Escuela Politécnica Superior

Control visual para el guiado de marcha de un robot humanoide

Grado en Ingeniería Robótica

Trabajo Fin de Grado

Autor:
Iñaki Saez Fuentes

Tutor:
Gabriel Jesús García Gómez

Junio 2019

Universitat d'Alacant
Universidad de Alicante

UNIVERSIDAD DE ALICANTE

Grado en Ingeniería Robótica

Trabajo Fin de Grado

Control visual para el guiado de marcha de un robot humanoide

Autor

Iñaki Saez Fuentes

Tutor

Gabriel Jesús García Gómez

Alicante, Junio 2019

Resumen

En este Trabajo Fin de Grado se realiza una investigación sobre los distintos métodos de control de guiado de marcha para robots humanoides que podemos encontrar en la actualidad, realizando para ello una búsqueda y comparación de muchos de los métodos de control para la tarea en cuestión que podemos encontrar a lo largo del estado del arte de la técnica, y comparando todos los métodos para comprobar las diferencias que podemos encontrar en estos. Tras ello, se describe el funcionamiento de algunas implementaciones propias basadas en visión artificial del control en cuestión, como son los desarrollos de ver y avanzar y de control visual basado en imagen implementados en el robot **NAO** de la empresa *Softbank Robotics*, viendo así de forma real y efectiva cómo varía el funcionamiento de algunos de los métodos planteados en un entorno real, y pudiendo así comparar los datos que podemos ir tomando de nuestra máquina para ver el avance de las leyes de control que se proponen a lo largo de la ejecución de estos controladores.

Palabras clave: Control, Visión, Marcha, NAO

Motivación, justificación y objetivo general

La motivación de este trabajo viene dada por un fuerte interés sobre los dos campos que vamos a tratar a lo largo de toda la memoria en todo momento, los cuales son las leyes de control y la visión artificial, ambos aplicados siempre en el contexto de la robótica. Podemos ver que cada uno de ellos tiene una elevada importancia en la robótica actual, siendo el primero la base de cualquier movimiento dentro de un robot, al realizar los cálculos necesarios de cada una de las articulaciones o mecanismos que queramos mover, y el segundo siendo un campo muy innovador y mediante el cual se han podido alcanzar grandes avances a la hora de realizar nuevos métodos de percepción para las máquinas con las que trabajamos. Creo firmemente que la combinación de ambos campos nos da una visión muy general sobre la robótica en la actualidad, y cómo ésta se puede entender como un conjunto de muchos campos teóricos distintos que dan pie a algo mayor, como puede ser la programación de un controlador visual de un robot en este caso.

Partiendo de la base planteada, vemos justificado la realización de un proyecto de este tipo al entenderlo como un repaso de los distintos métodos de control visual en robótica que tenemos en la actualidad, y realizando más tarde controladores para un robot real basándonos en todos los que repasemos en el estado del arte, viendo cómo podemos ir implementando éstos en un entorno de programación real que nos plantee algunos retos a superar, además de entendiendo los distintos interfaces con los que tendremos que trabajar a lo largo de nuestra implementación.

Conociendo la motivación y justificación de este desarrollo que vamos a realizar, podemos ver que el objetivo general que nos marcaremos en primer lugar será el de lograr comprender de la mejor forma posible el funcionamiento de los controladores visuales en robots humanoides, tratando para ello de repasar el estado del arte de los sistemas en cuestión, y más tarde tratando de implementar estos en un entorno realista que nos sea cómodo y mediante el cual podamos obtener el mayor número de datos que nos sea posible para futuras investigaciones que puedan basarse en esta implementación que vamos a realizar.

Agradecimientos

En primer lugar, quisiera agradecer a mi tutor Gabriel, que ha confiado en mí en todo momento a la hora de realizar este proyecto, y me ha animado siempre a seguir adelante, haciendo que todo el trabajo a realizar pareciese mucho más fácil de lo que estimaba en un principio, y consiguiendo que esta experiencia fuese algo placentero. También quiero agradecer a todo el grupo de investigación *Human Robotics*, pues me han brindado de una inestimable ayuda a lo largo de la realización de este desarrollo, habiendo hecho del laboratorio ya prácticamente una segunda casa con todos los profesores y compañeros que me han acompañado en el proceso de creación de esta parte del grado.

No quiero olvidarme tampoco en estas líneas de mis compañeros de grado, los cuales se han esforzado junto a mí a lo largo de estos cuatro años, y además me han ayudado y apoyado en todo momento a lo largo de mis estudios. Sin ellos esta época de mi vida no hubiese sido lo mismo, y les debo gran parte del poder estar realizando este trabajo, espero que puedan conseguir cualquier meta que se propongan.

Finalmente, dejo mis últimos agradecimientos a mi familia y amigos, los cuales son mi inspiración para continuar adelante en todo momento. En estos cuatro años he aprendido muchas cosas, y también he pasado por muchas experiencias, y ha sido gracias a ellos que dichas experiencias me han hecho ser quien soy. Os llevo siempre cerca, pase lo que pase.

Dedicatorias

A mis abuelos, que me han enseñado el verdadero valor de las cosas

A mi tío José, estés donde estés tu recuerdo me alienta a seguir luchando siempre

Citas

Recuerda mirar arriba, a las estrellas, y no abajo, a tus pies. Intenta encontrar el sentido a lo que ves y pregúntate qué es lo que hace que el Universo exista. Sé curioso. Por muy difícil que te parezca la vida, siempre hay algo que puedes hacer y en lo que puedes tener éxito. Lo único que cuenta es no rendirse.

Stephen Hawking

Índice de contenidos

Índice de figuras ix

1. Introducción 1
   1.1. Estructura y metodología para desarrollar nuestro trabajo 2
   1.2. Estado del arte 3
      1.2.1. Robótica humanoide, historia y actualidad 3
      1.2.2. Métodos de control visual, desarrollo y resumen de los más destacados 9
      1.2.3. Control visual en robótica humanoide. Desarrollos realizados 13
   1.3. Objetivos 16

2. Metodología: bases teóricas y técnicas del desarrollo 18
   2.1. Visión artificial, métodos implementados 18
      2.1.1. Métodos de detección de regiones de interés para el controlador 19
      2.1.2. Parámetros de interés de la cámara: el modelo Pinhole 20
      2.1.3. Control visual basado en imagen: fundamentos y base matemática 23
   2.2. Robot NAO: especificaciones técnicas para nuestro sistema 26
      2.2.1. Conceptos y datos genéricos de nuestro robot 27
      2.2.2 Programación del robot: lenguaje, sistemas y librerías utilizados 29
      2.2.2. Locomoción básica del NAO 30
      2.2.3. Sistema de cámaras del NAO y obtención de imágenes 34
      2.2.4. Sistemas de referencia de nuestro robot 35
   2.3. Control visual en nuestro sistema robótico. 38
      2.3.1. Transformaciones realizadas para los sistemas de referencia. 39
      2.3.2. Detección de referencias con nuestro robot NAO 42
      2.3.3. Implementación del controlador: estimación de características, ganancia adaptativa, y propiedades del controlador 44

3. Experimentación 51
3.1. Planteamiento inicial: posicionamiento del entorno y experimentos a realizar ... 51
3.1.1. Planteamiento para los experimentos de control visual basado en imagen 51
3.1.2. Planteamiento para los experimentos de control de ver y mover 54
3.2. Resultados de los experimentos realizados 54
3.2.1. Resultados para λ_0= 1, λ_∞ = 0.1 y λ'_0 = 10 55
3.2.2. Resultados para λ_0= 4.10, λ_∞= 0.4 y λ'_0= 40 57
3.2.3. Resultados para λ_0 = 5.10, λ_∞ 0.5 y λ'_0 = 50 62
3.2.4. Resultados para λ_0 = 6.10, λ_∞= 0.6 y λ'_0 = 60 66
3.2.5. Resultados para λ_0 = 10.1, λ_∞ = 1 y λ'_0 = 100 71
3.2.6. Experimento con referencia en movimiento 74
3.2.7. Resultados obtenidos para el controlador ver y mover 76
3.3. Comparativa de resultados y conclusiones sobre estos 77

4. Conclusiones 80
4.1. Trabajos futuros 81

Referencias 82

Anexos 84
Anexo A: Programa de controlador ver y mover 84
Anexo B: Programa del controlador visual basado en imagen 85

Índice de figuras

Figura 1: Ilustración del robot Elektro. Fuente: Belzec. https://blog.belzec.net/historia-de-los-robots-el-primer-robot/ (visitado el 06-05-2019) 1
Figura 2: Diagrama representativo de la metodología de espiral. Fuente: Wikipedia. https://es.wikipedia.org/wiki/Desarrollo_en_espiral (visitado el 07/05/2019) 2
Figura 3: Representación del funcionamiento del ZMP. Fuente: http://www.sohu.com/a/167908804_172964 (visitado el 07/05/2019) 3
Figura 4: Ilustración de algunos tipos de robots con locomoción bípeda pasiva. Fuente: Siciliano, Bruno; Oussama, Khatib. Springer Handbook of Robotics. Nápoles, Italia. Springer, vol 1, 2008. p 1312-1315. 4
Figura 5: WABOT-1. Fuente: Pinterest. https://www.pinterest.es/pin/549298485772019771 (visitado el 07/05/2019) 5
Figura 6: Robots desarrollados durante la duración de los proyectos de la serie P. Fuente: The Robot Report. https://www.therobotreport.com/honda-asimo-robot-discontinued/ (visitado el 08/05/2019) 6
Figura 7: Robot COG en su proceso de construcción. Fuente: Semantic Scholar. https://www.semanticscholar.org/paper/Learning-about-objects-through-action-initial-steps-Fitzpatrick-Metta/4cc4dc9d7d51b342510e8cc90b51b189ccf57dd4/figure/4 (visitado el 08/05/2019) 6
Figura 8: Nao, Romeo y Pepper respectivamente. Fuente: ResearchGate. https://www.researchgate.net/figure/Aldebaran-humanoids-From-left-to-right-the-robots-Nao-Romeo-and-Pepper_fig5_322748447 (visitado el 09/05/2019) 7
Figura 9: Robot Alpha 1 pro. Fuente: Wellbots. https://www.wellbots.com/ubtech-alpha-1-pro-connected-humanoid-robot/ (visitado el 09/05/2019) 7
Figura 10: Robot Atlas. Fuente: Boston Dynamics. https://www.bostondynamics.com/atlas#&gid=1&pid=1 (visitado el 09/05/2019) 8
Figura 11: Robot ICUB. Fuente: RobotCub. http://www.robotcub.org/ (visitado el 09/05/2019) 8
Figura 12: Controlador de tipo ver y mover. Fuente: Nigri, Ilana; Meggiolaro, Marco A.; Queiroz, Raul. Comparison Between look-and-move and visual servo control using sift transform in eye-in-hand manipulator system, 20th International Congress of Mechanical Engineering, 2009. 9
Figura 13: Controlador visual basado en posición. Fuente: Nigri, Ilana; Meggiolaro, Marco A.; Queiroz, Raul. Comparison Between look-and-move and visual servo control using sift transform in eye-in-hand manipulator system, 20th International Congress of Mechanical Engineering, 2009. 9
Figura 14: Esquema básico con los componentes destacados de un controlador visual. Fuente: Pomares, P., García, G. J., Pérez, J., Gil, P., Torres, F. “Control Visual”, Conceptos y Métodos en Visión por Computador, Ed. CEA, (2016) pp. 303-319. 10
Figura 15: Diferencias en las configuraciones de las cámaras. Fuente: Hutchinson, Seth; Hager, Gregory D.; Corke, Peter I., A tutorial on visual Servo Control, IEEE Transactions on Robotics and Automation, 12(5) (1996). 11
Figura 16: Control visual indirecto basado en posición. Fuente: Pomares, P., García, G. J., Pérez, J., Gil, P., Torres, F. “Control Visual”, Conceptos y Métodos en Visión por Computador, Ed. CEA, (2016) pp. 303-319. 11
Figura 17: Controlador visual basado en imagen indirecto. Fuente: Pomares, P., García, G. J., Pérez, J., Gil, P., Torres, F. “Control Visual”, Conceptos y Métodos en Visión por Computador, Ed. CEA, (2016) pp. 303-319. 12
Figura 18: Robot llevando a cabo tarea de manipulación con control visual y ambas manos. Fuente: Vahrenkamp, Nikolaus; Böge, Christian; Welke, Kai; Asfour, Tamim; Walter, Jürgen;

