este caso el movimiento del *target* generalmente no es conocido previamente, dado que la información sobre el *target*, incluyendo la información necesaria para su posicionado o seguimiento, se obtiene a partir de técnicas de visión artificial, no se introduce al controlador por parte del usuario.

Por ello, es necesario realizar una estimación de $\frac{\partial e}{\partial t}$ a partir de la información disponible. Una aproximación válida para el seguimiento de un *target* con un movimiento totalmente desconocido es:

$$\frac{\partial e}{\partial t} \approx \frac{\widehat{\partial e}}{\partial t} = \frac{e(k) - e(k-1)}{T_s} - L_s(k) \cdot v_c(k-1) \quad \{2.4.1.5\}$$

Lo que significa que se estima el efecto del movimiento del *target* sobre la variación del error despejando la propia expresión de partida ({2.4.1.4}) y utilizando datos de instantes anteriores.

Esta estimación puede dar lugar a acciones de control demasiado agresivas en caso de que se produzcan picos en la variación del error entre dos instantes consecutivos, de forma que es conveniente en experimentación real limitar su efecto, ya sea mediante un filtro o una ganancia constante.

Siguiendo un razonamiento análogo al de la ley de control de velocidad con *feedforward* para el seguimiento de trayectorias de referencia, la ley de control para un caso genérico *free camera* con *tracking* es:

$$v_c = -\lambda \cdot L_s^+ \cdot e - L_s^+ \widehat{\frac{\partial e}{\partial t}} \qquad \{2.4.1.6\}$$

2.4.1.3 Implementación en robots: [8]

El principal cambio que entra en juego al pasar de un caso *free camera* a un caso real de un robot, es decir, al pasar de un movimiento libre en el espacio cartesiano a un movimiento en el espacio de las articulaciones del robot, es en la interacción entre características visuales y movimiento.

Para realizar una aproximación gradual a este cambio, puede partirse del estudio de la relación entre el movimiento de las articulaciones del robot y el movimiento de la cámara, suponiendo que esta se encuentra fijada al extremo del robot (caso *eye-in-hand*):

$$v_c = X_n^c \cdot J_n \cdot \dot{q} \quad \{2.4.1.7\}$$

Donde:

- J_n es el la matriz Jacobiana geométrica del robot respecto del elemento terminal; es decir en lugar de definir la relación entre movimientos de articulaciones y movimientos del elemento terminal expresándolos respecto de la base, como se hacía en el apartado de control cinemático (2.3.2), se expresan respecto del sistema de referencia del elemento terminal del robot.
- X_n^c es la matriz de transformación espacial de movimiento, que relaciona la posición del efector final del robot con la posición de la cámara, siendo una matriz constante en el caso en que la unión entre cámara y efector final del robot sea rígida, como ocurren en el presente trabajo académico.

Esta matriz X_n^c tiene la siguiente forma:

$$X_n^c = \begin{pmatrix} R & [t]_x \cdot R \\ 0 & R \end{pmatrix}$$

Siendo R la matriz de rotación y t el vector de traslación que permiten pasar del sistema de referencia del plano imagen de la cámara al sistema de referencia del efector final del robot; y $[t]_x$ la matriz anti simétrica de t, de modo que se define como: [11]

$$[t]_x = \begin{pmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{pmatrix} \quad \{2.4.1.8\}$$

Una vez se tiene la relación {2.4.1.7}, se incorpora a la relación {2.4.1.4}, resultando:

$$\dot{e} = L_s \cdot X_n^c \cdot J_n \cdot \dot{q} + \frac{\partial e}{\partial t} \quad \{2.4.1.9\}$$

Ecuación que puede simplificarse si se define:

$$J_s = L_s \cdot X_n^c \cdot J_n$$

De modo que la evolución del error con el movimiento articular y el movimiento del *target* resulta:

$$\dot{e} = J_s \cdot \dot{q} + \frac{\partial e}{\partial t} \quad \{2.4.1.10\}$$

La ley de control resultante teniendo en cuenta la del caso *free camera* ({2.4.1.6}) y la relación entre la variación del error y el movimiento de las articulaciones del robot ({2.4.1.10}) será pues:

$$\dot{q}(k) = -\lambda \cdot J_s^+(k) \cdot e(k) - J_s^+(k) \widehat{\frac{\partial e}{\partial t}}(k) \quad \{2.4.1.11\}$$

Nótese que, en la nueva ley de control, el término de la estimación del efecto del movimiento del *target* sobre la variación del error, $\widehat{\frac{\partial e}{\partial t}}$, debe tener en consideración también el cambio debido al paso al espacio articular, de modo que la expresión {2.4.1.5} se modifica, pasando a ser:

$$\widehat{\frac{\partial e}{\partial t}} = \frac{e(k) - e(k-1)}{T_s} - J_s(k) \cdot \dot{q}(k-1) \quad \{2.4.1.12\}$$

En cuanto a la forma de aplicar el control al robot, se aplica la misma integración que en la expresión {2.3.2.2} y por el mismo motivo.

Nótese que en la expresión {2.4.1.12}, como resultado de la {2.4.1.11} influye también la ganancia proporcional y se acumula el efecto de la estimación anterior.

El caso *eye-to-hand* difiere ligeramente en la forma de obtener la matriz Jacobiana J_s, ya que la relación dada por X_n^c no es constante. No obstante, de dicho caso trasciende el objeto del presente trabajo académico, y puede encontrarse desarrollado con más detalle en la misma referencia sobre *Visual Servoing* del *Springer Handbook of Robotics* [8].

2.4.2 *Image Based Visual Servoing (IBVS)*: [8]

En IBVS, el vector de características *s* se determina a partir de las coordenadas en el plano imagen bidimensional de ciertos puntos de interés del *target*.

Así, las medidas de la imagen son precisamente dichas coordenadas de los puntos de interés de (en píxeles o en coordenadas del plano de la imagen) y el conjunto de parámetros que aporta información adicional sobre el sistema son los parámetros intrínsecos de la cámara.

En cuanto a la definición de la Matriz de Interacción (L_s), dado que relaciona la variación del vector de características visuales *s* con el movimiento de la cámara, es necesario primero comentar cómo se obtienen las coordenadas en el plano de la imagen para después pasar a explicar la construcción de la Matriz de Interacción:

Sea un punto de unas ciertas coordenadas (X, Y, Z) respecto del *frame* de la cámara, de manera que la coordenada Z represente la profundidad con respecto a la cámara. Dicho punto tridimensional da lugar a un punto bidimensional al proyectarse sobre el plano imagen de la cámara, y este punto bidimensional tiene las siguientes coordenadas (x, y) en el plano imagen:

$$\begin{cases} x = {}^{X}/_{Z} \\ y = {}^{Y}/_{Z} \end{cases} \quad \{2.4.2.1\}$$

Estas coordenadas lo son todavía en unidades de distancia normalizadas. Dado que en un caso de aplicación real las coordenadas en el plano de la imagen se obtendrían en píxeles, es de interés conocer la conversión de píxeles a coordenadas en el plano bidimensional que se puede realizar conociendo los parámetros intrínsecos de la cámara que toma la imagen:

$$\begin{cases} x = \dfrac{u - c_u}{f \cdot \alpha} \\ y = \dfrac{v - c_v}{f} \end{cases}$$

Donde f es la distancia focal en píxeles, (c_u, c_v) son las coordenadas del punto principal (centro de la imagen) en píxeles, (u, v) las coordenadas del punto proyectado en píxeles y α es el ratio de dimensiones del píxel, que permite corregir el cálculo en caso de que el tamaño horizontal y vertical del píxel no sea el mismo. Estos son los parámetros intrínsecos de la cámara.

Como método equivalente de cálculo, que se ha utilizado en el presente trabajo a la hora de simular, es posible realizar la conversión siguiente: [12]

$$\begin{cases} x = \dfrac{u - c_u}{f_u} \\ y = \dfrac{v - c_v}{f_v} \end{cases}$$

$$con\ f_u(píxeles) = \frac{f}{tamaño\ horizontal\ del\ píxel}, f_v(píxeles) = \frac{f}{tamaño\ vertical\ del\ píxel}$$

Donde f es la distancia focal, ahora en unidades de distancia.

Una vez obtenido $x_p = (x, y)$, para obtener la Matriz de Interacción es necesario encontrar la relación entre la velocidad de la cámara, (v_c, ω_c), y la variación de las coordenadas sobre el plano imagen, $\dot{x_p}$.

El punto de partida para ello es la relación entre la variación de las coordenadas del punto tridimensional respecto del *frame* de la cámara (X, Y, Z) y la velocidad de la cámara:

$$\dot{X_p} = -v_c - \omega_c \times X_p \leftrightarrow \begin{cases} \dot{X} = -v_x - \omega_y \cdot Z + \omega_z \cdot Y \\ \dot{Y} = -v_y - \omega_z \cdot X + \omega_x \cdot Z \\ \dot{Z} = -v_z - \omega_x \cdot Y + \omega_y \cdot X \end{cases} \quad \{2.4.2.2\}$$

A su vez, es posible obtener la relación entre $\dot{x_p}$ y $\dot{X_p}$ derivando las ecuaciones {2.4.2.1}:

$$\begin{cases} \dot{x} = \dfrac{\dot{X}}{Z} - \dfrac{X \cdot \dot{Z}}{Z^2} = \dfrac{\dot{X} - x \cdot \dot{Z}}{Z} \\ \dot{y} = \dfrac{\dot{Y}}{Z} - \dfrac{Y \cdot \dot{Z}}{Z^2} = \dfrac{\dot{Y} - y \cdot \dot{Z}}{Z} \end{cases} \quad \{2.4.2.3\}$$

De modo que la relación entre la velocidad de la cámara y la variación de los puntos del plano imagen viene dada por la sustitución de las expresiones {2.4.2.2} en el sistema {2.4.2.3}:

$$\begin{cases} \dot{x} = \dfrac{-v_x}{Z} + \dfrac{x \cdot v_z}{Z} + x \cdot y \cdot \omega_x - (1 + x^2) \cdot \omega_y + y \cdot \omega_z \\ \dot{y} = \dfrac{-v_y}{Z} + \dfrac{y \cdot v_z}{Z} + (1 + y^2) \cdot \omega_x - x \cdot y \cdot \omega_y - x \cdot \omega_z \end{cases} \quad \{2.4.2.4\}$$

Observando el sistema de ecuaciones {2.4.2.4}, resulta evidente que ambas ecuaciones son el resultado de multiplicar los elementos del vector de velocidad de la cámara v_c por unas serie de términos dependientes de las coordenadas del punto en el plano imagen (x, y).

Es decir, se ha hallado la interacción entre la variación de las coordenadas en el plano imagen de un punto, que es precisamente la característica visual necesaria para el control IBVS y el movimiento de la cámara, luego se ha hallado la forma de construir la Matriz de Interacción:

$$\dot{x}_p = L_x \cdot v_c$$

$$con\ L_x = \begin{bmatrix} \frac{-1}{Z} & 0 & \frac{x}{Z} & x \cdot y & -(1+x^2) & y \\ 0 & \frac{-1}{Z} & \frac{y}{Z} & (1+y^2) & -x \cdot y & -x \end{bmatrix} \quad \{2.4.2.5\}$$

Para que sea posible controlar los 6 grados de libertad de la cámara, el sistema de ecuaciones debe tener al menos tantas ecuaciones como incógnitas, es decir, la matriz de interacción debe tener, al menos, 6 filas. Como cada punto da lugar a dos filas, para controlar una cámara los 6 grados de libertad serán necesarios al menos 3 puntos.

No obstante, con 3 puntos existen múltiples configuraciones singulares de la Matriz de Interacción y, lo que es aún más importante, existen 4 posibles posiciones de la cámara que anulan el error, de manera que suelen utilizarse más de 3 puntos. En el caso del presente trabajo, se utilizan 4 puntos tanto en simulación como en experimentación real.

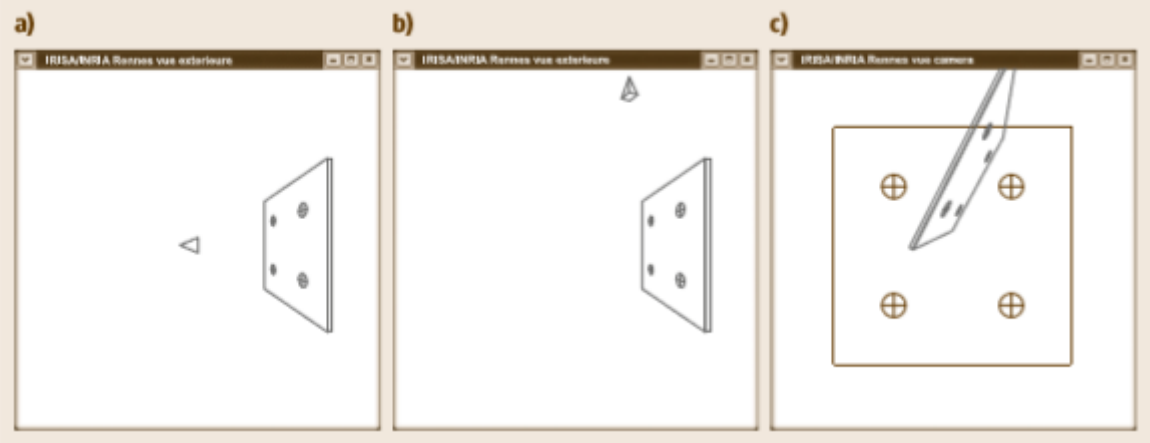


Figura 9 [8]- a)Posición deseada de la cámara respecto al target de 4 puntos, b) posición de partida de la cámara respecto al target de 4 puntos, c) comparativa de la imagen de referencia y la imagen de partida

La Matriz de Interacción y el vector de características visuales tomarán, por tanto, la siguiente forma en el presente trabajo:

$$L_s = \begin{pmatrix} L_{x1} \\ L_{x2} \\ L_{x3} \\ L_{x4} \end{pmatrix} \quad s = \begin{pmatrix} x_{p1} \\ x_{p2} \\ x_{p3} \\ x_{p4} \end{pmatrix}$$

En definitiva, el esquema de control IBVS toma la forma que se puede apreciar en la figura 10:

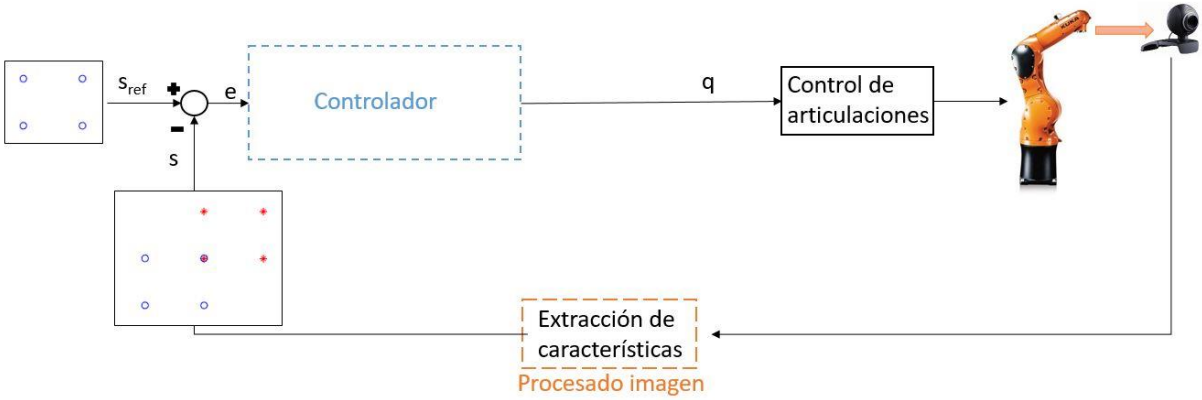


Figura 10- Esquema del control IBVS

2.4.2.1 Comportamiento del IBVS: [8]

El control IBVS tal y como se ha planteado, al trabajar con características obtenidas directamente de la imagen y presentar un descenso exponencial del error, busca siempre la convergencia más rápida entre los puntos de interés del *target* y sus respectivas posiciones de referencia, lo que lleva a que los puntos de interés describan trayectorias desde su posición de partida hasta su posición deseada sobre el plano imagen que se aproximan a rectas.

El control IBVS descrito presenta una ventaja derivada de dicho funcionamiento: al trabajar tal y como se ha indicado, los puntos de interés no abandonan el campo visual de la cámara como consecuencia del movimiento de la cámara para corregir el error, salvo que se dé el error de *camera retreat*, que se tratará más adelante.

En la figura 10 se muestra un caso representativo de este desplazamiento de los puntos de interés de la imagen aproximado a rectas, si bien no es un caso satisfactorio en cuanto al comportamiento del movimiento de la cámara, que realiza cambios demasiado bruscos.

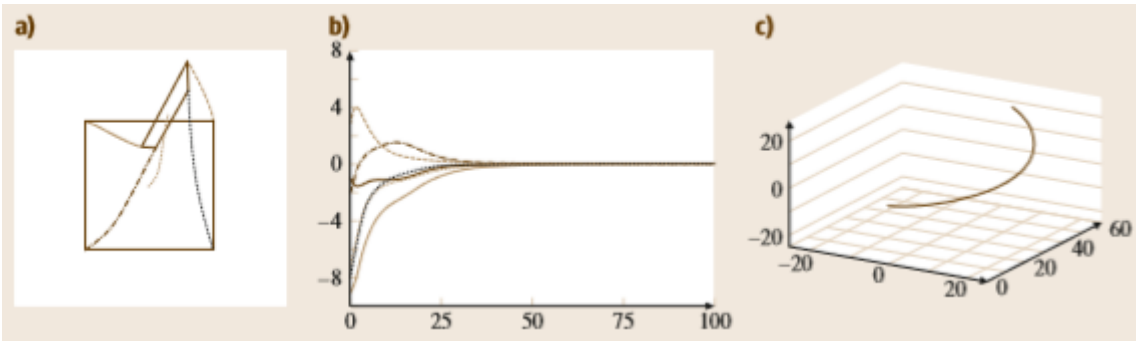


Figura 11 [8]- Convergencia en posicionado IBVS: a) Trayectoria de los puntos del target hasta la posición de referencia siguiendo líneas prácticamente rectas, b) evolución de la velocidad de la cámara, c) desplazamiento del centro óptico de la cámara.

El principal inconveniente que presenta el control IBVS es el error, ya mencionado, conocido como *camera retreat*. Este error consiste en que, ante errores que precisan una rotación entorno al eje óptico de la cámara, al tratar de conseguir una trayectoria de los puntos en línea recta, al movimiento de rotación entorno al eje óptico (eje Z de la cámara) se añade un retroceso a lo largo del eje Z óptico.

Este fenómeno se amplifica cuando el giro precisado entorno a Z es grande, creciendo el retroceso de la cámara en detrimento del giro, pudiendo alcanzar los límites de las articulaciones del robot.

El error de *camera retreat* se debe a las características visuales elegidas y a al acoplamiento entre las columnas de la Matriz de Interacción correspondientes a la traslación y la rotación en Z (columnas 3 y 6).

Además, como caso particular de este fenómeno, existe una singularidad cuando se requiere un giro entorno a Z de π radianes, caso en el cual simplemente no se produce rotación alguna, tan solo el retroceso en Z.

En cualquier caso, incluso sin darse el error de *camera retreat*, el control IBVS ignora el espacio tridimensional, dado que la ley de control trata de corregir el error en el espacio bidimensional, lo que puede llevar a las articulaciones del robot a alcanzar sus límites durante una tarea [13].

Además, a la hora de trabajar con un caso real, son necesarias técnicas de visión artificial para estimar la profundidad de cada punto, Z, y es necesario corregir la estimación de la pseudo-inversa de la matriz de interacción, $\widehat{L_s^+}$ con la matriz de interacción correspondiente al vector de características de referencia. Ambas consideraciones van más allá del presente trabajo, dado que son técnicas de procesado de los datos de la cámara, por lo que no se implementan en el presente trabajo académico, no obstante, pueden encontrarse tratado en detalle en el capítulo dedicado a *Visual Servoing* del *Springer Handbook of Robotics* [8].

En lo que respecta a la estabilidad, cabe señalar que es demostrable la estabilidad local del IBVS con un vector de características visuales correspondiente a más de 3 puntos, pero no la estabilidad global [8]. Esto se ve reflejado en el desempeño del control IBVS cuando se requiere un giro entorno al eje óptico de la cámara: es localmente satisfactorio, cuando el error a corregir es pequeño, pero puede no serlo cuando el error es grande.

Por último, cabe añadir que el control IBVS es robusto frente a errores de modelado del *target* y frente a errores de calibración de la cámara [13].

2.4.3 *Position Based Visual Servoing* (PBVS): [8]

En PBVS, el vector de características *s* se define a partir de la posición de la cámara con respecto al *target* (a un *frame* de referencia del mismo), posición que se estima a partir de medidas de la imagen bidimensional, para lo cual es necesario, además, contar con un modelo 3D del *target*. Cabe aclarar que las técnicas de visión artificial mediante las cuales se estima la posición del *target* en el espacio tridimensional a partir de medidas en el plano imagen no son objeto del presente trabajo académico.

Para definir *s*, en primer lugar es necesario definir el vector de características visuales de referencia, s^*, que es la posición deseada de la cámara respecto del *target*. Existen dos maneras típicas de plantear la posición de referencia:

- La posición deseada de la cámara respecto del objeto implica desplazamiento, pero no rotación. En este caso se puede probar que el control da lugar a un decrecimiento exponencial de la velocidad, la cámara no sigue un desplazamiento en línea recta y el origen del frame del objeto sí sigue un desplazamiento en línea recta hasta la posición deseada (en el caso del

presente trabajo, el desplazamiento en línea recta lo sigue el centro de los 4 puntos, pero no los 4 puntos) en el plano de la imagen [8].

- La posición deseada de la cámara respecto del objeto no implica ni desplazamiento ni rotación respecto de éste (de su *frame* de referencia). Puede probarse que, con esta elección, el desplazamiento de la cámara en el espacio tridimensional sí sigue una línea recta, pero es más probable la salida de algún punto de referencia fuera del campo visual [8].

Puede observarse el comportamiento de ambas opciones en las figuras 12 y 13:

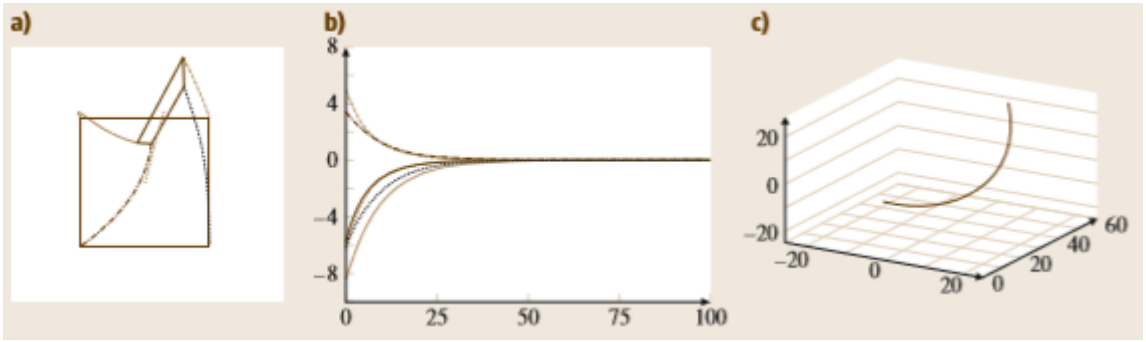


Figura 12 [8]-Desempeño al elegir la primera opción; a), b) y c) representan la misma información que en la figura 11

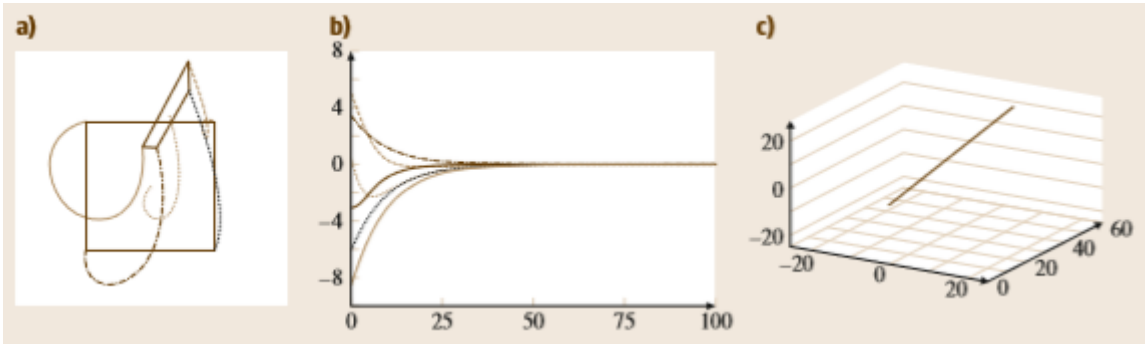


Figura 13 [8]-Desempeño al elegir la segunda opción; a), b) y c) representan la misma información que en la figura 11

En vista de que la primera opción tiene un riesgo menor de perder el contacto visual con el *target* durante la corrección del error, en el presente trabajo se desarrolla la primera opción. Una vez elegida la referencia, es necesario precisar la forma concreta que toma el vector de características, tanto el de referencia como el medido.

Para ello, se ha de trabajar con matrices de transformación homogénea, debido a lo cual se utilizará una notación tal que: *aMb* es una matriz de transformación que permite pasar del sistema de referencia *a* al sistema de referencia *b*, *aTb* es el vector de traslación existente desde *a* hasta *b*, correspondiente con *aMb(1:3,4)*, y *aRb* es la matriz de rotación de *a* a *b*.

Aplicando dicha notación al caso que se está tratando:

$$Cámara - target\ deseada: c^*Mo = [c^*To; 0]$$

$$Cámara - target\ actual: cMo = [cTo; cRo]$$

$$Cámara\ (pose\ deseada) - Cámara\ (pose\ actual): c^*Mc = [c^*Tc, c^*Rc]$$

Las transformaciones geométricas pueden observarse de forma más intuitiva en la figura 14, que se muestra a continuación:

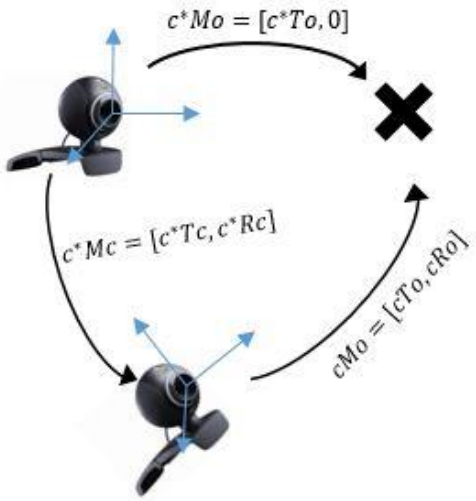


Figura 14- Transformaciones geométricas entre los sistemas de referencia

El vector de características visuales de referencia será sencillamente $s^* = [c^*To; 0_{3x1}]$, lo que implica que hay que asegurar que al definir *s* la rotación esté expresada en forma de vector de 3 elementos.

Esto es posible utilizando la conocida como *representación de ángulo-eje equivalente*, que consiste en expresar la rotación deseada como el giro de un ángulo, θ, entorno a un eje, u, denominado *eje equivalente*, dando como resultado un vector $\theta \cdot u$ de 3 elementos.

Para hallar esta representación equivalente, en primer lugar hay que calcular el ángulo de giro, θ, a partir de la matriz de rotación, *R*. Utilizando el cálculo mediante la arco tangente con rango ampliado entre [-π,π), atan2, se obtiene: [14]

$$\theta = atan2\left(\frac{r_{11} + r_{22} + r_{33} - 1}{\sqrt{(r_{12} - r_{23})^2 + (r_{13} - r_{31})^2 + (r_{21} - r_{12})^2}}\right) \quad \{2.4.3.1\}$$

$$Con\ \ r_{ij}\ elemento\ de\ la\ fila\ i, columna\ j\ de\ la\ matriz\ R$$

Y, utilizando el resultado de la expresión {2.4.3.1}, se obtiene el eje de la representación ángulo-eje equivalente: [15]

$$u = \frac{1}{2\sin\theta}\begin{pmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{pmatrix} \qquad \{2.4.3.2\}$$

A continuación, hay que definir el vector *s* de tal manera que corregir el error signifique corregir la diferencia entre *s* y s^*, llevando la cámara a la posición deseada, es decir, que la señal de error tome la forma:

$$e = s - s^* = [\ cTo - c^*To; \theta u_{c^*Rc}]$$

Por lo tanto, el vector de características será:

$$s = [\ cTo;\ \theta u_{c^*Rc}]$$

Y, como consecuencia, la Matriz de Interacción toma la forma:

$$L_s = \begin{pmatrix} -I_3 & [cTo]_x \\ 0 & L_{\theta u} \end{pmatrix} \quad \{2.4.3.3\}$$

El elemento $[cTo]_x$ de la matriz {2.4.3.3} es una matriz anti simétrica como en la expresión {2.4.1.8} y $L_{\theta u}$ viene dada por:

$$L_{\theta u} = I_3 + \frac{\theta}{2}[u]_x + \left(1 - \frac{sinc(\theta)}{sinc^2(\theta)/2}\right) \quad \{2.4.3.4\}$$

Siendo también $[u]_x$ una matriz anti simétrica (de *u*) y *sinc* el seno cardinal, cuyo resultado es 0 cuando el ángulo vale la unidad, y $^{\sin(\theta)}/_{\theta}$ en el resto de casos.

En definitiva, el esquema de control PgBVS toma la forma que se puede apreciar en la figura 15:

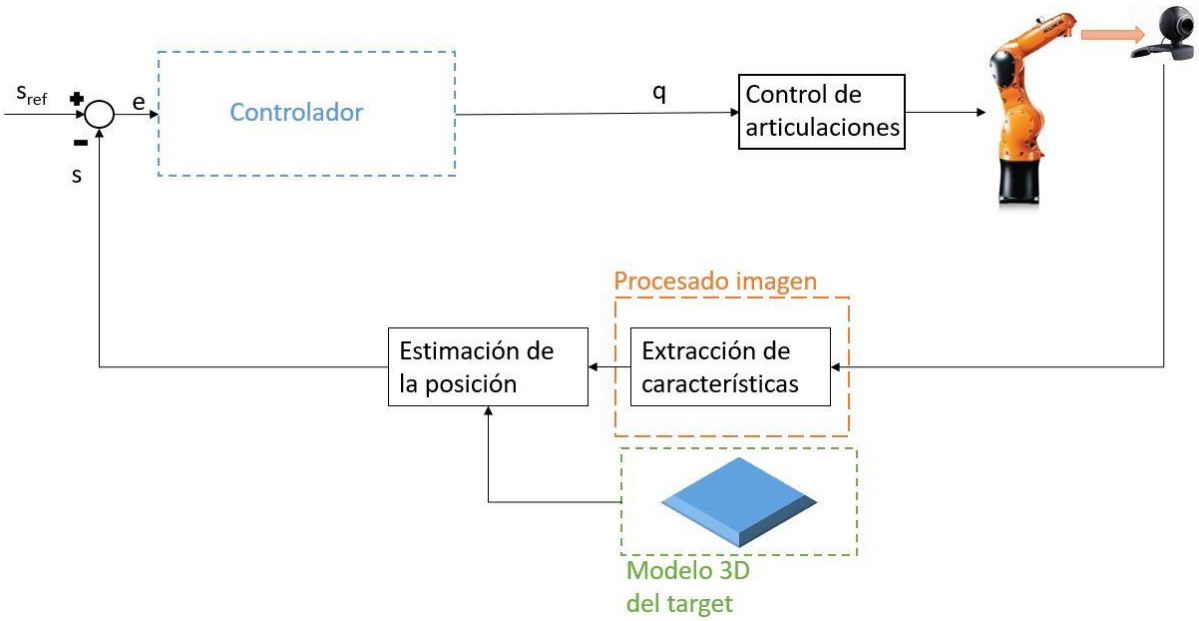


Figura 15-Esquema del control PBVS

2.4.3.1 Comportamiento del PBVS: [8]

El control PBVS, tal y como se ha planteado, hace disminuir exponencialmente el error de posición en el espacio tridimensional, de forma que su funcionamiento es más similar que en el caso IBVS a un control convencional de velocidad como el visto en el apartado 2.3.3.

En consecuencia, y como ya se ha adelantado al principio del apartado sobre PBVS, el control no busca que los puntos de la imagen realicen una trayectoria lo más breve posible (línea recta) desde el punto de partida a la referencia. Sin embargo, este recorrido más rápido posible, en línea recta, correspondiente al descenso exponencial del error, lo realiza o bien el centro del *frame* del objeto en el plano imagen, o bien la propia cámara en el espacio tridimensional, según la elección del vector de características visuales.

Mientras que al trabajar con un control que solo se da en el espacio tridimensional, el PBVS no presenta el error de *camera retreat* del IBVS, sí se plantea el riesgo de que alguno de los puntos de interés del *target*, necesarios para estimar las posiciones y realizar el control, salgan del campo visual de la cámara.

Esto se debe precisamente a que no existe ningún tipo de control que garantice que los puntos se mueven lo menos posible en el plano imagen. Como ya se ha comentado, si se elige definir una vector de características tal que el control trate de asegurar un movimiento de la cámara en línea recta, el control es más agresivo en el espacio tridimensional y, por tanto, los movimientos también son más grandes en el plano imagen, aumentando el riesgo de salida del campo visual.

Además, el error de estimación de los parámetros tridimensionales tiene, lógicamente, un efecto mayor sobre el PBVS que sobre el IBVS, dado que en el primero afecta a la precisión con que se alcanza la posición deseada, mientras que en el IBVS altera el movimiento de la cámara, pero no la convergencia, dado que ésta se termina midiendo sobre características extraídas directamente de la imagen [13].

En cuanto a la estabilidad del control PBVS, se puede garantizar su estabilidad global mientras los parámetros de posición estén perfectamente estimados [8].

Y, por último, en lo referente a su robustez, el control PBVS es poco robusto frente a errores en la computación de los puntos de la imagen y la consiguiente estimación de los parámetros tridimensionales, errores que, incluso siendo pequeños, pueden afectar negativamente a la precisión y estabilidad del control [8], lo cual confirma lo planteado anteriormente respecto a la estabilidad y precisión del sistema.

CAPÍTULO 3. SIMULADOR

3.1 Descripción general del simulador:

En el presente Trabajo Fin de Máster, se ha desarrollado una aplicación de *Matlab* con interfaz gráfica capaz de simular diferentes metodologías de control cinemático, tratadas en el capítulo 2 del presente documento, para una amplia gama de robots industriales y ante casos de control personalizables.

Dicho simulador ha recibido el nombre de *Robotic Arm Control Simulator*, y consiste en una aplicación de código libre y abierto, con una serie de paneles, indicadores y pulsadores que permiten la representación tridimensional de robots cuyos parámetros y modelo gráfico son proporcionados por el usuario, la modificación de sus posiciones articulares (directamente o mediante trayectorias personalizadas), así como la ejecución de simulaciones también personalizables de controles cinemáticos de velocidad, aceleración, IBVS y PBVS. Además, se ha elaborado un manual de usuario como apoyo documental.

Particularmente, el simulador es capaz de trabajar con brazos robóticos articulados de hasta 10 grados de libertad en los que todas sus articulaciones son rotativas. Se ha descartado incluir la opción de trabajar con articulaciones de tipo prismático por los siguientes motivos:

- En primer lugar, su uso en la industria es más limitado, debido a sus múltiples inconvenientes específicos: sensibilidad a la contaminación, uso indebido y efectos del entorno muy superior a la de las articulaciones de revolución, así como también una dificultad significativamente mayor a la hora de ser producidos, ensamblados y alineados [16].
- En segundo lugar, la simulación de una articulación prismática presenta problemas adicionales respecto al caso de revolución, puesto que requiere más información sobre la articulación para cada caso concreto [17] y, precisamente por ello, su documentación y funcionalidades en la librería que, como se comentará más adelante, se ha utilizado para la representación tridimensional de robots (*rvctools* de Peter Corke), es menor que en el caso de las articulaciones rotativas. Por tanto, la resolución de estos problemas informáticos no triviales entra en contradicción con los objetivos del presente trabajo académico.

Las metodologías de control cinemático del simulador incluyen:

- Control de velocidad, tanto para una referencia fija como para una referencia móvil (con pre alimentación de la velocidad de la referencia), tal y como se trata en el apartado 2.3.3 del presente documento. Son personalizables la ganancia proporcional del control y la referencia (punto o trayectoria).
- Control de aceleración tal y como se trata en el apartado 2.3.4 del presente documento, con las mismas funcionalidades que el control de velocidad.
- Control IBVS *eye-in-hand* para tareas de posicionado y *tracking,* tal y como aparece en el apartado 2.4 del presente documento. Son personalizables la ganancia proporcional del control, el tiempo total de simulación, el periodo de muestreo y la referencia, si bien ésta cuenta también con dos referencias de prueba previamente desarrolladas.

- Control PBVS *eye-in-hand* tal y como se muestra en el apartado 2.4 del presente documento, con las mismas funcionalidades que el control IBVS.

La motivación por la cual se ha optado por desarrollar un programa de simulación completo en lugar de elaborar simulaciones particulares para cada metodología de control responde a las siguientes consideraciones:

- Desarrollar un simulador con un enfoque generalista implica adquirir una comprensión más extensa y profunda de los controles implementados, dado que es necesario asegurar el correcto funcionamiento del control simulado ante una gran variedad de casos, también casos imprevistos.
- Un simulador como el desarrollado tiene una utilidad inmediata en dos sentidos: primero, la simulación de un control cinemático en un caso sin dinámica es un primer cortapisas a la hora de comprobar la viabilidad de una cierta tarea; y segundo, dado que se trata de *software* libre y de código abierto, supone un punto de partida para el aprendizaje por parte de terceros.
- El código del simulador es editable, lo que ofrece una plataforma de partida para futuros desarrollos más elaborados.

Para terminar, cabe añadir que la aplicación cuenta con una librería de 8 robots aptos para su simulación, si bien, como se tratará más tarde, es sencillo añadir nuevos robots a la librería.

3.2 Desarrollo del simulador:

La aplicación ha sido desarrollada en *Matlab R2017b*, recurriendo a las siguientes herramientas complementarias:

- El entorno de desarrollo de interfaces gráficas para aplicaciones de *Matlab*, GUIDE.
- La *toolbox* de *Matlab rvctools*, de Peter Corke, para la representación tridimensional del robot simulado, el cálculo de la matriz Jacobiana geométrica mediante métodos numéricos y la cinemática directa, si bien para esto último también se cuenta con una función de desarrollo propio.
- La *toolbox* de *Matlab* "ARTE" (*A Robotics Toolbox for Education*), de la cual se han utilizado su librería de archivos gráficos de robots y los correspondientes parámetros DH.

Dada la escasez y dispersión de la documentación del entorno GUIDE, así como su importancia para el desarrollo de la aplicación, conviene detenerse a explicar a grandes rasgos su funcionamiento y qué funcionalidades ha aportado al presente trabajo.

Las funciones de *rvctools* se explicarán brevemente más adelante a medida que se expongan las diferentes secciones de la aplicación, no obstante, más detalle puede encontrarse en la documentación elaborada por su desarrollador.

3.2.1 Entorno de desarrollo GUIDE:

El entorno de desarrollo GUIDE permite elaborar interfaces gráficas y, a través de ello, establecer la estructura básica para aplicaciones de *Matlab*.

En GUIDE se trabaja con archivos de tipo *fig*, y consiste en un editor que permite introducir elementos propios de una interfaz gráfica y modificar sus propiedades (tamaño, color, posición, nombre…) y, en ocasiones, su jerarquía en la aplicación (por ejemplo, botones que solo son visibles si el panel al que pertenecen lo es.

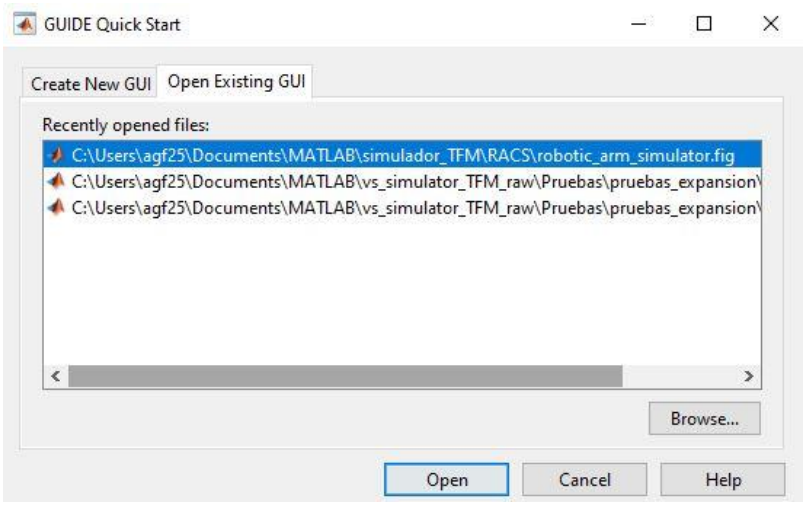


Figura 16-Carga de un proyecto de interfaz gráfica en GUIDE

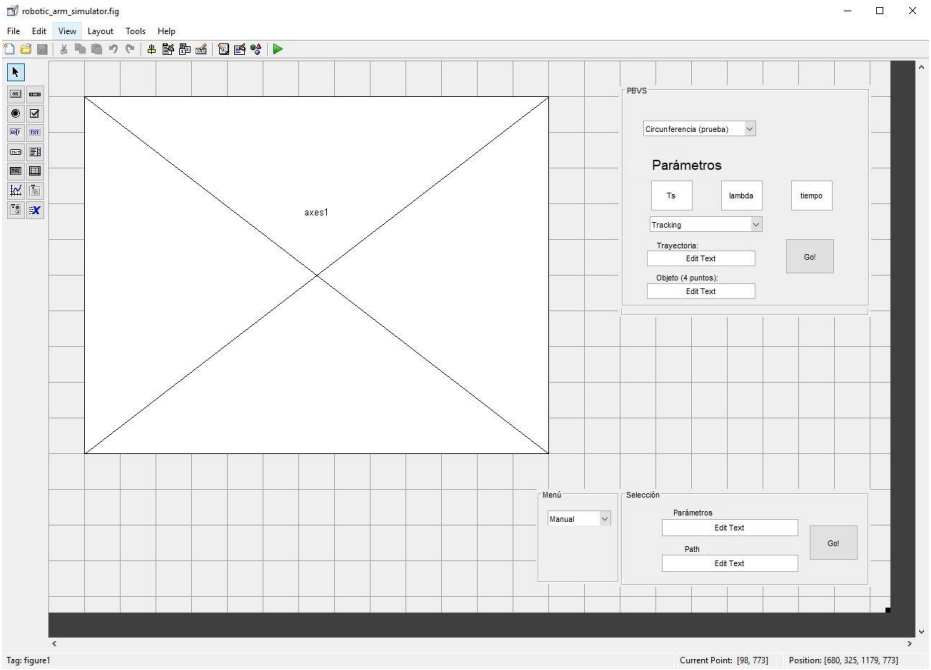


Figura 17-Editor de GUIDE

La principal ventaja de GUIDE consiste en que cada interfaz gráfica producida, genera inmediatamente un archivo de código fuente de la aplicación correspondiente en *Matlab*, con el mismo nombre y extensión *.m*, código que se actualiza automáticamente cada vez que se añade un nuevo elemento a la interfaz gráfica desde el editor, pero sin eliminar el código ya existente en el archivo *.m* de *Matlab*.

En lo que respecta a los elementos de la interfaz gráfica, cada uno de ellos cuenta con su correspondiente manejador en el archivo de código fuente de la aplicación, expresado como un campo

de una variable a la que suele dársele el nombre de *handles*. Además, en caso de que así se indique en el editor, los elementos añadidos a la interfaz gráfica también generan en el código de la aplicación sus correspondientes funciones de arranque y de *Callback* (función cuyo código se ejecuta al interactuar con el elemento en la interfaz gráfica de la aplicación o, naturalmente, cuando se le llama), que toman automáticamente los nombres "*Elemento_CreateFcn"* y "*Elemento_Callback*", donde *Elemento* se corresponde con la etiqueta dada al elemento en cuestión en el editor.

Las funciones de la aplicación son completamente independientes entre sí, y su actuación coordinada solo es posible mediante el uso de los manejadores de los elementos (campos de la variable *handles*) y otras variables creadas durante la ejecución del propio programa y que pueden añadirse como un campo más de la variable *handles* si se desea que sean accesibles por parte de otras funciones. Estos manejadores son administrados desde una función *main* generada automáticamente y externa al código de la aplicación.

Para que una función pueda utilizar los manejadores (tanto los de los elementos de la interfaz como los manejadores de variables creadas en otras funciones), debe llamarlas mediante la función "*guidata*", y para que los cambios que realice queden guardados para su futuro uso, debe actualizar los manejadores mediante la misma función, aunque con otra sintaxis.

Un ejemplo de llamada, modificación y posterior actualización de los manejadores se puede observar a continuación:

```
function pushbutton_suma_ejemplo_Callback(hObject, eventdata, handles)
handles=guidata(hObject); %Carga de los manejadores
handles.suma=2+3;%El código se ejecuta
guidata(hObject, handles);  %Actualización de los manejadores
```

En particular, el manejador de un elemento permite acceder a todas las propiedades de dicha clase de elemento. Esto es de utilidad para leer o modificar propiedades de los elementos de la interfaz gráfica (procesar texto escrito en una casilla editable, actualizar el texto de un indicador, hacer visible o no un cierto elemento…), pero también para acceder a propiedades de elementos esenciales de la simulación, como las posiciones de las articulaciones de un robot. En el ejemplo de código a continuación se muestra un caso representativo de cada uso:

```
handles.text_Kp.Visible='on'; %Hacer visible un indicador
q=handles.rob.getpos'; %Posición de las articulaciones del robot (lectura)
```

Así pues, el código fuente generado automáticamente por GUIDE consiste en: las funciones de creación y *callback* de los elementos que las necesiten, las funciones que ejecutan instrucciones al inicio y cierre de la aplicación ("*OpeningFcn*" y *"CloseFcn*") y una función no editable de puesta en marcha de la aplicación que arranca la función *main* ("*gui_mainfcn*") externo al archivo, y que se encarga de administrar el uso de los manejadores y, por tanto, permite la coordinación de todas las funciones del programa.

Para concluir la explicación, es de utilidad tratar brevemente los elementos de la interfaz gráfica utilizados en el presente trabajo académico:

- Pulsadores (*pushbutton*): Se trata de botones cuya función de *Callback* se activa al pulsarlos. En la aplicación desarrollada se han utilizado sistemáticamente para la activación de la

simulación de los distintos algoritmos de control, de manera que sus funciones de *callback* contienen todas las instrucciones necesarias para simular cada control.

- Casillas de texto editables (*edit text*): Se trata de elementos con la propiedad de almacenar variables de texto en el campo *String* y modificarlas tanto al escribir desde la interfaz en la casilla de texto, como al modificar la propiedad *String* desde el programa. Su función de *callback* se activa al terminar de escribir sobre la casilla. En la aplicación del presente trabajo se han utilizado principalmente para indicar los robots a simular, las trayectorias de referencia y los parámetros de control personalizables.
- Menús desplegables de selección múltiples (*pop-up menú*): Permiten definir opciones a seleccionar una vez puesta en marcha la interfaz gráfica. Las opciones únicamente son cadenas de texto asociadas al número de la posición que ocupen en el menú desplegable (primera, segunda…), de modo que para que cumplan con su objetivo de servir de selectores de opciones, es necesario elaborar un código en la función de *callback* del menú que evalúe cuál es la opción escogida y asigne una acción en consecuencia. La función de *callback* únicamente se activa al seleccionar una opción, así que el elemento no tiene una opción por defecto al arrancar el programa, por lo que es conveniente definirla en la función de creación del menú. En la aplicación desarrollada en el presente trabajo académico, los menús desplegables se utilizan principalmente para indicar qué modo de funcionamiento del simulador se debe activar y para seleccionar modos de marcha concretos dentro de cada algoritmo de control.
- Barras deslizadoras (*slide bars*): Las barras deslizadoras fueron utilizadas en las primeras versiones de la aplicación, si bien han sido descartadas en sus versiones finales. Esto se debe a que no permiten un control preciso del valor que se está eligiendo, y a que su funcionamiento no permite en la práctica un incremento gradual de un valor, que es el principal interés de una barra deslizadora, puesto que su función de *callback* solo se activa ante una selección instantánea, no manteniendo seleccionado el elemento o arrastrando la barra entre sus límites.

Por supuesto, también se han utilizado etiquetas de texto, cuyas cadenas de texto pueden ser editadas desde el programa, pero no desde la interfaz gráfica; y paneles, que permiten supeditar la visibilidad de los elementos situados sobre el panel a su propia visibilidad.

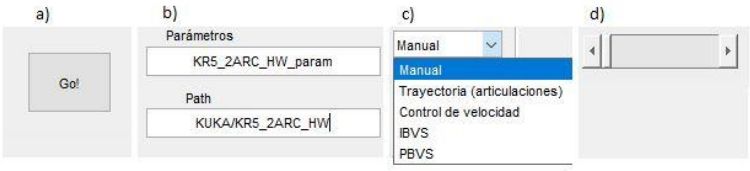


Figura 18-Elementos descritos: a)Pulsador, b)Casillas de texto editable, c)Menú desplegable de selección múltiple, d)Barra deslizadora

3.3 Secciones del simulador:

En este apartado se explican las distintas secciones que componen la aplicación de simulación desde el punto de vista de la programación desarrollada, incluyendo los algoritmos de control implementados.

Una exposición breve del funcionamiento general del simulador desde el punto de vista de su uso se puede encontrar en el manual de usuario de la aplicación, incluido entre los anexos del presente Trabajo Fin de Máster.

3.3.1 Carga del robot a simular:

La función esencial para definir un robot es *SerialLink* de *rvctools*, la cual crea una variable de clase *Serial-link robot*, definida también en la misma *toolbox*, partiendo de una matriz en que cada fila se corresponde con una articulación del robot y cada columna con uno de sus parámetros DH, el *offset* de la variable de la articulación (por defecto se considera articulación de revolución) y sus límites (en grados).

Esta clase permite a su vez asociar un modelo de representación tridimensional al robot serial de la variable, mediante la propiedad *plot3d*. Esta propiedad está diseñada para trabajar con la *toolbox* ARTE, de manera que para indicarle el *path* donde se encuentra el modelo tridimensional del robot, este modelo debe cumplir una serie de condiciones:

- Al activarse la propiedad/función *plot3d* del robot simulado, ésta busca entre las carpetas incluidas en el *path* disponible de *Matlab* hasta encontrar un archivo vacío que tenga por nombre *arte.m*. Una vez encontrado, la función define como *path* de partida el correspondiente a dicho archivo.
- El *path* del modelo gráfico debe coincidir con el del archivo *arte.m* y, adicionalmente, estar contenido en una carpeta de nombre *robots*. Por ejemplo, si se carga como *path* en la propiedad *model3d* la cadena de texto 'KUKA/Agilus', la función buscará el modelo en 'path_arte.m/robots/KUKA/Agilus'.
- Los archivos gráficos para componer el modelo tridimensional de simulación deben ser de tipo *stl* correspondientes a cada eslabón de la articulación, con el nombre 'linkN', donde N va de 0 hasta el número correspondiente al último eslabón. Otros detalles en para la configuración de los archivos *stl* pueden encontrarse en la documentación de la función *plot3d* de *rvctools*.

A nivel gráfico, con una adecuada disposición de ejes, luz, y posición y orientación del punto de vista de la escena (funciones *axis*, *light*, *campos*, *camva*) y la propiedad *plot3d* utilizada como se ha indicado, se obtiene la simulación tridimensional de un robot, tal y como se muestra en la figura 19.

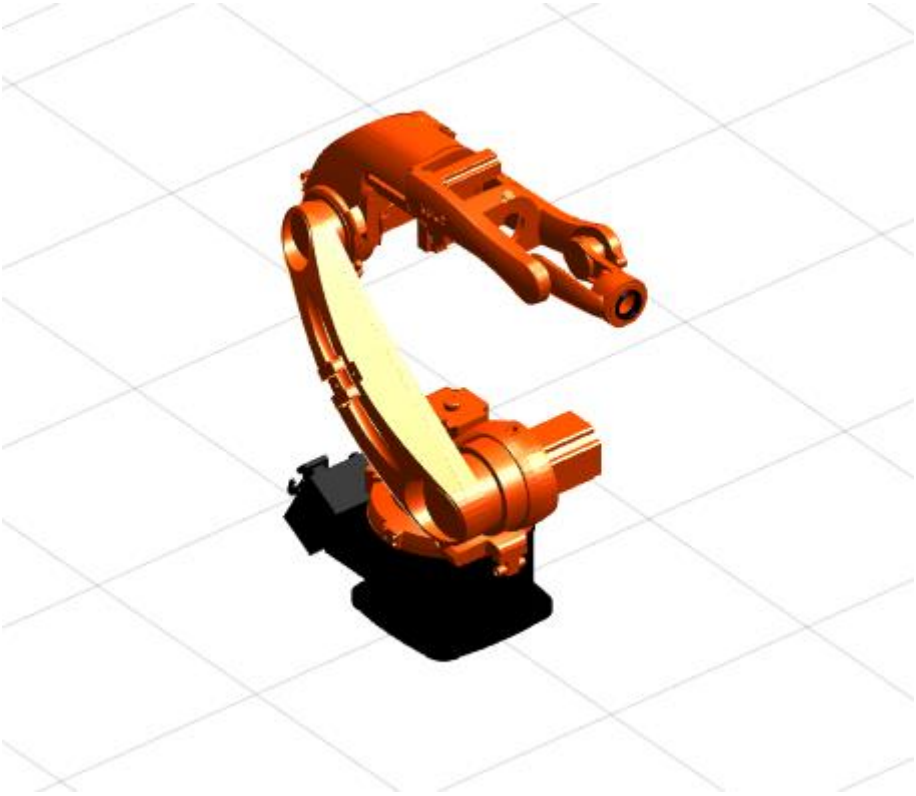


Figura 19-Ejemplo de representación gráfica tridimensional mediante funciones de rvctools

Más allá del uso de la clase *SerialLink* de *rvctools*, es necesario asegurar el funcionamiento de la carga del robot de forma que sea personalizable. Para ello, en el presente trabajo académico se ha optado por desarrollar una metodología basada en la definición de archivos de datos de *Matlab* (extensión *mat*) que contienen:

- Una matriz, de nombre '*P*', con los parámetros necesarios para construir el robot (los parámetros DH, el *offset* de cada articulación en radianes y los límites angulares). Cada fila corresponde a una articulación y cada columna a un parámetro.
- Una matriz, de nombre '*base*', que indica la posición y orientación de la base del robot en forma de transformación respecto del sistema de referencia global.
- Un vector, de nombre '*q_init*', que indica la configuración inicial de las articulaciones del robot en radianes.
- Un vector de células (*cell array*), de nombre '*color*', que indica el color de cada eslabón del robot.
- Una variable de texto, de nombre '*name*', que contiene el nombre del robot en cuestión.

De este modo, se ha construido una librería de diferentes brazos robóticos industriales, extrayendo las carpetas de archivos gráficos de la librería *ARTE* y componiendo las matrices de parámetros a partir de los datos que se encuentran en la misma librería. Para incluir nuevos robots en la librería del programa, basta con contar con sus archivos gráficos en el formato indicado por la función *plot3d* de *rvctools* y componer su correspondiente archivo con la matriz de parámetros y los demás datos a los que se ha hecho referencia anteriormente.

Una vez explicadas las claves del funcionamiento de la carga de robots, ya es posible analizar más en detalle el funcionamiento de la sección de la aplicación que permite la carga de robots.

La carga del robot a simular se gestiona por parte del usuario desde un panel de la interfaz gráfica etiquetado como "Selección". En él hay dos casillas de texto, una para indicar el *path* final del modelo gráfico del robot, tal y como se ha indicado al inicio de este apartado, y en otro el nombre del archivo de datos de *Matlab* que contiene los parámetros y el resto de información necesaria para cargar el robot. Por lo demás, las instrucciones que realmente se ocupan de la carga del robot se encuentran en la función *callback* del pulsador del panel.

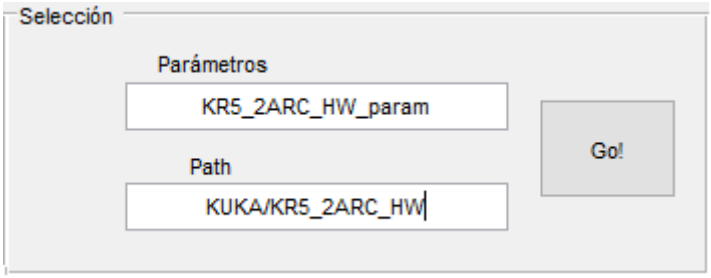


Figura 20-Panel de selección del simulador para la carga del robot.

Como se ha anticipado, la carga del robot en la aplicación no se limita simplemente a representar el robot tridimensionalmente como se ha comentado, sino que incluye funcionalidades adicionales importantes para el programa. Por ello, se comentan a continuación algunos fragmentos del código de dicha función.

Al inicio, tras haber obtenido los manejadores tal y como se ha indicado en el apartado 3.2.1, se guardan en una variable los datos necesarios para cargar el robot. Nótese que, para ello, primero hay que leer el texto de la casilla editable correspondiente, para lo cual es necesario convertirlo a formato de texto (función *strjoin*), puesto que, por defecto, se almacena en el manejador de la casilla como cadena de caracteres:

```
str=strjoin(handles.editparam.String);
handles.param=load(str);
```

A continuación, se construye la variable de clase robot llamando a una función llamada *build_robot* elaborada específicamente para esta aplicación. Esta función toma como entradas la matriz *P* de parámetros del robot, la matriz de transformación *base* y el nombre del robot, obtiene el número de articulaciones del robot leyendo el número de filas de la matriz *P* y después construye los vectores correspondientes a cada eslabón y define con ello el robot utilizando la función *SerialLink*:

```
function rob_out=build_robot(DHparam,nombre,base)
joints=size(DHparam,1);
```

[…]

```
for i=1:joints
        L(i,:)=Link('d', d(i),  'a', a(i),    'alpha', alfa(i),'offset',
off(i),'qlim', q_lim(i,:));
end
rob=SerialLink(L,'name',nombre);
rob.base=base;
```

```
rob_out=rob;
end
```

En la función de *callback* del pulsador se carga el *path* del modelo tridimensional del robot indicado desde la interfaz gráfica y se crea un manejador del robot creado para poder utilizarlo en el resto de funciones de la aplicación:

```
str=strjoin(handles.editpath.String);
rob.model3d=str;
handles.rob=rob;
```

A partir de ese punto, la aplicación se encarga de tareas que son necesarias para el correcto funcionamiento del resto de metodologías de control y del conjunto de la aplicación.

En primer lugar, para dotar a la carga del robot de la capacidad de servir para reiniciar la simulación sin necesidad de apagar el simulador, se hace un *reset* (*clareset*) del escenario que elimina todos los archivos gráficos y vuelve a definir el escenario deseado. Solo después se representa gráficamente el robot sobre este nuevo escenario (*plot3d*).

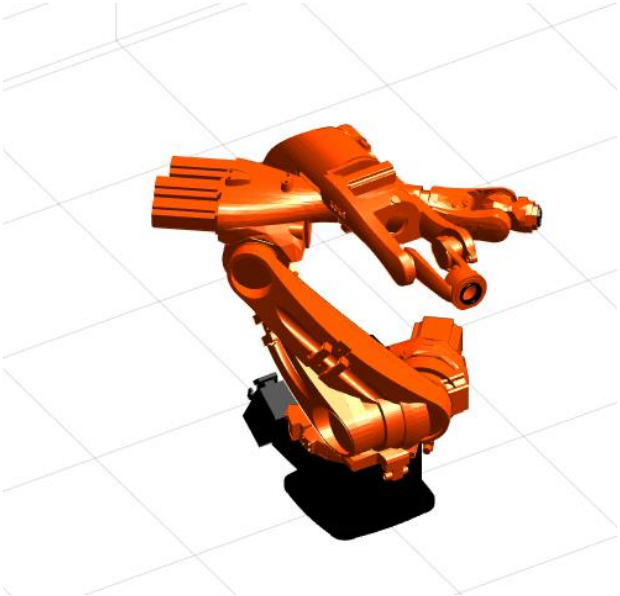


Figura 21-Efecto de no hacer un reset del escenario antes de representar el nuevo robot tras cargar un nuevo robot

La siguiente tarea reseñable es la obtención de las coordenadas de posición y orientación del efector final del robot para mostrarlas en indicadores del control de velocidad y aceleración. Si bien a lo largo de la aplicación se resuelve la cinemática directa recurriendo a la propiedad *fkine* de la clase *Robot-Serial* de *rvctools*, en este caso se ha dejado como muestra una función alternativa desarrollada en este trabajo (*calcular_cine_directa*):

```
p_act=calcular_cine_directa(handles.param.P,handles.rob.getpos,handles.para
m.base);
handles.text_x.String=strcat('x:',num2str(p_act(1)));
```

[…]

```
handles.text_xa.String=strcat('x:',num2str(p_act(1)));
```

La función *calcular_cine_directa* utiliza a su vez otra función desarrollada apropósito del presente trabajo, que calcula la matriz DH entre dos eslabones sucesivos (*DH_transHomogenea*), y la función *tr2rpy* de *rvctools*, que convierte a roll-pitch-yaw la rotación de una cierta matriz de transformación homogénea. Un breve extracto del código permite entender el funcionamiento de la función *calcular_cine_directa*:

```
n=size(parametrosDH,1);
T0i=eye(4);
for i=1:n
    A=DH_transHomogenea(pos_q(i)+offset(i),d(i),a(i),alfa(i)); %i-1Ai
    T0i=T0i*A; %A01, A01*A12, A01*A12*A23...
end
T=wTb*T0i; %La matriz completa ha de tener en cuenta la T entre base y
mundo
xyz=T(1:3,4);
rpy=tr2rpy(T);
```

A continuación, el programa resuelve otro problema no trivial debido a la generalidad de la aplicación: la colocación de la cámara simulada que se utiliza en IBVS y PBVS. Es un problema no trivial porque debe hallarse una forma de colocar la cámara que garantice que el eje Z de la cámara coincide con su eje óptico, que a su vez esté alineado con el efector final del robot, y su posición es la de dicho elemento terminal, con un pequeño *offset* que represente el cuerpo de la cámara, todo ello sin que el sistema de referencia del efector final sea siempre el mismo.

La forma en que se ha resuelto este problema es definir primero la posición y orientación del sistema de referencia de la cámara con respecto al mundo, sabiendo que su posición será la del efector final del robot respecto del mundo, y su rotación la necesaria para garantizar que el eje Z es el eje óptico de la cámara y que este está alineado con el efector final del robot (figura 22).

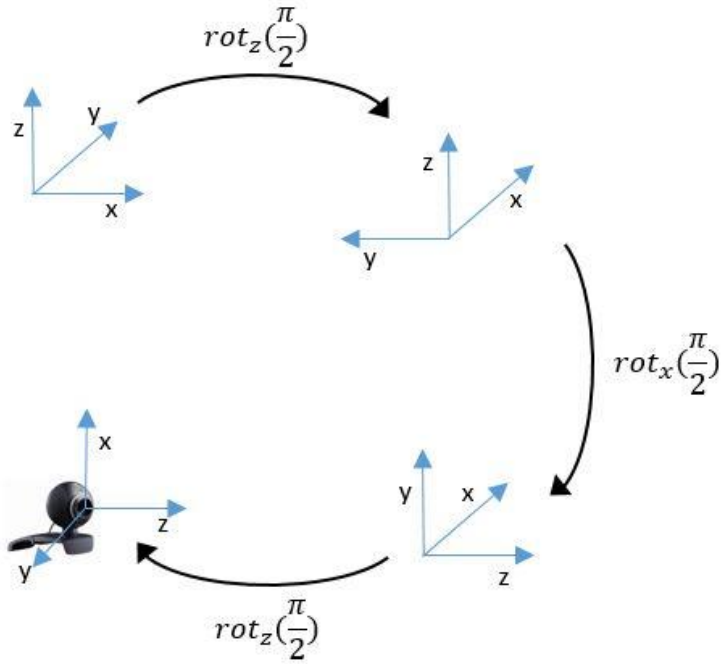


Figura 22-Rotación del sistema de referencia del camera-frame respecto del mundo en el simulador

La única restricción impuesta para ello a la definición del robot es que sus *offset* deben preverse de tal forma que el eje longitudinal de su efector final esté contenido en el plano XZ del simulador cuando todas sus articulaciones tienen el valor 0. Este requisito es sencillo de comprobar visualmente (figura 23), comprobando que cuando todos los ángulos son 0, el brazo está alineado con el eje X.

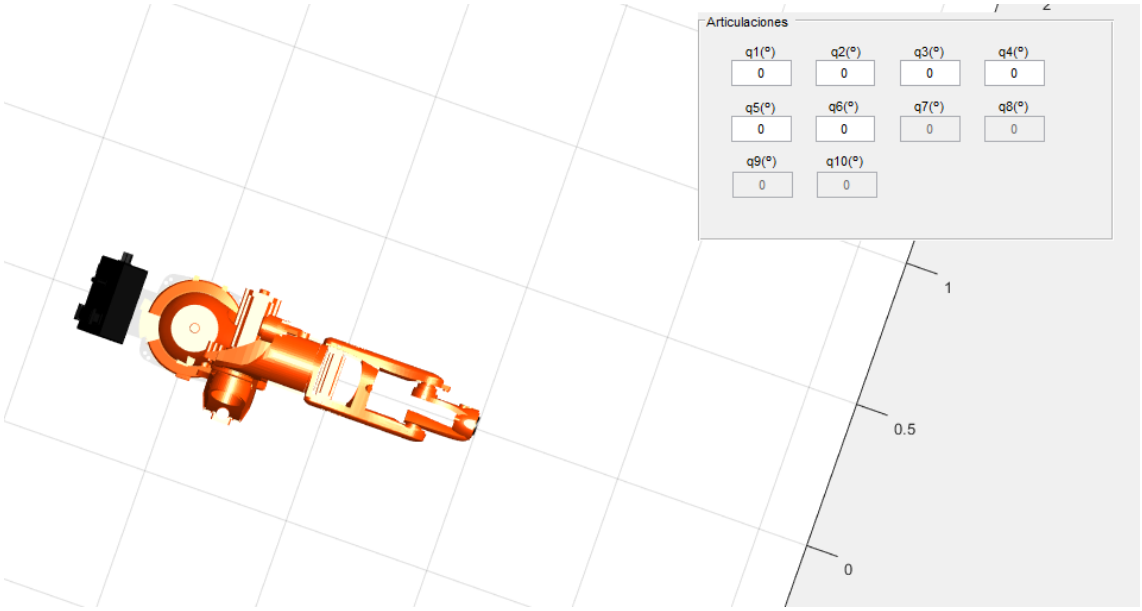


Figura 23-Posición requerida para colocar la cámara correctamente cuando todos los ángulos de las articulaciones se anulan

Con todo ello, se puede obtener la relación entre la cámara y el efector final de forma que $eMc = eMw \cdot wMc = eMw^{-1} \cdot wMc$, donde *wMe* es la matriz de transformación entre el mundo y el efector final. A continuación se muestra el código que realiza todo lo explicado anteriormente:

```
Trob=handles.rob.fkine(zeros(1,numjoints));
wMc=eye(4);
wMc(1:3,4)=Trob(1:3,4);
wMc(1,4)=wMc(1,4)+0.01; %offset de 1 cm
wMc(1:3,1:3)=rotz(pi/2)*rotx(pi/2)*rotz(pi/2);
handles.eMc=inv(Trob)*wMc;
```

Por último, la función *callback* que se está tratando inhabilita las casillas de texto correspondientes al control manual de articulaciones que no existan en el robot elegido, contribuyendo a la generalidad de la aplicación de simulación:

```
handles.wake_edit=[ones(1,numjoints),zeros(1,10-numjoints)];
handles.first_time=1;
guidata(hObject, handles);
edit1_Callback(handles.edit1, eventdata, handles);
```

[…]

```
edit8_Callback(handles.edit8, eventdata, handles);
edit9_Callback(handles.edit9, eventdata, handles);
edit10_Callback(handles.edit10, eventdata, handles);
```

Como se puede observar, la habilitación o inhabilitación de los elementos se ha resuelto llamando a las funciones de *callback* de cada uno, que tiene prevista esta primera llamada. A continuación se

muestra como ejemplo parte del código de una función de *callback* de una de las casillas a habilitar o inhabilitar:

```
handles=guidata(hObject);
if(handles.wake_edit(9)==1)
        if(handles.first_time==1)
            hObject.Enable='on';
            hObject.String=num2str(rad2deg(handles.q(9)));
```

[…]

```
else
    hObject.Enable='off';
```

En la función de *callback* de la última casilla disponible, la correspondiente a la décima articulación, se anula la condición que indica la primera iteración (*handles.first_time*), de manera que ya no se habilitará o inhabilitará una articulación hasta que se cargue un nuevo robot.

En la figura 23 también puede observarse cómo quedan inhabilitadas las casillas correspondientes a articulaciones no disponibles (en ese caso, para un robot de 6 articulaciones).

Es interesante mencionar que para resolver el problema que planteaba este modo de funcionamiento, en primer lugar se utilizó una versión basada en barras deslizadoras, que finalmente fue descartada por su falta de precisión y el resto de inconvenientes que se mencionan en el apartado 3.2 del presente documento.

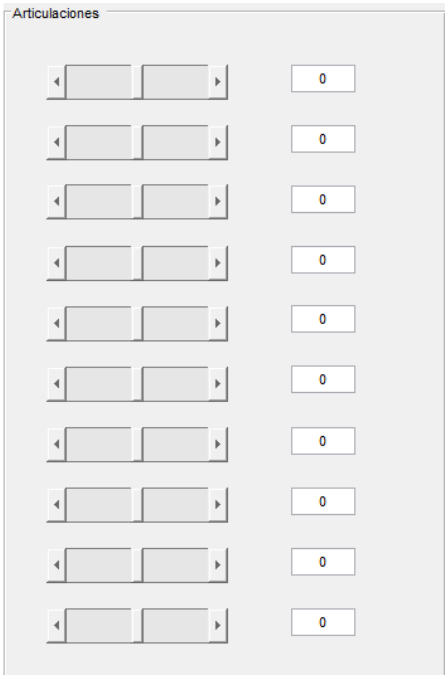


Figura 24-Apariencia del modo manual en las primeras versiones

Por último, cabe señalar que, para mayor facilidad de uso, los *path* y nombres de los archivos de parámetros de los robots presentes en la librería de la aplicación están anotados en un fichero de texto de nombre “Referencias”. La librería completa de robots puede observarse en la figura a continuación:

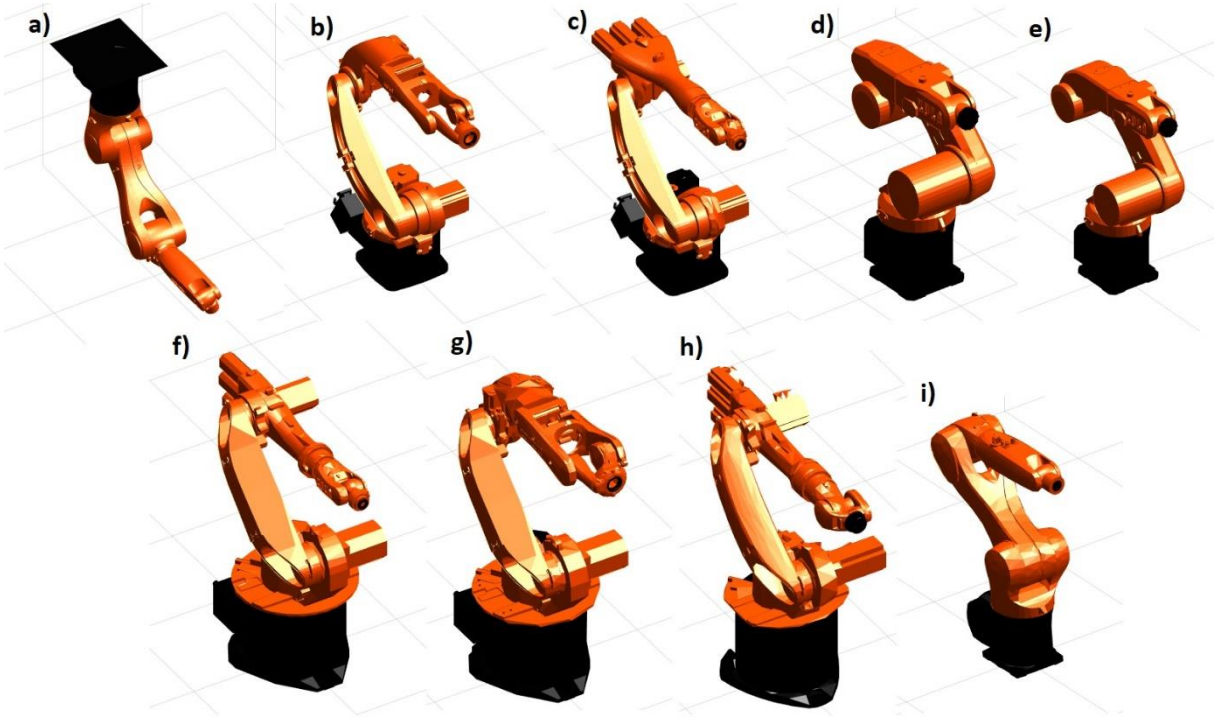


Figura 25-Librería completa de robots de la aplicación. KUKA: a) Agilus 900, b) KR5_2ARC_HW, c) KR5_arc, d) KR5_sixx_R650, e) KR5_sixx_R850, f) KR6_2, g) KR16_arc_HW, h) KR20_3, i) KR10_R900

3.3.2 Selección del modo de funcionamiento:

Esta sección del programa puede explicarse brevemente, dado que esencialmente consiste en un menú desplegable de selección múltiple. Este menú hace visible el panel correspondiente al modo de funcionamiento deseado y hace invisible el resto.

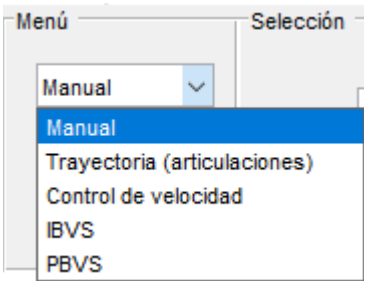


Figura 26-Menú de selección del modo de funcionamiento

Para posibilitar este funcionamiento, en la función de *callback* se incluye el código necesario para dilucidar qué hacer en el caso de seleccionar una u otra opción. Además, en el caso de seleccionar la opción "Manual", también se actualiza el valor que muestra por pantalla en dicha modalidad, como se tratará más adelante. Se reproducen a continuación fragmentos de este código a modo de ejemplo:

```
handles=guidata(hObject);
menu=hObject.String;
val=hObject.Value;
switch menu{val}
    case 'Manual'
```

```
        handles.uipanel3.Visible='off';
```

[…]

```
if handles.wake_edit(1)==1
        handles.edit1.String=num2str(rad2deg(handles.q(1)));
        end
```

Para garantizar que la opción por defecto es la primera del menú desplegable, el código de la función de *callback* se ha replicado en la función de creación del menú.

3.3.3 Modificación manual de la posición articular del robot:

Este modo de funcionamiento, accesible mediante la opción "Manual" del control de selección, permite dotar de forma instantánea a cada articulación del robot de una determinada posición angular, dentro de los límites especificados en el archivo de datos de parámetros del robot.

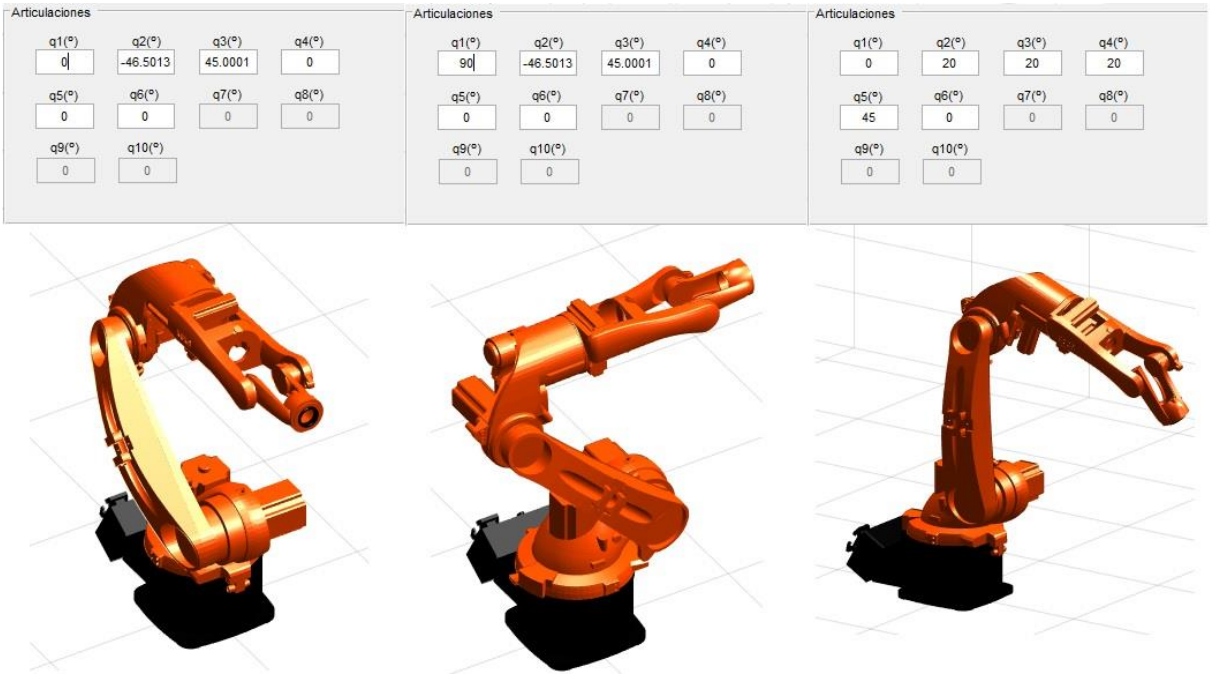


Figura 27-Ejemplo de funcionamiento del modo manual del simulador

Tiene por objetivos la colocación del robot en una cierta posición como forma de facilitar las simulaciones personalizadas, y la exploración de los límites reales del robot, de modo que es la única funcionalidad que incluye límites a las articulaciones del robot. En el resto de modos de funcionamiento esta limitación no se ha incluido, con el objetivo de facilitar la simulación de diferentes casos personalizados de control.

Figura 28-Ejemplo de saturación de una de las articulaciones

La posición angular deseada para una articulación se indica en grados mediante casillas de texto editables, de modo que, en realidad, este modo se basa en el funcionamiento de las funciones de *callback* asociadas a cada casilla de texto.

En dicha función, además de habilitarse o inhabilitarse durante su primera llamada, como se ha visto en el apartado 3.3.1 del presente documento, hace adquirir la posición indicada mediante la propiedad de *SerialLink* "*animate*", comprobando antes si dicha posición se encuentra dentro de los límites del actuador y, en caso contrario, haciéndolo saturar.

Para reducir el código utilizado en la comprobación de los límites, en lugar de comprobarlos obteniendo comprobando los límites con un vector creado apropósito, se ha utilizado la propiedad de *SerialLink* denoimnada "*islim*", que simplemente devuelve un valor de 1 si es el límite superior y -1 si es el inferior. A continuación, se reproduce parte del código de una de las funciones de *callback* de las casillas editables a modo de ejemplo:

```
handles.q(1)=deg2rad(str2num(hObject.String));
is_lim=handles.rob.islimit(handles.q);
if is_lim(1,1)==-1
    handles.q(1)=deg2rad(handles.qlim(1,1));
elseif is_lim(1,1)==1
    handles.q(1)=deg2rad(handles.qlim(1,2));
end
handles.rob.animate(handles.q);
```

Además, la casilla editable de texto muestra de forma actualizada la posición angular que tiene su articulación correspondiente, tanto durante la ejecución normal de este modo de funcionamiento, como en su primera llamada, como cuando se vuelve a este modo después de haber utilizado otro.

3.3.4 Modificación de la posición articular del robot mediante trayectoria personalizada:

En esta opción sencillamente se le proporciona a la aplicación un archivo de datos de *Matlab* (*mat*) una matriz, de nombre M, que contenga la trayectoria a desempeñar por las articulaciones del robot (cada fila, un instante; cada columna, una articulación), y el periodo de muestreo, T. El simulador simplemente representará cada una de estas configuraciones respetando el periodo de muestreo en la animación.

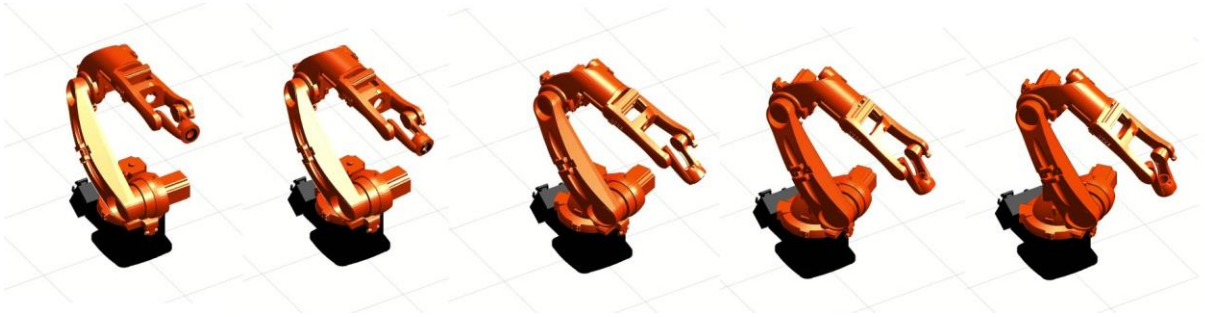


Figura 29-Ejemplo de trayectoria articular personalizada

Las instrucciones necesarias para ejecutar la animación se encuentran en la función de *callback* del pulsador con el que se lanza la simulación, tal y como se muestra en la figura 30. La trayectoria articular se debe introducir en grados.

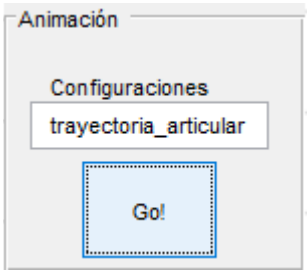


Figura 30-Panel de control del modo de funcionamiento de trayectoria articular

La función de *callback* llama a su vez para realizar la animación a una función externa desarrollada apropósito para el presente trabajo académico, denominada *anima_trayectoria*, que comprueba si el número de articulaciones implícito en la matriz de la trayectoria articular es coherente con el número de articulaciones del robot simulado e impide la simulación en caso contrario.

En esta sección no se utiliza ninguna funcionalidad que no haya sido explicada ya o que no sea básica en *Matlab*, de modo que no se reproducirán fragmentos de código aclarativos.

3.3.5 Control cinemático de velocidad:

Esta sección del programa se corresponde con la simulación del control cinemático de velocidad para tareas de posicionado de una referencia fija y de seguimiento de una referencia móvil (con *feedforward* de la velocidad de referencia) tratado en el apartado 2.3.3 del presente documento.

Para ello, utilizando las herramientas de la interfaz gráfica que ya se han ido explicando, ofrece en su panel de control, mostrado en la figura 31, casillas editables para personalizar la referencia y la ganancia proporcional del control, así como indicadores que muestran la posición actual del robot para poder compararla con la de referencia.

La simulación se pone en marcha cuando se acciona el pulsador del panel, cuya función de *callback* contiene los bucles de control tanto para la tarea de posicionado como para la tarea de *tracking*. Para realizar tareas de posicionado, la referencia se especifica en una casilla editable para cada coordenada de posición y orientación, mientras que en el caso de referencia móvil, es necesario proporcionar la

trayectoria en forma de archivo de datos de *Matlab* (*mat*) que contenga una matriz con nombre *p_ref* que contenga en cada fila el valor de las coordenadas de posición y orientación a cada instante.

En cuanto al periodo de muestreo, en el caso de posicionado se fija en el código de la función de *callback*, siendo su valor por defecto 0.1, que permite una simulación aproximadamente a tiempo real; mientras que en el caso de *tracking*, el periodo de muestreo debe proporcionarse en el mismo archivo de datos que la trayectoria de referencia, en segundos y con el nombre *ts*.