Dillman, Rüdiger, Visual Servoing for Dual Arm Motions on a Humanoid Robot, 9th IEEE-RAS International Conference on Humanoid Robots, Humanoids09 (2009). 13
Figura 19: Imagen donde el robot reconoce los objetivos para conocer los puntos de agarre para ambas manos. Fuente: Vahrenkamp, Nikolaus; Böge, Christian; Welke, Kai; Asfour, Tamim; Walter, Jürgen; Dillman, Rüdiger, Visual Servoing for Dual Arm Motions on a Humanoid Robot, 9th IEEE-RAS International Conference on Humanoid Robots, Humanoids09 (2009). 13
Figura 20: Pasos realizados para el agarre del objeto desconocido del robot. Fuente: Fantacci, Claudio; Vezzani, Giula; Pattacini, Ugo; Tikhanoff, Vadim; Natale, Lorenzo, Markerless visual servoing on unknown objects for humanoid robot platforms, 2018 IEEE International Conference on Robotics and Automation (ICRA) (2018): 3099-3106 14
Figura 21: Proceso de agarre de un objeto cualquiera del método destacado. Fuente: Ardón, Paola; Dragone, Mauro; Erden, Mustafa, Reaching and Grasping of Objects by Humanoid Robots Through Visual Servoing, Lecture Notes in Computer Science, vol 10894 (2018) 14
Figura 22: Ilustración del movimiento del robot con sus distintos sistemas de referencia. Fuente: Dune, C.; Stasse, O.; Yoshida, E; Herdt, A; Wieber, P-B.; Marchand, E, Visual Servoing for Dynamic Walking, IEEE/RSJ International Conference on Intelligent Robots and System (2010) 15
Figura 23: Robot Romeo controlando la posición de distintas partes de su cuerpo mediante control visual. Fuente: Agravante, Don; Claudio, Giovanni; Spindler, Fabien; Chaumette, François, Visual servoing in an optimization framework for the whole-body control of humanoid robots, EEE Robotics and Automation Letters, 2(2) (2017): 608-615..... 16
Figura 24: Ejemplo de cuatro píxeles dentro de una misma región debido a su conectividad e igualdad en nivel de gris. Fuente: Wikipedia. https://en.wikipedia.org/wiki/Pixel_connectivity#/media/File:Sasiedztwa_4_8.svg (visitado el 13/05/2019) 19
Figura 25: Esquema del proceso de detección utilizado para nuestro método. Fuente: ZBar bar code reader, http://zbar.sourceforge.net/about.html (visitado el 13/05/2019)..... 20
Figura 26: Representación de los datos principales del modelo Pinhole. Fuente: Monasse, Pascal; Morel, Jean-Michel; Tang, Zhongwei, Epipolar rectification, (2019)..... 21
Figura 27: Robot NAO en el entorno de trabajo 27
Figura 28: medidas generales de nuestro robot. Fuente: Aldebaran documentation. http://doc.aldebaran.com/2-1/family/robots/dimensions_robot.html (visitado el 16/05/2019)... 27
Figura 29: posiciones de las distintas cadenas cinemáticas que encontramos en nuestro robot. Fuente: Aldebaran Documentation. http://doc.aldebaran.com/2-1/family/robots/links_robot.html (visitado el 16/05/2019) 28
Figura 30: Articulaciones del NAO en sus respectivas cadenas cinemáticas. Fuente: Aldebaran Documentation. http://doc.aldebaran.com/2-1/family/robots/bodyparts.html (visitado el 16/05/2019) 29
Figura 31: Péndulo invertido lineal bípedo. Fuente: van Dalen, S. J., A linear inverted pendulum walk implemented on TUlip, Student tesis, Eindhoven university of technology (2012)..... 31
Figura 32: Péndulo invertido 3D. Fuente: van Dalen, S. J., A linear inverted pendulum walk implemented on TUlip, Student tesis, Eindhoven university of technology (2012) 31
Figura 33: Código ViSP para mandar un comando de velocidad a nuestro robot..... 33
Figura 34: Ilustración en lateral y en planta de la disposición y rango de las cámaras. Fuente: Aldebaran Documentation. http://doc.aldebaran.com/2-1/family/robots/video_robot.html (visitado el 16/05/2019) 34
Figura 35: Código básico para la obtención de imagen mediante ViSP y ROS..... 35
Figura 36: Sistemas de referencia de nuestro robot para cada una de sus articulaciones. Fuente: ResearchGate. https://www.researchgate.net/figure/Figura-2-Diagrama-Cinematico-Robot-NAO_fig2_303592455 (visitado el 16/05/2019) 35

Figura 37: Frame de referencia del robot. Fuente: Aldebaran Documentation. http://doc.aldebaran.com/2-1/naoqi/motion/control-cartesian.html (visitado el 16/05/2019)..... 36
Figura 38: Dibujo del robot con el sistema de referencia utilizado en nuestra cámara a la hora de trabajar. Fuente: SciELO. http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0123-77992014000200007 (visitado el 17/05/2019) 37
Figura 39: Referencia para los comandos de velocidad. Fuente: Aldebaran Documentation. http://doc.aldebaran.com/2-1/naoqi/motion/control-cartesian.html (visitado el 16/05/2019)..... 39
Figura 40: Referencia para la cámara. Fuente: SciELO. http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0123-77992014000200007 (visitado el 17/05/2019) 39
Figura 41: Diferencia entre las referencias dentro de la imagen 41
Figura 42: Fotografías de la detección del código Qr que realizamos con nuestro robot con información espacial extraída de la imagen 42
Figura 43: Código realizado para la estimación inicial de los puntos dentro de la imagen 43
Figura 44: Código requerido para la creación de puntos de características con nuestro controlador 44
Figura 45: Código para la estimación de posición de nuestro robot 45
Figura 46: Método para añadir las características que hemos calculado dentro de nuestro controlador 48
Figura 47: Funciones y valores para nuestra ganancia de controlador adaptativa 48
Figura 48: Código para iniciar el controlador que vamos a utilizar 49
Figura 49: Código necesario para calcular la velocidad que estimaremos para la cámara de nuestro robot..... 49
Figura 50: Código para estimar el final del bucle de control 50
Figura 51: Posicionamiento de referencia en perspectiva, planta, y desde el punto de vista del robot de los elementos del experimento para tomar los datos de referencia. 51
Figura 52: Posicionamiento inicial en trayectoria lineal en perspectiva, planta, y desde el punto de vista del robot de los elementos del experimento. 52
Figura 53:Posicionamiento inicial en trayectoria curvada en perspectiva, planta, y desde el punto de vista del robot de los elementos del experimento. 52
Figura 54: Posicionamiento inicial del robot y la referencia en perspectiva y planta para el control de tipo ver y mover 54
Figura 55: Evolución de la señal de velocidad para el caso $\lambda_0 = 1$, $\lambda_\infty = 0.1$ y $\lambda'_0 = 10$ 55
Figura 56: Evolución de la señal de error para el caso $\lambda_0 = 1$, $\lambda_\infty = 0.1$ y $\lambda'_0 = 10$ 55
Figura 57: Seguimiento de las referencias visuales en imagen del robot 55
Figura 58: Pose final en perspectiva y planta de nuestro robot para $\lambda_0 = 1$, $\lambda_\infty = 0.1$ y $\lambda'_0 = 10$ .. 56
Figura 59: Matriz de interacción y errores para el caso $\lambda_0 = 1$, $\lambda_\infty = 0.1$ y $\lambda'_0 = 10$ 56
Figura 60: Evolución de la señal de velocidad para $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria lineal 57
Figura 61: Evolución de la señal de error para $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria lineal 57
Figura 62: Seguimiento de las referencias visuales en imagen del robot para $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria lineal 58
Figura 63: Pose final en perspectiva y planta de nuestro robot para $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria lineal 58
Figura 64: Matriz de interacción y errores para el caso $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria lineal 59
Figura 65: Evolución de la señal de velocidad para $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria curva 59
Figura 66: Evolución de la señal de error para $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria curvada 59