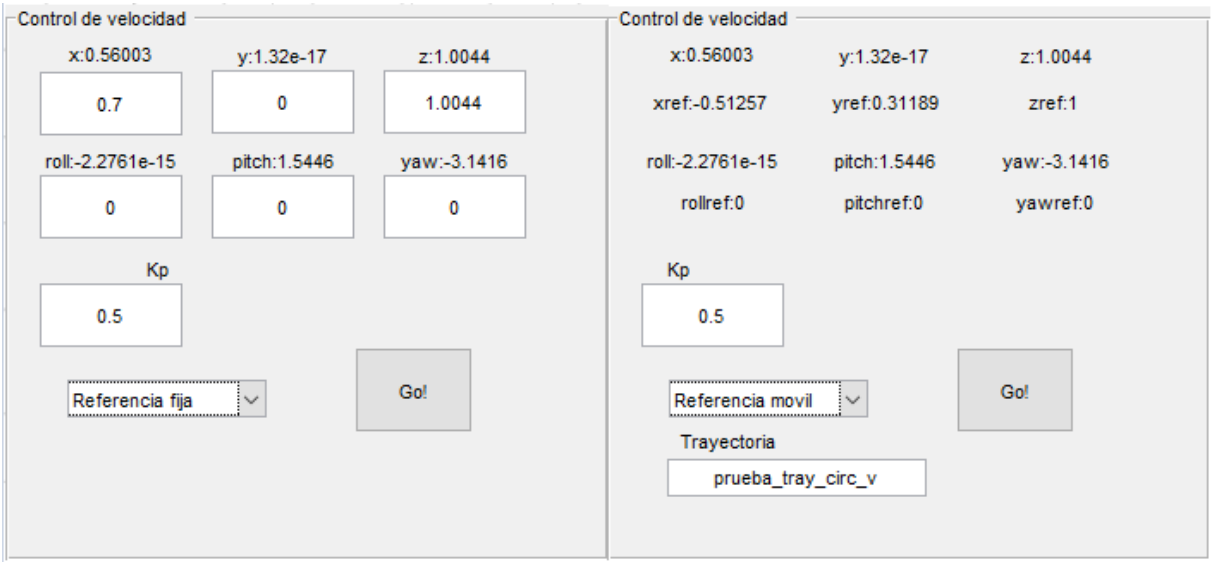


Figura 31-Panel de control de velocidad. Izq) tarea de posicionado, der) tarea de tracking.

Para indicar la orientación, en el presente trabajo académico se ha optado por utilizar la notación *roll, pitch, yaw*, puesto que es una notación intuitiva visualmente. Concretamente, las funciones de *rvctools* que trabajan con esta notación angular asocian por defecto *roll* a giro entorno a X, *pitch* entorno a Y y *yaw* entorno a Z, como se muestra en la figura 32.

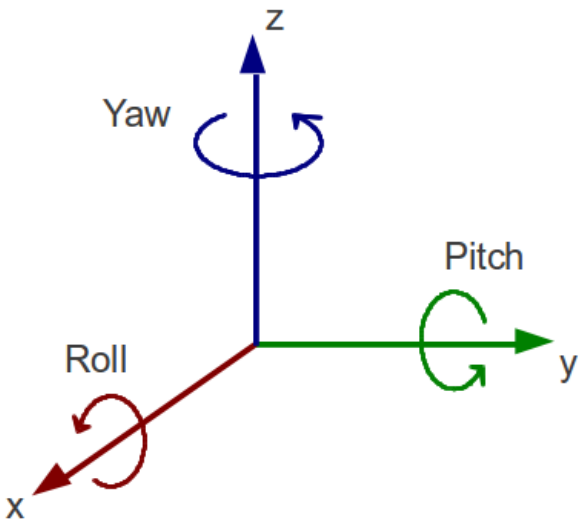


Figura 32 [18]-Roll Pitch Yaw de acuerdo con la notación de rvctools

En cuanto a la implementación, es de utilidad reproducir algunos fragmentos del código que resultan clave para comprender el funcionamiento de la simulación. No se volverán a explicar aquellos aspectos del programa que son análogos a los ya tratados anteriormente, como casillas de texto editable, indicadores de texto o menús de selección múltiple.

En la tarea de posicionado, correspondiente a la selección "Referencia fija" del menú del panel de control de velocidad, los elementos principales del bucle de control son los siguientes:

```
tau=1/Kp;
t_est=4*tau;
```

[…]

```
hold on;
ref=plot3(p_ref(1),p_ref(2),p_ref(3),'Marker','*','MarkerSize',10,'Color',[
1 0 0]);
```

[…]

```
i=0;
tiem_transc=0;
while (norm_error>0.01) && (tiem_transc<4*t_est)
    tic
    J0=jacob0(handles.rob,handles.rob.getpos,'rpy');
    Ji=pinv(J0);
    q_p=Ji*Kp*(p_ref-p_act);
    q=q+per*q_p;
    handles.q=q';
    handles.rob.animate(handles.q);
    T=fkine(handles.rob,handles.rob.getpos);
    xyz=T(1:3,4);
    rpy=tr2rpy(T);
    p_act=[xyz;rpy(1);rpy(2);rpy(3)];
[…]
    error=p_ref-p_act;
    norm_error=norm(error);
    tc=toc;
    t_total=t_total+tc;
    i=i+1;
    tiem_transc=per*i;
    pause(per-tc);
end
```

Es decir, la tarea de posicionado dura mientras la norma del error sea mayor que un cierto umbral aceptable y mientras el tiempo transcurrido (simulado, no real), sea menor de 4 veces el esperado, definiéndose el tiempo de establecimiento deseado como 4 veces la constante de tiempo (que es la inversa de la ganancia proporcional), lo que corresponde al tiempo de establecimiento del 98%. Esta segunda condición simplemente pretende poder simular qué ocurra cuando se le exige al robot llegar a una posición que no puede alcanzar, pero sin que el programa quede retenido en un bucle sin fin.

El cálculo de la matriz Jacobiana geométrica aprovecha una función de *rvctools* que permite además transformarla a notación *roll-pitch-yaw*, y, por lo demás, el control toma exactamente las mismas

expresiones que se trataron en el apartado 2.3.3 del presente documento. Para el cálculo de la cinemática directa, se ha recurrido a la propiedad de *SerialLink,* encargada de ello, *fkine*, obteniendo la posición angular de las articulaciones mediante otra propiedad, *getpos*.

En caso de no alcanzarse la posición deseada, el programa avisa de ello y reinicia la simulación recurriendo a la función de *callback* de carga del robot, en un fragmento de código a continuación del representado y que no se reproduce, pero puede consultarse en el anexo de código del presente Trabajo Fin de Máster.

Por último, cabe mencionar que la referencia aparece como un asterisco rojo representado tridimensionalmente en el mismo espacio que el robot, como se observa en el código anteriormente reproducido.

Figura 33-Posicionado con control de velocidad (referencia [0,0.7,1.0044,0,0,0])

Por otro lado, en la tarea de *tracking*, correspondiente a la selección de "Referencia móvil" del menú del panel de control de velocidad, los elementos principales del bucle de control son los siguientes:

```
N=size(p_ref,1);
hold on;
ref=plot3(p_ref(1,1),p_ref(1,2),p_ref(1,3),'Marker','*','MarkerSize',10,'Co
lor',[1 0 0]);
for i=1:N
```

[…]

```
    ref.XData=p_ref(i,1);
    ref.YData=p_ref(i,2);
    ref.ZData=p_ref(i,3);
    J0=jacob0(handles.rob,handles.rob.getpos,'rpy');
    Ji=pinv(J0);
    if i<N
    dp_ref=(p_ref(i+1,:)-p_ref(i,:))/per;
    elseif i==N
    dp_ref=0;
    end
    e=p_ref(i,:)'-p_act;
    q_p=Ji*Kp*e+Ji*dp_ref';
    q=q+per*q_p;
    handles.q=q';
```

```
    handles.rob.animate(handles.q);
    T=fkine(handles.rob,handles.rob.getpos);
    xyz=T(1:3,4);
    rpy=tr2rpy(T);
    p_act=[xyz;rpy(1);rpy(2);rpy(3)];
```

[…]

```
end
```

Como se observa, las principales diferencias, además de la ley de control que ahora incorpora la pre alimentación, como en la ecuación {2.3.3.3}, está en que el bucle de control tiene una duración delimitada por la propia trayectoria personalizada y en que, debido a que el término de *feedforward* se calcula de manera ideal (conocida la trayectoria, como en la expresión {2.3.3.4}), es necesario considerar para la última iteración del bucle la situación posterior a que haya finalizado la trayectoria, en la que, evidentemente, la velocidad de la referencia es 0.

Por otro lado, también se diferencia del caso de posicionado en que el asterisco rojo que representa a la referencia es actualizado en cada iteración de acuerdo con la trayectoria personalizada introducida por el usuario.

Figura 34-Tracking de una trayectoria en arco de circunferencia y orientación [0,0,0] (rpy)

Por último, al término de cada simulación, la aplicación genera gráficas de la evolución del error de posición y la velocidad de las articulaciones.

3.3.6 Control cinemático de aceleración:

Esta sección del programa se corresponde con la simulación del control cinemático de aceleración para tareas de posicionado de una referencia fija y de seguimiento de una referencia móvil (con *feedforward* de la aceleración de referencia) tratado en el apartado 2.3.4 del presente documento.

El funcionamiento y el desarrollo de la implementación en esta sección son esencialmente iguales a las de la sección correspondiente al control de velocidad, siendo las únicas salvedades aquellas que se corresponden con las ecuaciones necesarias para el control.

Por ello, el panel de control se utiliza de manera análoga al caso 3.3.5, salvo que en esta ocasión es necesario que el usuario proporcione no solo la ganancia proporcional para la posición (Kp) sino también para el error de velocidad (Kv).

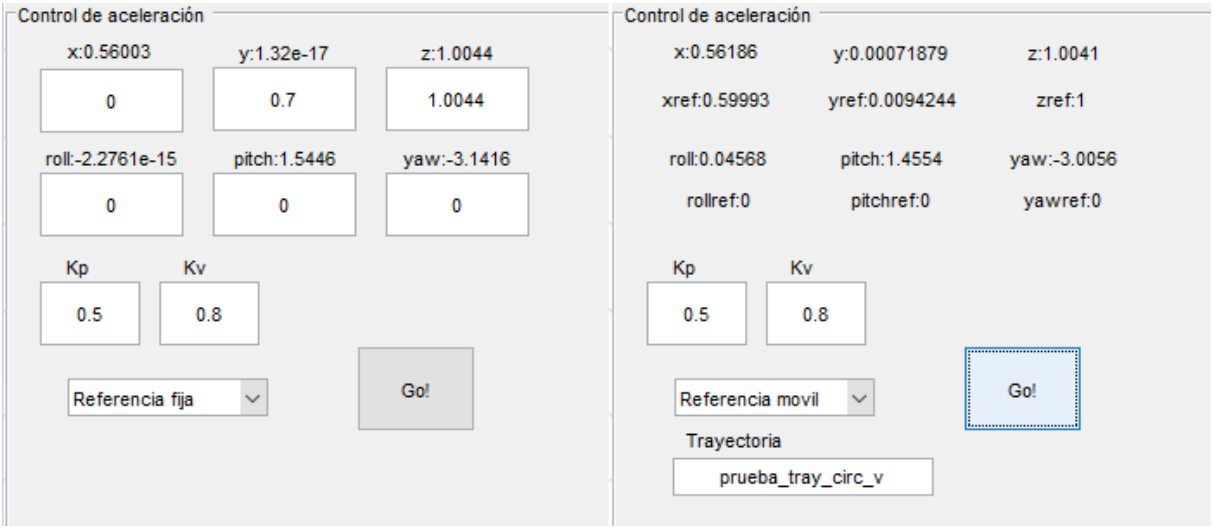


Figura 35-Panel de control de aceleración. Izq) tarea de posicionado, der) tarea de tracking

A nivel de código, la tarea de posicionado es idéntica al caso del control de velocidad, salvo porque la ley de control y su integración vienen dadas por las expresiones {2.3.4.3} y {2.3.4.4} y porque en el bucle de control de la tarea de posicionado ya no existe la condición del tiempo de establecimiento. Esto significa que, en caso de demandarse una referencia fuera del alcance del robot, quedará atrapado en el bucle de control hasta que el usuario reinicie la simulación (pulsador de carga del robot) o cierre la aplicación y vuelva a abrirla.

A continuación, se reproducen las partes más representativas del posicionado mediante control de aceleración, eliminando aquello que sea común al caso del control de velocidad:

```
p1=p;
dq1=0;
J01=jacob0(handles.rob,handles.rob.getpos,'rpy');
while (norm_error>0.01)
```

[…]

```
    J0=jacob0(handles.rob,handles.rob.getpos,'rpy');
    Ji=pinv(J0);
    dp_ref=0; %Referencia fija
    dp=(p-p1)/per;
    ep=dp_ref-dp;
    Jp=(J0-J01)/per;
    d2q=Ji*Kv*ep+Ji*Kp*e-Ji*Jp*dq1;
    d2q=d2q(:,1);
    dq=dq1+per*d2q;
    q=q+per*dq1+0.5*(per^2)*d2q;
    handles.q=q';
    handles.rob.animate(handles.q);
    T=fkine(handles.rob,handles.rob.getpos);
    xyz=T(1:3,4);
    rpy=tr2rpy(T);
    p1=p;
    dq1=dq;
```

```
    J01=J0;
    p=[xyz;rpy(1);rpy(2);rpy(3)];
    e=p_ref-p;
```

[…]

```
end
```

Es de interés comprobar cómo se cumple que la aceleración de las articulaciones en el instante de control (d2q) se calcula con la velocidad de las articulaciones del instante anterior (dq1), cómo la derivada del Jacobiano geométrico se calcula numéricamente (Jp=(J0-J01)/per), así como la actualización de variables que es necesario realizar al final del bucle.

En el caso del *tracking*, ocurre de una manera similar, introduciéndose además la pre alimentación de la aceleración. Con el fin de sintetizar lo más posible, únicamente se reproducen a continuación las partes del código correspondientes a la ley de control en el caso de *tracking*, dado que las diferencias con el caso de posicionado a nivel informático son las mismas que en el control de velocidad:

```
for i=1:N
```

[…]

```
    J0=jacob0(handles.rob,handles.rob.getpos,'rpy');
    Ji=pinv(J0);
    if i<N
    dp_ref=(p_ref(i+1,:)-p_ref(i,:))/per;
    elseif i==N
    dp_ref=0;
    end
    dp=(p-p1)/per;
    if i<N-1
    dp_ref_1=(p_ref(i+2,:)-p_ref(i+1,:))/per;
    else
    dp_ref_1=0;
    end
    d2p_ref=(dp_ref_1-dp_ref)/per; %Feedforward aceleración
    e=p_ref(i,:)'-p;
    ep=dp_ref'-dp;
    Jp=(J0-J01)/per;
    d2q=Ji*d2p_ref'+Ji*Kv*ep+Ji*Kp*e-Ji*Jp*dq1;
    dq=dq1+per*d2q;
    q=q+per*dq1+0.5*(per^2)*d2q;
```

[…]

```
end
```

Nótese cómo, igual que en el control de velocidad se requería contemplar el caso de la última iteración para igualar la velocidad de la referencia a 0, en este caso es necesario hacer lo mismo con la aceleración de la referencia en la penúltima iteración, puesto que el cálculo numérico de la aceleración requiere la velocidad de la trayectoria en el instante actual y la velocidad de la trayectoria en el instante inmediatamente futuro, y, dado que estas velocidades se expresan a su vez a partir del instante en que

se quieren obtener y el instante siguiente (expresión {2.3.3.4}), ello implica que se necesita acceder a la posición de referencia dos instantes en el futuro.

No se representan en este apartado imágenes del control de aceleración porque no serían distinguibles de las del control de velocidad, de modo que se abordarán más adelante mediante vídeos en el apartado de desempeño de los controles simulados.

3.3.7 Control IBVS *eye-in-hand*:

Esta sección del programa se corresponde con la simulación del control IBVS para tareas de posicionado de una referencia fija y de seguimiento de una referencia móvil tratado en los apartados 2.4.1 y 2.4.2 del presente documento.

La implementación de la simulación de los controles de *Visual Servoing,* particularmente del IBVS, es la más compleja, dado que implica simular una referencia solo conocida a través de una imagen. Es por ello que conviene aclarar una serie de consideraciones previas:

- La cámara simulada no cuenta con representación gráfica. Es un sistema de referencia no visible calculado en el momento de la carga del robot (ver apartado 3.3.1) para asegurar que está situado sobre el eje longitudinal del elemento terminal del robot, 1 cm adelantado respecto del sistema de referencia del elemento terminal, con su eje Z coincidiendo con el eje óptico de la cámara simulada.
- El *target* es siempre un conjunto de 4 puntos cuya posición es conocida. El programa cuenta con dos modalidades de *tracking* implementadas: una ideal en la que la trayectoria del *target* también es conocida, de modo que la pre alimentación de su velocidad es perfecta, y otra más similar al caso real, en la que la trayectoria es desconocida y por tanto la velocidad del *target* debe ser estimada según la expresión {2.4.1.12} vista anteriormente. La posición y orientación del *target* es gestionada mediante matrices de transformación homogénea que indican la posición y orientación del centro del *target* respecto del mundo (*wMp*) a cada instante.

Debido a ello, existen una serie de consideraciones específicas para la implementación del caso IBVS:

- La proyección de los 4 puntos del *target* sobre el plano imagen de la cámara se simula obteniendo primero las coordenadas de cada uno de los 4 puntos respecto del sistema de referencia de la cámara, $cMo = cMw \cdot wMo = inv(wMc) \cdot wMo = inv(wMe \cdot eMc) \cdot wMo$, donde "*o*" hace referencia al *target*, "*w*" al sistema mundo, "*e*" al efector final del robot y "*c*" a la cámara.
- Una vez obtenidas estas coordenadas, la profundidad se calcula de forma ideal, como la distancia euclidiana entre el origen de referencia de la cámara y cada uno de los 4 puntos, no simplemente como la coordenada Z de *cMo*.
- Por último, las coordenadas de los 4 puntos sobre el plano imagen se obtienen dividiendo entre la profundidad según la expresión {2.4.2.1}.
- Para dotar de mayor realismo a la simulación, si el *target* abandona el campo visual de la cámara simulada, el programa termina. Dicho campo visual se obtiene a partir de los parámetros intrínsecos de la cámara simulada, que son utilizados para obtener los límites del plano imagen, a partir de las expresiones de conversión píxel-coordenada del plano imagen

vistas en el apartado 2.4.2, de IBVS. Además, en esta línea, si la coordenada Z de *cMo* es negativa, el programa también termina, dado que significa que el *target* está detrás de la cámara, y por tanto ha abandonado el campo visual.

Aclaradas estas consideraciones previas sobre la implementación del control IBVS, ya es posible abordar más en detalle el funcionamiento de la aplicación.

El control IBVS implementado cuenta con un panel de control en que, mediante menús desplegables, se pueden seleccionar diferentes opciones respecto a la ley de control y respecto al *target*.

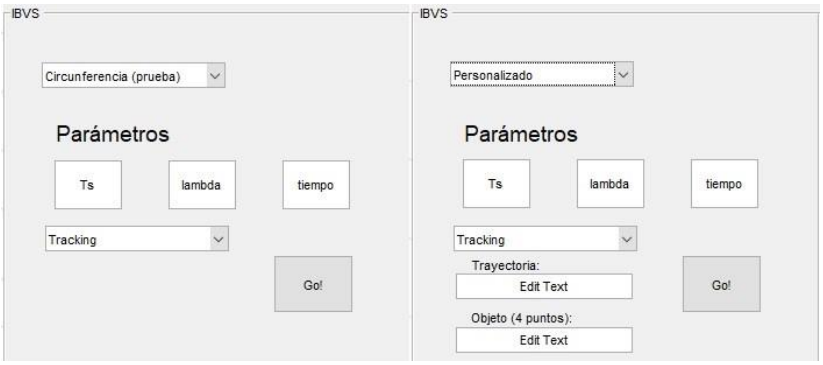


Figura 36-Panel de control IBVS. Izq) Trayectoria de prueba, centro) Modo personalizado, der) Posibles leyes de control

Las opciones disponibles en cuanto al *target* son:

- Trayectoria de prueba "Circunferencia": carga un objeto formado por 4 puntos separados 10 cm entre sí y situados sobre el plano XY, entorno al punto (0,0,0), y carga también la trayectoria que describirá dicho objeto. Esta trayectoria comenzará con el centro del objeto a la altura (posición Z del sistema de referencia global) del end-effector del robot, sobre el plano YZ y con un offset en X de 0.6 m respecto a la cámara del robot. Este offset pretende conseguir que haya una cierta profundidad entre la cámara y el objeto, de modo que es recomendable que la prueba parta con un robot alineado con el eje X. La trayectoria será una circunferencia alrededor del robot y con posición constante en Z global.
- Trayectoria de prueba "Corona": caso análogo a la circunferencia, pero con una variación periódica de Z de ±5 cm entorno a su punto de partida.
- Caso personalizado: El *target* y la trayectoria deben ser proporcionados por el usuario como archivos de datos de *Matlab*. La trayectoria debe ser una estructura que contenga una matriz, llamada "*wMp*", que exprese la posición y orientación del centro del *target* respecto del sistema de referencia mundo a cada instante. El *target* debe ser una matriz, llamada "P", que en cada fila indique las coordenadas XYZ de cada uno de los puntos que lo forman.

Y las opciones disponibles en cuanto a leyes de control son:

- Posicionado: Es un control proporcional tal y como se describe en la expresión {2.4.1.11} sin término de movimiento del *target* ($\frac{\partial e}{\partial t} = 0$). Puede utilizarse también para la persecución de *targets* en movimiento, dado que el bucle de control, a diferencia de en los controles de velocidad y aceleración, es el mismo.
- *Tracking*: Es un control proporcional con pre alimentación de la estimación del efecto del movimiento del *target* sobre el error, tal y como se describe en las expresiones {2.4.1.11} y

{2.4.1.12}.

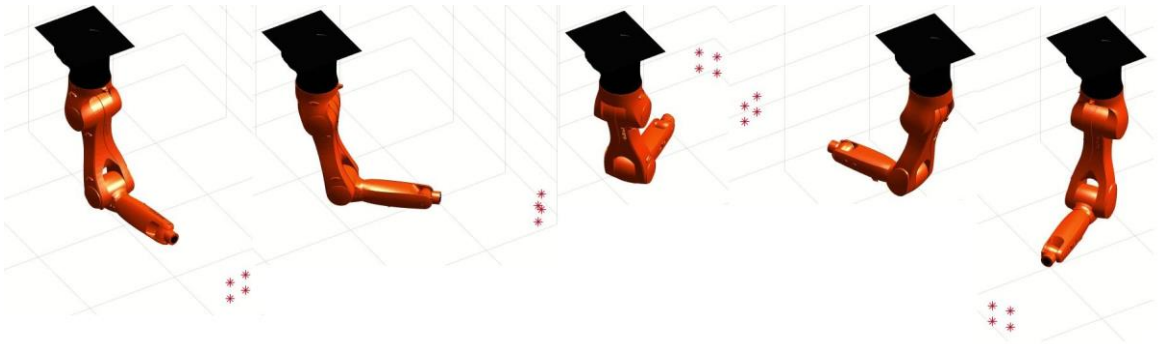


Figura 37- Secuencia de tracking IBVS

Como consecuencia de que lo único que cambie entre la tarea de posicionado y la de *tracking* sea la ley de control utilizada, la referencia debe proporcionarse siempre como una estructura de matrices de transformación *wMp*, aunque en el caso de un *target* fijo esta *wMp* sea la misma en todos los instantes.

A nivel visual, el simulador incorpora dos funcionalidades por defecto que contribuyen a comprobar que el funcionamiento del control IBVS sea correcto:

- Los puntos del *target* se muestran de forma actualizada como asteriscos rojos a lo largo de la simulación en el mismo espacio tridimensional que el robot.
- El simulador genera automáticamente una ventana donde se observa el control desde el punto de vista de la cámara situada en el elemento terminal del robot, de manera que la referencia aparece como 4 circunferencias azules y el *target* proyectado sobre el plano imagen como 4 asteriscos rojos.
- Al finalizar la simulación, se generan automáticamente gráficas que muestran la evolución de las variables del vector *s*, del error y de la estimación de $\frac{\partial e}{\partial t}$.

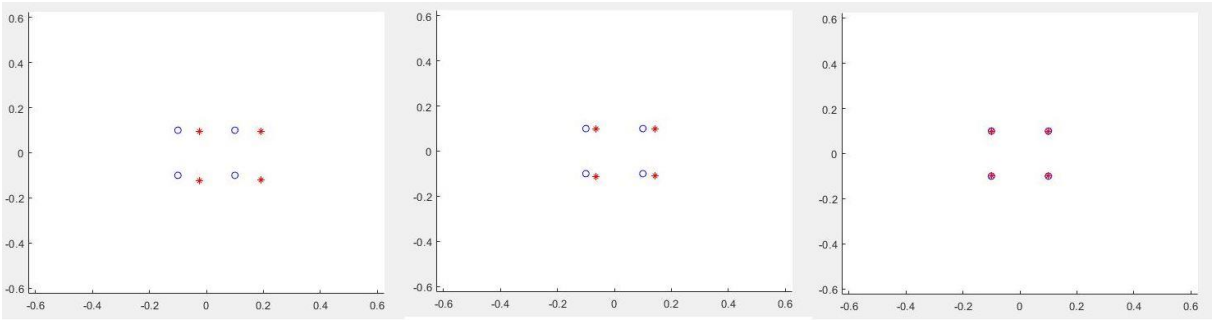


Figura 38- Punto de vista de la cámara, tracking IBVS

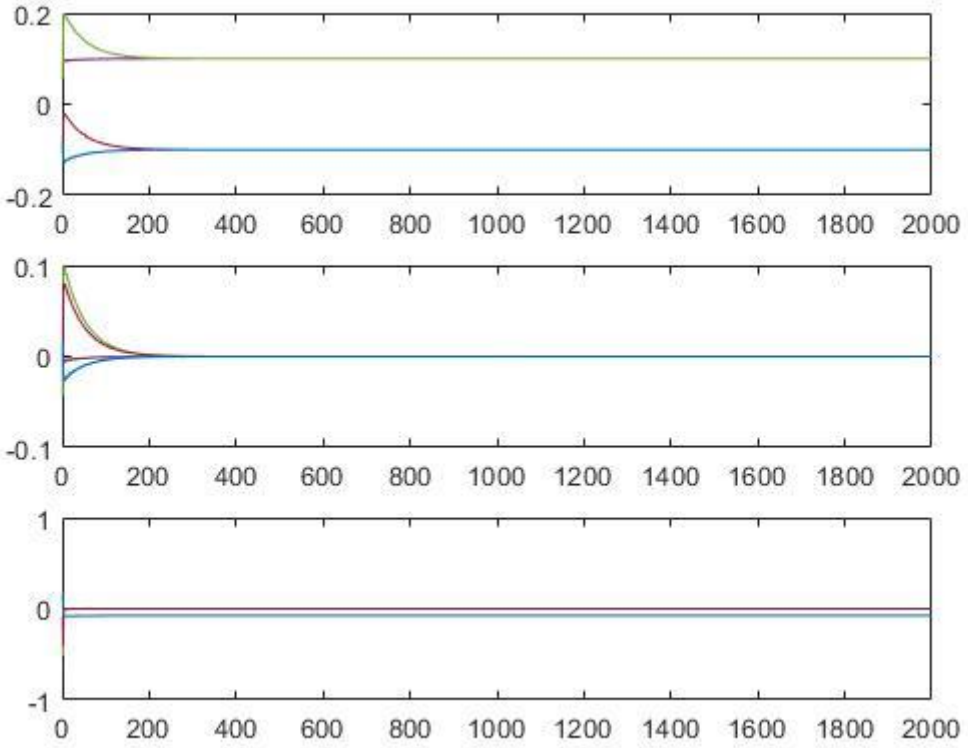


Figura 39- Ejemplo de gráficas generadas al finalizar la simulación. Desde arriba: variables de s, error y variación del error debida al movimiento del target

Una vez explicado el funcionamiento de la simulación a nivel cualitativo, es posible pasar a explicar los aspectos más significativos de su implementación informática. Para mayor brevedad, no se expondrán en el orden en que aparecen en el programa, dándose por hecho además el funcionamiento de elementos similares ya explicados en los apartados correspondientes a otras secciones del simulador.

-En primer lugar, las instrucciones que permiten la simulación del control IBVS se encuentran en el pulsador del panel de control. El fragmento del código que se encarga de aplicar las leyes del control es el siguiente:

```
sref=[-0.1; -0.1;0.1; -0.1;-0.1 ;0.1;0.1 ;0.1];
while i<=N && out~=0
    wMe=handles.rob.fkine(handles.rob.getpos);
    wMc=wMe*eMc;
```

[…]

```
%Coordenadas en el plano imagen:
    [XYZc,out]=coord_cam(wMc,wMo);
    Z=calcularZ(XYZc);
    x=[XYZc(1,1)/Z(1) XYZc(1,2)/Z(2) XYZc(1,3)/Z(3) XYZc(1,4)/Z(4)];
    y=[XYZc(2,1)/Z(1) XYZc(2,2)/Z(2) XYZc(2,3)/Z(3) XYZc(2,4)/Z(4)];
```

[…]

```
    L=calcularL_IBVS(x,y,Z);
    s=[x(1);y(1);x(2);y(2);x(3);y(3);x(4);y(4)];
```

[…]

```
    e=s-sref;
    Js=L*cXn*handles.rob.jacobn(handles.rob.getpos);
    Jsi=pinv(Js);
    switch mod_ibvs
        case 0 %posicionado
            q_p=-lambda*Jsi*e;
        case 1 %tracking
            e_p=((e-e1)/ts)-Js*q_p1;
            q_p=-lambda*Jsi*e-Jsi*e_p;
    end
    q=q+q_p*ts;
    handles.q=q';
    handles.rob.animate(handles.q);
```

Donde las coordenadas de los 4 puntos respecto del *target*, XYZc, se obtienen mediante la función *coord._cam*, desarrollada apropósito para el presente trabajo, cuyo código puede resumirse mediante el siguiente extracto:

```
cMw=inv(wMc);
cMw4=blkdiag(cMw,cMw,cMw,cMw);
cMo=cMw4*wMo;
XYZ=[cMo(1:3,4),cMo(5:7,8),cMo(9:11,12),cMo(13:15,16)];
```

Es decir, aplica la misma operación a la matriz de posición y orientación de cada punto del *target* para obtener sus coordenadas respecto del *frame* de la cámara simulada.

Además, esta función también indica al bucle de control si el objeto ha abandonado el campo visual porque se ha situado tras la cámara (Z<0), caso en el que da como salida *out=0*, dejándose de cumplir la condición para que el bucle de control continúe (*out*≠0). Por ser trivial, no se reproduce el fragmento de código que realiza dicha comprobación.

La profundidad se calcula con el criterio que ya se ha indicado (distancia euclidiana entre cada punto y el sistema de la cámara), pero mediante una función externa desarrollada apropósito para el presente trabajo, *calcularZ*. Como en ese punto del programa ya se cuenta con las coordenadas de cada punto respecto del sistema de la cámara, simplemente hay que calcular la norma euclidiana de cada vector origen de la cámara- punto. De nuevo, no se reproduce el código por ser trivial.

La matriz de interacción, L, se calcula según la expresión {2.4.2.5} para cada uno de los puntos mediante una función externa desarrollada para el presente trabajo, *calcularL_IBVS*, utilizando las coordenadas (x, y) normalizadas en el plano imagen y la profundidad. Aunque el simulador trabaja siempre con *targets* formados por 4 puntos, la función para calcular L está preparada para trabajar con cualquier número de puntos:

```
npoints=length(x);
Lprov=[];
for i=1:npoints
    Lx=[-1/Z(i) 0 x(i)/Z(i) x(i)*y(i) -(1+x(i))^2 y(i);
        0 -1/Z(i) y(i)/Z(i) 1+y(i)^2 -x(i)*y(i) -x(i)];
    Lprov=[Lprov;Lx];
end
```

```
L=Lprov;
```

Por lo demás, el control aplica las leyes de control explicadas anteriormente, con la ganancia proporcional (λ) es indicada desde la interfaz gráfica por el usuario mediante casilla de texto editable.

Es de interés detenerse en el caso de *tracking* ideal (*case 2* en el código), donde se comprueba que para calcular la derivada del error exacta, es necesario obtener el vector de características visuales en el instante inmediatamente futuro (*sf*), pero evidentemente respecto a la posición actual del robot, de modo que la proyección a calcular previamente es la de las coordenadas futuras según la trayectoria indicada por el usuario (*wMo_f*) respecto de la posición actual de la cámara (*wMc*).

-En segundo lugar, la gestión de la posición y orientación del *target* implica primero definir dicho *target* y después aplicar a cada instante la transformación del centro del *target* respecto del mundo a cada uno de sus 4 puntos.

La definición del *target* es común en las opciones de las trayectorias de prueba, y se consigue mediante la siguiente instrucción:

```
P=[-0.05 -0.05 0;0.05 -0.05 0;-0.05 0.05 0;0.05 0.05 0];

wMobject=blkdiag(transl(P(1,1),P(1,2),P(1,3)),transl(P(2,1),P(2,2),P(2,3)),
transl(P(3,1),P(3,2),P(3,3)),transl(P(4,1),P(4,2),P(4,3)));
```

De modo que es un plano de puntos separados 5 mm, definido como situado sobre el plano XY. Cada uno de los puntos se expresa como una matriz de transformación homogénea solo de traslación insertada como elemento de una matriz diagonal por bloques *wMobject*.

En el caso en que la trayectoria es personalizada, el *target* P se indica desde la interfaz gráfica mediante una casilla de texto editable, mientras que la definición de *wMobject* es la misma que en los casos de prueba.

Por otro lado, la trayectoria se define en los casos de prueba mediante funciones externas desarrolladas para el presente trabajo, y que pueden ser editadas en caso de querer realizar variaciones en ellas. Estas funciones son, respectivamente, *genera_tray_circunf* y *genera_tray_corona*.

Estas funciones toman como entradas la posición de la cámara respecto del mundo (*wMc)*, el tiempo de simulación y el periodo de muestreo deseados (indicados por el usuario desde la interfaz gráfica), calculan con ello las N iteraciones que durará en consecuencia la simulación, y con ello obtienen la trayectoria del centro del *target*, en el formato ya explicado de estructura de matrices de transformación.

A continuación, se reproduce el código más significativo de la función *genera_tray_circunf* para ilustrar su funcionamiento, de acuerdo a lo explicado anteriormente:

```
N=T/periodo;
xyz=wTc(1:3,4);
Tinit=transl(xyz(1)+0.6,xyz(2),xyz(3)); %Posición inicial: offset en x
respecto a la cámara
T_provisional=[];
R_ini=roty(pi/2); %Z hacia fuera, coincidente con cámara
Tinit(1:3,1:3)= rotx(pi)*R_ini;
Tobj=eye(4);
```

```
for i=1:N
    q1=(2*pi*(i-1)/N); %Gira a la velocidad especificada
    Tobj(1:3,1:3)=rotz(q1);
    T_provisional(:,:,i)=Tobj*Tinit;
end
T_provisional(:,:,N+1)=T_provisional(:,:,N);
wTp=T_provisional;
```

La función para el caso de la corona simplemente incorpora una variación en Z que es ocho veces más rápida que el desplazamiento en X e Y:

```
q2=16*pi*(i-1)/N;
pz=0.05*sin(q2);
```

En el caso en que se desee utilizar una trayectoria personalizada, el formato debe ser el mismo, y debe calcularse para que sea coherente con el tiempo de simulación y el periodo de muestreo que se indiquen desde la interfaz gráfica.

Una vez dentro del bucle, se obtiene la posición y orientación de los puntos del *target* a cada instante combinando esa trayectoria de la estructura *wMp* con la matriz *wMobject*:

```
wMo=blkdiag(wMp(:,:,i),wMp(:,:,i),wMp(:,:,i),wMp(:,:,i))*wMobject;
```

-En tercer lugar, la simulación del funcionamiento de la cámara y de su campo visual, más allá de lo explicado, trabaja por defecto con unos parámetros intrínsecos arbitrariamente elegidos:

```
resol_h = 640;
resol_v = 480;
sx=1/resol_h; %Tamaño del pixel en X
sy=1/resol_v; %Tamaño del pixel en Y
f=0.8; %Distancia focal
fu = f/sx; % fu = f/sx
fv = f/sy; % fv = f/sy
cu = resol_h/2; % punto principal en horizontal (u)
cv = resol_v/2;
```

Con ellos, se calculan los límites del campo visual en coordenadas normalizadas en el plano imagen:

```
xlim=[(resol_h-cu)/fu;-cu/fu]; %límite máx y min en x
ylim=[(resol_v-cv)/fv;-cv/fv];
```

Y se utilizan estos límites como comprobación de que el *target* no ha abandonado el campo visual de la cámara en el bucle de control:

```
x_islim=x>xlim(1)| x<xlim(2);
y_islim=y>ylim(1)| y<ylim(2);
    if any(x_islim) || any(y_islim)
        out=0;
    end
```

Es decir, se obtienen vectores de booleanas, de modo que todos los elementos de dichos vectores valdrán 0 si todos los puntos están dentro de los límites, mientras que algunos valdrán 1 en caso de estar más allá de los límites. Esta comprobación se suma así a la comprobación de que la coordenada Z de *cMo* no sea negativa.

3.3.8 Control PBVS *eye-in-hand*:

Esta sección del programa se corresponde con la simulación del control PBVS para tareas de posicionado de una referencia fija y de seguimiento de una referencia móvil tratado en los apartados 2.4.1 y 2.4.3 del presente documento.

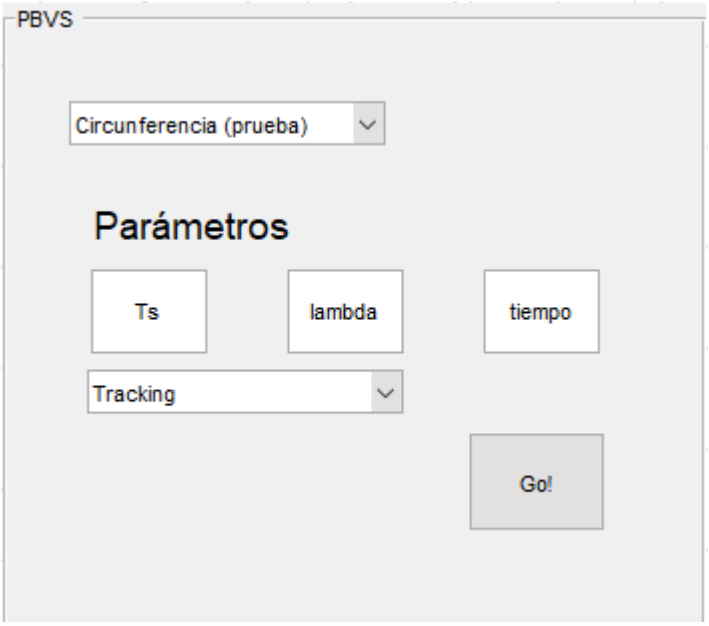


Figura 40-Panel de control PBVS

Esta sección es esencialmente igual que la del control IBVS, tanto en consideraciones generales, como en opciones a la hora de simular el movimiento del *target* y la ley de control. Las principales particularidades del caso PBVS se exponen a continuación:

- A diferencia del control IBVS, en el control PBVS no se trabaja con las coordenadas de los puntos del *target*, sino que se trabaja tomando como conocidas la posición y orientación del centro del *target*, es decir, no se realiza una estimación de dicha posición a partir de una imagen simulada, lo cual es coherente con el objeto del presente trabajo académico, tal y como se ha expuesto en el capítulo 2.
- La proyección de los 4 puntos del *target* sobre el plano imagen sigue realizándose, pero ahora simplemente con el objetivo de comprobar si el *target* sigue dentro del campo visual y para obtener la representación del punto de vista de la cámara, tal y como se realiza en la sección de IBVS.

A nivel visual, la aplicación también ofrece, por tanto, las mismas características que en la sección de IBVS.

Por ello, a nivel de implementación la principal diferencia se encuentra en el cálculo del vector de características *s* y en el cálculo de la matriz de interacción, *L*, como ya ocurría de hecho en el apartado 2.4 del presente Trabajo Fin de Máster. Es por ello que, a continuación, tan solo se explican los fragmentos del código que se diferencian de la sección IBVS:

```
csMo=transl(0,0,0.48); %c*Mo, transformación deseada. Sin rotación y con
traslación
```

```
csTo=csMo(1:3,4);
sref=[csTo;zeros(3,1)];
```

[…]

```
while i<=N && out~=0
    %Cálculo de la posición relativa cMo (ideal)
    wMe=handles.rob.fkine(handles.rob.getpos);
    wMc=wMe*eMc;
    cMo=inv(wMc)*wMp(:,:,i);

    %Cálculo de s y pinv(L):
    cTo=cMo(1:3,4);
    cTo_x=skew(cTo);
    csMc=csMo*inv(cMo); %Relativa de C a C*
    csRc=csMc(1:3,1:3);
    %Rotación en formato ángulo-eje:
    theta=atan2(csRc(1,1)+csRc(2,2)+csRc(3,3)-1,sqrt((csRc(1,2)-
csRc(2,3))^2+(csRc(1,3)-csRc(3,1))^2+(csRc(2,1)-csRc(1,2))^2));
    u=(1/(2*sin(theta)))*[csRc(3,2)-csRc(2,3);csRc(1,3)-
csRc(3,1);csRc(2,1)-csRc(1,2)];

    s=[cTo;theta*u];

    u_x=skew(u);

    L_thu=eye(3)+(theta/2)*u_x+(1-(sinc(theta)/sinc(theta/2)^2))*u_x^2;

    Ls=[-eye(3) cTo_x;zeros(3) L_thu];
```

[…]

```
    e=s-sref;
    Js=Ls*cXn*handles.rob.jacobn(handles.rob.getpos);
    Jsi=pinv(Js);
```

[…]

```
end
```

Como puede observarse, se trata de una aplicación punto por punto de las ecuaciones planteadas en los apartados 2.4.1 y 2.4.3. Las leyes de control una vez obtenido el error y la matriz Jacobiana de características (*Js*), son exactamente iguales a las de la sección de IBVS.

En cuanto a la representación del punto de vista de la cámara, dado que la referencia en el caso PBVS no se encuentra sobre la imagen, para poder representar los puntos de referencia es necesario deducirlos en la primera iteración a partir de la posición de referencia.

3.4 Desempeño de los controles simulados:

A continuación, se prueba que el comportamiento de los controles implementados en la aplicación se ajusta a lo tratado en su correspondiente desarrollo teórico.

Para ello, se realizan una serie de simulaciones para los controles de velocidad, aceleración, IBVS y PBVS, extrayendo vídeos y gráficas que permiten probar los puntos tratados. Los vídeos se han compartido públicamente en la web *Youtube*.

Por lo general, se ha utilizado para ello el robot KUKA KR 5-2 ARC HW, si bien se han documentado también simulaciones con otro robot (KUKA Agilus) en una posición significativamente diferente (suspendido boca abajo) para demostrar que el simulador efectivamente puede funcionar con condiciones diferentes.

También en la mayoría de las simulaciones se ha utilizado un periodo de muestreo de 0.1 segundos, que es viable para el control de brazos robóticos industriales (de hecho, es el periodo utilizado para la validación experimental con un KUKA Agilus) y se aproxima al que tarda cada iteración del bucle de control incluyendo la actualización gráfica, que está entre los 0.1 y 0.12 segundos. De este modo, se obtiene una simulación prácticamente a tiempo real. Cabe señalar que el elevado periodo de duración de cada iteración se debe a la necesidad de simulación gráfica, pues eliminando la actualización gráfica de cada iteración, el periodo de cada bucle cae hasta valores alrededor de los 0.005 segundos.

Cabe señalar que las gráficas presentan en el eje horizontal no el tiempo simulado sino las iteraciones realizadas, de modo que el instante de cada iteración se puede obtener de forma trivial (k·Ts).

3.4.1 Control de velocidad:

El principal punto a probar respecto al control de velocidad implementado es que efectivamente logra seguir las referencias tanto fijas como móviles asegurando un descenso exponencial del error de posición, y que la velocidad de convergencia crece con la ganancia proporcional del control. También se han graficado las velocidades de las articulaciones para poder realizar una comparativa en el apartado del control de aceleración.

En todas las simulaciones se ha utilizado un periodo de muestreo de 0.1 segundos.

-Referencia fija:

Con referencia fija se ha recogido documentación de dos simulaciones, con la misma referencia y diferente ganancia proporcional.

La referencia es: [0, 0.7, 1.0044, 0, 0, 0], de modo que es sencillo comprobar visualmente que la posición es alcanzada, no solo gracias a los indicadores del panel de control, sino también porque la orientación del elemento terminal debe alinearse con la del sistema de referencia mundo.

La primera simulación se ha realizado con una ganancia proporcional de 0.2, y su resultado puede observarse en el vídeo a continuación:

https://youtu.be/OADJJM1pxnM

Como se comprueba en las gráficas correspondientes a dicha simulación, en la figura 41, el descenso del error es, efectivamente, exponencial, hasta anularse:

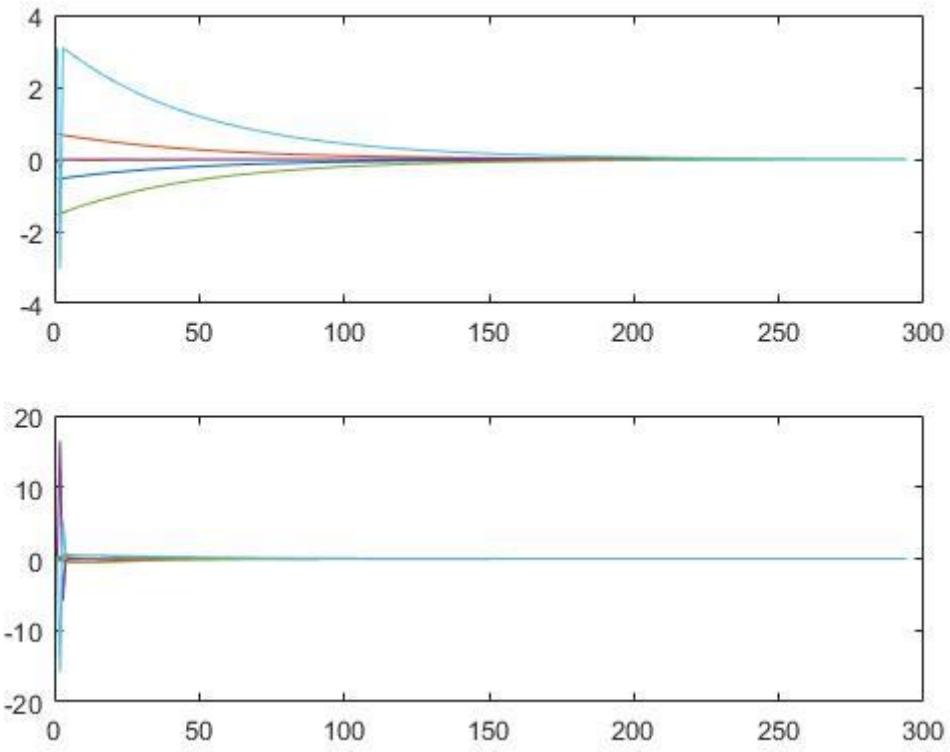


Figura 41-Evolución del error de posición y la velocidad de las articulaciones del control de velocidad con referencia fija para una ganancia proporcional de 0.2 y una referencia [0, 0.7, 1.0044, 0, 0, 0].

La segunda simulación toma la misma referencia fija que la primera, con una ganancia proporcional de 0.4, y demuestra cómo un aumento de la ganancia proporcional incide en la velocidad de convergencia manteniendo la corrección completa del error de posición. Puede observarse el resultado en el vídeo enlazado a continuación:

https://youtu.be/XfujU5XZLKA

De nuevo, el comportamiento del error y la velocidad de las articulaciones se observa en la figura 42. Nótese cómo la convergencia se consigue en un número de iteraciones significativamente menor, a costa de unas mayores velocidades en las articulaciones.

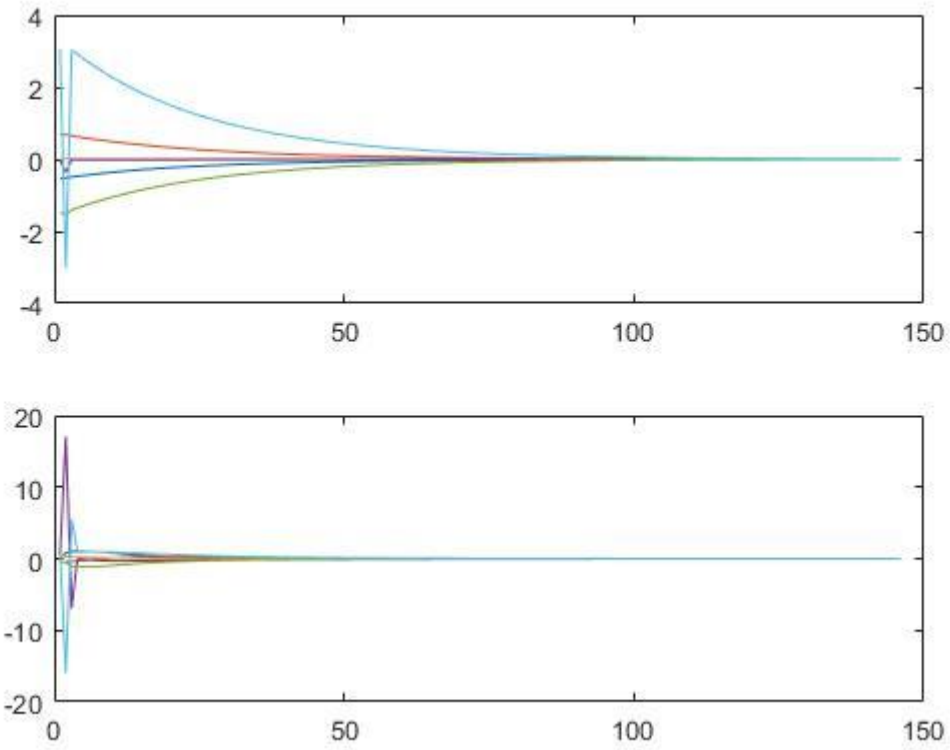


Figura 42- Evolución del error de posición y la velocidad de las articulaciones del control de velocidad con referencia fija para una ganancia proporcional de 0.4 y una referencia [0, 0.7, 1.0044, 0, 0, 0].

Además de en los vídeos, una secuencia fotográfica del comportamiento del control puede observarse en la figura 33.

-Referencia móvil:

Con referencia móvil se han documentado dos simulaciones, una con el robot utilizado por defecto y otra con el KUKA Agilus, ambas siendo una circunferencia alrededor del robot que empieza y termina en el punto [0.6,0,1] con una orientación de referencia [0,0,0] y mantiene la coordenada Z y la orientación constante. La referencia completa la trayectoria en 200 segundos.

Nótese que en ninguna de las dos simulaciones la referencia móvil coincide con la del robot, de manera que se puede probar más claramente el descenso exponencial del error y el papel de la pre alimentación en eliminar el problema de perseguir la referencia. En ambos casos la ganancia proporcional utilizada es de 0.5.

La primera simulación es la realizada con el KUKA Agilus suspendido boca abajo, cuyo comportamiento puede comprobarse en el vídeo que se enlaza a continuación, así como en la frecuencia fotográfica de la figura 34:

https://youtu.be/LEAz6wvC4Qs

Si se observan sus gráficas de la figura 43, análogas a las anteriores, se comprueba cómo el descenso del error sigue un comportamiento exponencial pese al movimiento de la referencia, lo cual se da

gracias a la pre alimentación, puesto que, de lo contrario, el controlador solo podría corregir el movimiento con una iteración de retraso.

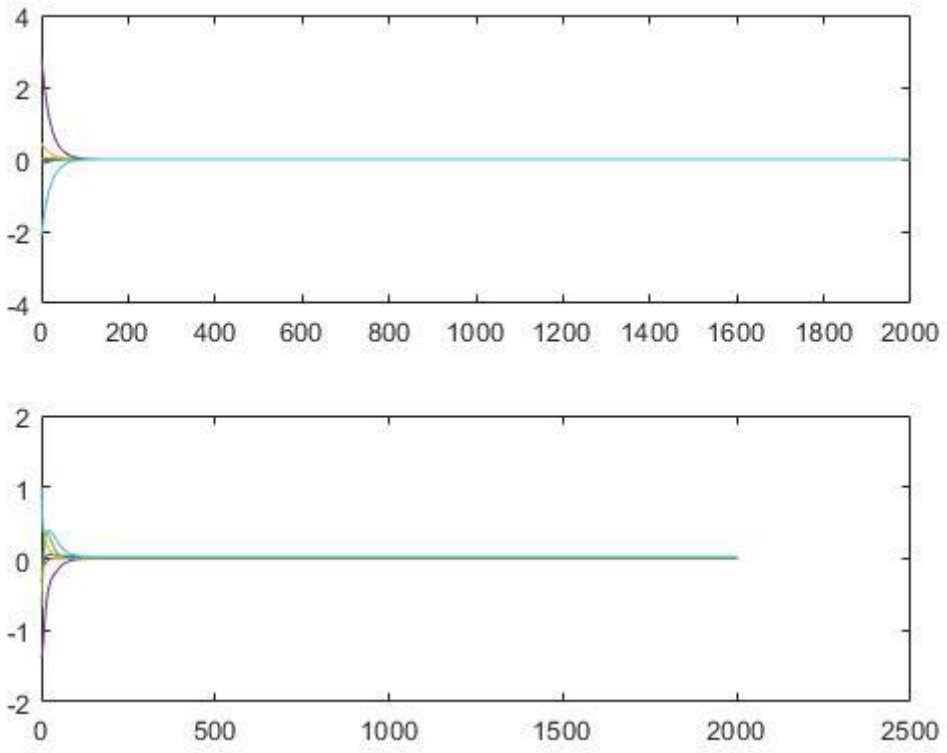


Figura 43-Evolución del error de posición y la velocidad de las articulaciones del control de velocidad con referencia móvil para una ganancia proporcional de 0.5 y una referencia de trayectoria circular con altura 1 y orientación [0,0,0]. KUKA Agilus.

La segunda simulación es la realizada con el robot utilizado por defecto, y tiene el mismo comportamiento, que se puede observar parcialmente en el vídeo enlazado a continuación. Sus gráficas pueden observarse en la figura 45:

https://youtu.be/CmgHyZVbHjc.

Figura 44-Secuencia del control de velocidad para referencia móvil circular

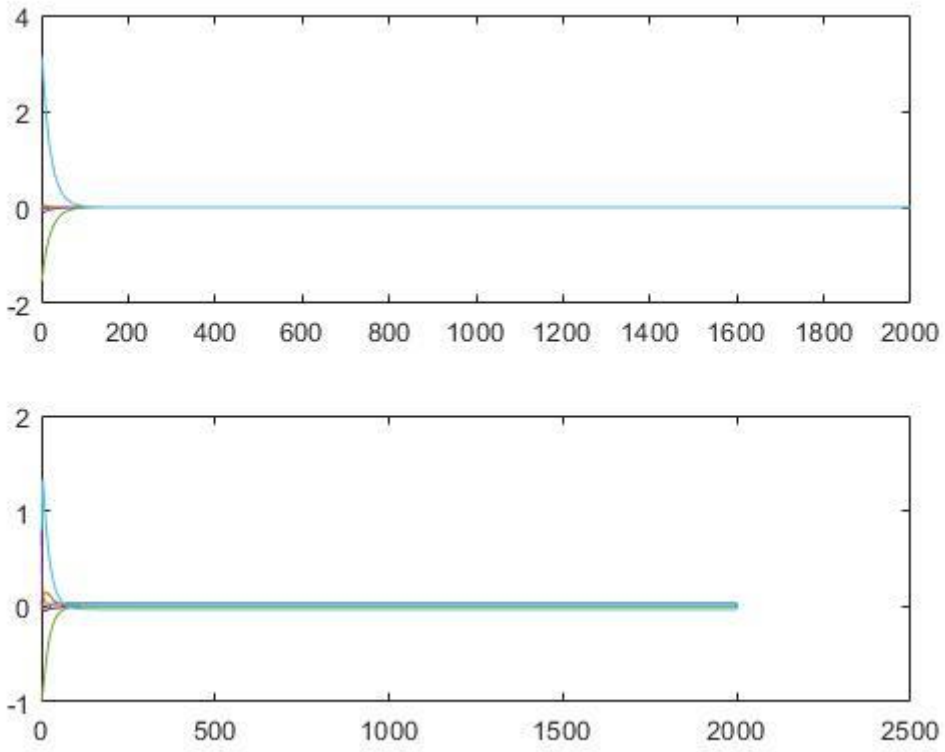


Figura 45- Evolución del error de posición y la velocidad de las articulaciones del control de velocidad con referencia móvil para una ganancia proporcional de 0.5 y una referencia de trayectoria circular con altura 1 y orientación [0,0,0].

Así pues, la aplicación simula correctamente el comportamiento de un control de velocidad para diferentes situaciones y referencias.

3.4.2 Control de aceleración:

En cuanto al control de aceleración, la principal diferencia con el caso anterior, tal y como se ha tratado en el apartado 2.3.4, es la incorporación de la corrección del error de velocidad y la obtención de la posición articular deseada a partir de la aceleración y no de la velocidad, de manera que la evolución del error se corresponde con una EDO de 2º orden.

Ello se traduce en que, con la variación de la ganancia asociada a la corrección del error de velocidad, debe variar el amortiguamiento del sistema, de manera que una menor ganancia Kv suponga un control con más sobre oscilación.

Se han realizado dos simulaciones con referencia fija para y una con referencia móvil, tal y como se trata a continuación.

-Referencia fija:

Las dos simulaciones con referencia fija utilizan la misma referencia que en el control de velocidad, de manera que es sencillo comprobar el efecto de corregir el error de velocidad.

La primera simulación utiliza una ganancia Kp de 0.5 y una ganancia Kv de 0.8, que es baja, de modo que se observa una sobre oscilación significativa. La simulación al completo puede observarse en el vídeo enlazado a continuación, y la evolución del error y la velocidad de las articulaciones aparece en la figura 46:

https://youtu.be/O60pfgbTrm8

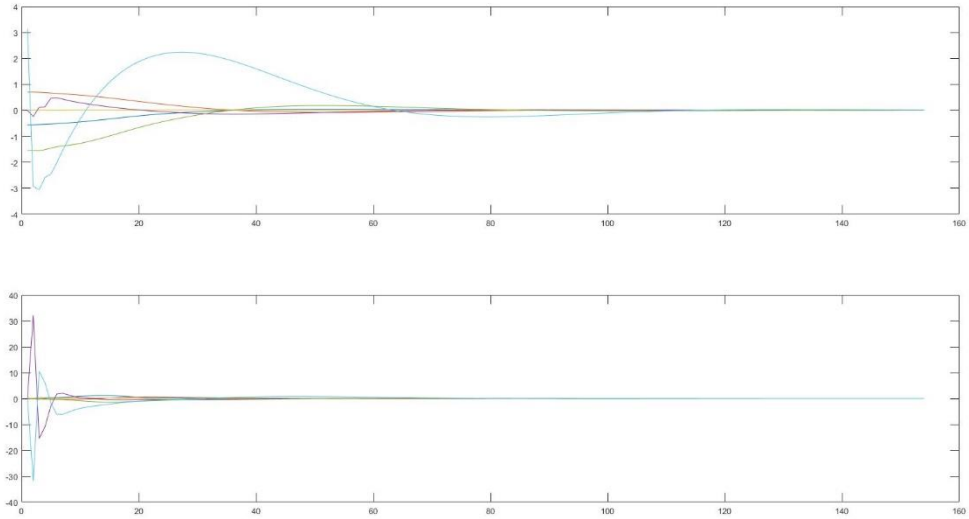


Figura 46- Evolución del error y la velocidad de las articulaciones en control de aceleración con referencia fija [0,0.7,1.0044,0,0,0], Kp=0.5 y Kv=0.8

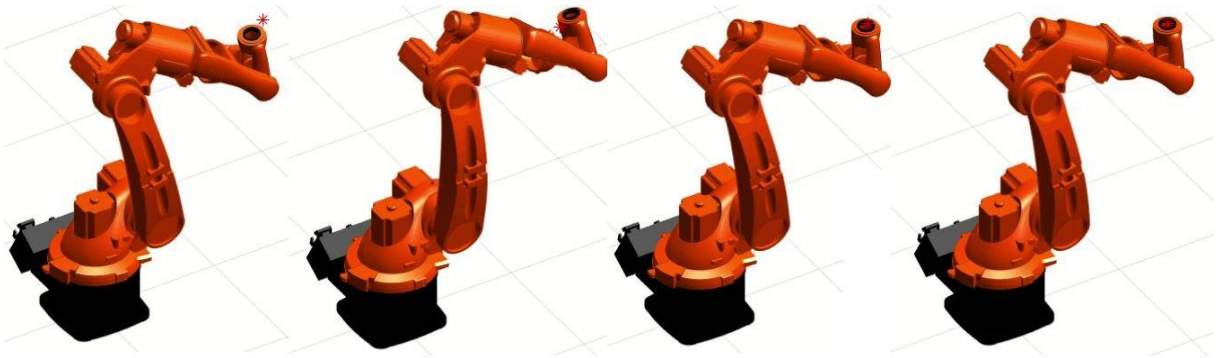


Figura 47-Secuencia demostrativa de la sobre oscilación del caso de control correspondiente a la figura 46

La segunda simulación utiliza la misma Kp de 0.5 y una ganancia Kv mayor, de 1.5, que elimina la sobre oscilación de la primera simulación. La simulación completa puede visionarse en el vídeo enlazado a continuación:

https://youtu.be/2cNY9XAsNOU

En la figura 49, que contiene las gráficas de la segunda simulación, se puede ver cómo se elimina la sobre oscilación visible en la figura 46 correspondiente en la primera simulación.

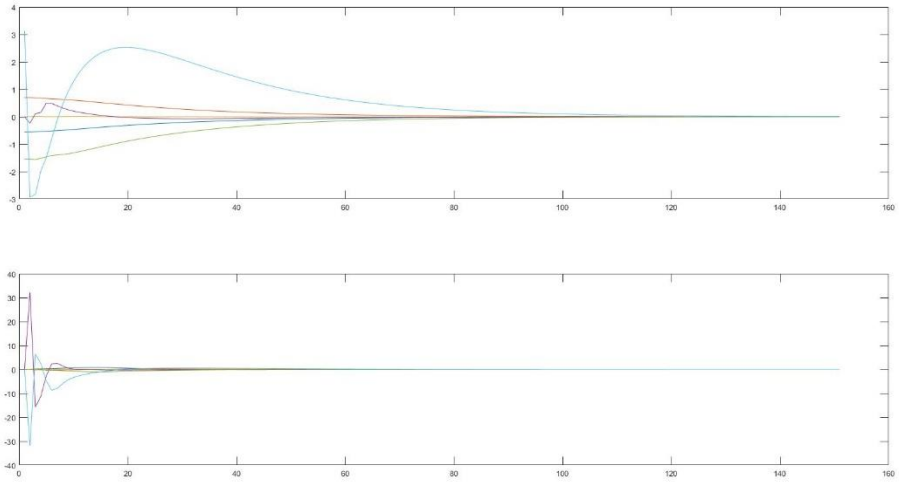


Figura 48- Evolución del error y la velocidad de las articulaciones en control de aceleración con referencia fija [0,0.7,1.0044,0,0,0], Kp=0.5 y Kv=1.5

-Referencia móvil:

Utilizando la misma referencia que en el caso del control de velocidad, pero con un tiempo de simulación menor (la trayectoria debe completarse en 30 segundos), con una Kp de 0.8 y Kv de 1.5, se comprueba que el control de aceleración cumple su objetivo con éxito. Esto puede comprobarse en el vídeo enlazado a continuación y las gráficas correspondientes a dicho caso de simulación, representadas en la figura 50:

https://youtu.be/XHLF_3k-Ft4

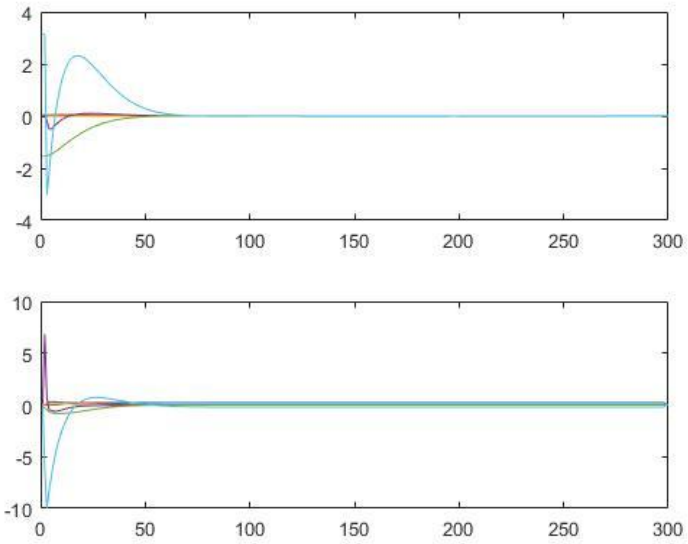


Figura 49- Evolución del error de posición y la velocidad de las articulaciones del control de aceleración con referencia móvil para Kp 0.8, Kv 1.5 y una referencia de trayectoria circular con altura 1 y orientación [0,0,0].

3.4.3 Control IBVS *eye-in-hand:*

Los distintos aspectos de interés del control IBVS se han probado en la aplicación mediante múltiples simulaciones, tanto en tareas de posicionado como en tareas de *tracking* con diferentes trayectorias.

En todos los casos, el vector de características de referencia, *s**, es [-0.1; -0.1;0.1; -0.1;-0.1 ;0.1;0.1 ;0.1], prácticamente coincidente con la referencia del control PBVS.

3.4.3.1 Posicionado:

Las simulaciones de tareas de posicionado incluyen la documentación de cuatro casos en que se demuestran el descenso exponencial del error, el desplazamiento de los puntos sobre el plano imagen siguiendo trayectorias que se aproximan a líneas rectas y el aumento de la velocidad de convergencia con la ganancia proporcional; y tres simulaciones que exploran el fenómeno de *camera-retreat*.

Las primeras dos simulaciones tienen en común un *target* fijo situado a la misma altura inicial que la cámara simulada, 0.6 metros adelantado respecto a la misma, y posteriormente rotado 0.2513 radianes tanto alrededor de su propio eje Z como sobre el eje Z del mundo partiendo desde su posición inicial.

La primera de las simulaciones tiene una ganancia proporcional λ de 0.2, garantizando una convergencia lenta y un desplazamiento de los puntos sobre el plano imagen en línea recta, como puede comprobarse tanto en el vídeo de la simulación completo enlazado a continuación, como en las gráficas de la figura 50:

https://youtu.be/_Yvcv1Wjwn4

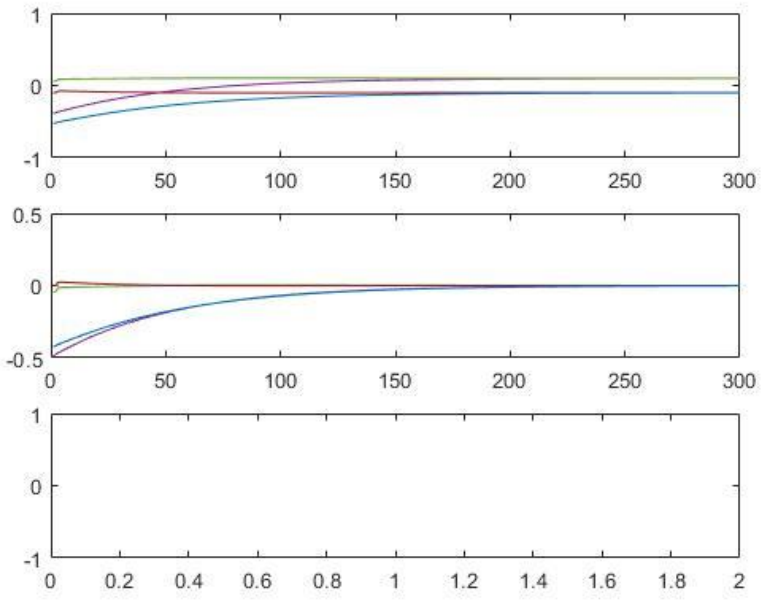


Figura 50- Desde la gráfica superior: evolución de las coordenadas de los puntos del target sobre el plano imagen, evolución del error y término del efecto del movimiento del target, que no existe en este caso por ser una referencia fija

La segunda simulación incrementa la ganancia proporcional λ hasta 0.5, de manera que la convergencia se consigue mucho antes, como puede comprobarse en el vídeo enlazado a continuación y en las gráficas de la figura 51:

https://youtu.be/zMq65b_XiMA

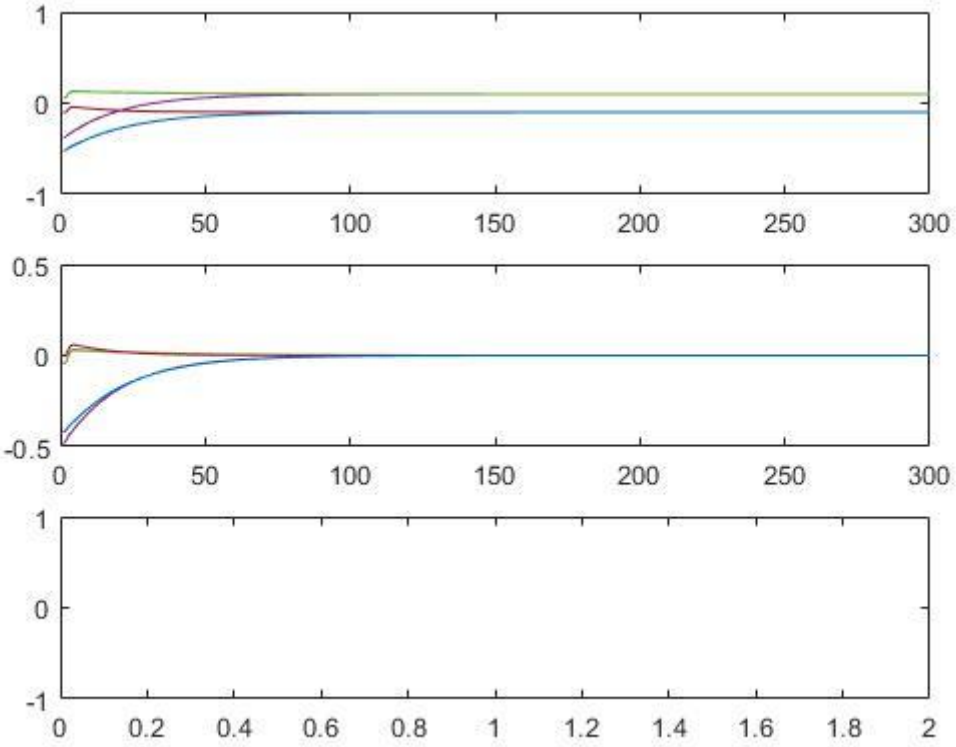


Figura 51- Ídem que la figura 50, segundo caso de posicionado

Cabe recordar que, como se adelantó en el apartado anterior, la simulación de las tareas de posicionado tienen un tiempo definido por el usuario, no terminan cuando se posiciona el *target*, de modo que en ambas simulaciones se ha elegido el mismo tiempo de simulación (30 segundos), para que sea fácil comprobar en las figuras 50 y 51 cómo la convergencia es más rápida en el segundo caso por el incremento de la ganancia proporcional, pero en ambos casos con un descenso exponencial y trayectorias de los puntos sobre la imagen que se aproximan a líneas rectas. El fenómeno del desplazamiento aproximado a una línea recta se observa mejor en los vídeos.

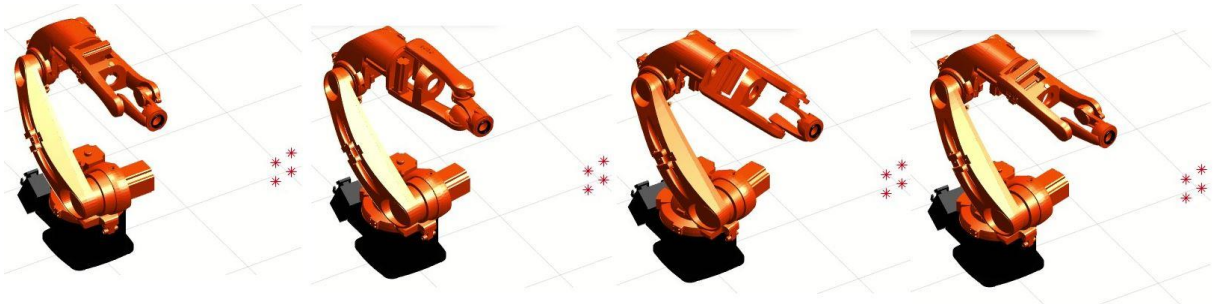


Figura 52-Secuencia del posicionado correspondientes a los casos 1 y 2

Las simulaciones de posicionado 3 y 4 tienen en común un nuevo *target* fijo, situado 0.3 metros por encima del de las simulaciones 1 y 2. De nuevo se han documentado simulaciones con una ganancia proporcional λ de 0.2 y 0.5, respectivamente, obteniendo resultados similares a los casos 1 y 2.

El principal interés de estos nuevos casos de simulación es que muestran el comportamiento del control IBVS visto en el apartado 2.4.2.1, de modo que, al controlar tan solo el movimiento de los puntos sobre el plano imagen, se pueden alcanzar límites de las articulaciones. En estas simulaciones se está muy cerca de ello en el caso de menor λ y se da esta situación en el caso de mayor λ, dado que la acción de control es más agresiva. No obstante, dado que no se han introducido límites en las articulaciones a la hora de los controles implementados, la simulación continúa con normalidad.

Los vídeos correspondientes a estos casos de simulación pueden visualizarse con los siguientes enlaces:

https://youtu.be/nCgsJixZUDw

https://youtu.be/IwxSB6-pe1E

Dado que el principal interés de estos casos de simulación es precisamente comprobar cómo el control IBVS es proclive a llegar a los límites de las articulaciones, y lo es más cuanto más agresivo es, no se reproducen las gráficas de ambas simulaciones, sino tan solo de menor ganancia proporcional, siendo las de la otra esencialmente iguales, pero con una convergencia más rápida. En cualquier caso, en las gráficas, visibles en la figura 53, se sigue observando el mismo comportamiento esperado en una tarea de posicionado IBVS.

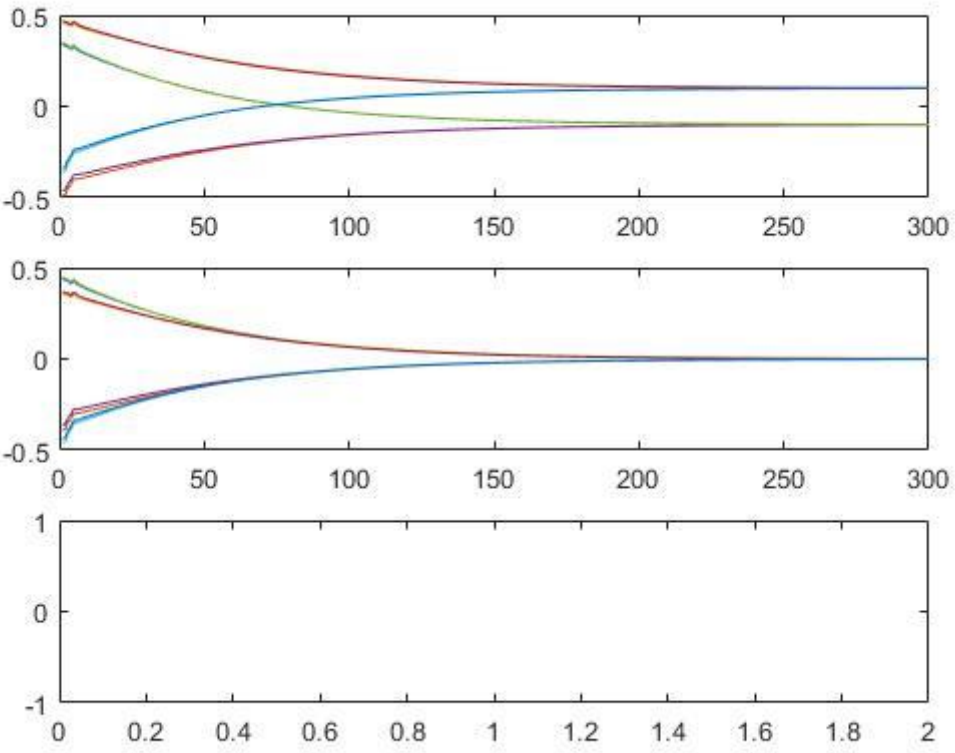


Figura 53-Ídem que la figura 50, para el tercer caso de posicionado

En cuanto a las tres simulaciones que reproducen el fenómeno de *camera retreat*, simplemente se ha dispuesto un *target* inicialmente posicionado y orientado de manera que coincida con la referencia, y se ha rotado la cámara (modificando manualmente la posición de la última articulación del robot) de tal manera que la tarea de posicionado requiriese una u otra rotación. En todos los casos se ha utilizado una ganancia λ de 0.2, que provoca una acción de control lo suficientemente suave como para observar el fenómeno adecuadamente.

En el primer caso la cámara se ha rotado 90º respecto del *target*, de tal manera que el fenómeno de *camera retreat* se da, pero sin llegar a la inestabilidad del control porque el giro requerido no es lo suficientemente grande. Este caso puede observarse en el vídeo enlazado a continuación:

https://youtu.be/ON7qLxM-QsE

En las gráficas de la figura 54 se puede comprobar cómo, pese a tratarse de una rotación, el fenómeno de *camera retreat* consigue que los puntos sobre el plano imagen se desplacen prácticamente siguiendo líneas rectas, y además, como se ha indicado, el control logra converger.

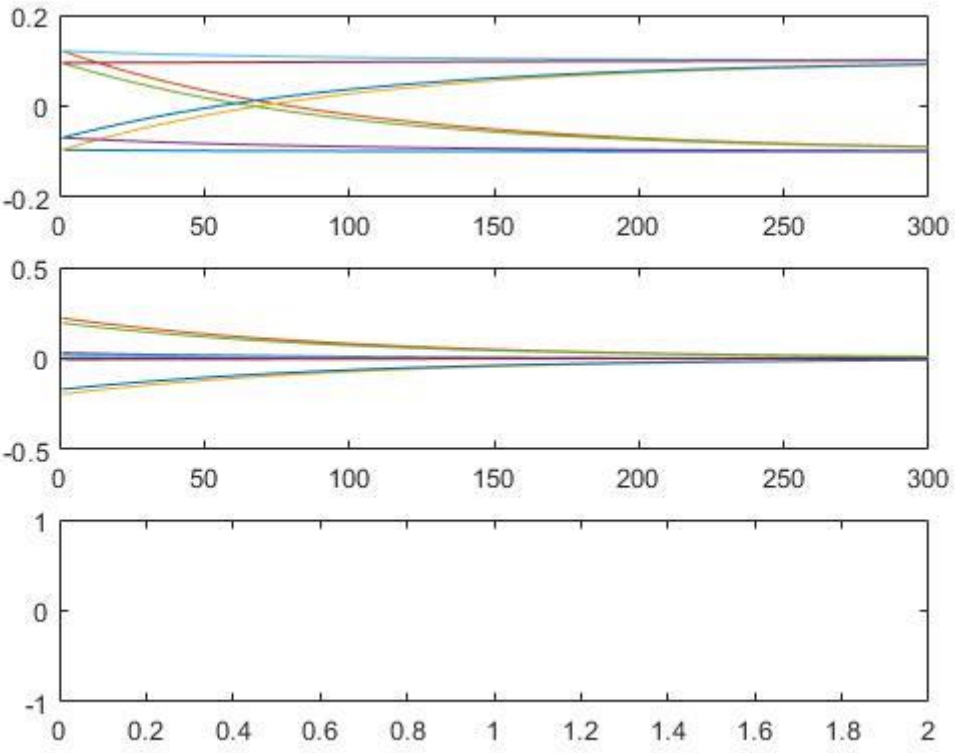


Figura 54-Ídem que la figura 50, para el caso de camera retreat con convergencia

En la secuencia de la figura 55, obtenida a partir de la simulación del vídeo enlazado anteriormente, puede contemplarse de forma clara cómo se resuelve la rotación siguiendo trayectorias que se aproximan a líneas rectas en el plano imagen.

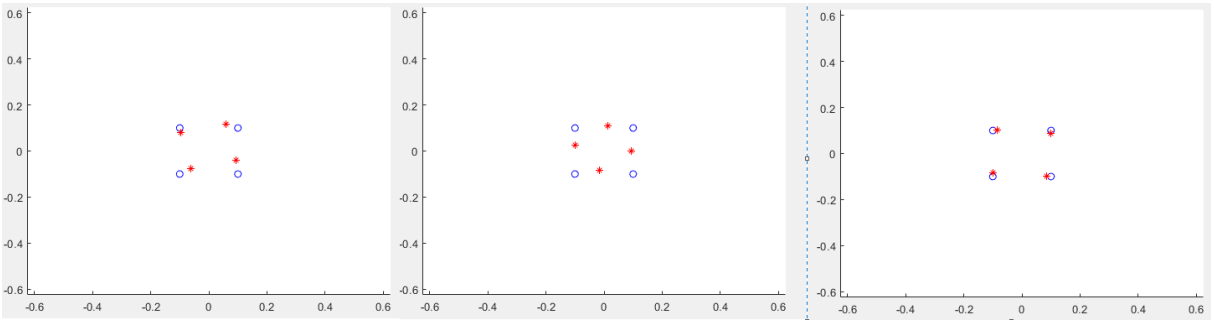


Figura 55- Secuencia de camera retreat con convergencia, punto de vista de la cámara

En el segundo caso, la cámara se ha rotado 120º, de manera que el giro demandado es demasiado grande y el control no logra estabilizarse, de manera que no hay convergencia, comprobándose así que el control IBVS garantiza estabilidad local pero no global. En el enlace siguiente se puede observar la simulación completa:

https://youtu.be/65H_sPq5dSk

En el último caso, se pone a prueba el caso límite mencionado en el apartado 2.4.2.1, según el cual cuando el giro demandado es de 180º, el fenómeno de *camera retreat* no produce giro alguno entorno al eje Z de la cámara, sino tan solo retroceso:

https://youtu.be/eG1kgxpYSyY

En las gráficas correspondientes a este caso, en la figura 56, se ve claramente cómo, hasta el momento en que el control deja de ser estable porque se alcanza un límite del robot, la cámara se aleja cada vez más del target, lo que se traduce en un acercamiento (sobre trayectorias rectilíneas) de los puntos del *target* proyectados en el plano imagen.

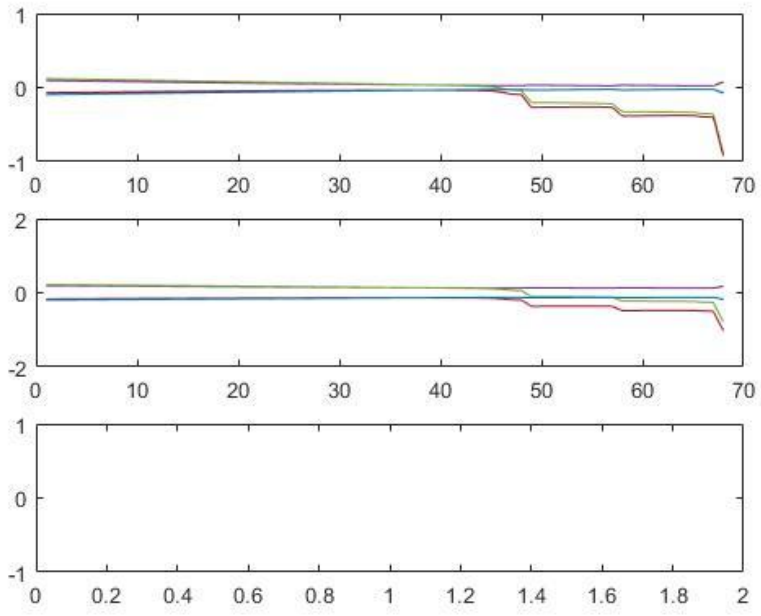


Figura 56-Ídem que la figura 50, para el caso límite de camera retreat con giro demandado de 180 grados

3.4.3.2 Seguimiento:

Para la simulación del seguimiento de *targets* móviles mediante IBVS se han documentado 8 casos de simulación utilizando la ley de control con pre alimentación y un caso adicional en que se utiliza tan solo la ley de control proporcional. Con dichos casos se ha comprobado tanto el correcto funcionamiento del control implementado como el efecto de diferentes parámetros sobre el desempeño del control, de acuerdo a lo tratado en el apartado 2.4.

En primer lugar, se han realizado dos simulaciones con la trayectoria de prueba circular descrita en el apartado 3.3.7 realizada en 200 segundos, una λ de 0.2 y con los dos robots indicados anteriormente (Agilus y KR 5-2 ARC HW). Con ello se ha demostrado el correcto funcionamiento del control simulado en términos generales.

Las simulaciones con el KUKA Agilus y el KUKA KR 5-2 pueden observarse en los vídeos enlazados a continuación:

https://youtu.be/XkfUGAdMUoo

https://youtu.be/20KFsTPiqp0

En las gráficas correspondientes a la simulación con el KUKA KR 5-2, visibles en la figura 57, se comprueba la convergencia con un descenso exponencial del error, gracias a la pre alimentación de la estimación del efecto del movimiento del *target* sobre la variación del error.