Figura 67: Seguimiento de las referencias visuales en imagen del robot para $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria curvada 60
Figura 68: Pose final en perspectiva y planta de nuestro robot para $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria curvada 61
Figura 69: Matriz de interacción y errores para el caso $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria curvada 61
Figura 70: Evolución de la señal de velocidad para $\lambda_0 = 5.1$, $\lambda\infty = 0.5$ y $\lambda'_0 = 50$ con trayectoria lineal 62
Figura 71: Evolución de la señal de error para $\lambda_0 = 5.1$, $\lambda_\infty = 0.5$ y $\lambda'_0 = 50$ con trayectoria lineal 62
Figura 72: Seguimiento de las referencias visuales en imagen del robot para $\lambda_0 = 5.1$, $\lambda_\infty = 0.5$ y $\lambda'_0 = 50$ con trayectoria lineal 63
Figura 73: Pose final en perspectiva y planta de nuestro robot para $\lambda_0 = 5.1$, $\lambda_\infty = 0.5$ y $\lambda'_0 = 50$ con trayectoria lineal 63
Figura 74: Matriz de interacción y errores para el caso $\lambda_0 = 5.1$, $\lambda_\infty = 0.5$ y $\lambda'_0 = 50$ con trayectoria lineal 64
Figura 75: Evolución de la señal de velocidad para $\lambda_0 = 5.1$, $\lambda\infty = 0.5$ y $\lambda'_0 = 50$ con trayectoria curvada 64
Figura 76: Evolución de la señal de error para $\lambda_0 = 5.1$, $\lambda_\infty = 0.5$ y $\lambda'_0 = 50$ con trayectoria curvada 64
Figura 77: Seguimiento de las referencias visuales en imagen del robot para $\lambda_0 = 5.1$, $\lambda_\infty = 0.5$ y $\lambda'_0 = 50$ con trayectoria curvada 65
Figura 78: Pose final en perspectiva y planta de nuestro robot para $\lambda_0 = 5.1$, $\lambda_\infty = 0.5$ y $\lambda'_0 = 50$ con trayectoria curvada 65
Figura 79: Matriz de interacción y errores para el caso $\lambda_0 = 5.1$, $\lambda_\infty = 0.5$ y $\lambda'_0 = 50$ con trayectoria curvada 66
Figura 80: Evolución de la señal de velocidad para $\lambda_0 = 6.1$, $\lambda\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria lineal 67
Figura 81: Evolución de la señal de error para $\lambda_0 = 6.1$, $\lambda_\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria lineal 67
Figura 82: Seguimiento de las referencias visuales en imagen del robot para $\lambda_0 = 6.1$, $\lambda_\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria lineal 68
Figura 83: Pose final en perspectiva y planta de nuestro robot para $\lambda_0 = 6.1$, $\lambda_\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria lineal 68
Figura 84: Matriz de interacción y errores para el caso $\lambda_0 = 6.1$, $\lambda_\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria lineal 69
Figura 85: Evolución de la señal de velocidad para $\lambda_0 = 6.1$, $\lambda\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria curvada 69
Figura 86: Evolución de la señal de error para $\lambda_0 = 6.1$, $\lambda_\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria lineal 69
Figura 87:Seguimiento de las referencias visuales en imagen del robot para $\lambda_0 = 6.1$, $\lambda_\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria curvada 70
Figura 88: Pose final en perspectiva y planta de nuestro robot para $\lambda_0 = 6.1$, $\lambda_\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria curvada 70
Figura 89: Matriz de interacción y errores para el caso $\lambda_0 = 6.1$, $\lambda_\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria curvada 71
Figura 90: Evolución de la señal de velocidad para el caso $\lambda_0 = 10$, $\lambda_\infty = 1$ y $\lambda'_0 = 100$ 71
Figura 91: Evolución de la señal de error para el caso $\lambda_0 = 10$, $\lambda_\infty = 1$ y $\lambda'_0 = 100$ 72
Figura 92: Seguimiento de las referencias visuales en imagen del robot para $\lambda_0 = 10$, $\lambda_\infty = 1$ y $\lambda'_0 = 100$ 72
Figura 93: Pose final en perspectiva y planta de nuestro robot para $\lambda_0 = 10$, $\lambda_\infty = 1$ y $\lambda'_0 = 100$. 73

Figura 94: Matriz de interacción y errores para el caso $\lambda_0 = 10$, $\lambda_\infty = 1$ y $\lambda'_0 = 100$ 73
Figura 95: Evolución de la señal de velocidad para el caso de la referencia móvil 74
Figura 96: Evolución de la señal de error para el caso de la referencia móvil 75
Figura 97: Seguimiento de las referencias visuales en imagen del robot para el caso de la referencia móvil 75
Figura 98: Matriz de interacción y errores para el caso de la referencia móvil 76
Figura 99: Valores de posición finales obtenidos con el controlador ver y mover 76
Figura 100: Posición final alcanzada con el controlador de ver y mover 77

1. Introducción

En numerosas ocasiones a lo largo de la historia de la robótica se ha tratado de imitar el comportamiento humano en muchos aspectos a la hora de realizar muchas de las tareas que han sido requeridas, como ha podido ser el movimiento de un brazo en los robos industriales, o la capacidad de análisis visual en los últimos años mediante la visión artificial. Como podemos observar, siempre se suele realizar un enfoque parcial a la hora de aplicar las capacidades humanas a una máquina, cogiendo en muchos casos tan solo una propiedad de nosotros las personas, para no complicar más de la cuenta el proceso de creación del sistema que queremos plantear.

Sin embargo, en algunos casos se ha tratado de replicar el cuerpo y comportamiento humano de forma más precisa teniendo en cuenta más de un aspecto de estos campos, y tratando de combinarlos en un mismo robot para obtener robots **humanoides**. El primer robot que se realizó tratando de imitar el comportamiento de una persona de forma lo más realista que se pudo realizar fue el robot **Elektro** (ver Fig. 1) de la compañía *Westinghouse Electric Corporation* en el año 1937.

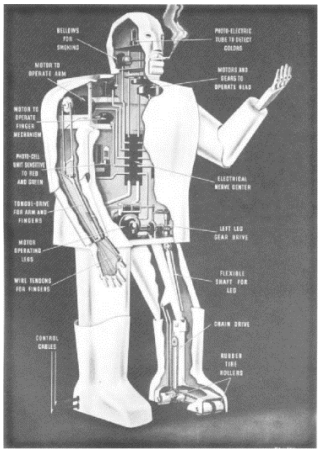

Figura 1: Ilustración del robot Elektro. Fuente: Belzec. https://blog.belzec.net/historia-de-los-robots-el-primer-robot/ (visitado el 06-05-2019)

Tras este robot simple con aspecto humano que tan solo era capaz de caminar por comando de voz, decir 700 palabras grabadas previamente y distinguir el rojo del verde [1], a lo largo de la historia se han ido realizando más y más implementaciones de robots humanoides que han ido mejorando su forma de integrar las capacidades humanas en conjunto que se han ido seleccionando hasta llegar a la actualidad, donde encontramos grandes avances en la robótica humanoide.

Hoy en día, el desarrollo de robots humanoides ha avanzado hasta el punto de poder realizar métodos de control basados en el procesamiento de imagen que realizamos desde las cámaras del mismo robot, como veremos en este trabajo. Sabiendo esto, a lo largo de esta memoria podremos ver cómo este avance nos permite realizar un sistema capaz de guiar al robot humanoide NAO haciendo uso de distintas librerías y teniendo en cuenta las propiedades que conocemos de este tipo de robots que hemos ido adquiriendo a lo largo de la historia.

1.1. Estructura y metodología para desarrollar nuestro trabajo

En este primer punto introductorio, trataremos de realizar un análisis del estado del arte de algunas de las técnicas y sistemas relacionados con el trabajo a realizar en su primera parte, para luego más tarde realizar una breve explicación de los objetivos que nos planteamos con este trabajo a partir de toda la teoría analizada para saber hasta dónde queremos llegar.

En el punto dos que le sigue, trataremos de explicar de la forma más técnica posible algunos de los puntos teóricos que se han expuesto en el estado del arte, y además también daremos algunas explicaciones sobre el funcionamiento del robot que vamos a utilizar a lo largo del desarrollo de este trabajo, para más tarde explicar finalmente cómo vamos a juntar ambos campos para conocer cómo van a funcionar las experimentaciones que hagamos.

En el siguiente punto, hablaremos sobre las pruebas y experimentaciones que hemos ido realizando con el sistema propuesto en el anterior punto, viendo también los resultados que vamos obteniendo de este sistema, comentando éstos y luego comparándolos más tarde con otros experimentos similares para conocer su validez respecto a otros métodos.

Finalmente, en el último punto realizaremos unas breves conclusiones a partir de todo el trabajo que se ha ido realizando respecto a este método de control, viendo cómo se han cumplido los objetivos que hemos ido proponiendo, y además tratando de proponer nuevas investigaciones que se deriven del trabajo que se ha realizado en esta memoria.

En cuanto a la metodología que se ha seguido a la hora de desarrollar estos puntos y realizar las experimentaciones necesarias para nuestro trabajo, podemos destacar que se ha seguido una metodología de tipo espiral (ver Fig. 2), donde hemos definido en primer lugar hasta dónde queremos llegar, luego hemos planificado cómo queríamos llegar a ese punto, hemos hecho las pruebas que se nos ha ido requiriendo para ello, y finalmente se ha hecho un análisis de los resultados que se han ido obteniendo para volver a plantear nuevos objetivos a los que ir llegando poco a poco.

Figura 2: Diagrama representativo de la metodología de espiral. Fuente: Wikipedia. https://es.wikipedia.org/wiki/Desarrollo_en_espiral (visitado el 07/05/2019)

1.2. Estado del arte

En este apartado, trataremos de comentar brevemente cómo han evolucionado los métodos y sistemas que vamos a ir viendo a lo largo de nuestro proyecto, todo siempre ligado al campo del control de marcha en robots humanoides y de visión artificial en la robótica.

Así, para tratar todos estos temas, en el primer punto que analizaremos hablaremos sobre la historia y el desarrollo de la robótica humanoide, viendo cómo ha ido evolucionando este tipo de robots y cuáles son los más destacables en la actualidad. Tras esto, pasaremos a hablar sobre algunos métodos de control desarrollados mediante el uso de visión artificial, comparando estos por encima para saber cómo funcionan y viendo algún ejemplo de robots que utilicen estos métodos. Finalmente, hablaremos sobre algunas investigaciones ya realizadas sobre robots humanoides con métodos de control de marcha con visión artificial, viendo así en conjunto todos los temas de los que vamos a ir hablando, y viendo cómo se ha conseguido que estos métodos funcionen de manera adecuada de forma muy breve.