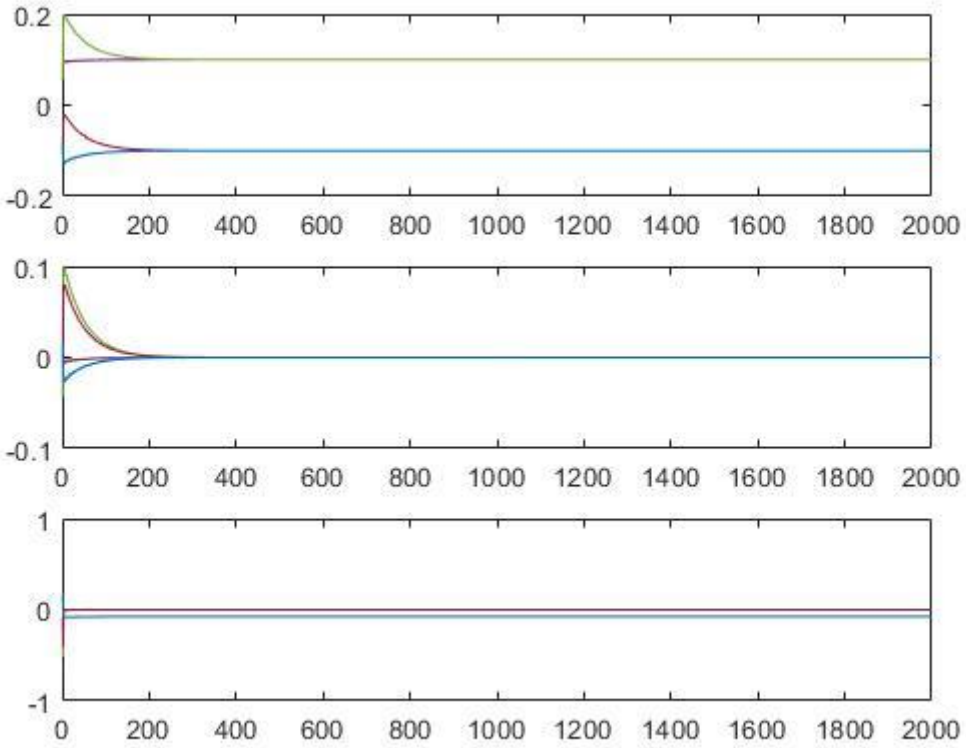


Figura 57-Ídem que la figura 50, para el caso de tracking de una trayectoria circular de 200 segundos de duración con el KUKA KR 5-2 ARC HW.

En la figura 37 puede observarse una secuencia de la simulación utilizando el KUKA Agilus.

Si se retira el término del *feedforward*, utilizando únicamente la ley de control proporcional, el control simulado tan solo puede realizar una persecución del *target*, asumiendo como consecuencia un error de posición no nulo. Este caso, con la misma trayectoria circular pero ahora realizada en 100 segundos, se puede observar en el vídeo a continuación:

https://youtu.be/rIdR_xNHSrI

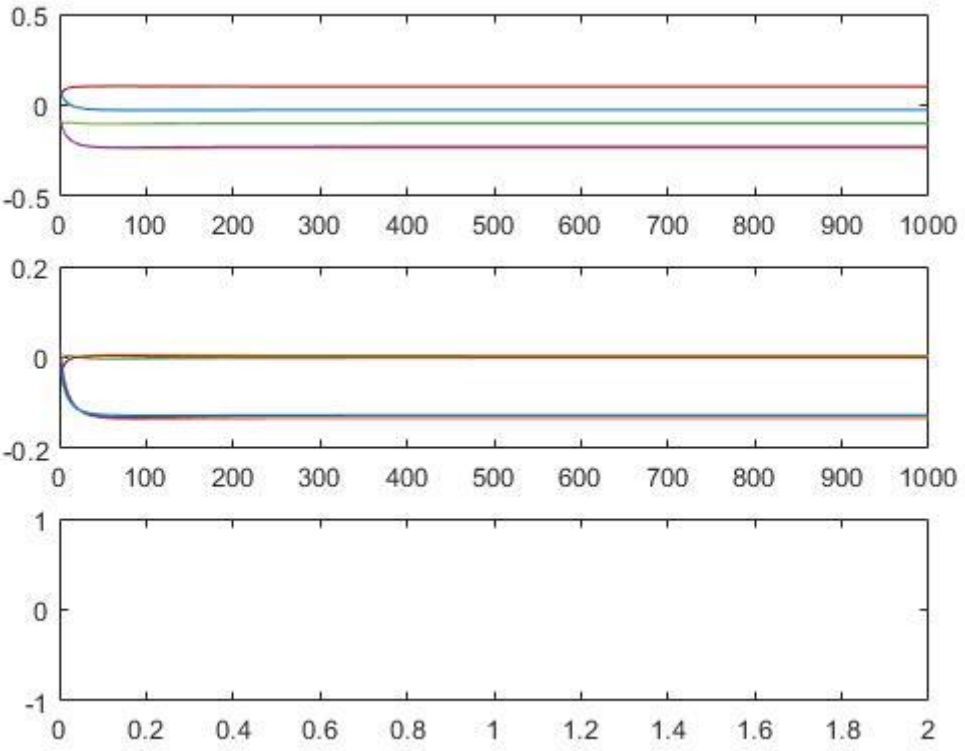


Figura 58-Ídem que la figura 50, para el caso de seguimiento de una referencia móvil de trayectoria circular recurriendo tan solo a control proporcional. Nótese el error de posición asumido.

Como puede comprobarse en el vídeo enlazado, la ganancia proporcional necesaria para poder cumplir la tarea de seguimiento es muy elevada en comparación a la necesaria si se utiliza la ley de control con *feedforward* (1.2 frente a 0.15 que se utilizarán más adelante para el mismo caso), asumiendo además un error de posición significativo.

La contrapartida del uso del término de *feedforward* con la aproximación dada por la expresión {2.4.1.12} es que puede contribuir a la desestabilización del control IBVS, ya que introduce un término que no necesariamente converge con el *target*, a diferencia del término proporcional. Es decir, en ocasiones la combinación de la acción proporcional al error y un término de *feedforward* resultado de una aproximación, dan lugar a una acción de control que no persigue adecuadamente al *target*, cosa que puede llegar a ser un problema para la estabilidad.

Dado que la derivada del error es una estimación a partir de datos pasados, los factores que tiendan a aumentar el efecto de dicha derivada aportan un riesgo, dado que un cambio rápido en la dirección en

que se mueve el *target* o en la distancia que recorre en el siguiente periodo de muestreo hace que la derivada contribuya a alejarse del *target* y aumentar el error, lo cual puede a su vez agrandar todavía más el término de *feedforward*.

Evidentemente, si la velocidad del *target* es mayor en un ensayo que en otro, pero su trayectoria y el periodo de muestreo utilizado son iguales, el riesgo de desestabilización crece, porque entre una iteración del bucle de control y la siguiente el error es más grande y, por tanto, la estimación de la derivada crece.

Además, un control más agresivo, con una λ mayor, influye en el efecto de la pre alimentación, porque, si bien en el término proporcional es cierto que favorece una convergencia más rápida, también hay que tener en cuenta que, como se ha visto en el apartado 2.4.1.3, la ganancia proporcional λ aparece de forma implícita en el segundo término de la expresión de la estimación ({2.4.1.12}).

Por ello, se han realizado dos simulaciones demostrativas tomando la trayectoria circular con un *target* que la realiza en 100 segundos, y demostrando cómo con una ganancia proporcional menor (0.15) y el mismo periodo de muestreo, es posible conseguir el recorrido, y con una ganancia proporcional mayor (0.5) falla. Cabe señalar que se podría haber utilizado también la misma ganancia proporcional de la primera simulación de *tracking* (0.2), pero ésta falla de forma mucho más tardía.

Así, la simulación con λ 0.15, trayectoria circular de 100 segundos y periodo de muestreo de 0.1 segundos puede observarse en el vídeo enlazado a continuación, y su desempeño en las gráficas de la figura 58:

https://youtu.be/QnGoc5UoQgI

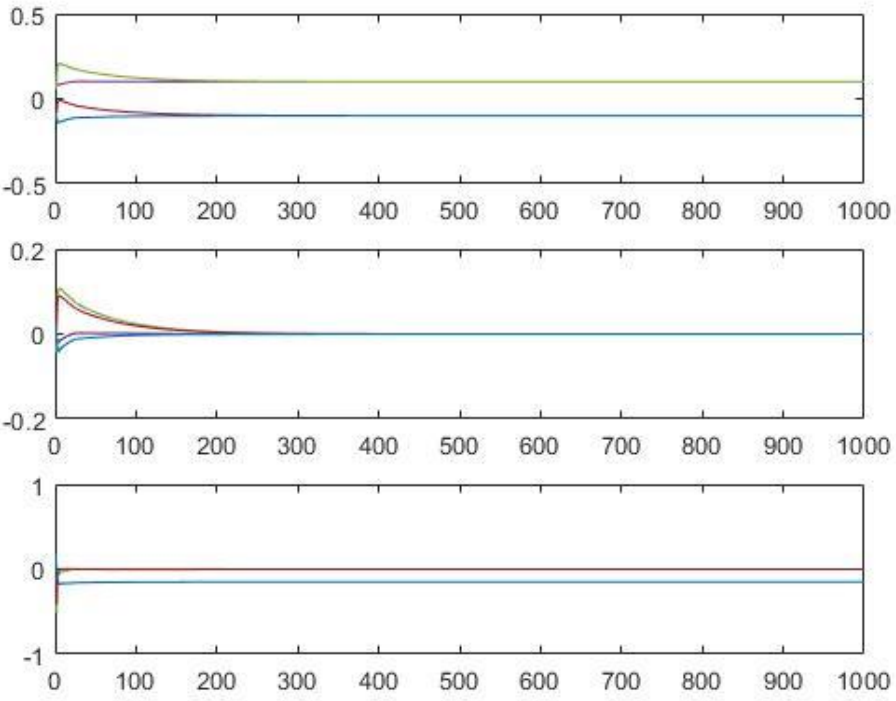


Figura 59- Ídem que la figura 50, para el caso de tracking de una referencia móvil de trayectoria circular de 100 segundos

Y la simulación con los mismos parámetros que la anterior y una λ de 0.5 falla al poco de iniciarse la simulación, como se comprueba en el vídeo a continuación:

https://youtu.be/5I1AXApqo0M

La confirmación de que este fallo se debe al término de *feedforward* estimado se tiene al recordar que la simulación con persecución pura utilizando solo la ley de control proporcional, cuyas gráficas se pueden ver en la figura 58, utiliza de hecho una λ de más del doble de 0.5 con una trayectoria más rápida, sin que haya desestabilización.

Continuando con las simulaciones y la contribución del término de *feedforward* al fallo del control, hay que tener en cuenta que otro factor que afecta al peso del *feedforward* en el control, haciendo que la estimación de la derivada del error sea más aguda o menos, es el periodo de muestreo. Para una misma trayectoria y velocidad del *target*, un periodo de muestreo más grande significa que la diferencia del error actual y el anterior será mayor, lo que produce picos en la derivada del error que afectan negativamente al control. En términos analíticos, esto se corresponde con la noción de que una derivada refleja una tendencia de una función, y los cambios bruscos en la función se aproximan a picos no diferenciables que conllevan un crecimiento de la derivada.

Es por esto que la siguiente simulación mantiene la trayectoria circular y el tiempo de 100 segundos de la anterior, así como la λ que antes desencadenaba un fallo (0.5) pero reduce el periodo de muestreo de 0.1 segundos a 0.05 segundos. Como resultado, el control es capaz de completar la tarea de *tracking* con éxito, si bien se aleja significativamente de la simulación en tiempo real. Todo ello puede comprobarse en el siguiente vídeo y en las gráficas de la figura 60:

https://youtu.be/jCIy40LxeYg

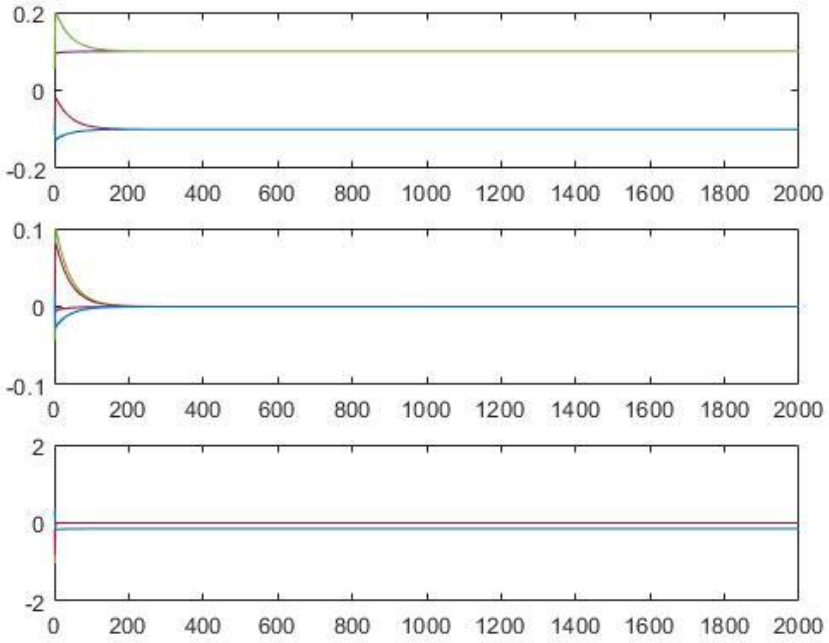


Figura 60- Ídem que la figura 50, para el caso de tracking de una trayectoria circular en 100 segundos, λ 0.5 y disminuyendo el periodo de muestreo a 0.1 segundos

Se ha hecho referencia también al efecto de los cambios de tendencia rápidos como un factor que contribuye al fallo del *tracking* IBVS, porque hace que la estimación del término de *feedforward* se aleje del valor real del efecto del movimiento del *target* sobre la evolución del error de posición.

Para mostrar este efecto se han documentado dos simulaciones con la trayectoria de prueba llamada "corona". La primera, con una duración de 200 segundos, una λ de 0.2 y el periodo de muestreo usual de 0.1 segundos, pretende demostrar que el control implementado es capaz de cumplir la tarea; mientras que la segunda, con un movimiento más rápido (misma trayectoria en 100 segundos) y λ de 0.15, que antes sí lograba el *tracking* de una trayectoria circular en las mismas condiciones, falla, lo que se debe precisamente a esos cambios de tendencia rápidos en el mismo tiempo de simulación.

La primera simulación, que realiza la tarea con éxito, puede verse en el vídeo a continuación y sus gráficas en la figura 61:

https://youtu.be/NkEVu8LFi7M

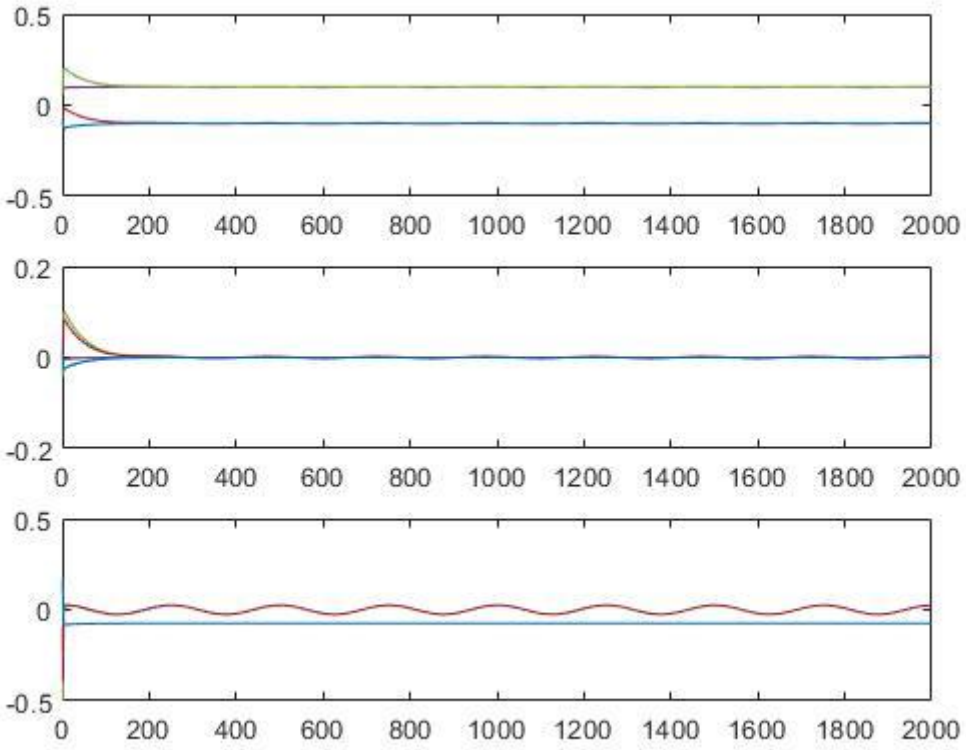


Figura 61-Ídem que la figura 50, para el caso de tracking de una trayectoria de prueba de tipo "corona" en 200 segundos y λ 0.2

Nótese cómo, aunque el error de posición tiende a 0, el término de *feedforward* que permite alcanzar el *target* utilizando una ganancia proporcional pequeña, también da pie a una leve oscilación entorno a la posición de referencia, lo cual puede observarse en la segunda gráfica (evolución del error de posición), precisamente debido a los cambios de tendencia.

La segunda simulación, que falla debido al efecto de estos cambios de tendencia sobre la precisión de la aproximación de la derivada del error, puede verse en el vídeo a continuación:

https://youtu.be/EtUaQCqLj5E

Por último, se ha elaborado una trayectoria personalizada y documentado la simulación de un *tracking* IBVS exitoso utilizando dicha trayectoria.

La trayectoria personalizada tiene una duración de 60 segundos, realizando los siguientes movimientos: el movimiento parte con el centro del *target* situado a 0.6 metros sobre el eje óptico de la cámara, retrocede 0.2 metros en dicha dirección (en X del sistema global) gradualmente en 10 segundos, a continuación desciende (en Z del sistema global) 0.2 metros en otros 10 segundos, durante los siguientes 20 segundos rota 30º sobre su propio eje Z (el eje X del sistema global) en sentido anti horario y deshace el giro en sentido horario, y para finalizar, durante los últimos 20 segundos avanza 0.1 metros hacia el robot (en X del sistema global), asciende 0.1 metros (en Z del sistema global) y se inclina hacia arriba (en Y del sistema global, giro horario) todo ello gradualmente.

La tarea de *tracking* es resuelta con éxito en la simulación documentada en los vídeo a continuación y con las gráficas de la figura 62, con una λ de 0.2 y un periodo de muestreo de 0.1 segundos. Los enlaces siguientes corresponden a una grabación desde diferentes perspectivas y otras dos grabaciones con perspectiva frontal y lateral del robot, respectivamente:

https://youtu.be/lfeiScmYPUM

https://youtu.be/D6jzd-D-zmo

https://youtu.be/oyBnQ-aif_E

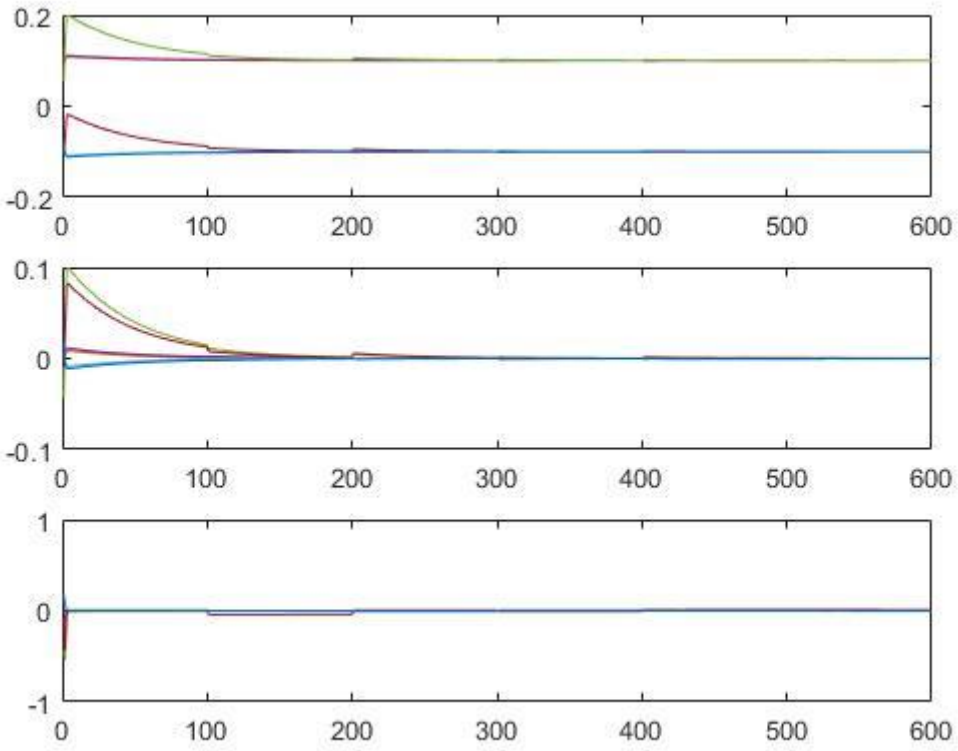


Figura 62-Ídem que la figura 50, para el caso de tracking una trayectoria personalizada de 60 segundos de duración con λ 0.2

En las gráficas de la figura 62 pueden observarse picos correspondientes a los momentos de cambio brusco en la trayectoria personalizada.

En la figura 63 puede verse una secuencia de la tarea de *tracking* correspondiente al vídeo con perspectiva lateral.

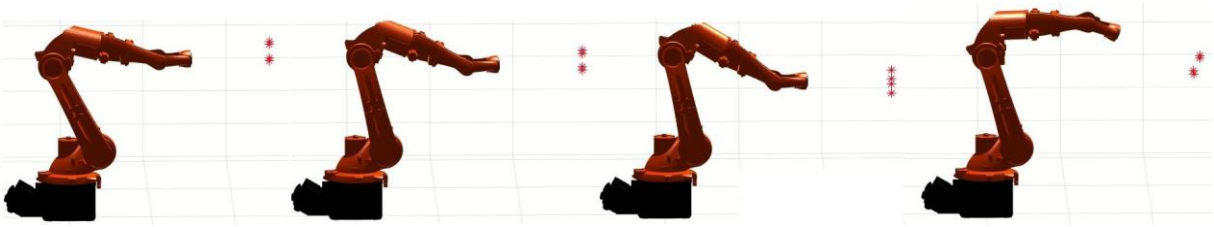


Figura 63- Secuencia del tracking IBVS de una trayectoria personalizada

3.4.4 Control PBVS eye-in-hand:

Los distintos aspectos de interés del control PBVS se han probado en la aplicación mediante múltiples simulaciones, tanto en tareas de posicionado como en tareas de *tracking* con diferentes trayectorias.

En todos los casos, el vector de características de referencia, s^*, es [0,0,0.48,0,0,0]. La referencia se ha elegido para que la posición deseada del robot respecto del *target* sea prácticamente coincidente con la de las simulaciones del control IBVS. Puede comprobarse que es así en la figura 64, que compara la vista de la cámara en la primera iteración de un control de trayectoria de prueba circular con las mismas condiciones para el caso IBVS y el caso PBVS.

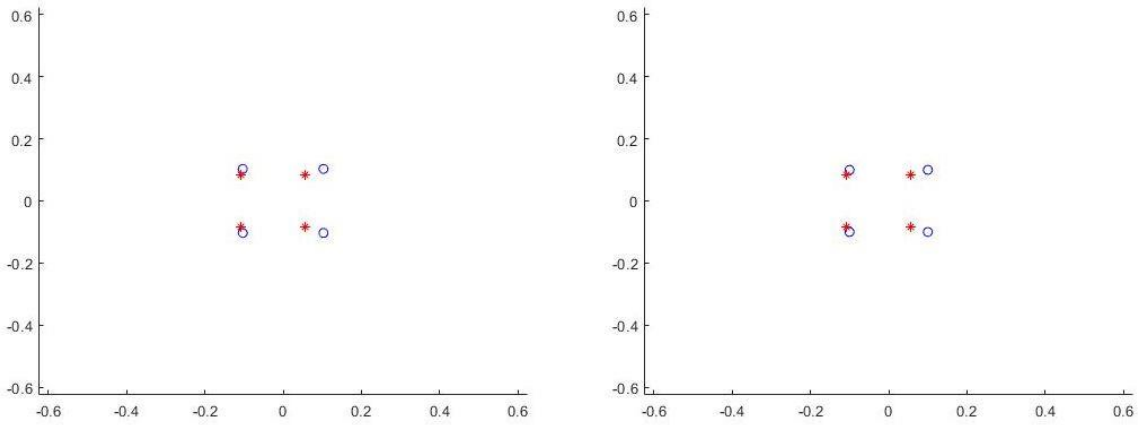


Figura 64- Para la misma posición del target y el robot Der) Vista de la cámara PBVS, izq) Vista de la cámara IBVS

3.4.4.1 Posicionado:

Las simulaciones documentadas de posicionado pretenden mostrar que el descenso del error es exponencial, que la convergencia es más rápida con una λ mayor y que, a diferencia del caso IBVS, los puntos del *target* no se trata de que se desplacen siguiendo líneas rectas en el plano imagen y pueden resolverse rotaciones deseadas sin que se dé el fenómeno de *camera-retreat*.

Las primeras dos simulaciones muestran precisamente la evolución del error y el efecto de λ, así como las trayectorias no rectilíneas de los puntos del *target* sobre el plano imagen. El *target* se encuentra fijo en la misma posición que las simulaciones de posicionado 3 y 4 del apartado 3.4.3.1.

La primera simulación utiliza una λ de 0.2, puede verse en el vídeo enlazado a continuación y su correcto desempeño se puede apreciar en las gráficas de la figura 65:

https://youtu.be/v5Q69o33NR0

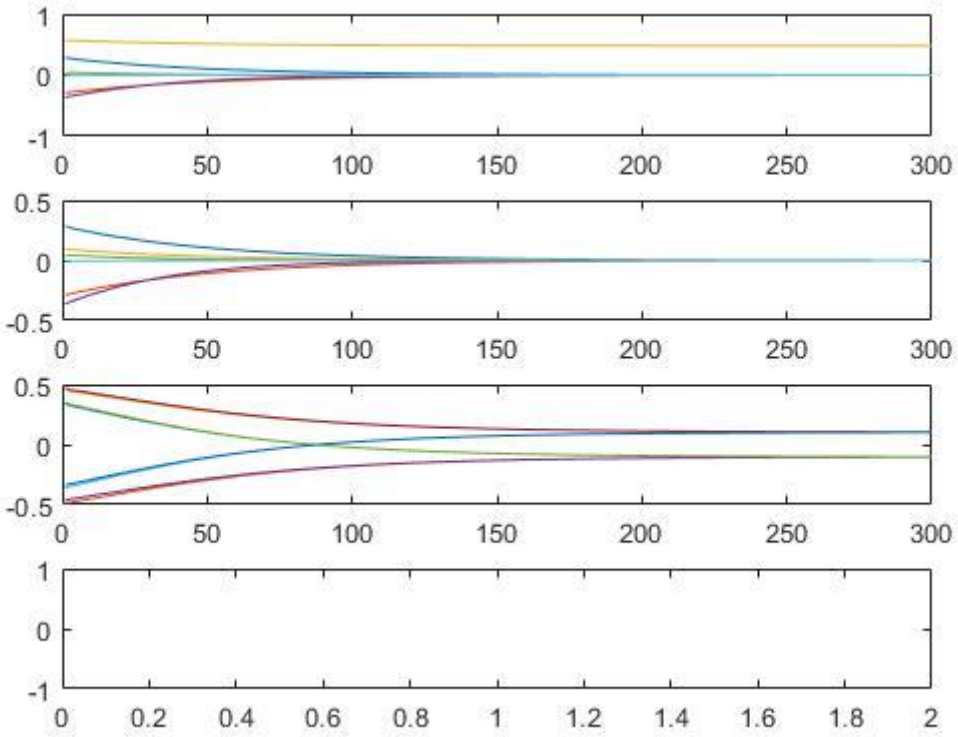


Figura 65- Desde la gráfica superior: evolución de las características del vector s, del error de posición, movimiento de los puntos del target sobre el plano imagen y término de feedforward, para el caso de posicionado con λ 0.2

La segunda simulación utiliza una λ de 0.5, de modo que el desempeño es igualmente correcto y la convergencia es más rápida, como se esperaba. Esto puede observarse en el vídeo siguiente y en la figura 66:

https://youtu.be/AK7UedH2-Fg

Nótese cómo, aunque la mayor diferencia con el caso de IBVS en cuanto al desplazamiento de los puntos del *target* sobre el plano imagen se observa en el punto de vista de la cámara representado en los vídeos correspondientes, en este segundo caso de simulación es más evidente al observar las gráficas que el control no pretende aproximar ese movimiento a trayectorias rectilíneas.

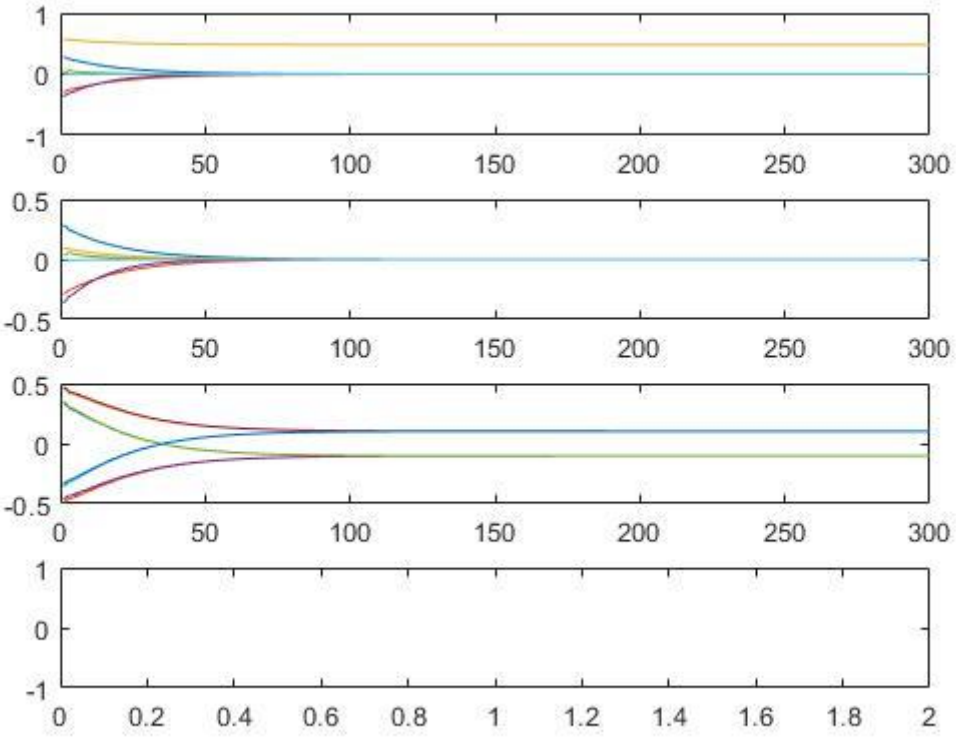


Figura 66-Ídem que la figura 65, para el caso de posicionado con λ de 0.5

En cuanto a la prueba de que el control PBVS no está afectado por el fenómeno de *camera-retreat*, la simulación siguiente reproduce uno de los casos de fallo por este fenómeno en IBVS (giro demandado de 120º), con una λ de 0.5 y se confirma cómo, efectivamente, es capaz de cumplir la tarea siguiendo el comportamiento esperado para el control PBVS. Ello puede comprobarse en el vídeo a continuación y las figuras 67 y 68

https://youtu.be/BaEBH-jcm4U

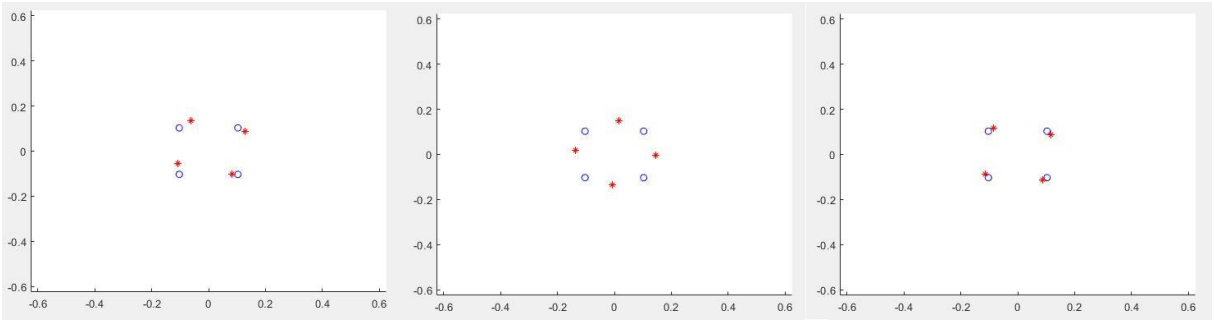


Figura 67--Secuencia del posicionado PBVS que causaría fallo de camera retreat en IBVS, punto de vista de la cámara

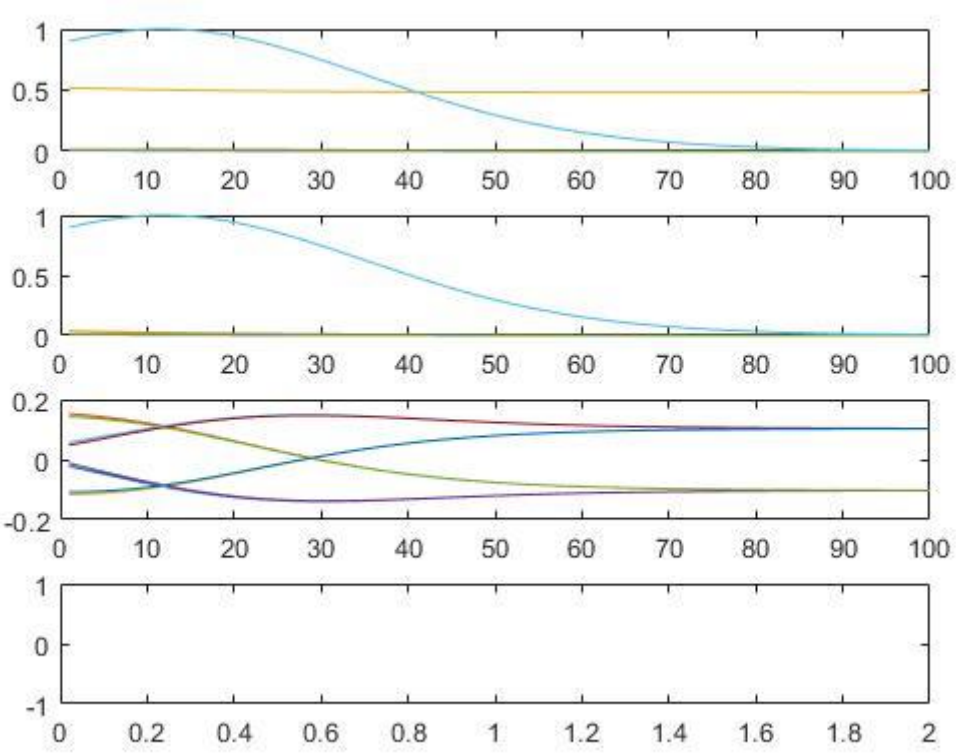


Figura 68-Ídem que la figura 65, para el caso de posicionado PBVS que provocaría fallo de camera-retreat en IBVS

3.4.4.2 Seguimiento:

Para la simulación del seguimiento de *targets* móviles mediante PBVS se han documentado 84 casos de simulación utilizando la ley de control con pre alimentación. Con dichos casos se ha comprobado tanto el correcto funcionamiento del control implementado como el efecto de diferentes parámetros sobre el desempeño del control, de acuerdo a lo tratado en el apartado 2.4.

Todas las simulaciones de este apartado se han realizado con el robot por defecto (KUKA KR 5-2 ARC HW) y un periodo de muestreo de 0.1 segundos.

En primer lugar, se recoge un caso simulado de control PBVS que logra completar con éxito una tarea de *tracking* análoga al segundo caso de simulación de *tracking* IBVS, una trayectoria circular de 200 segundos de duración, con λ de 0.2. El desempeño es el adecuado, como se confirma en el vídeo enlazado a continuación y las gráficas de la figura 69:

https://youtu.be/X29YF_Ar45k

Se observa cómo, al igual que ocurría en el caso IBVS, el término de pre alimentación logra el descenso exponencial del error de posición hasta hacerlo nulo.

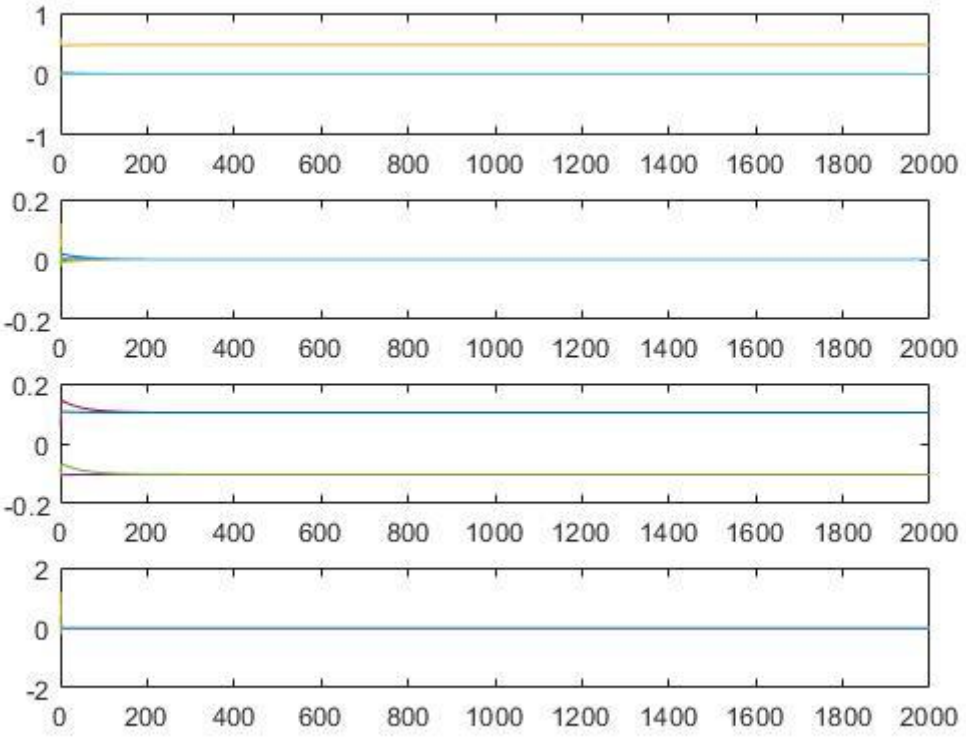


Figura 69-Ídem que la figura 65, para el caso de tracking PBVS con trayectoria circular de 200 segundos y λ 0.2

En cuanto al efecto del término de pre alimentación sobre el desempeño del control: el efecto del movimiento del *target* sobre la evolución del error de posición sigue calculándose como en la expresión {2.4.1.12}, pero en PBVS la estabilidad global del control está garantizada siempre que los parámetros de posición del *target* estén bien estimados, y en el caso de la simulación son conocidos, como se explica en el apartado 3.3.8.

Por ello, la estabilidad global del control PBVS simulado está garantizada en todo momento. No obstante, el término *feedforward* sigue siendo el resultado de una aproximación, lo que implica que, si la aproximación no es correcta, afectará negativamente al desempeño del control, por ejemplo al error de posición durante la tarea de *tracking*, aunque no se desestabilice.

Puede comprobarse en la simulación siguiente cómo, en las mismas condiciones en que el control IBVS falla (trayectoria circular de 100 segundos, λ de 0.5 y periodo de muestreo de 0.1 segundos), el control simulado PBVS cumple la tarea satisfactoriamente. Las gráficas correspondientes se encuentran en la figura 70:

https://youtu.be/Qx0VdAoPWGU

Efectivamente, el error de posición se anula y no hay desestabilización.

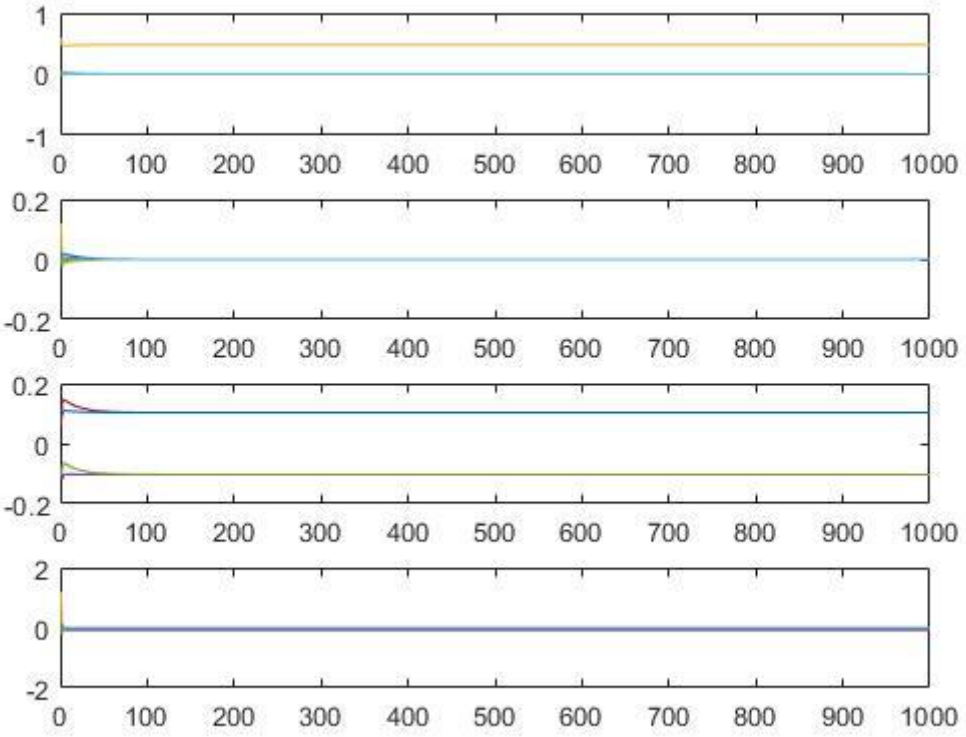


Figura 70- Ídem que la figura 65, para el caso de tracking de trayectoria circular de 100 segundos y λ de 0.5

En la siguiente simulación, se muestra un caso de movimiento del *target* más veloz y con más cambios de tendencia, utilizando la trayectoria de prueba "corona" recorrida en tan solo 30 segundos, con una λ de 0.5. Lo que se pretende mostrar es cómo el control no se desestabiliza en un caso en que sin duda el efecto del *feedforward* estimado se aleja significativamente del ideal, pero sí tiene un efecto negativo sobre el error de posición, no siendo capaz de anularlo.

Ello está recogido en el vídeo a continuación y las gráficas de la figura 71:

https://youtu.be/u8sC8VZ7xq4

En las gráficas de la figura 71 puede comprobarse cómo el término de *feedforward*, al calcularse con datos de la iteración actual y la anterior, no es una estimación correcta frente a los rápidos cambios de dirección del *target*, y cómo ello se refleja en un error de posición y unos puntos del *target* sobre el plano imagen que no convergen al valor deseado, sino que oscilan entorno a dicho valor.

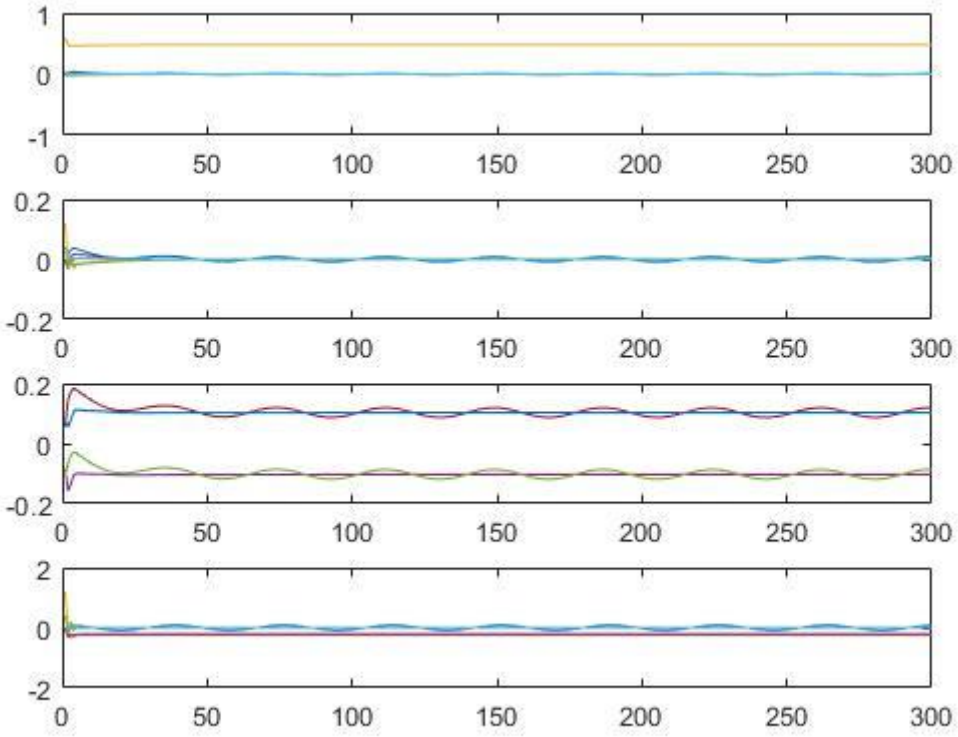


Figura 71-Ídem que la figura 65, para el caso de tracking de trayectoria de prueba "corona" de 30 segundos y λ de 0.5

Para concluir, se muestra como en el caso de IBVS, el desempeño correcto que se logra ante la trayectoria personalizada, en este caso con una λ de 0.5. De nuevo, puede observarse la simulación en los vídeos a continuación, grabados con una perspectiva frontal y lateral, respectivamente:

https://youtu.be/2byd6r6RCcI

https://youtu.be/4v1CFnPmZ1M

Las gráficas correspondientes a esta tarea se pueden observar en la figura 72. De nuevo, como ocurre en el control IBVS análogo, pueden observarse picos correspondientes a los cambios de dirección bruscos que realiza el *target.*

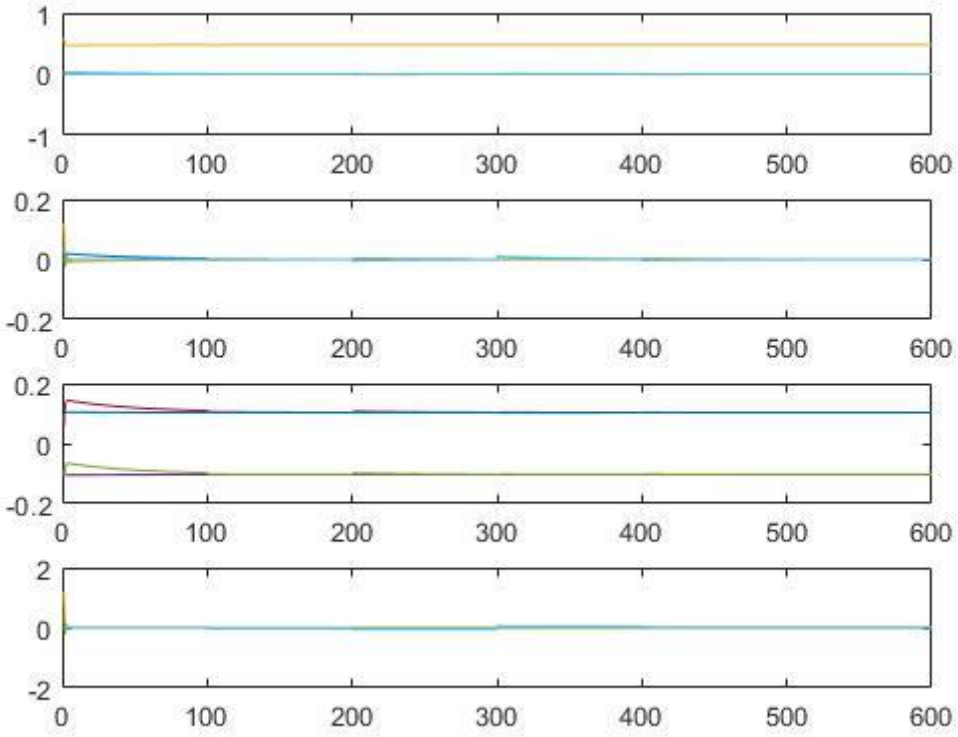


Figura 72-Ídem que la figura 65, para el caso de tracking de la trayectoria personalizada con λ de 0.5

CAPÍTULO 4. VALIDACIÓN EXPERIMENTAL

4.1 Disposición del entorno de experimentación:

La validación experimental de los conocimientos adquiridos sobre *Visual Servoing* ha tenido lugar en las instalaciones del Instituto de Diseño y Fabricación (IDF) de la UPV, trabajando para ello sobre una configuración previamente establecida como resultado de trabajos de investigación realizados por el IDF en el pasado.

Dicha configuración incluye los siguientes elementos:

- Un brazo robótico industrial *KUKA Agilus KR6 900 sixx* con 6 articulaciones de rotación, situado dentro de una estructura (figura 73) que delimita su espacio de trabajo y fijado al techo de la misma a través de un eje lineal que le otorga un grado de libertad adicional (figura 74), que no se ha utilizado para el control pero sí para obtener la configuración definitiva para la experimentación.

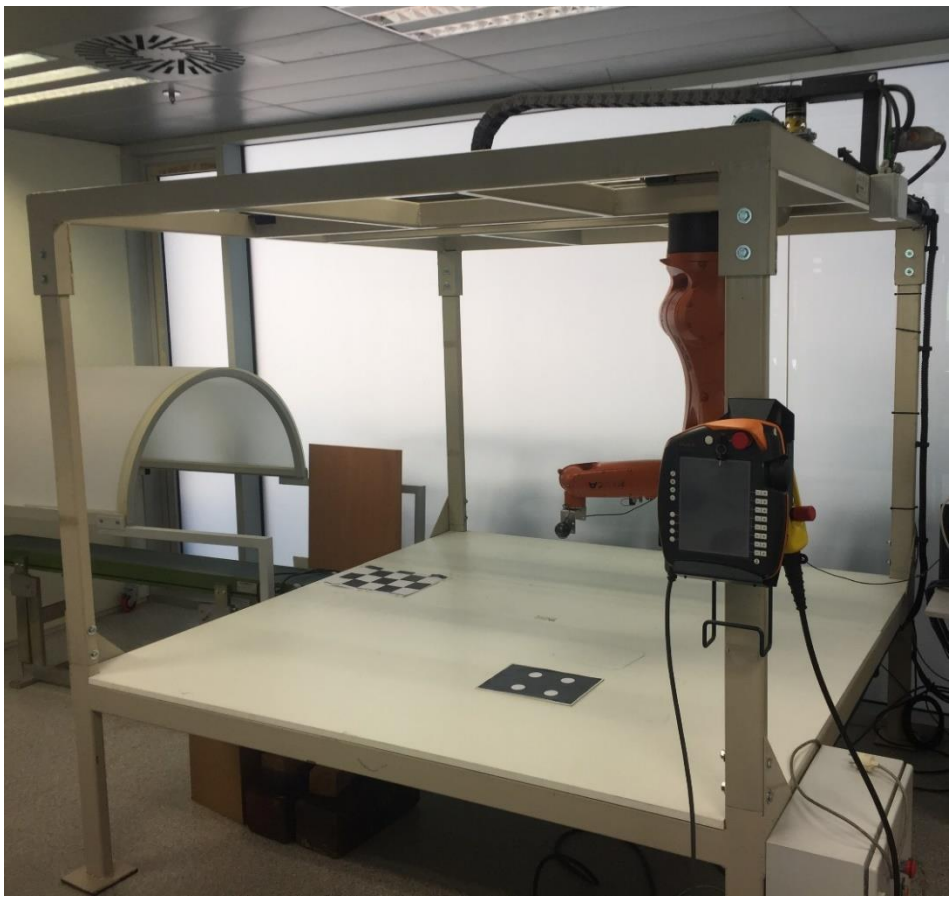


Figura 73-Espacio de trabajo del robot KUKA Agilus

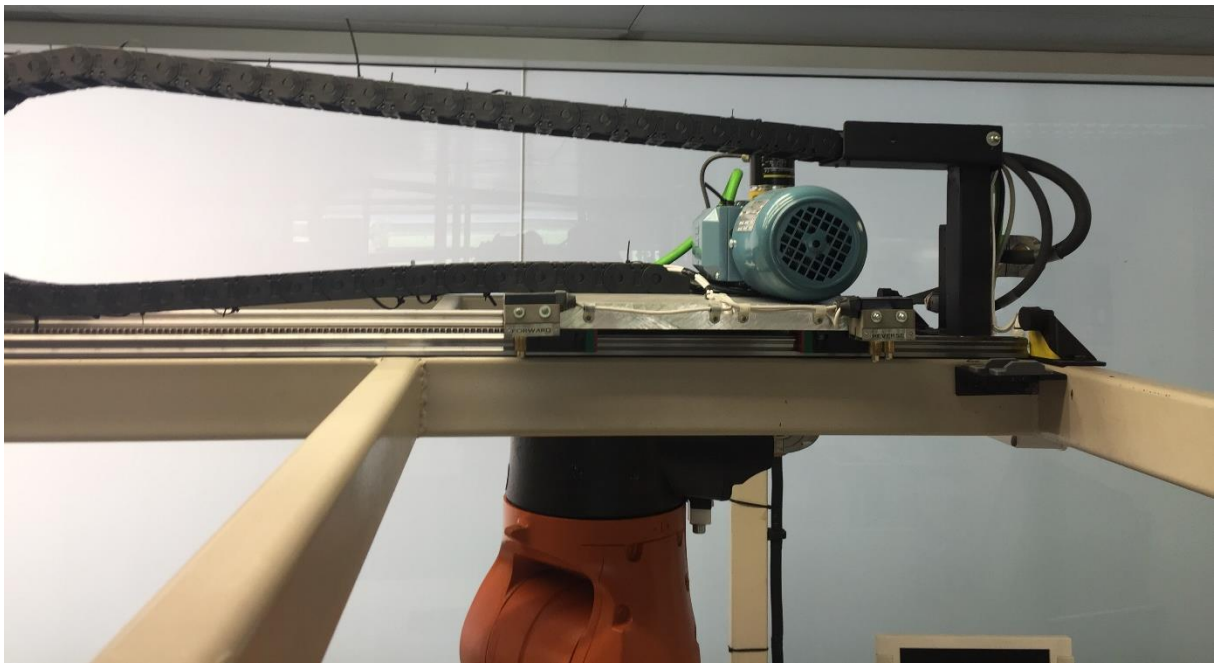


Figura 74-Carril lineal al que está fijado el robot KUKA

- Un controlador *KUKA KRC4 Compact* (figura 75) [19], que permite la comunicación empleando la tecnología *KUKA RSI XML Ethernet*, que permite la comunicación a tiempo real (haciendo un *by-pass* del núcleo del sistema operativo del robot KUKA) con equipos externos utilizando un cable *Gigabit Ethernet* [19] a través de protocolos TCP/IP o UDP/IP, enviando los datos como cadenas XML [20]. El controlador incluye su correspondiente *KUKA smartPAD* (figura 75) [19], que permite el control manual y automático del robot, y permite la programación en lenguaje KRL necesaria para determinar la configuración de partida para los experimentos y albergar el programa cliente del robot.

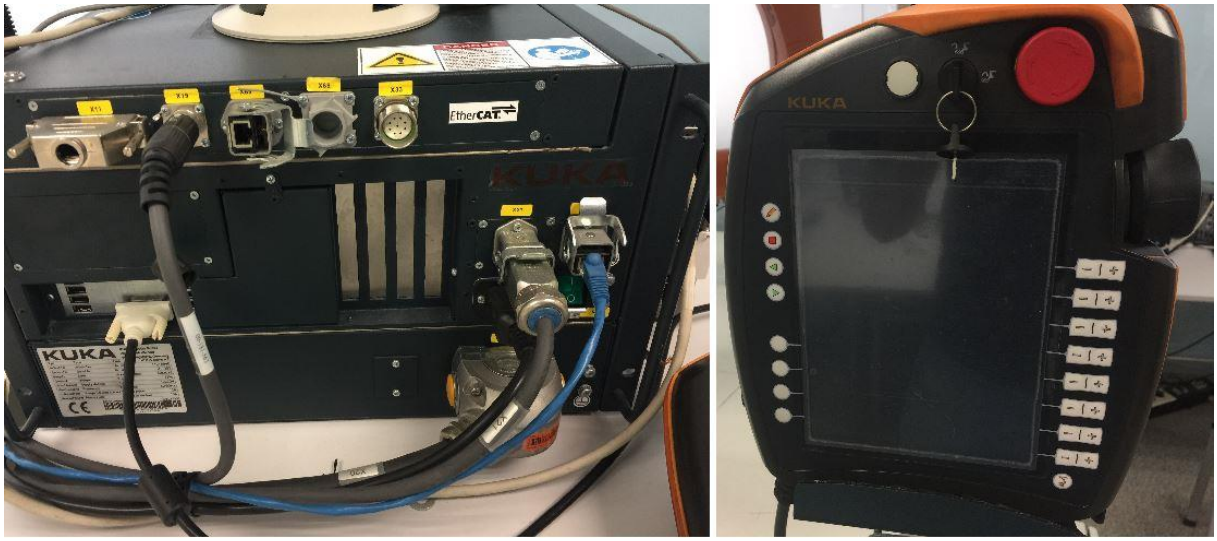


Figura 75-Izq) Controlador KUKA KRC4 Compact, der) KUKA smartPAD

- Una *webcam Logitech C300* capaz de capturar vídeo a 30 fps y con una resolución de hasta 1280x1024. La cámara es solidaria al efector final del robot KUKA, fijada en él tal y como se

muestra en la figura 76. Esta cámara está conectada por USB a un PC que se tratará a continuación.

Figura 76-Webcam Logitech C300 fijada solidariamente al efector final del robot KUKA Agilus

- Un PC (figura 77) con el sistema operativo *Ubuntu* 12.04 con el meta sistema operativo de código libre enfocado a robótica, ROS (Robotics Operating System) [21], la librería de visión artificial *OpenCV*, la librería VISP (*Visual Servoing Platform)* basada en *OpenCV* y encargada de las tareas de procesado de la imagen y obtención de las características visuales y otros parámetros necesarios para el *Visual Servoing* a partir de la imagen [22], y la librería *OROCOS Toolchain* que permite crear aplicaciones de control de robots en tiempo real trabajando con ROS [23].

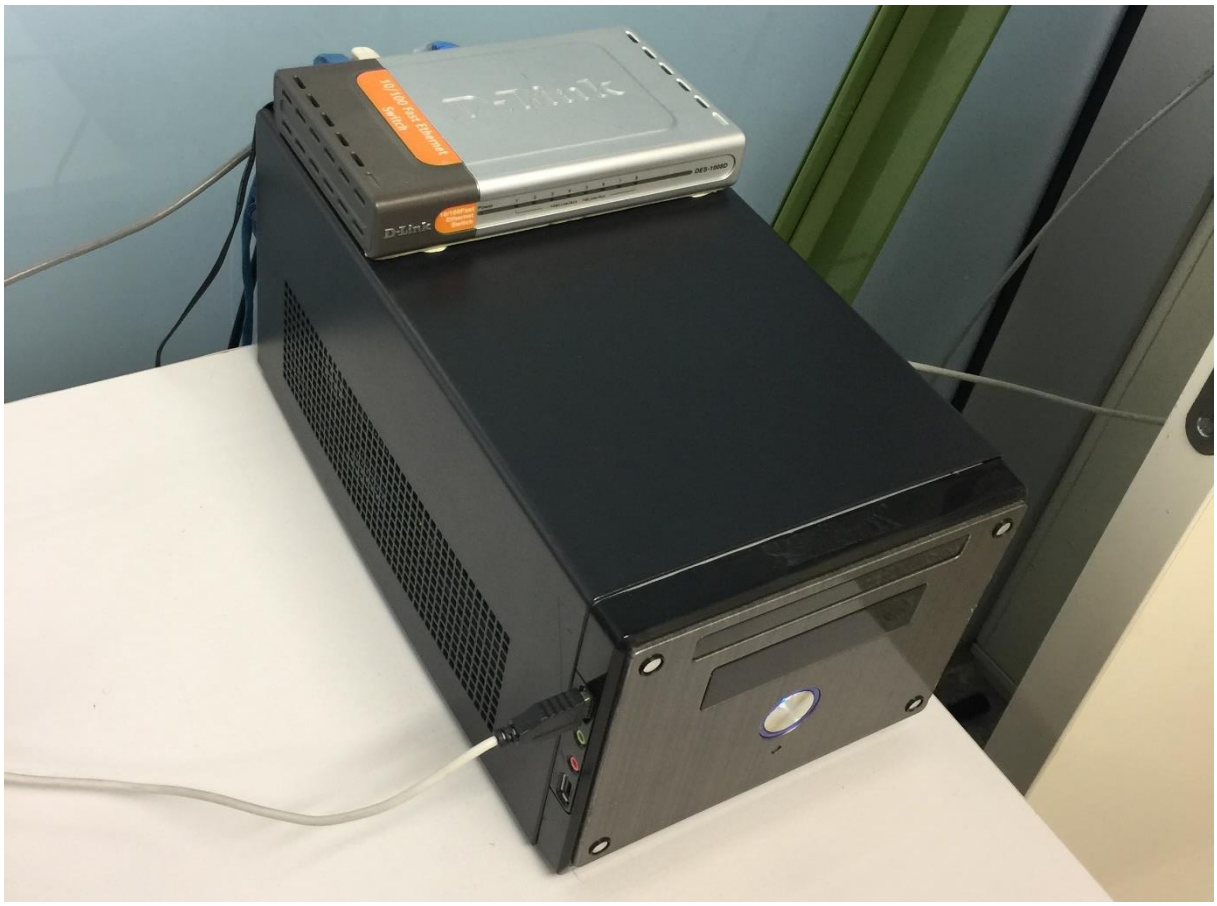


Figura 77-PC externo utilizado para la experimentación

- Un *target* que consiste en una matriz cuadrada de 10 cm de lado de puntos blancos sobre fondo negro (figura 78).

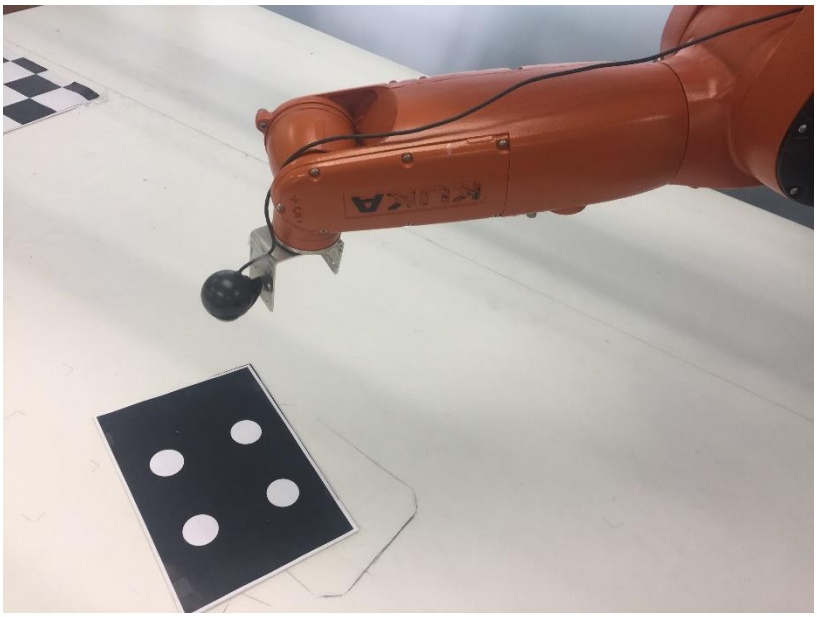


Figura 78-Target utilizado para la experimentación real

Particularmente, la configuración heredada de anteriores trabajos académicos y de investigación realizados en el IDF utiliza el protocolo de comunicación entre ordenador y controlador del robot de tipo UDP/IP, debido a que es más rápida que la TCP/IP, lo que la hace muy extendida en la industria [24], en la que el programa cliente está en el controlador KRC4 y el servidor en el PC.

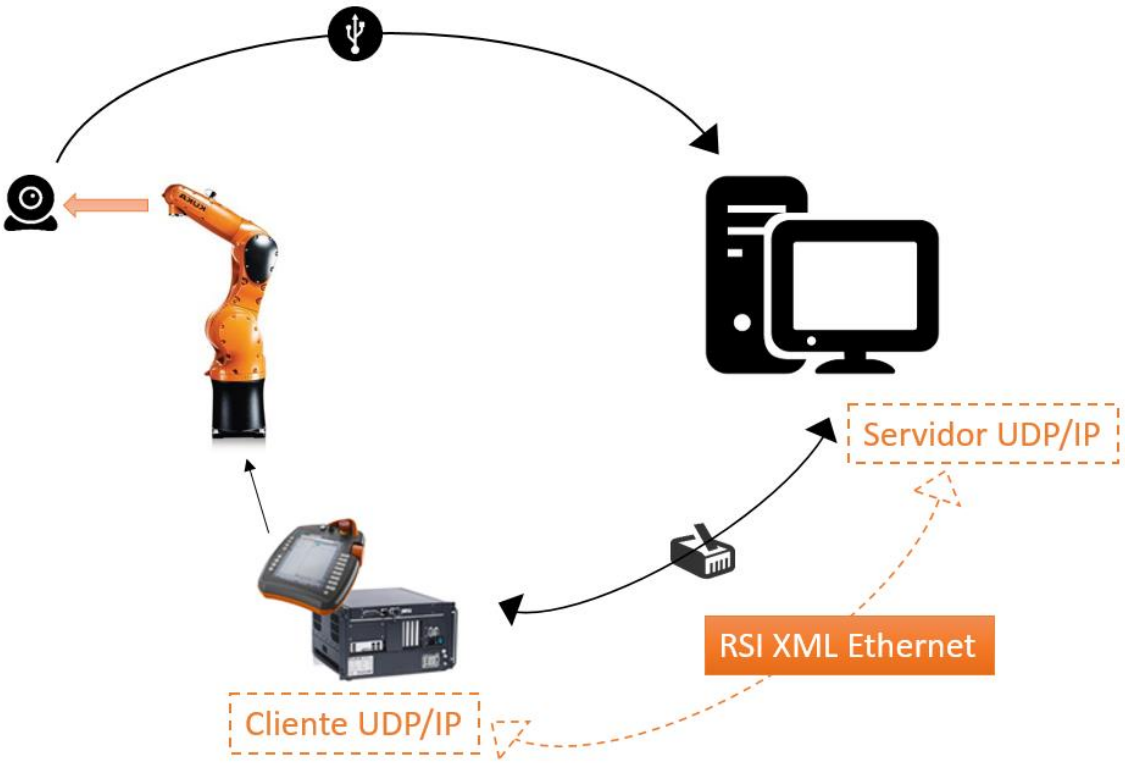


Figura 79-Diagrama de bloques de la comunicación para la experimentación

Además, también como punto de partida, se cuenta para el presente trabajo académico con proyectos desarrollados en C++ para el posicionado IBVS y PBVS, de manera que el control del robot mediante *Visual Servoing* está ya implementado. Estos proyectos cuentan con un módulo de control, otro de visión y otro de comunicación (el servidor); de estos tres módulos, el de interés para el presente trabajo académico es el de control.

Así, las tareas desarrolladas de cara a la experimentación real en el marco del presente Trabajo Fin de Máster han sido:

- Familiarización con los elementos de la configuración de partida.
- Modificación de la configuración de partida alterando la configuración inicial del robot, mediante el uso del eje lineal y el *KUKA smartPAD*.
- Diseño, ejecución, documentación y análisis de los experimentos.
- Modificación de los proyectos C++ de control IBVS y PBVS, particularmente de los archivos que contienen el bucle de control (*IBVSControl y PBVSControl*) y de los archivos que contienen los parámetros de control (*Control_config*) y que determinan la referencia (*DesiredPoints_real_IBVS* y *DesiredPoints_real_PBVS*).

4.1.1 Modificaciones realizadas sobre la configuración inicial:

En primer lugar, se ha desplazado el robot utilizando el eje lineal móvil hasta su final de carrera derecho con el objetivo de facilitar la grabación de vídeos para documentar las pruebas.

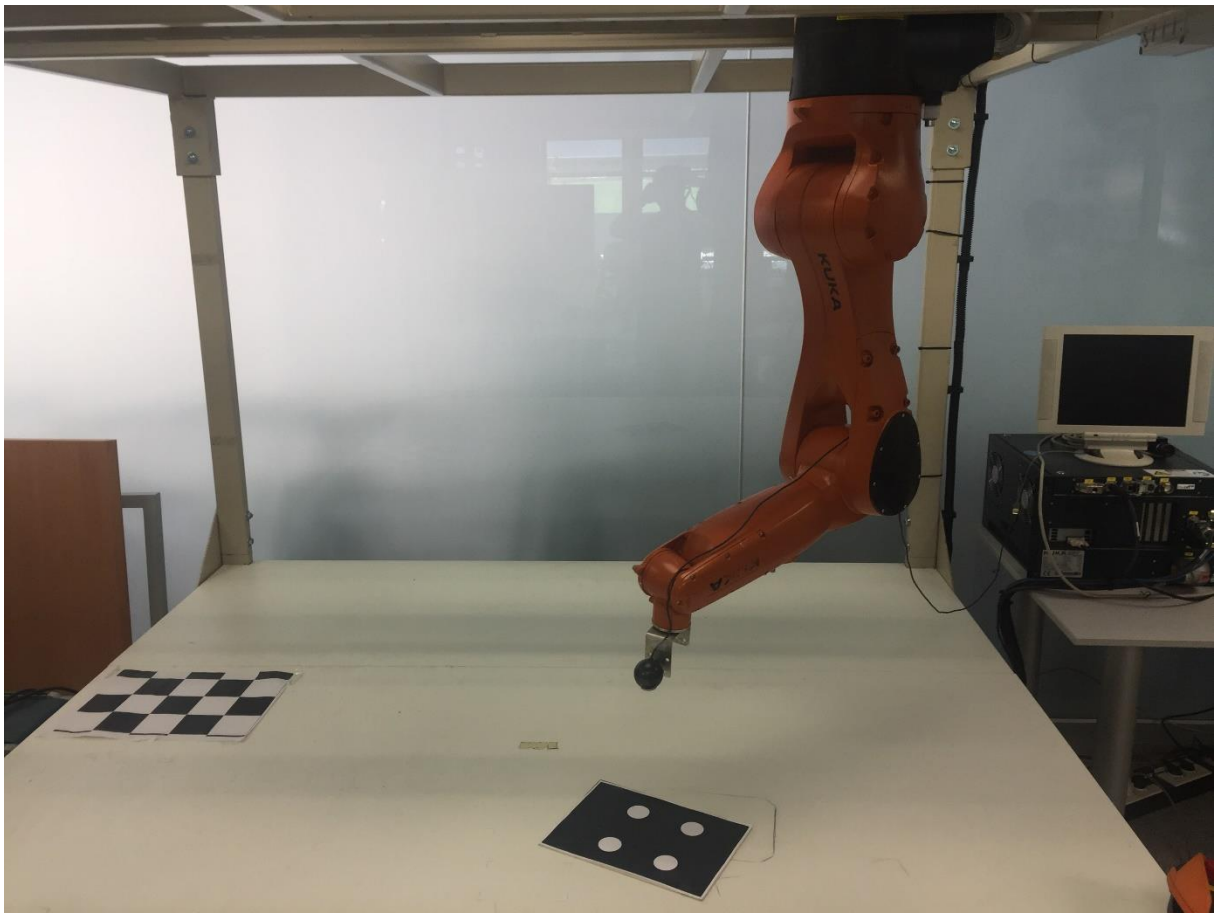


Figura 80-KUKA Agilus en la posición deseada dentro del espacio de trabajo

En segundo lugar, la posición articular de partida del robot también se ha modificado en diversas ocasiones según las necesidades de cada experimento. Ello requiere cambiar la posición de partida en el programa que habilita el cliente UDP, lo cual es posible desde el *KUKA smartPAD* recurriendo a su capacidad de comunicación USB para conectar un teclado, y modificando la línea del programa cliente UDP correspondiente a dicha posición de partida. La posición deseada se puede hallar pasando a modo manual o tomando como posición deseada la posición final de una prueba de *Visual Servoing*, y después accediendo en el *smartPAD* al valor angular de las articulaciones.

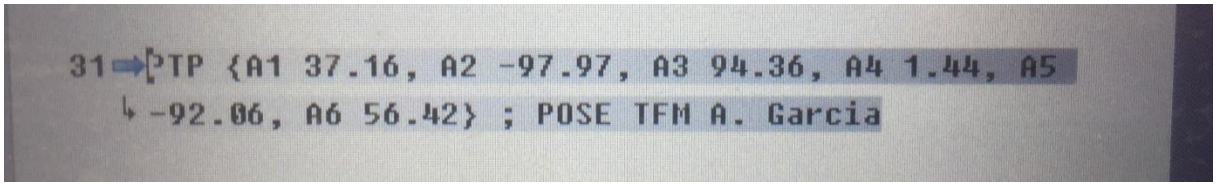


Figura 81-Posición de partida del programa cliente UDP en el smartPAD del controlador KRC4

En cuanto a los archivos que contienen los bucles de control *Visual Servoing* y otros archivos necesarios para realizar los experimentos, se han realizado las siguientes modificaciones:

- En los archivos C++ que contienen el bucle de control, se ha eliminado una condición que incrementa la ganancia proporcional λ cuando el valor del error es pequeño. Esta es una propiedad interesante para un control más rápido, pero dificulta la comprobación de las conclusiones respecto al efecto de una mayor o menor ganancia proporcional sobre el tiempo de establecimiento.
- En el archivo C++ que contiene el bucle de control del posicionado IBVS se ha añadido el término de *feedforward* para habilitar el *tracking* como funcionalidad. Se volverá sobre esta cuestión más adelante en el apartado correspondiente.
- Modificación del valor de la ganancia proporcional en el archivo XML de nombre *Control_config* para ajustarla a los distintos experimentos.
- Modificación de los archivos que recogen el valor de referencia del vector de características visuales, s*, de manera que la referencia sea la misma para ambos controles.

4.2 Validación experimental:

Antes de exponer los experimentos realizados y las conclusiones extraídas, es conveniente detenerse a explicar brevemente el funcionamiento de los programas implicados.

Como ya se ha dicho, dentro de los proyectos C++ que realizan el control IBVS y PBVS para tareas de posicionado, el módulo de control tiene como elemento más relevante el archivo de código fuente con la ley de control, que funciona de forma esencialmente igual al mismo bucle en el simulador, con la excepción de que el programa finaliza cuando el error está por debajo de un cierto umbral (5 mm) o cuando se incumple alguna de las condiciones de seguridad que se tratarán en el apartado de *tracking*.

Por otro lado, el módulo de visión muestra por pantalla el punto de vista de la cámara, y espera a que se hayan seleccionado con el puntero del ordenador los que serán los 4 puntos del *target*, que deben escogerse en sentido horario, empezando por la esquina superior izquierda si se desea que cada punto del *target* esté lo más cerca posible de la referencia.

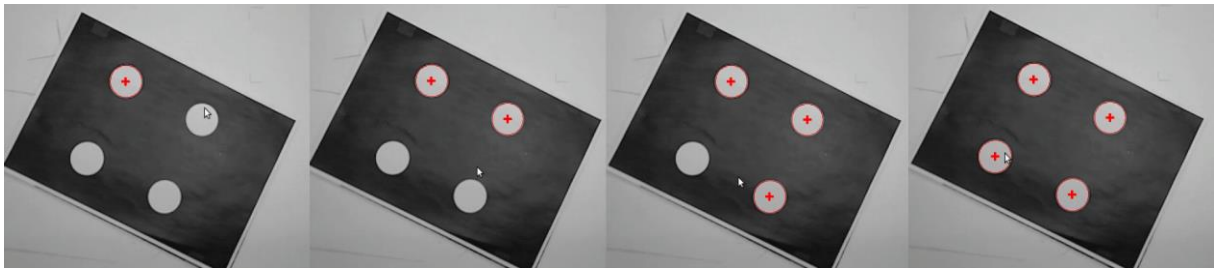


Figura 82-Selección de puntos del target en sentido horario

Una vez se eligen los 4 puntos del *target*, el programa intenta enviar los valores de referencia de las posiciones articulares del robot al controlador mediante el módulo de comunicación, actuando como servidor. Para que puedan enviarse los datos al controlador del robot, el programa cliente debe haberse puesto en marcha antes desde el *KUKA smartPAD* y haber llegado a la línea del programa que pone al robot a la espera de datos del servidor. Por supuesto, a su vez, para que el cliente UDP funcione, el programa servidor debe estar previamente en marcha y, como se ha dicho, sin haber enviado datos todavía.

Si se marca el cuarto punto del *target* y el cliente no está conectado, el programa finaliza. Del mismo modo, si el programa cliente llega a la línea de espera de datos del servidor y el servidor no está operativo, el programa cliente concluye y es necesario reiniciarlo. Para que el controlador pueda hacer funcionar el programa cliente, es necesario haberlo puesto antes en modo automático y haber habilitado los accionamientos del robot.

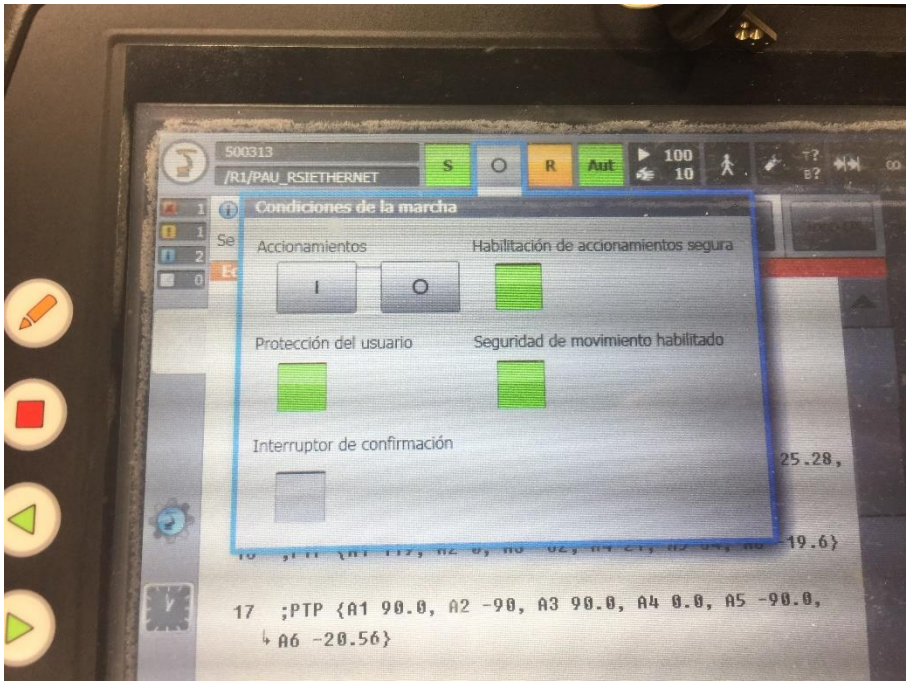


Figura 83-KUKA smartPAD, habilitación de los accionamientos en modo automático

Una vez se han cumplido estas condiciones, como se ha comentado, comienza el control IBVS o PBVS y el módulo de visión se encarga de mostrar por pantalla de forma actualizada el punto de vista de la cámara, sobreimprimiendo el recorrido del centro los puntos del *target* y la referencia. Además, se encarga de finalizar el programa si alguno de los puntos sale del campo visual de la cámara, lo cual ocurre cuando ya no es posible calcular un centro del área blanca del *target*.

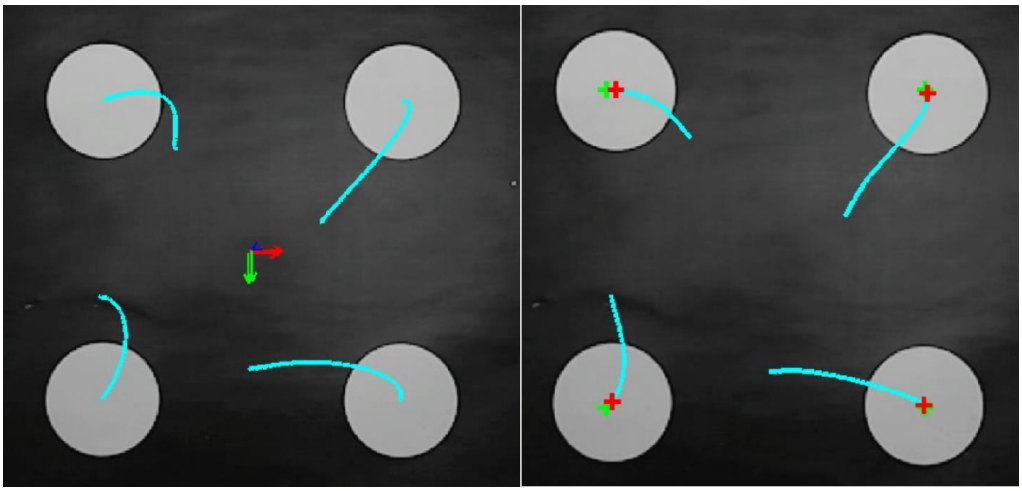


Figura 84-Punto de vista de la cámara, izq) PBVS, der) IBVS

4.2.1 Posicionado:

En todos los experimentos se ha utilizado un periodo de muestreo de 100 milisegundos, y las referencias son:

- IBVS: [-0.1888, -0.1888, 0.1888, -0.1888, 0.1888, 0.1888, -0.1888, 0.1888]
- PBVS: [0, 0, 0.3590414123, 0, 0, 0]

De manera que son referencias coincidentes.

Además, se han grabado todos los experimentos tanto desde el punto de vista de la cámara como desde una perspectiva amplia que capta el movimiento del robot. También se han desarrollado gráficas de los experimentos, que muestran la evolución de de los puntos del *target* sobre la imagen y del vector de características visuales *s*.

Los experimentos diseñados son:

-Experimento 1:

El *target* está situado sobre el plano horizontal, y los puntos se seleccionan para que sean lo más cercanos posible a la referencia. El experimento se realiza tanto en IBVS como en PBVS para poner de manifiesto la diferencia en la trayectoria de los puntos sobre el plano imagen, y además con λ 0.2 y 0.5 en cada tipo de control, para mostrar cómo se reduce el tiempo de establecimiento cuando aumenta la ganancia proporcional, tal y como se concluye también en las pruebas de simulación.

Los vídeos correspondientes a este experimento pueden verse en los enlaces a continuación:

Casos IBVS:

Con λ de 0.2: https://youtu.be/C2_BaNd0vqg

Con λ de 0.5: https://youtu.be/llRGARI5u6c

Casos PBVS:

Con λ de 0.2: https://youtu.be/Q3Z4U1yX1zc

Con λ de 0.5: https://youtu.be/wI9eodptpeM

También se han realizado y descartado pruebas con una ganancia proporcional mayor (0.7) que, debido a las vibraciones del robot, no han presentado un desempeño satisfactorio para mostrar un comportamiento representativo de los controles PBVS e IBVS.

En las figuras 85 y 86, que muestran el punto de vista de la cámara y la posición del robot al final de los experimentos, puede comprobarse cómo efectivamente en IBVS las trayectorias de los puntos del *target* se aproximan a líneas rectas mientras que en el caso de PBVS no es así.

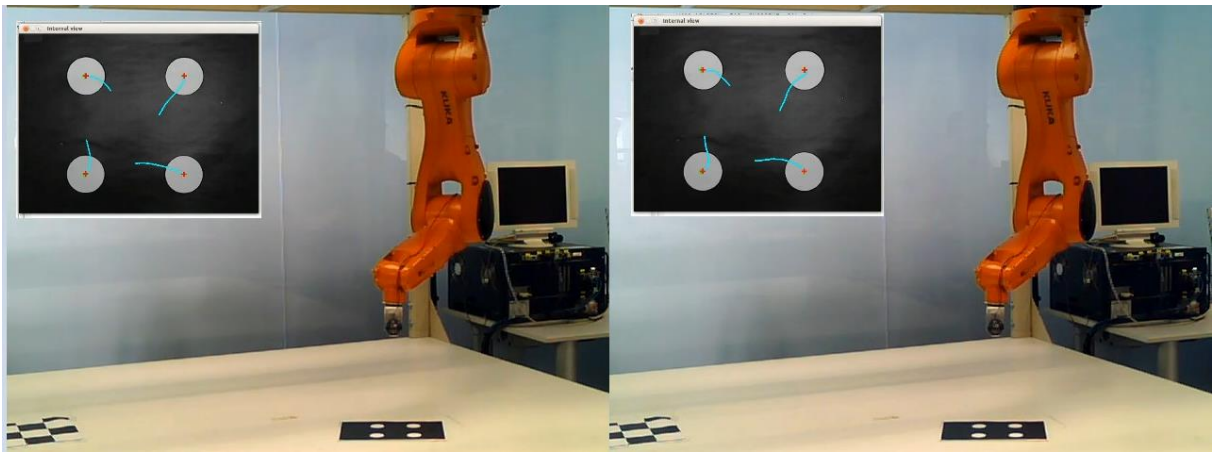


Figura 85-Posición final del primer experimento de posicionado IBVS. Izq) λ de 0.2, der) λ de 0.5

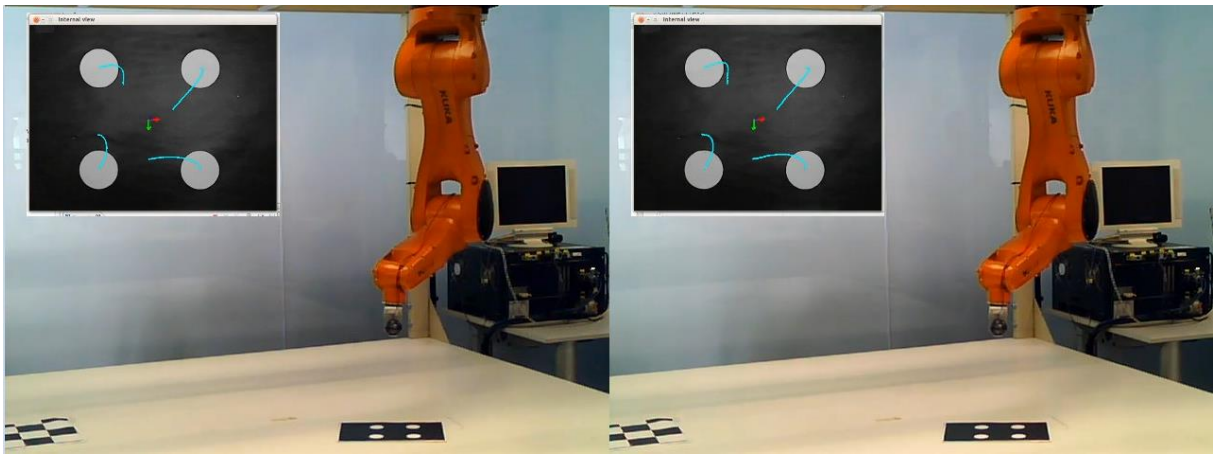


Figura 86- Posición final del primer experimento ded posicionado PBVS. Izq) λ de 0.2, der) λ de 0.5

En la figura 88, que representa la evolución de *s* (también se representa *s** para mayor claridad) frente a los instantes transcurridos, se observa claramente la convergencia correcta y la mayor velocidad al aumentar la ganancia proporcional, que también trae consigo más vibraciones.