1.2.1. Robótica humanoide, historia y actualidad

En primer lugar, antes de destacar ningún robot, debemos de hablar sobre uno de los problemas principales que plantean este tipo de robots, para el cual veremos cuáles son las soluciones más destacables que se han ido planteando a lo largo de la historia, el cual es el problema de la **locomoción bípeda.**

El primer concepto que se encontró para solventar este problema fue encontrado en 1968 por Miomir Vokobratovic al tratar de desarrollar un exoesqueleto. Dicho método sería conocido como ***Zero Moment Point*** (ZMP de aquí en adelante), y se basaría en conseguir el punto respecto al cual cualquier esfuerzo que realice el pie contra el suelo no produzca ningún momento en la dirección horizontal (ver Fig. 3).

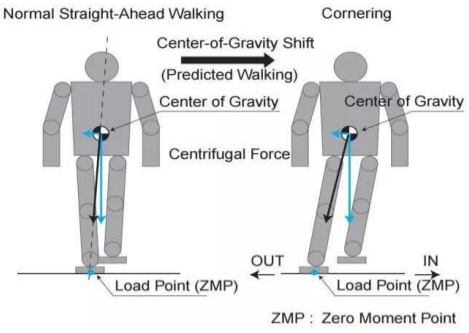

Figura 3: Representación del funcionamiento del ZMP. Fuente: http://www.sohu.com/a/167908804_172964 (visitado el 07/05/2019)

Dicho método se ha seguido utilizando hasta la actualidad, y se ha usado para planificar patrones de marcha. Así, teniendo una trayectoria deseada del punto de momento cero, se van resolviendo las ecuaciones diferenciales que surgen para contrarrestar las fuerzas necesarias en *offline* respecto al centro de gravedad del robot [2].

A partir de este método, se han propuesto nuevos estudios basados en su teoría que tratan de perfeccionarla para mejorar la locomoción humana en robots, como bien puede ser el método de generación de patrones en tiempo real, el cual resuelve las ecuaciones diferenciales en tiempo real, generando un patrón de marcha así de forma *online*, y teniendo una actualización de la marcha en función de las fuerzas externas del robot y de las características del terreno [3]. Otros métodos destacables que se han desarrollado a lo largo de la historia de la robótica basados en el ZMP son el *generalized ZMP*, el cual tiene en cuenta información de seis dimensiones en vez de sólo de dos, pudiendo añadir información de fuerza y momentos externos, o las investigaciones de movimiento en carrera, con las cuales se tiene en cuenta también la fase de vuelo dentro de la teoría destacada [4].

Además de estos métodos destacados que se basan en el ZMP, también podemos encontrar otros aparte que funcionan de manera adecuada y que tratan de innovar en la implementación de la locomoción bípeda, como bien pueden ser las aproximaciones que se han realizado hasta el momento de la marcha pasiva (ver Fig. 4) y de los modelos de péndulo invertido, los cuales se basan en explotar dinámicas naturales de los mismos mecanismos de los robots, consiguiendo así una mayor eficiencia en el movimiento que pretenden realizar [4].

Figura 4: Ilustración de algunos tipos de robots con locomoción bípeda pasiva. Fuente: Siciliano, Bruno; Oussama, Khatib. Springer Handbook of Robotics. Nápoles, Italia. Springer, vol 1, 2008. p 1312-1315.

También cabe destacar que, durante el desarrollo de todas estas teorías, también se desarrollaron otros métodos importantes para temas relacionados con la locomoción y que podían afectar a los robots, como pueden ser bucles de control para recuperarse de caídas, o implementaciones del ZMP que fuesen capaces de ser validas a la vez que el robot carga un objeto [4].

Una vez que ya conocemos unas pinceladas básicas sobre cómo se empezaron a plantear métodos de locomoción bípeda para nuestros robots, y cómo estos métodos han evolucionado y han variado a lo largo del tiempo, podemos pasar a ver los distintos robots destacables que se han ido realizando a lo largo de la historia moderna que se basan en esta teoría y que han supuesto pasos importantes en el desarrollo de este tipo de máquinas.

Aunque en la introducción hemos visto unos primeros breves pasos de desarrollo de un robot humanoide, no fue hasta el año 1973 cuando se desarrolló y construyó el primer robot humanoide haciendo uso de electrónica moderna de forma eficiente. Este robot fue el **WABOT-1**, desarrollado en la universidad Waseda de Tokio, en Japón (ver Fig. 5). Dicho robot constaba de un sistema de control de miembros, un sistema básico de visión, y otro de conversación. Con dicho robot, se inició el desarrollo de robots humanoides modernos [5].

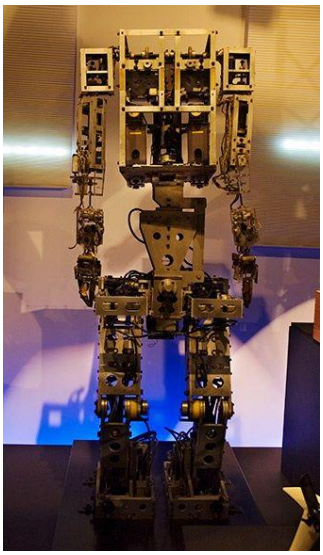

Figura 5: WABOT-1. Fuente: Pinterest. https://www.pinterest.es/pin/549298485772019771 (visitado el 07/05/2019)

Tras el desarrollo de este robot, habría que esperar unos años, hasta 1986, para poder ver un desarrollo nuevo de robots basados en la fisiología humana. Así, en este año veríamos como Honda empezó a desarrollar el proyecto de robot P2. Este fue el primer robot a escala real capaz de caminar por sí solo, a diferencia del WABOT-1 que requería de ayudas para poder realizar la locomoción. Dicho proyecto no terminó de desarrollarse hasta el año 1996, presentando así un diseño bastante más ligero del que se presentó inicialmente, y habiendo realizado un robot P1 previamente. Así, en Honda se comenzó a desarrollar la serie de robots P, la cual en un futuro derivaría en los robots **ASIMO** de la compañía (ver Fig. 6).

Con dichos desarrollos se consiguió ir mejorando el estilo de los robots bípedos para que estos tuviesen un movimiento más natural y dinámico, reduciendo además en cada nueva entrega de este tipo de robots el peso de éstos, para conseguir dichos avances en el movimiento y el comportamiento de los robots [6].

Cabe destacar que la diferencia de pesos y medidas en orden decreciente se mantuvo hasta los robots ASIMO, en los cuales se invirtió dicho orden, creando en primer lugar una versión más pequeña de la que luego fue la más reciente hasta la época. Esta serie de robots supuso un gran avance en la investigación de la robótica humanoide, creando así una base para los siguientes desarrolladores a la hora de crear robots de este tipo [6].

Figura 6: Robots desarrollados durante la duración de los proyectos de la serie P. Fuente: The Robot Report. https://www.therobotreport.com/honda-asimo-robot-discontinued/ (visitado el 08/05/2019)

Además, a la vez que se realizaba el desarrollo de esta serie de robots, desde el MIT Artificial Laboratory se empezó a desarrollar el proyecto **COG** en 1993 (ver Fig.7), el cual se basaba en realizar la construcción de un robot que aprendiese a realizar movimientos cada vez más complejos a partir de sus experiencias corporales, juntando en una investigación el campo de la inteligencia artificial y de la robótica humanoide, y siendo de gran importancia a la hora de comprender cómo integrar las capacidades sensoriales que pueda requerir un robot de este tipo a la hora de realizar tareas de carácter más complejo [7].

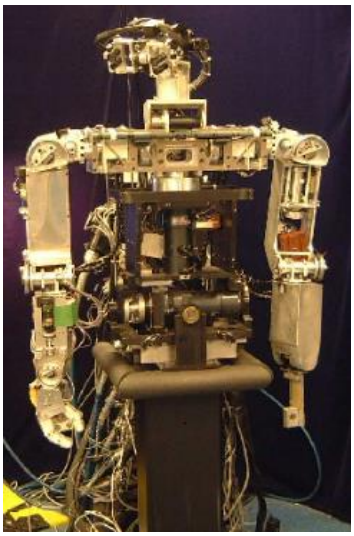

Figura 7: Robot COG en su proceso de construcción. Fuente: Semantic Scholar. https://www.semanticscholar.org/paper/Learning-about-objects-through-action-initial-steps-Fitzpatrick-Metta/4cc4dc9d7d51b342510e8cc90b51b189ccf57dd4/figure/4 (visitado el 08/05/2019)

Tras esta sucesión de importantes desarrollos, a partir del año 2000 hasta la actualidad observamos un gran avance en el campo de la robótica humanoide, dándose grandes desarrollos de robots de este tipo que serían decisivos a la hora de mejorar nuestro entendimiento para trabajar con ellos, e incluso abriendo estos a un público más amplio y pudiendo usarse éstos en contextos más abiertos que los que se dan en los entornos de investigación.

A partir de este año podemos dividir entre dos tipos de robots humanoides según su tipo de uso, distinguiendo entre robots más comerciales, con un uso más extendido y con producciones en cadena, que se venden al por mayor, y luego robots más orientados a investigación, más complejos y caros, y que están pensados para ser probados en avances para este tipo de robótica.

En cuanto a los robots del primer tipo de los que hemos hablado, podemos destacar algunos como pueden ser los que se producen desde la compañía Aldebaran (ahora Softbank Robotics), los cuales son el Romeo, el Pepper, o el robot que usaremos en este trabajo, el NAO (ver Fig. 8). De estos tres, el robot más avanzado que se ha realizado es el Romeo, el cual aún se encuentra en desarrollo, y se planea utilizar como una versión avanzada de los dos robots anteriores mencionados, perfeccionando cada uno de sus puntos.

Figura 8: Nao, Romeo y Pepper respectivamente. Fuente: ResearchGate. https://www.researchgate.net/figure/Aldebaran-humanoids-From-left-to-right-the-robots-Nao-Romeo-and-Pepper_fig5_322748447 (visitado el 09/05/2019)

Otro robot de este tipo que también podemos destacar es el Alpha 1 pro, robot de la empresa Ubtech que se creó como juguete para niños (ver Fig. 9). Este robot es capaz de cantar, bailar e interactuar para ser usado como un compañero de juegos.

Figura 9: Robot Alpha 1 pro. Fuente: Wellbots. https://www.wellbots.com/ubtech-alpha-1-pro-connected-humanoid-robot/ (visitado el 09/05/2019)

En cuanto a los robots enfocados al estudio e investigación, también encontramos algunos bastante destacables, como bien puede ser el famoso robot Atlas desarrollado por *Boston Dynamics* (ver Fig. 10). Este robot en especial es muy destacable, pues está realizando mejoras muy notables en cuanto a los bucles de control para la locomoción de robots humanoides, viendo cómo este robot es capaz de moverse por terrenos inestables, es capaz de compensar fuerzas externas fuertes en su marcha, y puede correr y realizar acrobacias de forma aparentemente fácil gracias a sus sistemas y bucles de control.