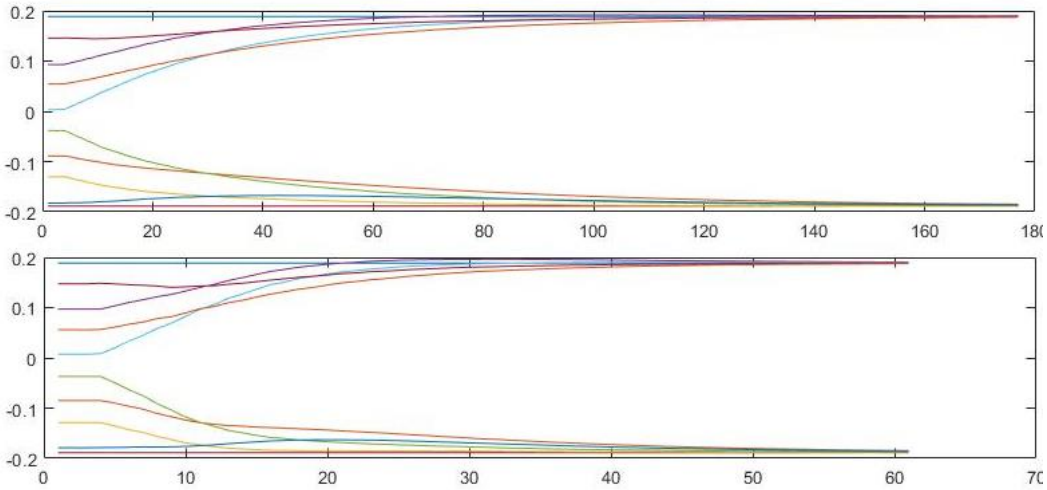


Figura 87-Evolución de s en el primer experimento de IBVS. Sup)) λ de 0.2, inf)) λ de 0.5

En la figura 89, que representa la evolución de los puntos sobre la imagen y del vector *s*, se aprecian tanto las trayectorias parabólicas previstas, como el aumento de la velocidad de convergencia al incrementar la ganancia proporcional, como también con más claridad el efecto de las vibraciones.

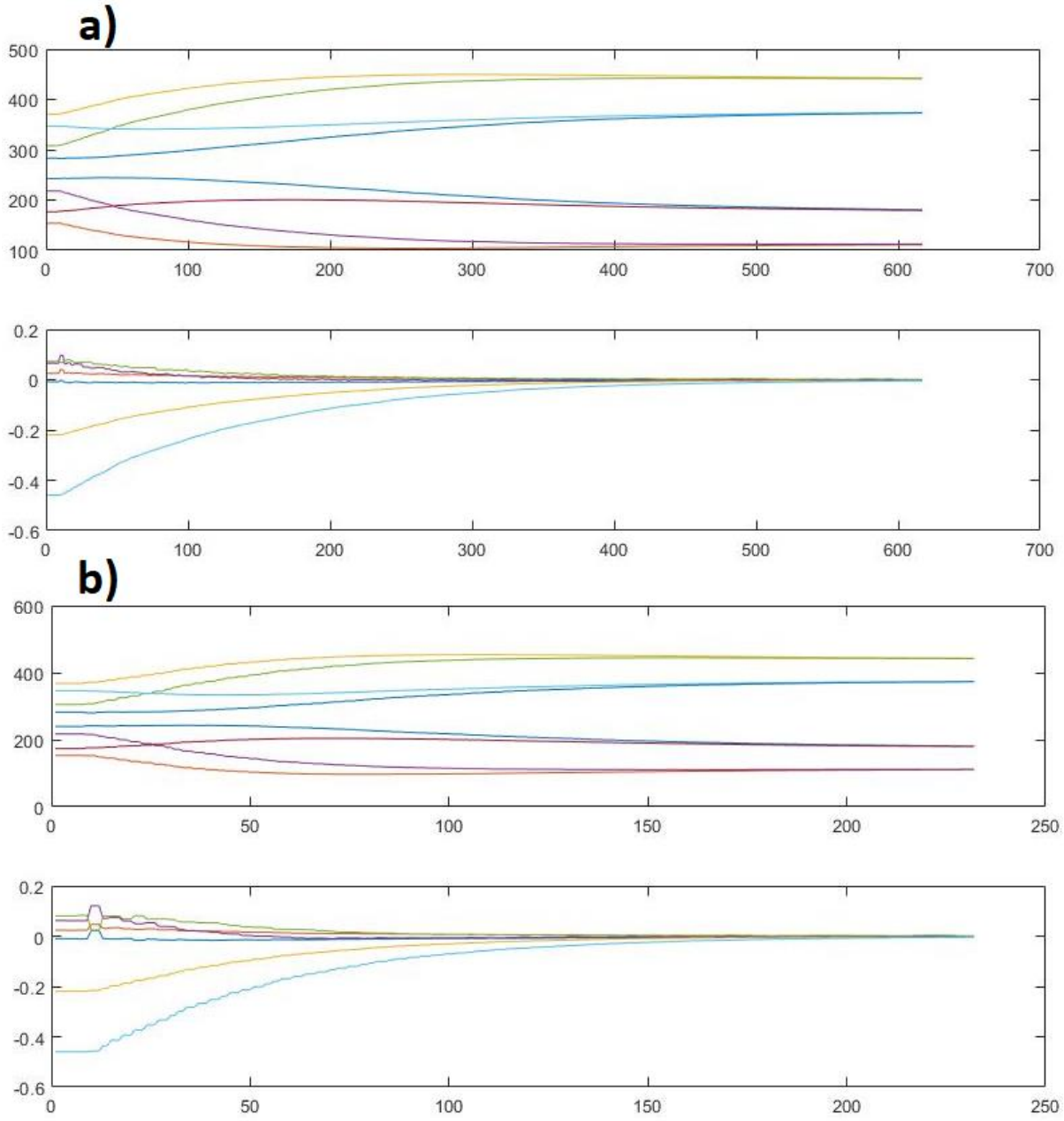


*Figura 88-Evolución de los puntos del target sobre el plano imagen y de c*Mc en el primer experimento de PBVS, a) λ de 0.2, b) λ de 0.5*

-Experimento 2:

El *target* está situado sobre el plano horizontal, y los puntos se seleccionan para que se requiera una rotación amplia. De nuevo, con la misma finalidad que en el primer experimento, se realiza en IBVS y PBVS, con λ 0.2 y 0.5 en cada tipo de control.

Los vídeos correspondientes a este experimento pueden verse en los enlaces a continuación:

Casos IBVS:

Con λ de 0.2: https://youtu.be/siDdK2kjoX8

Con λ de 0.5: https://youtu.be/rtLLuw-6vY8

Casos PBVS:

Con λ de 0.2: https://youtu.be/FnHMWQjzgKY

Con λ de 0.5: https://youtu.be/-ve8opp7hLM

En las figuras 89 y 90, análogas a la 85 y 86, se aprecia todavía más claramente la diferencia en cuanto a la trayectoria de los puntos del *target* en la imagen.

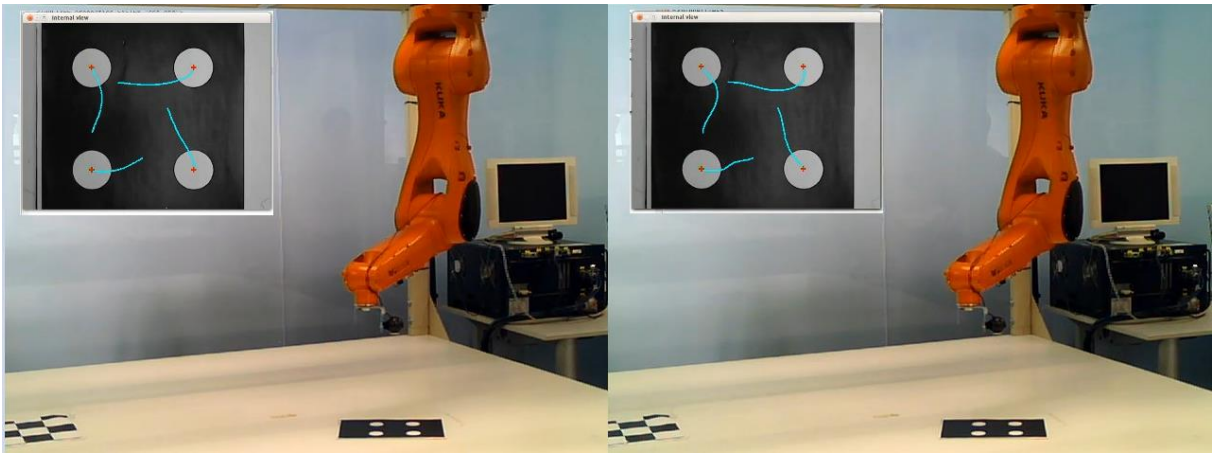


Figura 89- Posición final del segundo experimento de posicionado IBVS. Izq) λ de 0.2, der) λ de 0.5

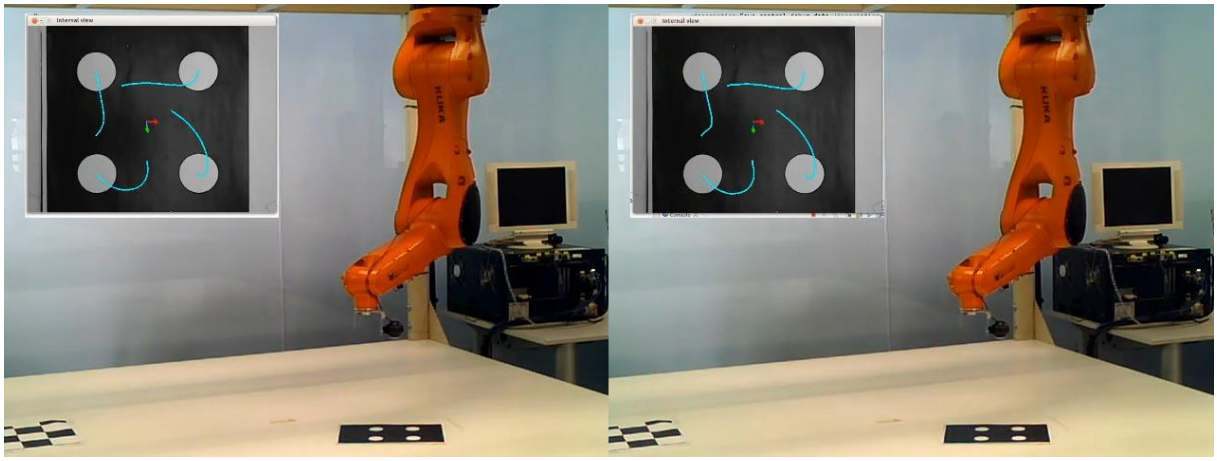


Figura 90- Posición final del segundo experimento de posicionado PBVS. Izq) λ de 0.2, der) λ de 0.5

Nótese además como, en la figura 89, se observa que un control más agresivo también lleva a un desplazamiento de los puntos del *target* sobre el plano imagen que se aproxima menos a una línea recta. Tanto en la figura 89 como 90 se puede comprobar cómo el aumento de la ganancia proporcional da lugar en ambos casos a un movimiento más brusco, con más vibración y oscilación.

Este aumento de las vibraciones puede observarse en las figuras 91 y 92, análogas a las figuras 87 y 88.

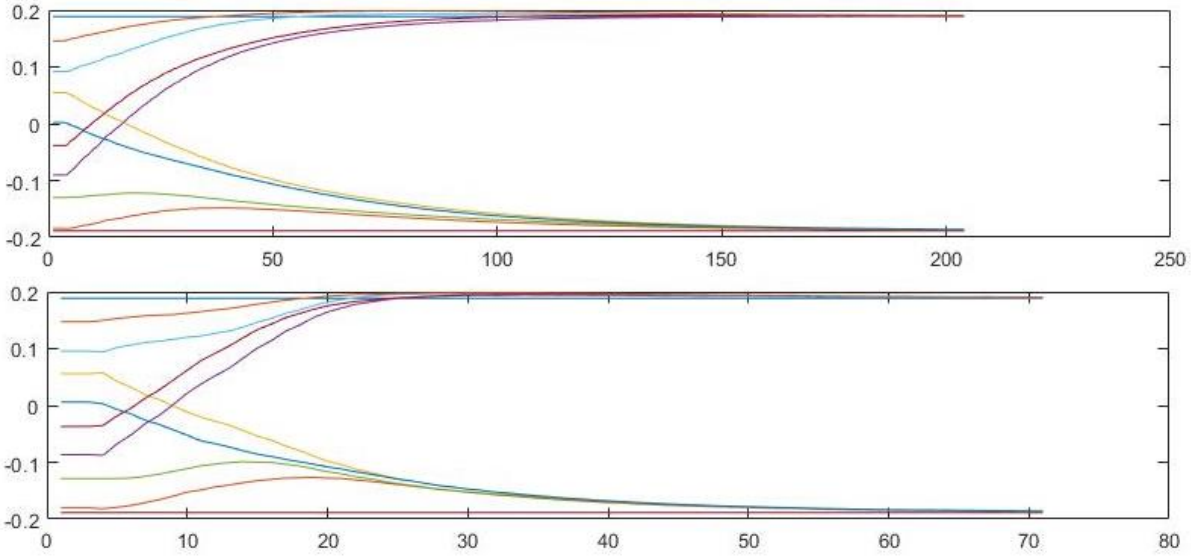


Figura 91- Análoga a la figura 87, para el segundo experimento de IBVS

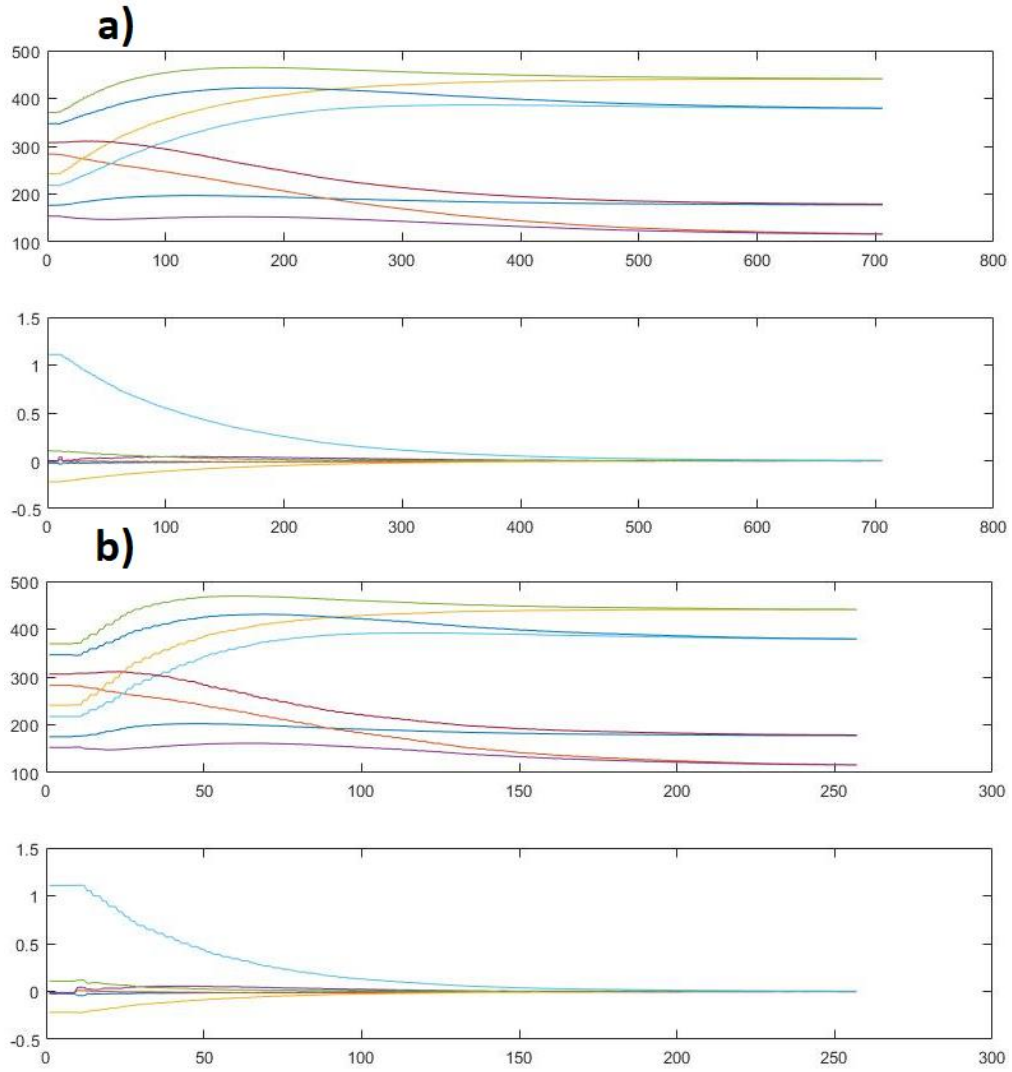


Figura 92- Análoga a la figura 88, para el segundo experimento de PBVS

Una muestra especialmente clara del efecto de las vibraciones a medida que se incrementa la agresividad del control puede observar en la figura 93, que representa la evolución de la velocidad de las articulaciones en el presente experimento para el caso IBVS.

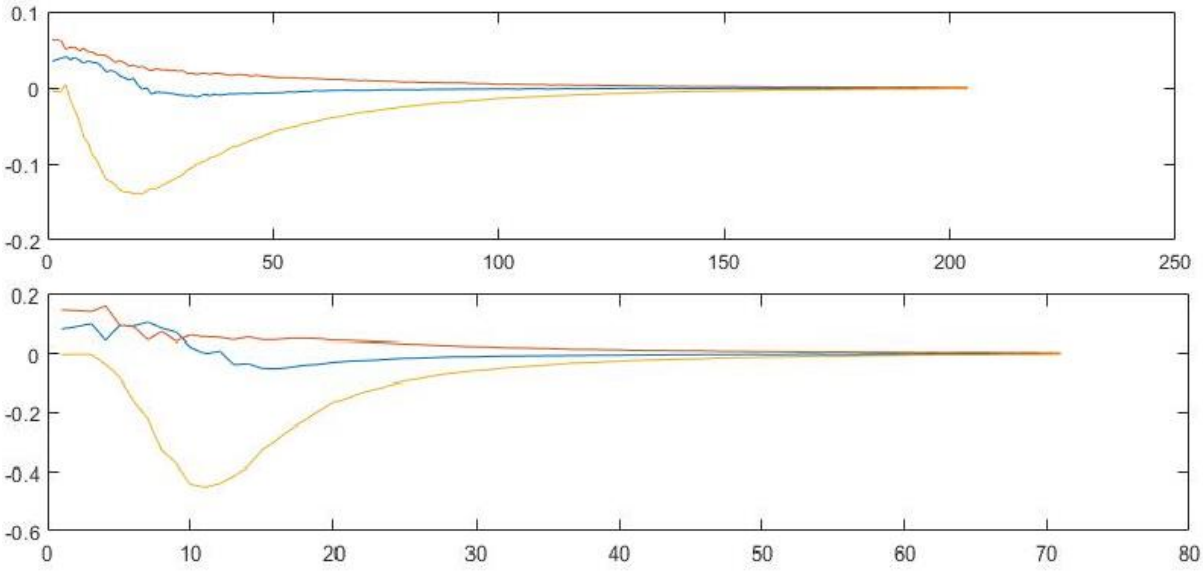


Figura 93- Evolución de la velocidad de las articulaciones en el segundo experimento de IBVS, sup) λ de 0.2, inf) λ de 0.5

-Experimento 3:

En este experimento el *target* está situado en un plano inclinado respecto al horizontal, y los puntos se seleccionan para que sean lo más cercanos posible a la referencia. El experimento se realiza para IBVS y PBVS con λ 0.5, de manera que además de apreciarse el mismo comportamiento que en experimentos anteriores, también se observa en el caso de PBVS cómo se desarrolla una trayectoria de los puntos del *target* en el plano imagen con desplazamientos mucho mayores que en el caso IBVS, lo que en otros casos puede causar un fallo si el *target* abandona el campo visual.

Los vídeos correspondientes a este experimento pueden verse en los enlaces a continuación:

Caso IBVS: https://youtu.be/f2a1a_ofU8o

Caso PBVS: https://youtu.be/MjJHtboIpbg

En la figura 94, que compara la posición final del caso IBVS y el PBVS se puede observar claramente la diferencia apuntada en cuanto al desplazamiento realizado por los puntos de la imagen.

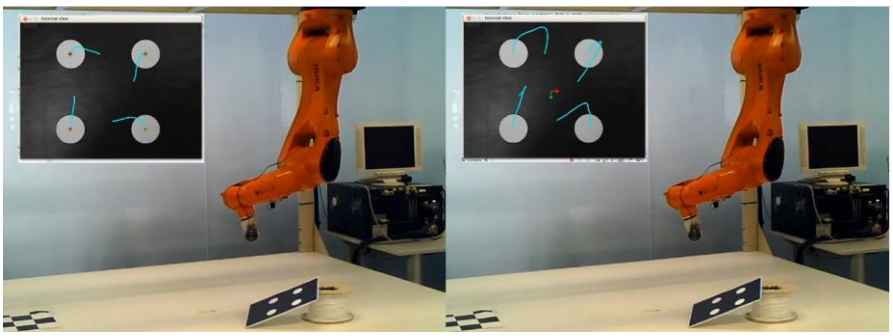


Figura 94-Posición final del tercer experimento de posicionado, izq) IBVS, der) PBVS

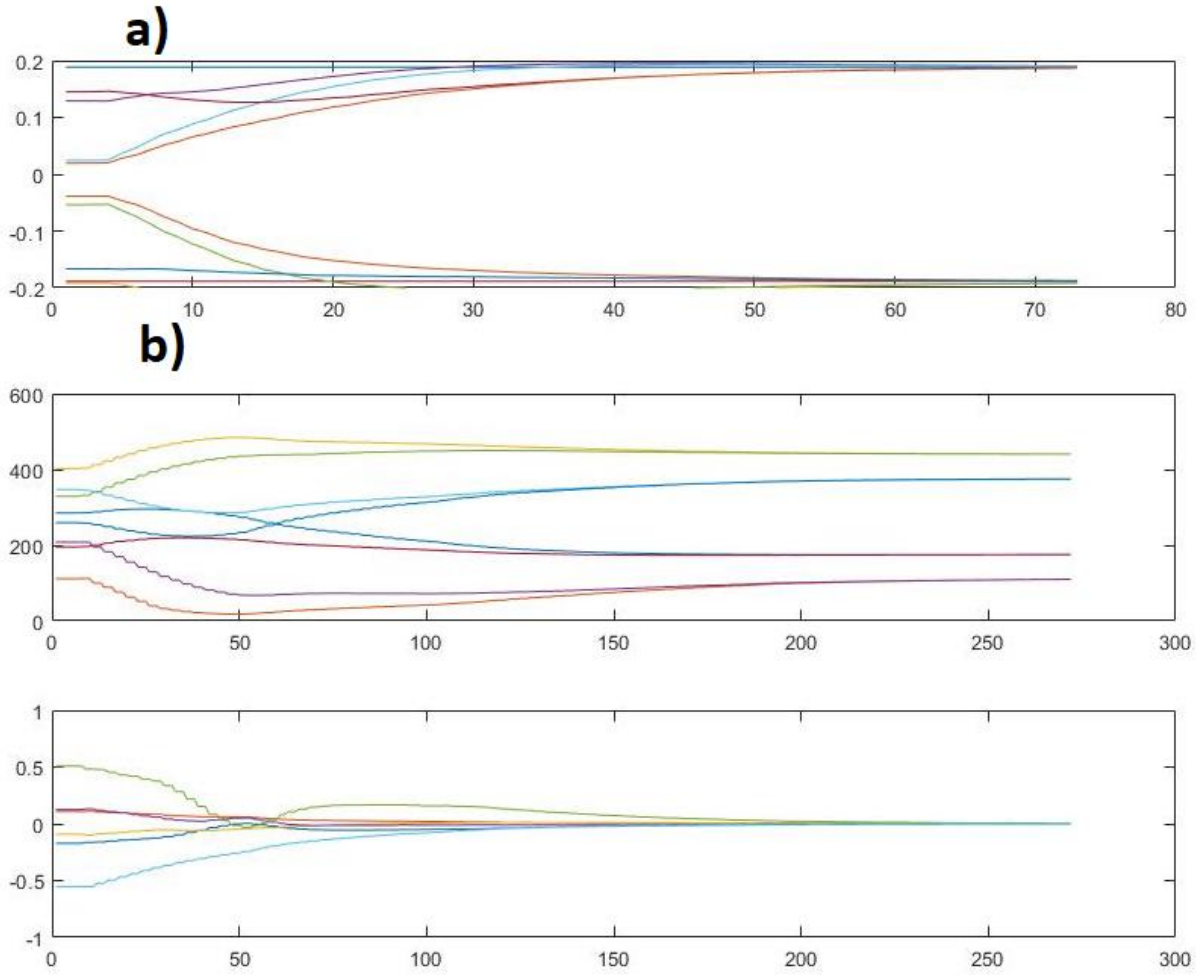


Figura 95-Evolución de variables en el tercer experimento; a) s en IBVS comparado con s, b) puntos del target sobre el plano imagen (sup) y c*Mc (inf)*

En la figura 95 puede observarse que, en efecto, ambos controles logran alcanzar la referencia, pero en el caso del control PBVS el esfuerzo para converger rápidamente se hace a nivel de la posición de la cámara respecto a la posición deseada, y ello tiene como contrapartida que los puntos sobre el plano imagen hacen un desplazamiento mayor que en el caso IBVS.

De hecho, en la figura 96 se ha seleccionado un instante del vídeo del experimento desde el punto de vista de la cámara con el control PBVS en que parte de uno de los puntos del *target* ha abandonado el campo visual, y se observa cómo el centro del punto se desplaza, porque se recalcula con el área restante dentro del campo visual.

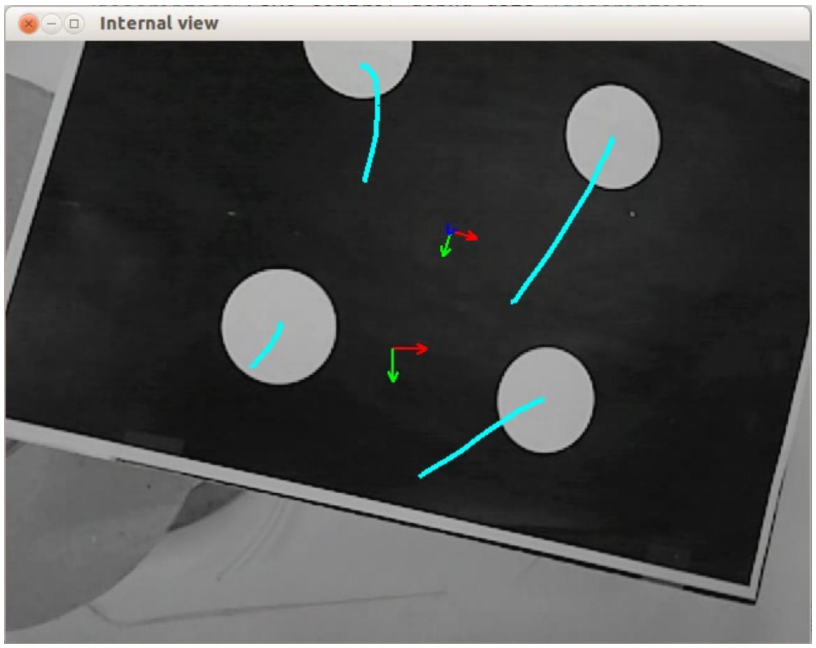


Figura 96-Punto de vista de la cámara, tercer experimento de PBVS

-Experimento 4:

En este experimento el *target* está situado en un plano inclinado respecto al horizontal, y los puntos se seleccionan para que se requiera una rotación amplia respecto a la referencia. El experimento se realiza para IBVS y PBVS con λ 0.5, con el objetivo de mostrar un fallo por abandono del campo visual de la cámara por parte del *target* en el control PBVS, mientras que este mismo problema no se da en IBVS.

Los vídeos correspondientes a este experimento pueden verse en los enlaces a continuación:

Caso IBVS: https://youtu.be/i_VQYJnhqlg

Caso PBVS: https://youtu.be/wI9sAFMzFL8

La figura 97, análoga a la figura 94 para el cuarto experimento de posicionado, muestra cómo el control IBVS ha sido capaz de resolver la tarea y el control PBVS no lo ha sido por el fallo debido al abandono del campo visual por parte de un punto del *target* como consecuencia de que no se controla el movimiento de los puntos sobre el plano imagen.

De esta manera, al combinarse movimientos demasiado amplios por parte de la cámara con una posición de referencia cercana al *target* que hace que el campo visual alrededor del mismo sea menor, uno de los puntos abandona el campo visual y el programa termina automáticamente sin haberse concluido la tarea de posicionado con éxito.

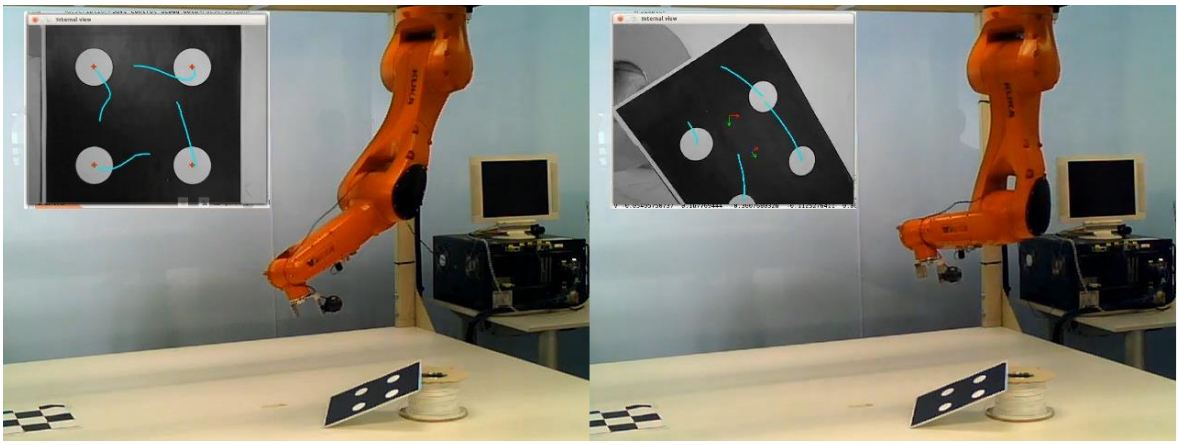


Figura 97- Análoga a la figura 94, para el cuarto experimento de posicionado

Representando unas gráficas análogas a las de la figura 95 para este experimento, se puede comprobar cómo, de nuevo, el recorrido de los puntos sobre el plano imagen en el control IBVS es corto y directo hacia la referencia, aproximándose así a una línea recta, mientras que en el control PBVS no llega a alcanzarse esta referencia porque un descenso rápido del error respecto a la posición de referencia (c*Mc) provoca que un punto abandone el campo visual y no sea posible seguir realimentando mediante la información estimada a partir de la imagen.

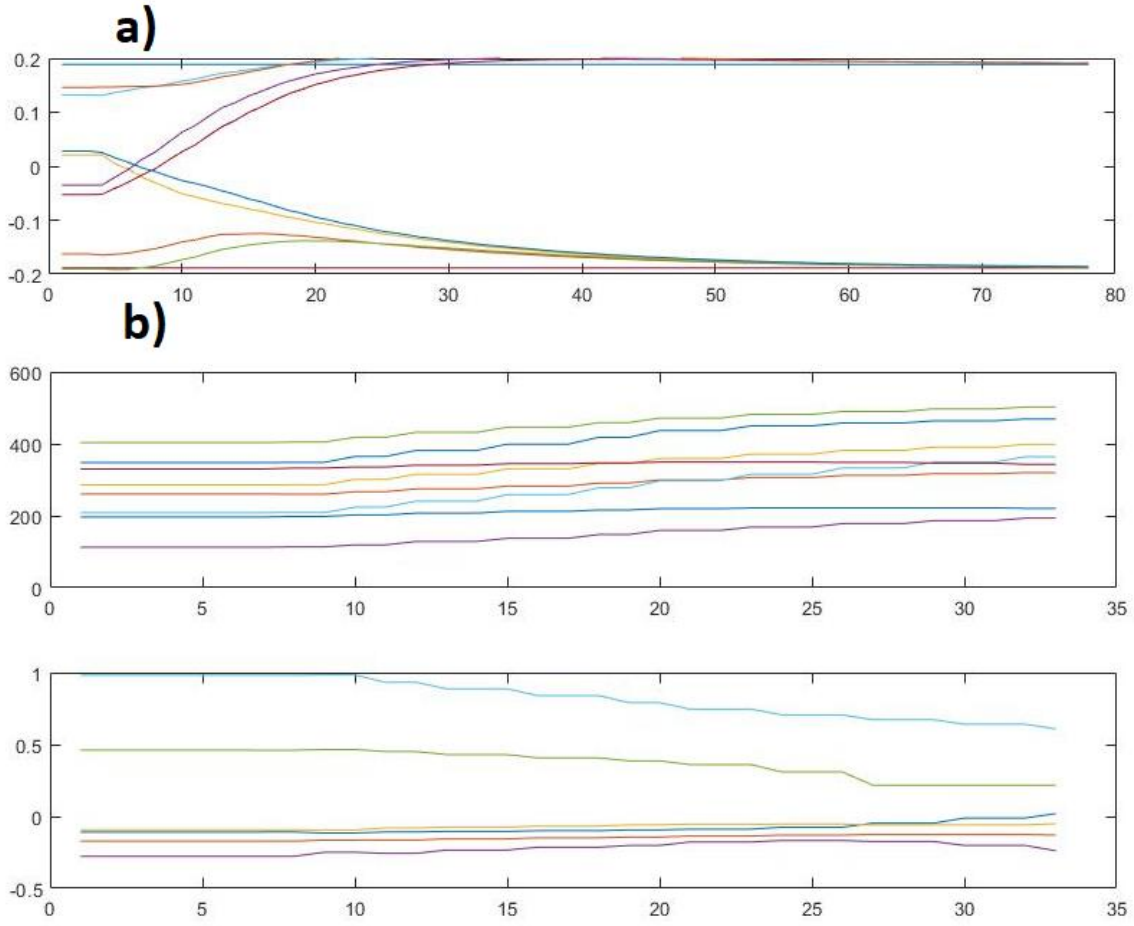


Figura 98- Análoga a la figura 95, para el cuarto experimento de posicionado

-Experimento 5:

En este experimento el *target* está situado de nuevo en el plano horizontal, con la cámara en una posición de partida que hace que la profundidad inicial sea prácticamente coincidente con la profundidad de referencia, de manera que los puntos se eligen par que se requiera un giro de 90^{o}. El experimento se realiza para IBVS y PBVS con λ 0.5, con el objetivo de mostrar un fallo por *camera-retreat* en el control IBVS, mientras que el control PBVS es capaz de solucionar la tarea.

Los vídeos correspondientes a este experimento pueden verse en los enlaces a continuación:

Caso IBVS: https://youtu.be/AJblgcesVvs

Caso PBVS: https://youtu.be/wcwdunixHks

En la figura 99, análoga a la 94 para el quinto experimento de posicionado, se observa cómo el control PBVS ha solucionado con éxito la tarea realizando los puntos del *target* trayectorias parabólicas sobre el plano imagen, mientras que el control IBVS no ha solucionado la tarea debido al fenómeno *camera-retreat* que ya ha sido discutido en apartados anteriores.

En concreto, en este caso el fenómeno de *camera retreat* ha provocado que el programa termine al alcanzarse un límite de rango de una de sus articulaciones cosa que, como se verá más adelante, es una de las medidas de seguridad existentes en el robot.

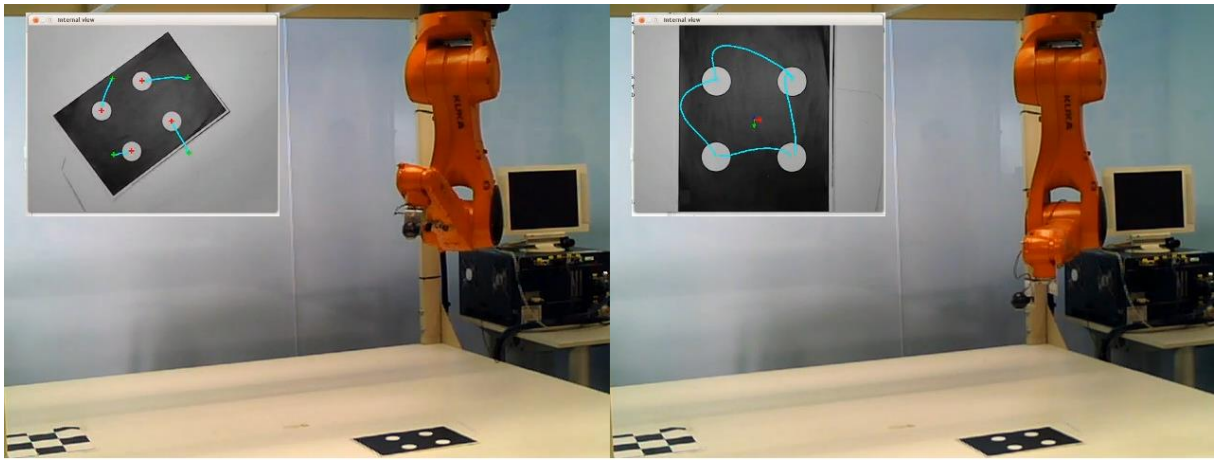


Figura 99-Análoga a la figura 94, para el quinto experimento de posicionado

Si se observan las gráficas de la figura 100, análoga a la 95, se confirma cómo en el caso IBVS los puntos del *target* parecen camino de unirse, lo que se corresponde con el retroceso de la cámara que hace que el *target* aparezca cada vez más pequeño al ser representado en la imagen. Sin embargo, en el caso PBVS, como ya se ha comentado, el movimiento de los puntos sobre el plano imagen no es controlado, lo que en otros casos puede, como el experimento 4, puede llevar a un fallo pero, en este caso, permite la convergencia.

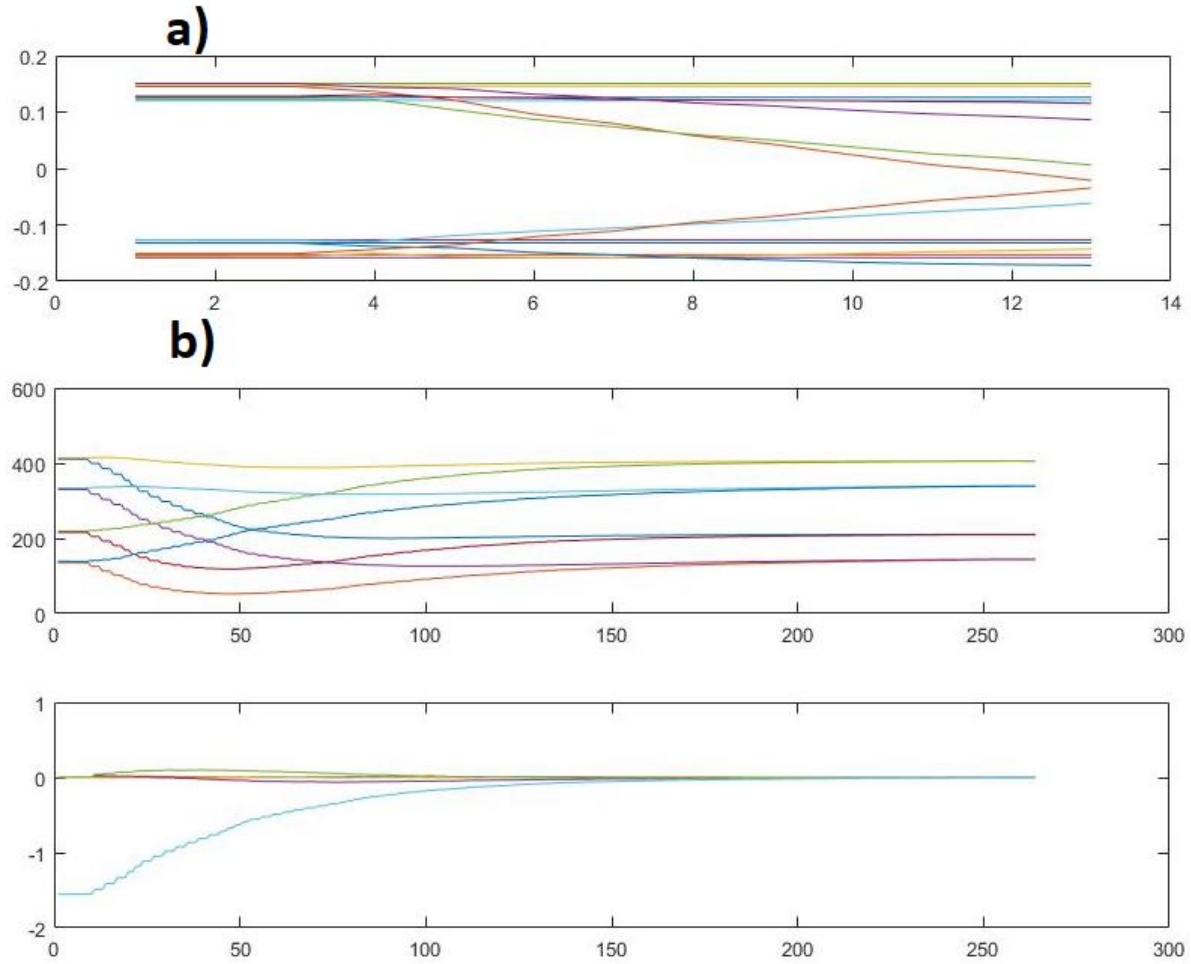


Figura 100- Análoga a la figura 95, para el quinto experimento de posicionado

4.2.1 Seguimiento mediante control IBVS:

Para la validación del control IBVS se ha incorporado la pre alimentación tal y como se ha tratado con anterioridad (expresiones {4.1.1.11} y {4.1.1.12}), si bien se ha limitado su efecto incorporando una ganancia reductora de valor constante e igual a 0.5, que, aunque empeoraría el desempeño de un control ideal con *feedforward*, en este caso evita que las vibraciones afecten a la estabilidad del control haciendo crecer y variar rápidamente la aproximación de la derivada del error de posición.