Figura 10: Robot Atlas. Fuente: Boston Dynamics. https://www.bostondynamics.com/atlas#&gid=1&pid=1 (visitado el 09/05/2019)

Otro robot destacable de este tipo es el ICUB (ver Fig. 11), el cual ha sido desarrollado por *RobotCub* para simular el aprendizaje cognitivo como lo haría un niño [8], muy parecido a lo que se intentó con el robot COG, pero esta vez con sistemas y sensores mucho más avanzados, que permiten avanzar más en la investigación cognitiva de este tipo de robots.

Figura 11: Robot ICUB. Fuente: RobotCub. http://www.robotcub.org/ (visitado el 09/05/2019)

Una vez vistos todos estos robots, y la metodología principal de locomoción en los bípedos, cabe destacar que en este trabajo nos centraremos precisamente en este tipo de robots con dos piernas, pues vamos a estudiar métodos de control que funcionen con este tipo de robots. Por esto mismo también, hemos destacado en este punto la historia y avances en las teorías para el control de estos robots, centrándonos sobre todo en el método ZMP, que es en el que además se basa nuestro robot NAO.

1.2.2. Métodos de control visual, desarrollo y resumen de los más destacados

En este segundo punto del estado del arte, hablaremos brevemente sobre las bases de los métodos de control visual, y luego explicaremos brevemente el funcionamiento de las teorías más destacadas en la actualidad. Para ello, cabe destacar que el control visual se basa en el uso de visión por computadora para realizar el control de movimiento de un robot [9].

En 1980, Sanderson y Weiss clasificaron dos sistemas de control visual, el método de ver y mover, y los sistemas de control visual. El primer método (ver Fig. 12) usaba los métodos de visión para generar los puntos objetivos para las articulaciones del robot, sin tener en cuenta la información visual de forma actualizada. Por otra parte, el método de controladores visuales (ver Fig. 13) no hacía uso de controladores de posición convencionales para estimar la posición del robot, si no que estimaba las posiciones para las articulaciones mediante el *feedback* que se iba obteniendo de las cámaras que se utilizaban [10].

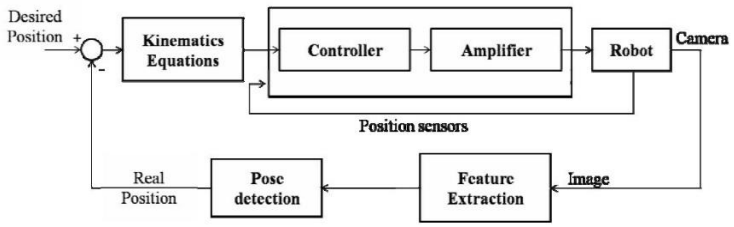

Figura 12: Controlador de tipo ver y mover. Fuente: Nigri, Ilana; Meggiolaro, Marco A.; Queiroz, Raul. Comparison Between look-and-move and visual servo control using sift transform in eye-in-hand manipulator system, 20th International Congress of Mechanical Engineering, 2009.

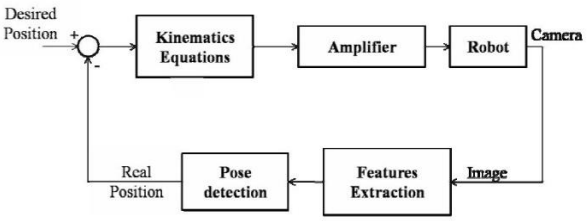

Figura 13: Controlador visual basado en posición. Fuente: Nigri, Ilana; Meggiolaro, Marco A.; Queiroz, Raul. Comparison Between look-and-move and visual servo control using sift transform in eye-in-hand manipulator system, 20th International Congress of Mechanical Engineering, 2009.

Aunque veremos más adelante cómo aplicar en un robot real el método de ver y mover, en este trabajo nos centraremos sobre todo en los métodos de control visual como tal, observando así, cómo las características que extraemos de la imagen pueden ser utilizadas para realizar bucles de control válidos que se utilicen en un sistema real.

Sabiendo esto, podemos pasar a destacar otros puntos importantes sobre el control visual, como sus componentes principales (ver Fig. 14). Así, podemos ver que dichos componentes serán los siguientes en cualquier tipo de controlador que nos encontremos de aquí en adelante:

- Referencia. Es la configuración final que queremos que alcancen las articulaciones de nuestro robot. Esta la podemos conseguir o bien teniendo en cuenta una posición 3D extraída en base a las características de la imagen (controlador basado en posición), o bien teniendo en cuenta cómo varían las características en la misma imagen en función del movimiento de nuestro robot (controlador basado en imagen).
- Controlador. Como ya sabemos, es el encargado de mover las articulaciones de nuestro robot basándose en la información que nos va llegando desde nuestra cámara.
- Sistema de visión artificial. Es la parte computacional que se encarga de tratar con las características de la imagen que queremos extraer para extraer de éstas las referencias que nos sean necesarias para hacer que nuestro robot se mueva hasta el objetivo que deseamos de manera adecuada [11].

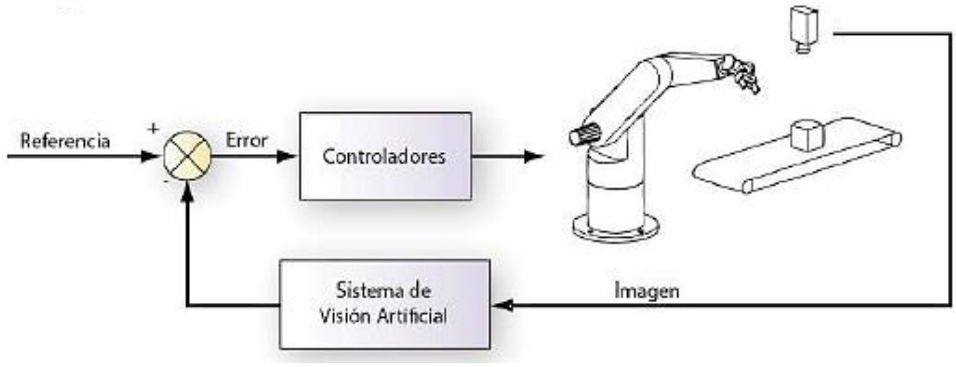

Figura 14: Esquema básico con los componentes destacados de un controlador visual. Fuente: Pomares, P., García, G. J., Pérez, J., Gil, P., Torres, F. "Control Visual", Conceptos y Métodos en Visión por Computador, Ed. CEA, (2016) pp. 303-319.

A partir de estos puntos básicos, se derivan todos los tipos de control visual que vamos a ver a continuación, sirviendo este esquema como base de cualquier tipo de control con visión, ya sea basándose en la imagen o en la posición que se quiera alcanzar.

Otro punto destacable sobre este tipo de controladores es que los podemos encontrar de dos tipos, pudiendo ser estos directos o indirectos. Con los primeros tipos de controladores, la acción de control viene dada como pares a aplicar en las articulaciones de nuestro robot, mientras que los controladores indirectos se basan en aspectos cinemáticos de nuestro robot, dándole directamente a éste simplemente la velocidad que queremos que tenga [11].

Además, también cabe destacar que la disposición de la cámara dentro del entorno de trabajo en estos controladores también es un punto muy importante, pues al cambiar la cámara de disposición nos encontramos con que los cálculos que tenga que realizar nuestro controlador a partir de la imagen serán muy distintos. Así, podemos encontrar dos aproximaciones principales a la forma de colocar la cámara en nuestro entorno (ver Fig. 15), siendo las configuraciones de la cámara colocada en el extremo del robot, conocida como "*eye-in-hand*", o una configuración con la cámara externa al robot apuntando a su manipulador, conocida como "*eye-to-hand*" [12].

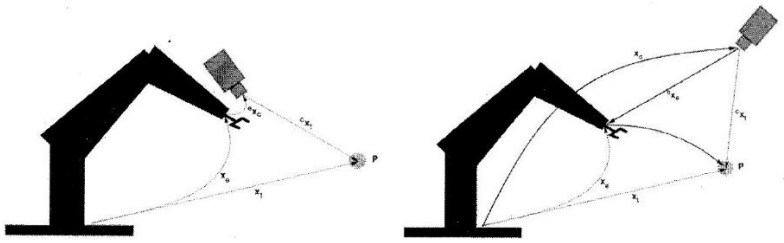

Figura 15: Diferencias en las configuraciones de las cámaras. Fuente: Hutchinson, Seth; Hager, Gregory D.; Corke, Peter I., A tutorial on visual Servo Control, IEEE Transactions on Robotics and Automation, 12(5) (1996).

A partir de todos estos datos que hemos destacado, finalmente ya podemos pasar a hablar sobre la diferencia más notable entre controladores, que es la referencia que estos toman con respecto a la imagen para realizar las estimaciones de control, que como hemos destacado anteriormente, podíamos encontrar basados en posición, y basados en imagen.

En cuanto a los controladores basados en posición (ver Fig. 16), de estos podemos decir que la entrada del regulador corresponde a la diferencia entre la localización deseada del objeto y la estimada a partir de las características que hemos obtenido de nuestro sistema de visión. En este control, la referencia será la posición y orientación deseada del robot, o un objeto existente en el espacio de trabajo [11].

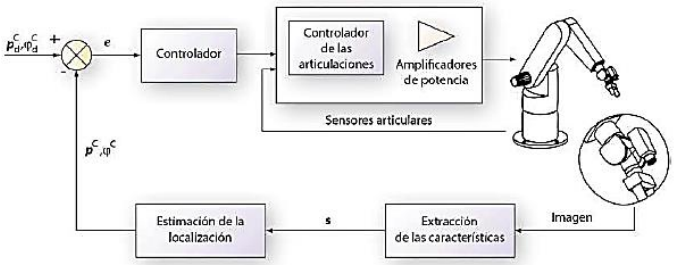

Figura 16: Control visual indirecto basado en posición. Fuente: Pomares, P., García, G. J., Pérez, J., Gil, P., Torres, F. "Control Visual", Conceptos y Métodos en Visión por Computador, Ed. CEA, (2016) pp. 303-319.

En cuanto al control visual basado en imagen (ver Fig. 17), el regulador se encarga en este caso de generar acciones de control que hagan que las características visuales de las imágenes que vamos obteniendo vayan convergiendo con las características deseadas que queremos en la imagen. A diferencia de los controladores basados en posición, en este caso no requerimos de estimar una posición 3D del objeto, haciendo así que el control se realice directamente en el espacio de la imagen. Así, el controlador generará un movimiento sobre el robot que haga que las características visuales describan una línea recta en el plano de la imagen, sin conocer de antemano que trayectoria Cartesiana va a realizar nuestro robot.

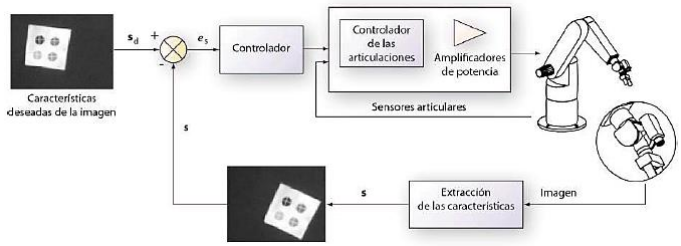

Figura 17: Controlador visual basado en imagen indirecto. Fuente: Pomares, P., García, G. J., Pérez, J., Gil, P., Torres, F. "Control Visual", Conceptos y Métodos en Visión por Computador, Ed. CEA, (2016) pp. 303-319.

Sabiendo todos estos datos sobre el control visual, a continuación, podemos pasar a ver más a fondo algunos usos en robots reales humanoides que se han dado de este tipo de controles de los que hemos hablado a lo largo de este punto de nuestro estado del arte. Cabe destacar que, en nuestro caso, en la implementación final que hemos hecho de todos estos controladores, hemos optado por probar a realizar el controlador de ver y mover en primer lugar para tener una primera aproximación de cómo trabajar con el NAO y este tipo de control, y luego hemos realizado un control basado en imagen indirecto con una configuración de la cámara en el mismo robot ("*eye-in-hand*") para realizar el estudio que queremos analizar sobre el control de marcha en robots humanoides.

Sabiendo esto, también hay que recalcar que más adelante, hablaremos en más profundidad sobre este tipo concreto de control, explicando para ello en detalle el fundamento matemático y teórico sobre el que nos hemos basado para realizar la implementación del control visual basado en imagen para nuestro robot bípedo, y hablando luego de cómo lo hemos adaptado a nuestro entorno de trabajo para que todo funcione de la manera adecuada.

1.2.3. Control visual en robótica humanoide. Desarrollos realizados

Una vez que hemos visto por separado la historia y puntos importantes de la robótica humanoide y el control visual, podemos pasar a ver cómo ha funcionado la combinación de ambos campos hasta la actualidad, viendo para ello algunos desarrollos importantes que se hayan realizado en los que nos podamos basar a la hora de realizar nuestro trabajo de investigación y experimentación.

En las investigaciones que trataremos, podremos ver cómo se utiliza el control basado en imagen en este tipo de robots, no sólo en control de marcha, si no en otras muchas tareas, como bien pueden ser tareas de agarre de objetos con las extremidades superiores, o tareas de seguimiento de objetivo con la cabeza del robot. Veremos de cada caso las investigaciones más destacables para poder sacar toda la información posible que nos pueda ser útil en un futuro.

Un desarrollo interesante de este tipo en robots antropomórficos que podemos encontrar es el control visual de estos robots para utilizar dos brazos de forma simultánea, pudiendo realizar trabajos de agarre y manipulación con ambas manos (ver Fig. 18 y Fig. 19). Así, los objetos que marcamos y ambas manos son seguidas de forma alterna y una combinación de controladores de bucle abierto y cerrado se usa para posicionar las manos con respecto a dichos objetivos. Así, haciendo uso simplemente de los sensores del mismo robot y de los bucles de control propios del sistema, se puede llevar a cabo la tarea que destacamos [13].

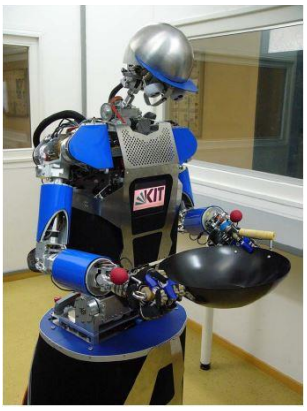

Figura 18: Robot llevando a cabo tarea de manipulación con control visual y ambas manos. Fuente: Vahrenkamp, Nikolaus; Böge, Christian; Welke, Kai; Asfour, Tamim; Walter, Jürgen; Dillman, Rüdiger, Visual Servoing for Dual Arm Motions on a Humanoid Robot, 9th IEEE-RAS International Conference on Humanoid Robots, Humanoids09 (2009).

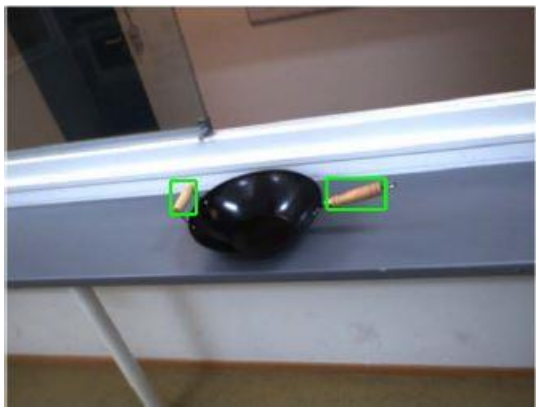

Figura 19: Imagen donde el robot reconoce los objetivos para conocer los puntos de agarre para ambas manos. Fuente: Vahrenkamp, Nikolaus; Böge, Christian; Welke, Kai; Asfour, Tamim; Walter, Jürgen; Dillman, Rüdiger, Visual Servoing for Dual Arm Motions on a Humanoid Robot, 9th IEEE-RAS International Conference on Humanoid Robots, Humanoids09 (2009).

Dicho desarrollo es importante, pues nos permite ver cómo podemos coordinar varios elementos de un mismo robot haciendo uso del control visual, como veremos que nos ocurrirá al tener que andar de forma bípeda con el robot que vamos a trabajar.

Otros desarrollos que también pueden resultar interesantes pueden ser las investigaciones con métodos de visión para el agarre de objetos sin marcas ni referencias del objeto a coger. Este desarrollo se realizó para el robot ICUB que hemos visto anteriormente, y con él lo que se pretendía es que el robot aprendiese de una forma natural a manejar objetos que viese en su entorno como una persona humana real (ver Fig. 20). Mediante este método, se requiere que el robot realice una estimación de pose y de orientación tanto del objeto como de su propio efector final para realizar una estimación de puntos válidos para el agarre en cada objeto nuevo que le ordenemos al robot que coja, teniendo así por cada iteración de objeto nuevo, todos estos cálculos a estimar para que el agarre se realice de manera adecuada [14]

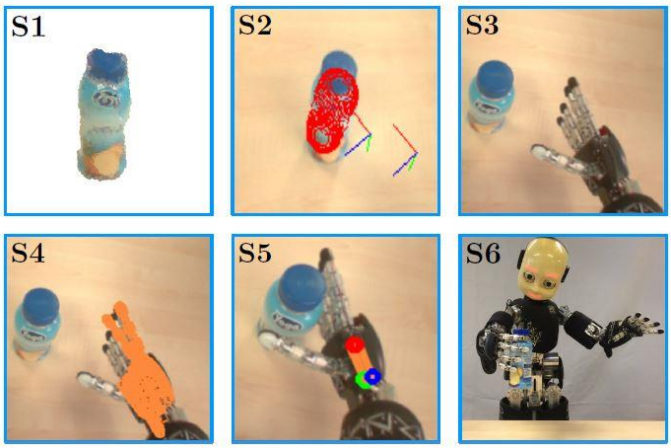

Figura 20: Pasos realizados para el agarre del objeto desconocido del robot. Fuente: Fantacci, Claudio; Vezzani, Giula; Pattacini, Ugo; Tikhanoff, Vadim; Natale, Lorenzo, Markerless visual servoing on unknown objects for humanoid robot platforms, 2018 IEEE International Conference on Robotics and Automation (ICRA) (2018): 3099-3106

Otro enfoque interesante de este tipo que podemos ver en otras investigaciones en relación con la que acabamos de destacar, es la de agarre de objetos sin marcas y además teniendo que buscar éstos en un entorno, teniendo así que realizar una estimación parecida a la del robot anterior, pero esta vez conociendo el objeto que vamos a utilizar para coger (ver Fig. 21). En este caso, se realizan unas marcas virtuales sobre la imagen para realizar el control visual de manera adecuada y poder guiar al robot al destino que deseamos [15].

Figura 21: Proceso de agarre de un objeto cualquiera del método destacado. Fuente: Ardón, Paola; Dragone, Mauro; Erden, Mustafa, Reaching and Grasping of Objects by Humanoid Robots Through Visual Servoing, Lecture Notes in Computer Science, vol 10894 (2018)

Una vez vistos los métodos de agarre con control visual de este tipo de robots, podemos pasar a destacar ahora los métodos para el control de marcha en humanoides, de los cuales podremos sacar más datos que nos serán útiles a la hora de implementar estos métodos en un experimento real con nuestro robot.

Uno de los desarrollos que podemos destacar es el de control visual de marcha dinámica, pues en algunos estudios podemos observar cómo, para este enfoque, se tiene en cuenta una mayor cantidad de datos sobre el sistema de control que en otros, como bien puede ser la varianza que hay en la imagen a cada movimiento que realiza el robot debido a cómo va cambiando la referencia de la cámara con el movimiento del robot (ver Fig. 22), viendo así como se relaciona el movimiento del sistema de referencia que se usa para mover a nuestro robot con el sistema de referencia de nuestra cámara desde donde sacamos el objetivo que deseamos [16].