El experimento realizado ha consistido en el movimiento manual del *target* por parte del autor del presente trabajo académico mientras el robot realiza el *tracking* con control IBVS tal y como se ha descrito, de manera que se ha probado la capacidad del control de hacer que la cámara siga al objeto en distintos cambios de orientación y posición.

Para ello, ha sido necesario realizar múltiples pruebas con el fin de familiarizarse con los límites de las articulaciones del robot y la velocidad y gradualidad con la que es posible mover el *target* sin riesgo de abandonar el campo visual o provocar una acción de control demasiado brusca por parte del robot.

El resultado del experimento puede comprobarse en el vídeo enlazado a continuación:
https://youtu.be/3NsjG0zku6o

También puede observarse un extracto de dicho vídeo en la figura 101.

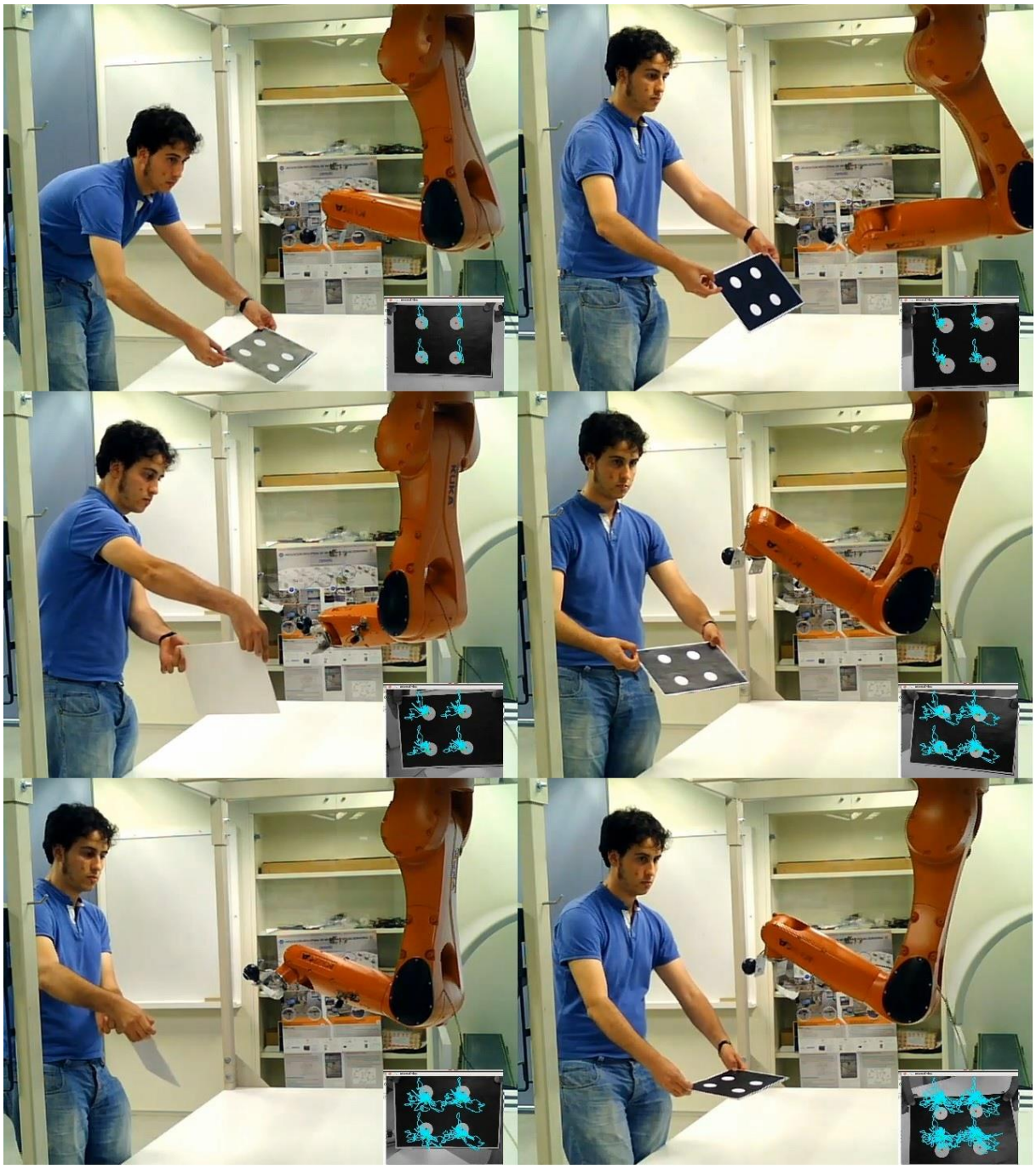


Figura 101-Secuencia del experimento de tracking IBVS

Este experimento se ha llevado a cabo de acuerdo a las siguientes medidas de seguridad:

- Por parte del *software* presente en el programa que se encarga del control desde el PC externo, las medidas ya comentadas: el programa se detiene si un punto del *target* abandona el campo visual.

- La tecnología *RSI XML Ethernet* de KUKA que permite la comunicación a tiempo real también incorpora como medida de seguridad un *Watchdog*, que se encarga de detener el robot si el controlador deja de recibir datos desde el servidor.
- El controlador detiene el robot si alcanza un límite de sus articulaciones y si se le envía un movimiento que entre dos periodos supera un cierto par máximo de seguridad.
- El *KUKA smartPAD* tiene una seta de emergencia que detiene el robot cuando se pulsa, y durante el experimento, personal cualificado del IDF ha supervisado la operación sosteniendo el *smartPAD* por si fuera necesario pulsar la seta de emergencia.

CAPÍTULO 5. CONCLUSIONES Y TRABAJOS FUTUROS

5.1 Conclusiones:

Para concluir la memoria descriptiva del presente Trabajo Fin de Máster, se puede afirmar que se han cumplido con éxito todos los objetivos planteados:

- Se ha adquirido una base sólida de conocimientos sobre el control cinemático y el servo control visual de robots, particularmente sobre sus dos principales metodologías: IBVS y PBVS. Este conocimiento no se ha limitado únicamente a la comprensión superficial de su ley de control, sino también a las especificidades del desempeño de ambas metodologías.
- Dichos conocimientos han sido probados mediante la implementación en simulación, generando una gran cantidad de documentación de las diferentes pruebas, con resultados concordantes con los comportamientos esperados según la teoría estudiada.
- Para ello, se ha desarrollado una aplicación de simulación para brazos robóticos de hasta 10 articulaciones rotativas, que permite simular casos de control con alto grado de personalización para todas las variantes de control cinemático y *Visual Servoing* tratadas en el presente trabajo académico. Como complemento a su uso, también se ha redactado un manual de usuario.
- Además, el comportamiento esperado de las metodologías de *Visual Servoing* estudiadas también ha sido validado en experimentación real utilizando para ello un robot industrial, lo que ha permitido familiarizarse con algunas de las particularidades de la implementación real.

5.2 Trabajos futuros:

El cumplimiento de los objetivos del presente trabajo académico trae consigo dos vías principales para el desarrollo de proyectos futuros:

- Los conocimientos adquiridos en *Visual Servoing* suponen una base sólida desde la cual es posible partir para profundizar en este campo del control de robots que tiene gran número de aplicaciones y también gran potencial.
- La aplicación de simulación desarrollada es la otra vía que es posible explotar en futuros proyectos:
    - Por un lado, es un medio adaptable y sencillo de utilizar para fines educativos, puesto que puede consultarse la manera en que se implementan las leyes de control de diversas metodologías y probar su comportamiento ante distintos casos de simulación.
    - Por otro lado, al ser una aplicación de código libre y abierto, es posible su futura edición para incluir más funcionalidades, ya sea por la vía de adquirir nuevos conocimientos simulando nuevas metodologías de control, por la vía de realizar simulaciones más realistas, que incorporen comportamientos dinámicos, o por la vía de mejorar su adaptabilidad y grado de personalización.

CAPÍTULO 6. BIBLIOGRAFÍA

[1] ISO 8373:2012 (definiciones de robótica):

https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en

[2] Bruno Siciliano, Oussama Khatib y otros, 2016. Springer Handbook of Robotics. Victor Scheinman, J.Michael McCarthy y Jae-Bok Song. Capítulo 4: Mechanism and Actuation. Serial robots.

[3] International Federation of Robotics, informe World Robotics (2016), capítulo 1.2:

https://ifr.org/img/office/Industrial_Robots_2016_Chapter_1_2.pdf

[4] Antonio Barrientos, Luis Felipe Peñín y otros, 2007. Fundamentos de Robótica. Capítulo 6: Control cinemático.

[5] Antonio Barrientos, Luis Felipe Peñín y otros, 2007. Fundamentos de Robótica. Capítulo 4: Cinemática del robot.

[6] Antonio Barrientos, Luis Felipe Peñín y otros, 2007. Fundamentos de Robótica. Capítulo 4: Cinemática del robot. Punto 3: Modelo diferencial. Matriz Jacobiana. Sub apartado 2: Jacobiana geométrica.

[7] María Dolores Soto Torres, 1986. La inversa de Penrose.

https://dialnet.unirioja.es/servlet/articulo?codigo=785490

[8] Bruno Siciliano, Oussama Khatib y otros, 2016. Springer Handbook of Robotics. François Chaumette, Seth Hutchinson y Peter Corke. Capítulo 34: Visual Servoing.

[9] Seth Hutchinson, Peter Corke y Gregory D. Hager, 1996. A Tutorial on Visual Servo Control.

http://masters.donntu.org/2012/etf/nikitin/library/article10.htm

[10] Documentación de OpenCV. Real Time pose estimation of a textured object.

https://docs.opencv.org/3.1.0/dc/d2c/tutorial_real_time_pose.html

[11] Skew-symmetric matrix. Encyclopedia of Mathematics.

http://www.encyclopediaofmath.org/index.php?title=Skew-symmetric_matrix&oldid=39104

[12] Antonio José Sánchez Salmerón, material docente de la asignatura Visión Artificial del MUII-UPV. Tema 6.2, modelo de cámara.

[13] Pau Muñoz Benavent, 2017. Robot Visual Servoing Using Discontinuous Control. Sub aparatado 2.1.5.3: PBVS-IBVS Comparison.

[14] Andrés Jaramillo Botero, 2005. Descripciones y transformaciones espaciales.

http://www.wag.caltech.edu/home/ajaramil/libro_robotica/transformaciones_espaciales.pdf

[15] John Craig, 2006. Robótica. Capítulo 2: Descripciones espaciales y transformaciones.

http://www.wag.caltech.edu/home/ajaramil/libro_robotica/transformaciones_espaciales.pdf

[16] Bruno Siciliano, Oussama Khatib y otros, 2016. Springer Handbook of Robotics. Victor Scheinman, J.Michael McCarthy y Jae-Bok Song. Capítulo 4: Mechanism and Actuation. Joint Mechanisms.

[17] Google Groups, Robotics & Machine Vision Toolboxes. Hilos donde el creador de *rvctools*, Peter Corke, trata los problemas de la simulación de articulaciones prismáticas mediante Matlab:

https://groups.google.com/forum/#!searchin/robotics-tool-box/prismatic$20joint%7Csort:date/robotics-tool-box/sEMqJnC15zo/zY1QMoGpBQAJ

[18] Buscar en google imágenes "roll pitch yaw xyz":

https://www.google.es/search?q=roll+pitch+yaw+xyz&source=lnms&tbm=isch&sa=X&ved=0ahUKEwi9--rIoIDcAhWHtBQKHS55DagQ_AUICigB&biw=1920&bih=947#imgrc=QvI_O5T9VVO9ZM:

[19] Ficha técnica de los controladores KUKA KRC4:

https://www.kuka.com/-/media/kuka-downloads/imported/9cb8e311bfd744b4b0eab25ca883f6d3/kuka_pb_controllers_en.pdf

[20] Ficha técnica de la tecnología de comunicación KUKA ETHERNET RSI XML, página 9:

http://vip.gatech.edu/wiki/images/f/f2/RSI_XML.pdf

[21] Enciclopedia virtual (Wiki) de ROS. Introducción.

http://wiki.ros.org/ROS/Introduction

[22] Página de introducción de VISP:

http://visp-doc.inria.fr/doxygen/visp-daily/index.html

[23] Página de presentación de OROCOS toolchain:

http://www.orocos.org/toolchain

[24] Siemens *Industry Online Support*. "¿Qué propiedades, ventajas y particularidades ofrece el protocolo UDP?"

https://support.industry.siemens.com/cs/document/26484229/%C2%BFqu%C3%A9-propiedades-ventajas-y-particularidades-ofrece-el-protocolo-udp-?dti=0&lc=es-WW

PRESUPUESTO

1 Justificación del presupuesto:

La elaboración del presupuesto correspondiente al proyecto completado en el presente Trabajo Fin de Máster se ha realizado en base a los criterios para la elaboración de presupuestos en actividades de I+D+i del Centro de Apoyo a la Innovación, la Investigación y la Transferencia de Tecnología (CTT), debido a la vinculación de la materia tratada en el proyecto con la investigación, desarrollo e innovación.

2 Estudio económico:

2.1 Costes de personal:

El coste de personal se calcula según:

$$CP = Coste\ horario \cdot Horas\ de\ trabajo$$

De acuerdo a las tablas retributivas de la UPV, el sueldo mensual de un trabajador a jornada completa del PAS (Personal de Administración y Servicios) del grupo A, subgrupo A2, que incluye a la escala intermedia de ingeniería técnica, es de 28,42 €/h.

Además, de acuerdo a las bases y tipos de cotización del Régimen General de la Seguridad Social para 2018, para la categoría profesional correspondiente a Ingenieros técnicos, peritos y ayudantes titulados (grupo de cotización 2) el tipo de cotización para contingencias comunes es del 23,60%. De la misma referencia, con un tipo de cotización por concepto del desempleo en un contrato de duración determinada y tiempo parcial del 6.7%. También del Régimen General de la Seguridad Social, el tipo de cotización al FOGASA es de 0,2% y el tipo por concepto de Formación Profesional es de 0,6%. Todo ello supone un coste del 31,1% del salario, lo que implica añadir 8,84 €/h al coste.

Por último, de acuerdo con la información proporcionada por el portal web de la Seguridad Social, la tarifa para la cotización por accidentes de trabajo y enfermedades profesionales para actividades de investigación y desarrollo es del 1%, es decir, 0,28 €/h.

Tomando estas cifras como referencia, el coste horario total es de 37,54 €. De esta manera, teniendo en cuenta las horas trabajadas:

Tarea	Horas	Coste (€/h)	Coste (€)
Documentación y estudio	70	37,54	2627,8
Desarrollo del simulador	110		4129,4
Diseño y ejecución de los casos de simulación	12		450,48

Análisis y documentación de los resultados de las pruebas de simulación	8		300,32
Validación experimental	20		750,8
Redacción de documentos	80		3003,2
TOTAL	**300**		**11262**

Lo que también puede desglosarse teniendo en cuenta los porcentajes que corresponden a los honorarios, las cotizaciones a la Seguridad Social y por Accidente de Trabajo y Enfermedad Profesional:

Concepto	Coste (€)
Honorarios	8526
SS (31,1%)	2652
AT y EP (1%)	84
TOTAL	**11262**

2.2 Coste del material inventariable:

El coste de amortización puede calcularse como:

$$CMI = Precio\ material \cdot \frac{Meses\ uso}{12\ \left(^{meses}/_{año}\right) \cdot Periodo\ de\ amortización\ (años)}$$

Concepto	Precio (€)	Meses uso	Periodo de amortización	Uds	Coste (€)
Ordenador portátil	1200	6	4	1	150
Ordenador de sobremesa	600	0,25	4	1	3,12
KUKA Agilus KR6 900 sixx	20000	0,25	6	1	69,44

con controlador KRC4					
Cámara Logitech C300	20	0,25	1	1	0,42
Matlab (licencia académica)	0	-	-	-	0
TOTAL	-	-	-	-	**222,98**

2.3 Material fungible:

Concepto	Coste (€)
Folios (500 uds)	3
Encuadernación	15
TOTAL	**18**

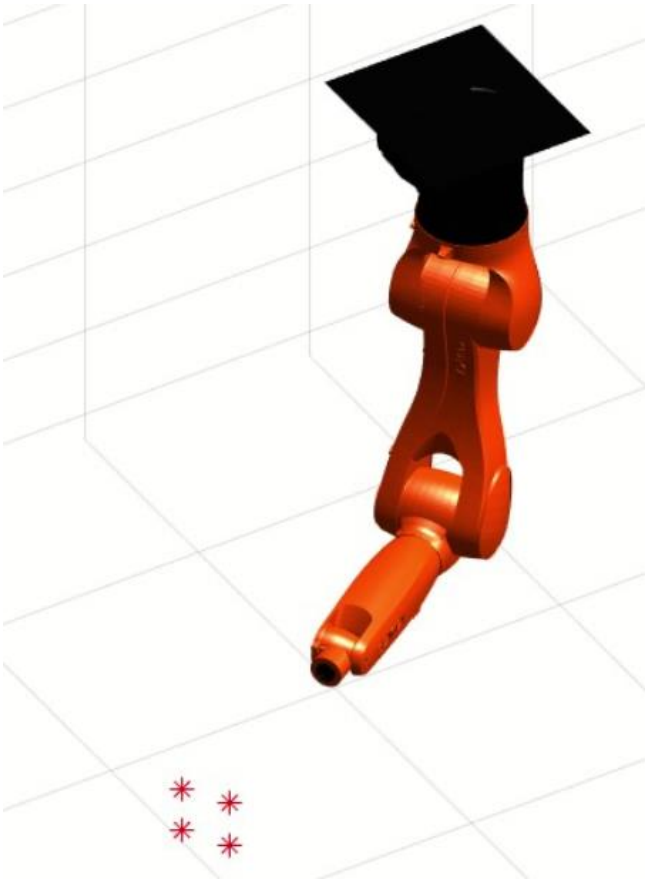
3 Resumen del presupuesto:

Concepto	Coste (€)
Mano de obra	11262
Material inventariable	222,98
Material fungible	18
Subtotal	**11502,98**
IVA (21%)	2415,63
TOTAL	**13918,61**

ANEJO: MANUAL DEL SIMULADOR

Robotic Arm Control Simulator

Manual de usuario



Qué es *Robotic Arm Control Simulator*:

Robotic Arm Control Simulator (RACS) es un simulador de código abierto y libre desarrollado en *Matlab*.

La finalidad de RACS es simular el control de brazos robóticos con articulaciones de revolución mediante diferentes metodologías.

Qué metodologías de control incluye:

Las metodologías de control incluidas en RACS son:

-Control manual: Permite indicar la posición angular de cada articulación disponible.

-Trayectoria articular: Permite indicar una secuencia de posiciones angulares para cada una de sus articulaciones. El tiempo total de ejecución de la secuencia también es personalizable.

-Control de velocidad: Permite el posicionado o seguimiento de una referencia fija o móvil personalizada para el *end-effector* del robot, utilizando un control de la velocidad de las articulaciones. La ganancia proporcional del control también es personalizable.

-Control de aceleración: Permite resolver las mismas tareas que el control de velocidad, utilizando para ello el control de la aceleración de las articulaciones del robot. También son personalizables las ganancias proporcionales del error de posición y del de velocidad.

-*Image Based Visual Servoing (IBVS)*, *eye-in-hand*: Permite la simulación de la metodología de servocontrol visual basada en imágenes (IBVS) con cámara en el *end-effector* del robot para posicionado y seguimiento ante referencias fijas y móviles, de prueba (ya implementadas) y personalizables. También son personalizables los parámetros de control: tiempo total de simulación, tiempo de muestreo y ganancia proporcional del control.

-*Position Based Visual Servoing (PBVS), eye-in-hand*: Permite la simulación de la metodología de servocontrol visual basada en la posición (PBVS) con cámara en el *end-effector* de forma análoga al caso IBVS.

Además, dado que se trata de un simulador de código libre y abierto, pueden ser incluidas nuevas metodologías de control editando el código fuente y la propia interfaz gráfica mediante *Matlab*.

Cómo realizar una simulación:

Todas las metodologías de control indicadas están incluidas en una misma interfaz gráfica, así que, para realizar una simulación de cualquiera de ellas, hay que seguir una serie de pasos comunes que explicaremos a continuación.

1-Incluir el robot en la librería:

Para simular gráficamente el robot, RACS utiliza modelos *stl*, siendo los modelos incluidos por defecto procedentes de la librería *ARTE* (*A Robotics Toolbox for Education)* y algunas funciones de la librería *rvctools* (*Robotics, Vision and Control Toolbox*, de Peter Corke), así que es necesario que *rvctools* esté accesible durante la ejecución de la simulación (añadidas al *Path* en *Matlab*) y que las carpetas que

contienen los modelos *stl* de los eslabones del robot estén a su vez en una carpeta de nombre "robots" que comparta espacio con un archivo de código *Matlab* (*.m*) vacío de nombre "*arte.m*", de acuerdo a la documentación de *rvctools* para la simulación tridimensional de robots.

Además, es necesario crear un archivo *mat* que contenga los parámetros para construir el robot simulado:

-Una matriz llamada "P" con los parámetros de la matriz Denavit-Hartenberg correspondientes al robot en cada columna. La primera columna corresponde al parámetro *d*, la segunda al parámetro *a*, la tercera al parámtero *α*, la cuarta al *offset* de la posición angular de cada articulación (en radianes), y las columnas quinta y sexta a los límites inferior y superior (en grados) de la posición angular de cada articulación. A continuación, se muestra un ejemplo de matriz P para un robot con 6 grados de libertad:

```
P =

   -0.4000    0.0250    1.5708         0 -170.0000  170.0000
         0   -0.4550         0    3.1416 -190.0000   45.0000
         0   -0.0350   -1.5708   -1.5708 -120.0000  156.0000
   -0.4200         0    1.5708         0 -190.0000  190.0000
         0         0   -1.5708         0 -120.0000  120.0000
   -0.0800         0    3.1416         0 -358.0000  358.0000
```

-Una matriz llamada "base" que indique la posición y orientación de la base del robot mediante el formato de matriz de transformación homogénea.

-Un vector llamado "q_init" que indique la posición angular inicial (en radianes) de cada articulación.

-Un vector de células (*cell array*) llamado "color" que indiquen el color de cada segmento del robot. A continuación, se muestra un ejemplo de la estructura "color" para un robot de 6 grados de libertad:

```
color =

  1×6 cell array

    {'black'}    {'OrangeRed'}    {'OrangeRed'}    {'OrangeRed'}    {'OrangeRed'}    {'OrangeRed'}
```

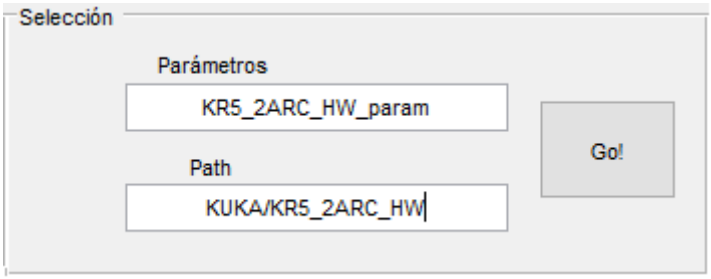
-Una variable de texto llamada "name" que contenga el nombre del robot.

Actualmente existe ya una librería con los parámetros de 8 robots de KUKA que cuentan con sus correspondientes carpetas de la librería *ARTE* con los archivos *stl* necesarios para la simulación. Si se utilizan los *stl* de ARTE, tan solo es necesario crear el archivo *mat* con los parámetros tal y como se ha indicado para proceder a simular.

2-Cargar el robot en la interfaz gráfica:

Para cargar el robot en la interfaz gráfica, basta con especificar el nombre del archivo *mat* con los parámetros y el *path* donde se encuentran los archivos *stl* en las casillas correspondientes del panel del simulador.

En el caso de los parámetros se indica el nombre del archivo sin la extensión. En el caso del *path*, se indica con el formato *Nombre_Carpeta_Marca/Nombre_Carpeta_Modelo*. A continuación, se muestra un ejemplo:



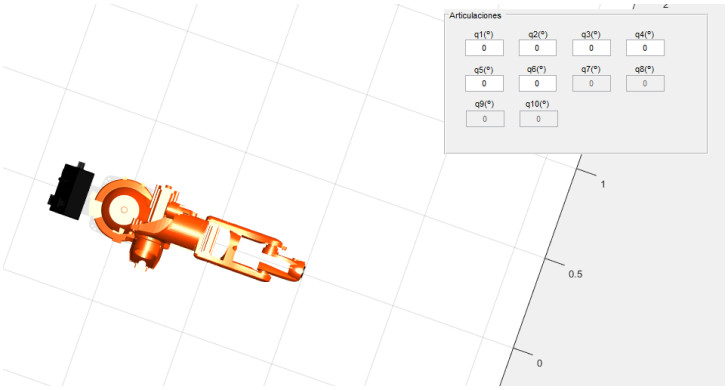
Para que el robot se cargue, una vez rellenados los campos indicados, hay que hacer clic en el botón "*Go*!".

En caso de haberlo realizado correctamente, veremos el robot en la posición deseada respecto de un sistema de coordenadas global.

Este mismo método puede utilizarse consecutivamente sin necesidad de reiniciar la interfaz gráfica, tan solo introduciendo nuevos datos en las casillas de parámetros y *path* y cargando el nuevo robot.

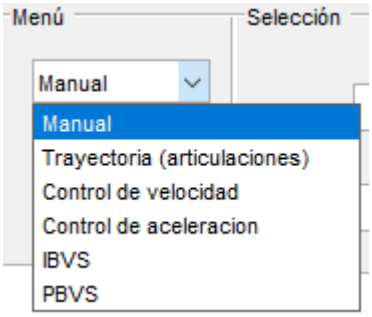
Es importante señalar que, a la vez que se carga el robot, se posiciona la cámara simulada que se utilizará para los cálculos en los controles tipo *Visual Servoing* con un eje Z siempre coincidente con la profundidad de la cámara. Esto es completamente automático y el único requisito es que, al definir los parámetros, el robot tenga el eje longitudinal de su *end-effector* contenido en el plano XZ cuando todos sus ángulos son 0.

Este requisito es sencillo de comprobar visualmente, como se muestra en el ejemplo a continuación, comprobando que cuando todos los ángulos son 0, el brazo está alineado con el eje X:



3-Seleccionar la metodología y realizar la simulación:

Las distintas metodologías son accesibles mediante un control desplegable situado junto al panel para cargar el robot:



Seleccionar una mostrará por pantalla el panel de control de dicha metodología y ocultará el resto. En cada panel de control se rellenará una serie de campos obligatorios y la simulación se iniciará mediante un botón "*Go!*", excepto en el control manual, que es instantáneo.

Las metodologías pueden utilizarse secuencialmente y en cualquier orden. Aquellas que dependen de la posición de partida están debidamente interconectadas para respetar dicha posición de partida.

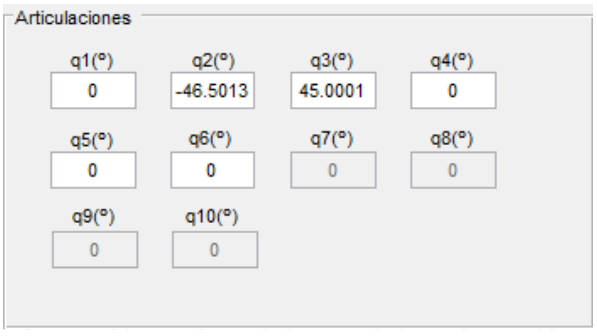
Una simulación no puede tener lugar a la vez que otra, ni puede interrumpir a otra. Para detener una simulación en curso por cualquiera de las metodologías que se accionan mediante un botón "*Go!*" es necesario reiniciar el robot volviendo a pulsar el botón "*Go!*" del panel de carga del robot.

Cómo utilizar cada metodología de control:

Aunque hemos tratado el funcionamiento general del simulador, cada metodología de control tiene sus especificidades que conviene comentar.

1-Control manual:

El control manual incluye un sencillo panel de control con una casilla de texto editable para cada una de las articulaciones. Las casillas correspondientes a grados de libertad no disponibles en el robot que se ha cargado son automáticamente inhabilitadas. A continuación, un ejemplo del panel con un robot de 6 grados de libertad:



El control manual permite manipular la configuración del robot de una forma cómoda, actualizando la posición de manera instantánea cada vez que se introduce una nueva posición angular en la casilla de cada articulación. La posición angular debe indicarse en grados.

Su objetivo también es explorar los límites de las posibles configuraciones del robot, de modo que es la única metodología de las incluidas en el simulador que incorpora la saturación cuando se alcanzan estos límites.

Puede desactivarse la saturación de las articulaciones accediendo al código del simulador y comentando o eliminando las siguientes líneas en cada función de *callback* de las casillas de texto editable (*edit1_callback, edit2_callback…*).

Por último, cabe añadir que las casillas activas de cada articulación se actualizan cada vez que se accede a la metodología manual tras ejecutar una simulación mediante otra metodología, de modo que siempre indican la posición en la que se encuentra el robot.

2-Trayectoria articular:

Esta metodología es la que tiene un panel más sencillo. Simplemente hay que introducir en la casilla de texto editable un archivo *mat* que contenga:

-Una matriz llamada "M" donde en cada fila tenga la posición angular (en radianes) de cada articulación en cada instante. De este modo, las columnas representan articulaciones (en orden creciente: q1, q2, q3…) y cada fila un instante.

-Una variable llamada "T" que indique el tiempo total de simulación.

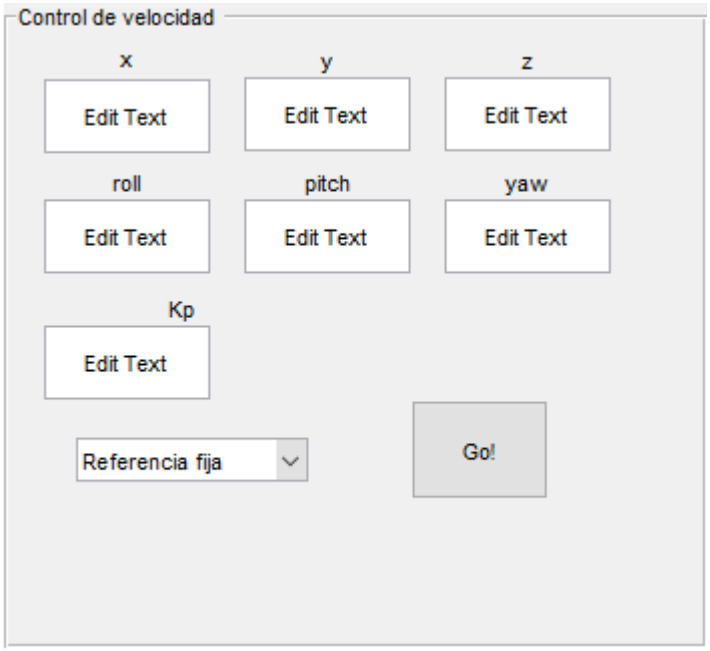
El periodo transcurrido entre cada instante será calculado automáticamente por el programa a partir del número de instantes (filas de M) y del tiempo total de simulación deseado. La simulación comenzará al hacer clic en el botón "*Go!*".

3-Control de velocidad:

El panel del control de velocidad permite indicar las coordenadas de posición (metros) y de orientación (roll-pitch-yaw, en radianes) del *end-effector* respecto del sistema de coordenadas global.

Por un lado, para referencia fija, están las casillas de texto editable donde introduciremos la referencia y la ganancia proporcional deseados. Por otro, las etiquetas de texto no editable indican, de manera actualizada, las coordenadas del *end-effector*.

La simulación comenzará al pulsar el botón "*Go!*".



La simulación se realiza con un periodo de muestreo por defecto de 0.1 s, que es cercano al tiempo que consume cada iteración del bucle de control y simulación gráfica (aproximadamente entre 0.1 y 0.12 s), de manera que la simulación es aproximadamente a tiempo real. Para alterar el tiempo de muestreo, es necesario editar el código del programa en la función de *callback* del pulsador "*Go!*".

En caso de que la referencia deseada no sea alcanzable, el simulador mostrará lo que ocurre al tratar de alcanzarla, pero el programa no se bloqueará, ya que el bucle finalizará al transcurrir un número de iteraciones correspondiente a 4 veces el tiempo de establecimiento deseado. Cuando esto ocurra, el robot se reiniciará no a su posición de partida sino a la posición en la que fue cargado inicialmente.

Por otro lado, para referencia es móvil, en el panel tan solo se indica la ganancia proporcional y el archivo *.mat* que contiene la trayectoria y el periodo de muestreo deseado para realizarla con la siguiente forma:

-Una matriz llamada "p_ref" que contenga la trayectoria de referencia del *end effector* en posición y orientación, como en el caso de referencia fija, de tal manera que cada fila de la matriz se corresponda con un instante y cada columna con una coordenada (x, y, z, roll, pitch, yaw).

-Una variable numérica llamada "ts" que indique el periodo de muestreo deseado en segundos.

Cabe señalar que el control de velocidad para una referencia fija incorpora además del control de posición, la pre alimentación de la velocidad con la que se mueve la referencia, y no es necesario que el punto de partida de la trayectoria de referencia coincida con el punto de partida del *end-effector* del robot, ya que el control tratará de alcanzar la referencia *online*.

Tanto en el caso de referencia fija como de referencia móvil, la referencia aparece señalada como un punto rojo en el espacio de simulación:

4-Control de aceleración:

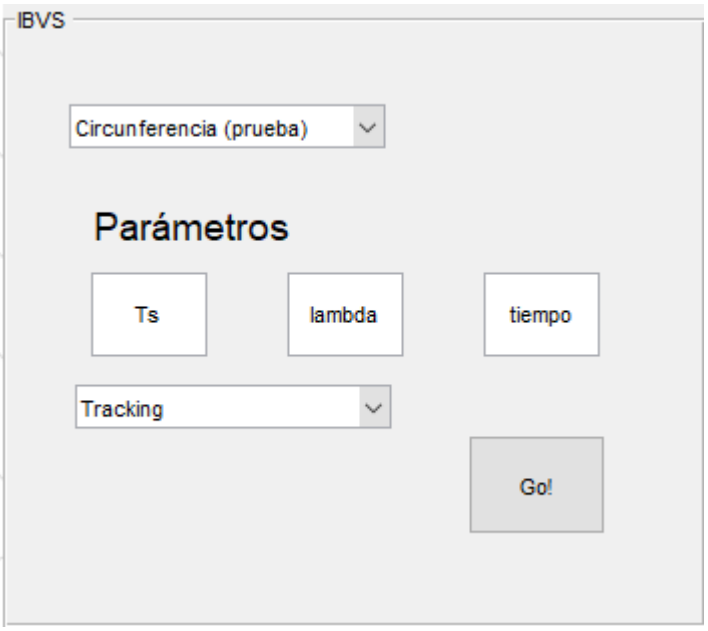
El control de aceleración se utiliza de manera esencialmente idéntica al de velocidad, con la salvedad de que, en el panel de control, tanto para referencia fija como para referencia móvil, se ha de indicar la ganancia que se encarga de corregir el error de posición con respecto a la referencia, como ocurría en el de velocidad, pero también la ganancia correspondiente al error de velocidad respecto a la referencia.

5-IBVS:

Las metodologías de *visual servoing* tienen los paneles más complejos y ofrecen múltiples opciones.

La cámara simulada no aparece representada gráficamente, y se encuentra siempre alineada con el *end-effector* del robot, estando el plano de imagen de la cámara 1 cm por delante del *end-effector*, con el eje Z coincidiendo con su profundidad. Para alterar el *offset* del plano de la cámara respecto al *end-effector* hay que editar este valor en el código del programa, concretamente en la función en que se carga el robot. Para más detalles, consultar el apartado de cómo realizar una simulación de este manual.

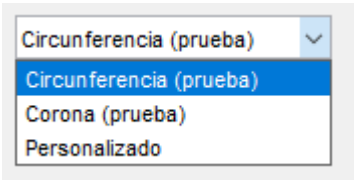
Por defecto, aparece el siguiente panel en la metodología IBVS:



Como en el resto de metodologías de control, la simulación comenzará al pulsar el botón "*Go!*" una vez seleccionadas las distintas opciones.

5.1-El objeto y su trayectoria:

En el menú desplegable superior, se selecciona el objeto que se va a seguir o posicionar. Seleccionaremos si se desea uno de los dos modos de prueba (circunferencia y corona), que cargan automáticamente un objeto en movimiento que orbita alrededor del robot actual, o un modo personalizado para realizar simulaciones de casos concretos:



-La opción "circunferencia (prueba)" carga un objeto formado por 4 puntos separados 10 cm entre sí y situados sobre el plano XY, entorno al punto (0,0,0), y carga también la trayectoria que describirá dicho objeto. Esta trayectoria comenzará con el centro del objeto a la altura (posición Z del sistema de referencia global) del *end-effector* del robot, sobre el plano YZ y con un *offset* en X de 0.6 m respecto a la cámara del robot. Este *offset* pretende conseguir que haya una cierta profundidad entre la cámara y el objeto, de modo que es recomendable que la prueba parta con un robot alineado con el eje X.

La trayectoria será una circunferencia alrededor del robot y con posición constante en Z global.

-La opción "corona (prueba)" funciona de manera análoga a la opción anterior, pero con una trayectoria que sube y baja en Z.

-La opción "Personalizado" permite cargar un objeto y trayectoria propios, mostrando por pantalla los siguientes campos a rellenar:

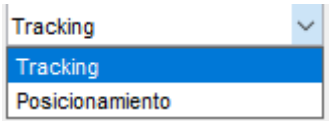
En las casillas Trayectoria y Objeto deben indicarse los nombres de archivos *mat* que contengan:

-En el caso de la trayectoria, una estructura de matrices de transformación homogénea llamada “wMp” que indique la traslación y rotación del objeto en coordenadas globales respecto a su posición de partida.

-En el caso del objeto, una matriz llamada “P” que contenga en cada fila las coordenadas globales XYZ iniciales de cada uno de los cuatro puntos que deben formar el objeto.

5.2-Posicionado y *tracking*:

En el menú desplegable inferior podemos elegir entre dos opciones: posicionado y *tracking*.

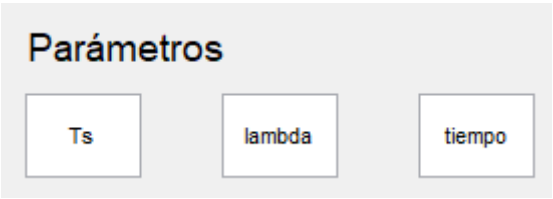


El posicionado tan solo aplica un control proporcional a partir de la medida del error a cada instante, de modo que es apropiado para tareas de posicionado de objetos fijos, pero tiene un error de posición a la hora de hacer seguimiento de objetos móviles.

El tracking incorpora la derivada del error, estimándola a partir de los datos pasados del error de posición. Es especialmente recomendables para el seguimiento de objetos de movimiento sin error de posición.

5.3-Parámetros de la simulación:

Los parámetros restantes se introducen en las respectivas casillas de texto editable:

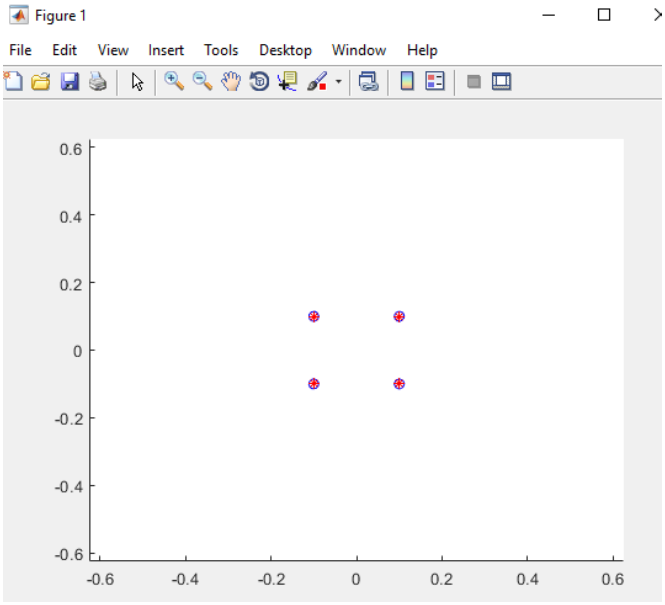


De derecha a izquierda: tiempo de muestreo (en segundos), ganancia proporcional del control y tiempo total de simulación (en segundos). Para que la simulación sea aproximadamente en tiempo real, el periodo de muestreo debe ser de al menos 0.1 segundos.

Para modificar los parámetros intrínsecos de la cámara simulada hay que editar el código del programa, concretamente en la función correspondiente al pulsador de activación de la simulación.

4.4-Plano imagen de la cámara:

Automáticamente cuando arranca la simulación, aparece una figura donde se muestra de forma actualizada el objeto proyectado sobre el plano imagen (asteriscos rojos) y la referencia que indica cómo deberían verse en caso de error de posición 0:



Aunque la ventana no se cierra automáticamente al terminar la simulación, no interfiere con las siguientes simulaciones, sino que éstas crean una nueva figura.

6-PBVS:

A efectos del usuario, los pasos necesarios para ejecutar una simulación PBVS y las opciones disponibles son exactamente iguales que en el caso IBVS. Por ello, se refiere al apartado anterior para resolver posibles dudas.