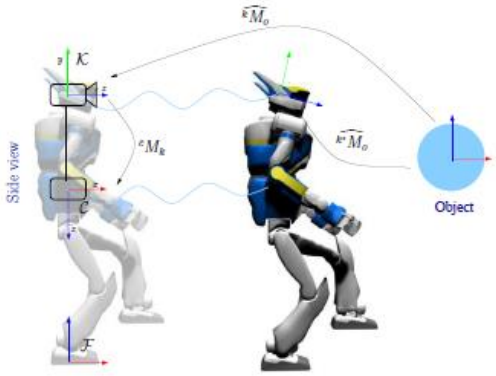

Figura 22: Ilustración del movimiento del robot con sus distintos sistemas de referencia. Fuente: Dune, C.; Stasse, O.; Yoshida, E; Herdt, A; Wieber, P-B.; Marchand, E, Visual Servoing for Dynamic Walking, IEEE/RSJ International Conference on Intelligent Robots and System (2010)

Otros enfoques algo más simples, pero más cercanos al trabajo que deseamos realizar en nuestro estudio, pueden ser los estudios generales para marcha con robots humanoides que hay en la actualidad, destacando algunos por realizar un sistema de evitación de obstáculos funcional mediante el mismo sistema de control visual [17], o por realizar compensaciones en el movimiento del robot [18]. De estos artículos referenciados es destacable el hecho de que, para realizar el sistema, se ha utilizado un robot NAO como el que utilizaremos en este trabajo, demostrando que se puede utilizar para tareas de control de marcha haciendo uso de visión artificial.

Sabiendo esto, vemos que podemos basarnos en muchos de los datos aportados por estos artículos para realizar nuestro trabajo, pudiendo obtener de estos varias transformaciones y datos interesantes de nuestro robot que nos ayudaran a implementar un método de control propio que vamos a realizar con las herramientas de las que disponemos en el laboratorio.

Antes de acabar este punto, cabe destacar una investigación en concreto donde se implementó un sistema de control de cuerpo entero mediante control basado en visión, el cual era capaz de integrar cualquier pose de todas las extremidades del robot con el que se trabajaba, que es el Romeo (ver Fig. 23). Para realizar este trabajo, se utilizó un control visual formulado como un problema de optimización cuadrática [19].

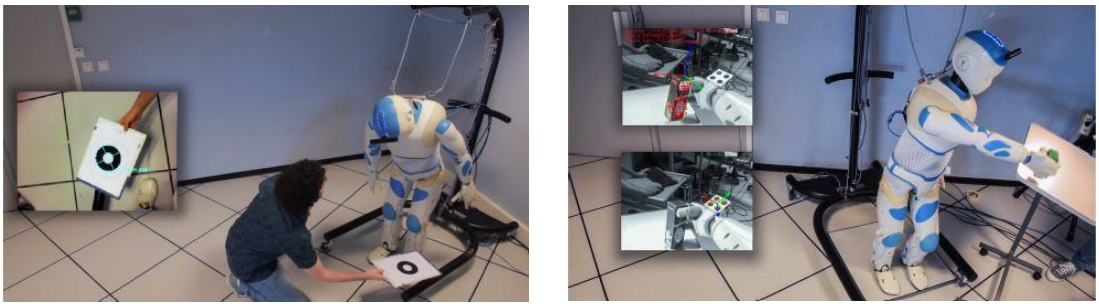

Figura 23: Robot Romeo controlando la posición de distintas partes de su cuerpo mediante control visual. Fuente: Agravante, Don; Claudio, Giovanni; Spindler, Fabien; Chaumette, François, Visual servoing in an optimization framework for the whole-body control of humanoid robots, EEE Robotics and Automation Letters, 2(2) (2017): 608-615

Una vez hemos realizado este repaso a los distintos métodos de control visual aplicados en algunos robots humanoides, podemos pasar ya a hablar sobre los objetivos que deseamos cumplir en este trabajo, utilizando para ello algunos de los métodos que se han ido exponiendo en este estado del arte, y además explicando más en profundidad dichos métodos, y tratando de adaptar éstos a nuestro entorno de trabajo, explicando cómo realizamos dicha adaptación en nuestro robot NAO.

1.3. Objetivos

Habiendo estructurado ya el trabajo, y habiendo hablado sobre la historia y avances más significativos de los sistemas que queremos implementar en nuestro robot, podemos pasar a hablar de los objetivos que nos hemos planteado para este trabajo, tratando con estos de tocar todos los temas de control de marcha con visión que nos sean posibles, e intentando así que podamos tener una idea general amplia y válida sobre dichos sistemas y su funcionamiento, tratando para ello de crear un sistema lo más avanzado posible y explicando paso a paso las teorías que vamos aplicando y cómo vamos creando nuestro entorno con los métodos y herramientas de los que disponemos.

Así, podremos ver que, con esta investigación, lo que deseamos es lograr cumplir los siguientes objetivos para abarcar el mayor número de conocimientos posibles sobre los temas de los que vamos a tratar:

- Comprender el funcionamiento de los métodos de control basado en imagen que queremos desarrollar en este trabajo, hablando de su funcionamiento para ello de forma más detallada, viendo su fundamento teórico y planteando cómo podemos aplicar todas estas teorías a un entorno real.
- Aprender a utilizar el robot NAO con el que trabajaremos de manera lo más adecuada posible, viendo para ello la forma de ponerlo en marcha, hacer que mueva sus articulaciones, haciendo especial hincapié en investigar cómo vamos a hacer que camine, la manera en la que vamos a obtener las imágenes de sus cámaras y con qué librerías y sistemas informáticos vamos a procesar éstas, y finalmente viendo cuales son los sistemas de referencia con los que nuestro robot trabaja para realizar las transformaciones que nos sean necesarias a la hora de moverlo.
- Implementar métodos de control basado en imagen para los sistemas que hemos investigado en nuestro robot, empezando inicialmente por un sistema de ver y avanzar simple para comprender en primer lugar cómo reaccionará el robot a las órdenes que vaya atendiendo en función de lo que vaya leyendo por su procesamiento de imagen, y luego tratando de implementar un controlador visual basado en imagen más complejo que sea capaz de hacer un seguimiento lo más preciso que le sea posible de un objetivo que le marquemos.

Una vez hayamos analizado todos estos puntos y hayamos obtenido resultados de estos, podremos llegar a una serie de conclusiones útiles sobre el control visual basado en imagen para el guiado de marcha de robots humanoides, conociendo para ello la historia y distintos avances de este tipo de sistemas, y además teniendo datos reales de un entorno propio con los cuales podremos plantear ciertos puntos destacables sobre el funcionamiento de este tipo de métodos, y además podremos plantear ciertos puntos a partir de los cuales poder realizar nuevas investigaciones en un futuro basándose en el control que hemos realizado para nuestra máquina.

2. Metodología: bases teóricas y técnicas del desarrollo

En este apartado trataremos las bases teóricas sobre las que nos basaremos a la hora de desarrollar las soluciones necesarias para nuestro proyecto, tratando con éstas de cumplir los objetivos que hemos marcado inicialmente para el desarrollo.

Sabiendo esto, podemos ver que, en primer lugar, hablaremos más a fondo sobre el control basado en visión artificial, viendo para ello las bases matemáticas del control visual basado en imagen, y tratando más a fondo sobre ciertos aspectos de este tipo de controladores para saber más a fondo cómo estos van a afectar al movimiento del robot.

Tras esto, pasaremos a hablar sobre el NAO, viendo algunas características técnicas generales sobre éste, para más adelante hablar sobre sus sistemas de cámara, sus métodos de marcha bípeda en los cuales nos basaremos para realizar el control, y finalmente hablaremos sobre sus sistemas de referencia para conocer qué transformaciones deberemos de utilizar para poder realizar el control de marcha de manera adecuada.

Como ya veremos más adelante, una vez hayamos tratado todos estos puntos, podremos pasar a hablar de las experimentaciones que hemos realizado, viendo sobre éstas cómo hemos implementado toda la teoría que trataremos en este apartado a nuestro sistema robótico, y más tarde viendo los resultados que hemos obtenido con nuestro sistema, y comparando dichos resultados con otros desarrollos para conocer la bondad de los datos que vamos obteniendo para realizar nuevas experimentaciones en un futuro.

2.1. Visión artificial, métodos implementados

En este primer apartado sobre la metodología, hablaremos sobre los métodos de control por visión que hemos implementado en nuestra investigación, hablando para ello sobre cómo detectamos las marcas de referencia que deseamos para nuestro robot, sobre los modelos ópticos en los que nos basaremos para obtener datos sobre la imagen que queremos tratar, y finalmente sobre la base matemática de los controladores que queremos implementar en nuestro desarrollo, que en este caso serán los de ver y mover, y los de control visual basado en imagen. Cabe destacar que el primer caso no tendrá un gran desarrollo matemático, puesto que ese tipo de control es muy básico, por lo que en este apartado trataremos sobre todo con la implementación del método de control basado en imagen, viendo cómo adaptar la información que nos llega de la imagen para obtener los datos necesarios para mover a nuestro robot a la zona que deseamos

2.1.1. Métodos de detección de regiones de interés para el controlador

En este primer apartado, trataremos de explicar, desde el punto de vista de la visión artificial, algunas de las técnicas que se han utilizado durante este trabajo para estimar regiones de interés para tomar éstas de referencia a la hora de realizar el control de nuestro sistema mediante los métodos que iremos viendo más adelante. Así, veremos la forma de estimar regiones mediante la detección de regiones simples para detectar los patrones de cuatro puntos del control basado en imagen, y también los métodos que podemos implementar para detectar códigos Qr para la implementación del método ver y mover.

En primer lugar, veremos cómo funciona la detección de regiones simples para nuestro desarrollo. Veremos que simplemente lo que haremos será detectar si ciertos píxeles de una imagen que tratamos se encuentran conexos entre ellos, es decir, si sus bordes son colindantes, y luego veremos dentro de los píxeles conexos que queremos detectar, si el nivel de gris con el que tratamos se encuentra dentro de un umbral por el cual consideramos que nos encontramos en todo momento en la misma región (ver Fig. 24). Así, de forma muy simple podemos estimar las regiones simples de nuestro sistema, acotando estas a los píxeles conexos que tengan el mismo nivel de gris en un entorno acotado.

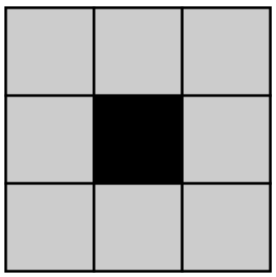

Figura 24: Ejemplo de cuatro píxeles dentro de una misma región debido a su conectividad e igualdad en nivel de gris. Fuente: Wikipedia. https://en.wikipedia.org/wiki/Pixel_connectivity#/media/File:Sasiedztwa_4_8.svg (visitado el 13/05/2019)

Como observaremos más adelante, con esta técnica simple podremos realizar la estimación de cuatro regiones simples para cumplir con el patrón de cuatro puntos que necesitaremos para el control basado en posición, obteniendo las regiones de las que queremos obtener la referencia, y más tarde calculando su centro de gravedad mediante el código que implementemos para considerarlo como el punto mediante el cual obtendremos las referencias necesarias para guiar a nuestro robot con el método de control que vamos a implementar.

Conociendo el método que hemos utilizado para las regiones simples, podemos pasar a ver cómo detectaremos los códigos Qr para el método de ver y mover, pues esta implementación tiene más base teórica en la que debemos de basarnos para obtener la posición que deseamos para nuestro robot en cada caso.

Aunque pueda parecer una tarea más compleja que la realizada anteriormente, la detección de un código de este tipo tampoco es difícil, pudiendo realizar su detección de forma bastante sencilla y efectiva. Así, podemos ver el método que utilizaremos (ver Fig. 25), como un simple escáner, el cual se encargará de escanear de la imagen que le pasemos, el código en cuestión, para luego sacar de este los muestreos de negro y blanco que se encuentra en él para luego darle valores de información a cada bloque de blanco y negro según la normativa que sigamos, y más tarde, interpretar los valores que vayamos obteniendo de esa información, para considerar si esa secuenciación de intensidades de color la podemos considerar como un código Qr con el que trabajar. [20]

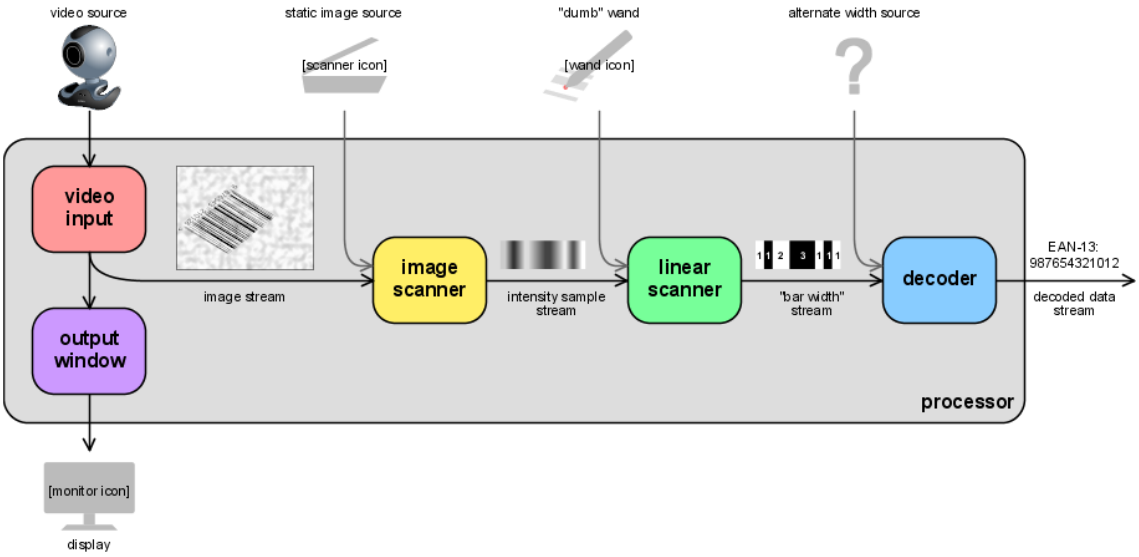

Figura 25: Esquema del proceso de detección utilizado para nuestro método. Fuente: ZBar bar code reader, http://zbar.sourceforge.net/about.html (visitado el 13/05/2019)

Una vez que ya conocemos todos los métodos que vamos a utilizar de tratado de imagen para detectar las referencias que utilizaremos en nuestro trabajo, podemos pasar a hablar sobre los parámetros de la cámara y cómo influyen estos en el tratamiento de nuestra imagen, para más tarde ya entrar en materia de forma completa sobre el controlador más complejos que realizaremos en el trabajo, viendo de este sus características principales para entender cómo funciona su base teórica.

2.1.2. Parámetros de interés de la cámara: el modelo Pinhole

Una vez hemos analizado las referencias que vamos a tomar en la imagen para su procesado, podemos pasar a hablar de los parámetros que nos serán de interés para obtener información sobre la cámara que utilicemos. Para ello, nos basaremos en la teoría expuesta en el **modelo Pinhole**.

Este modelo de cámara se basa en considerar que la cámara que utilizamos tiene una anchura de lente despreciable, en la cual encontramos una pequeña apertura por la que pasan todos los rayos de luz, a partir de los cuales estimaremos nuestra imagen. Sabiendo esto, hay ciertos elementos de este modelo que son importantes y se deben destacar para entenderlos (ver Fig. 26), los cuales son:

- **Punto focal** (centro de la cámara, o centro óptico): es el punto por el que pasan todos los rayos que componen nuestra imagen.
- **Plano de la imagen**: plano en el que se forma la imagen que nos llega a partir de los rayos dentro de la cámara, donde se proyectan las formas 3D en el plano 2D. El punto focal no se encuentra contenido en este plano, sino que se encuentra apartado de él.
- **Eje principal:** eje que pasa por el punto focal que es perpendicular al plano de la imagen. A través de él construimos todos los sistemas de referencia que nos serán necesarios en nuestro desarrollo.
- **Distancia focal**: es la distancia entre el punto focal y el plano de la imagen.
- **Referencia de la cámara**: es la base de coordenadas en la cámara que tiene el punto focal como origen y como eje principal el Z.
- **Referencia del mundo**: un sistema de coordenadas fijo donde cualquier punto 3D puede ser representado [21].

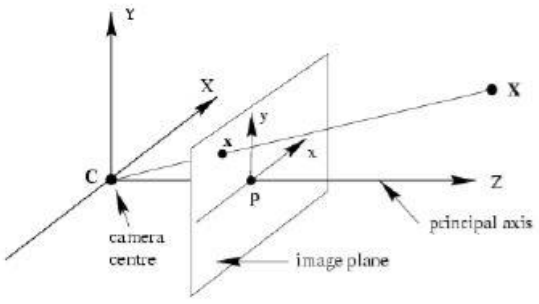

Figura 26: Representación de los datos principales del modelo Pinhole. Fuente: Monasse, Pascal; Morel, Jean-Michel; Tang, Zhongwei, Epipolar rectification, (2019)

A partir de estos puntos, podemos plantear el modelo de cámara con el que vamos a trabajar. Además, a partir de este mismo modelo, se derivan algunas propiedades de la cámara que también son destacables dentro de cualquier investigación con este modelo, los cuales son llamados parámetros intrínsecos, ya que se basan en propiedades de la misma cámara sin tener en cuenta factores externos a ésta, y que pueden afectar a la toma de la imagen que obtengamos al utilizar una cámara de este tipo.

Los parámetros intrínsecos con los que trabajaremos, y que son más relevantes a la hora de trabajar con este modelo, son los siguientes:

- **Distancia focal**: tiene el mismo significado que la explicada para el modelo pinhole anteriormente comentado, marcando la distantica entre el centro óptico y el plano de la cámara.
- **Coordenadas del centro óptico**: nos servirá para saber cómo se relaciona la posición de cualquier representación en el plano de la cámara con el espacio 3D que ésta capta en todo momento.
- **Número de píxeles por milímetro**: nos indica la relación entre el espacio real que se quiere recrear y la representación que se realiza, relacionando las medidas del mundo real con las que se utilizan en forma de píxel en la imagen.
- **Valor de inclinación**: como su nombre indica, nos indica el grado de inclinación con respecto al plano externo que tiene nuestra cámara.

Con dichos valores, podemos ver que se puede generar la matriz de parámetros intrínsecos, la cual es la encargada de transformar las coordenadas 3D de la cámara a coordenadas 2D homogéneas de la imagen. Así, podemos ver que dicha matriz tendría la siguiente forma según los parámetros que hemos destacado:

$$K = \begin{pmatrix} fm_x & skew & u_0 \\ 0 & fm_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} \quad (1)$$

De esta matriz cabe destacar que la multiplicación de la distancia focal con el número de píxeles por milímetro nos dará valores de píxeles cuadrados, relacionando la cantidad de píxeles que podemos encontrar en nuestra imagen en función de la distancia focal que tenga nuestra cámara basada en el modelo Pinhole [22].

Como podemos observar, esta matriz la utilizaremos para transformar entre coordenadas del mundo real 3D y la representación 2D que creamos en el plano de nuestra cámara, conteniendo así ésta una gran cantidad de información sobre cómo se forma nuestra imagen, y cómo ésta interpreta la información que le llega del mundo real.

Conociendo todos estos datos, finalmente ya podemos pasar a ver cómo funciona nuestro método de control más importante, viendo para ello cómo trabaja tanto con la información de este modelo, como con la de las regiones que reconocemos y los datos de control del mismo robot, y tratando de explicar paso a paso el significado matemático y teórico de cada una de las operaciones que se realizan con estos datos.

2.1.3. Control visual basado en imagen: fundamentos y base matemática

Una vez que hemos visto las nociones básicas sobre cómo vamos a reconocer los patrones que encontremos en nuestro entorno, y además qué modelo matemático vamos a utilizar para plantear los parámetros necesarios de nuestra cámara, podemos pasar a ver el funcionamiento del control visual basado en imagen, viendo para ello toda la base matemática en la que nos basaremos para realizar dicho control.

El objetivo de cualquier control basado en imagen es el de minimizar el error implícito de cualquier bucle de control, que en nuestro caso se puede definir de la siguiente manera matemáticamente:

$$e(t) = \mathbf{s}(m(t), a) - \mathbf{s}^* \tag{2}$$

Donde podemos ver que **s** son las características actuales que tomamos de la imagen, y **s*** son las características que deseamos obtener finalmente en el control para reducir el error a cero. Cabe destacar que la variable *m* serán una serie de medidas de la imagen, como las coordenadas de puntos de interés, y la variable *a* representa los parámetros que pueden suponer información adicional importante sobre el sistema, como pueden ser parámetros intrínsecos de la cámara como vimos en el punto anterior.

Así, sabiendo que partimos de un conjunto de características visuales **s**, debemos buscar la relación cinemática entre el movimiento de la cámara y el movimiento de las características visuales para reducir el error que representa la diferencia entre referencias. Sabiendo esto, podemos ver que la relación entre la velocidad espacial de la cámara denotada como $v_c = (v_c, \omega_c)$, siendo v_c la velocidad lineal del origen de la referencia de la cámara y ω_c la velocidad angular de esta misma referencia, la relación de estos valores con la velocidad de las características $\dot{\mathbf{s}}$ tendrá la siguiente forma:

$$\dot{\mathbf{s}} = \mathbf{L}_s v_c \tag{3}$$

En dicha relación, observamos la introducción de la variable $\mathbf{L}_s$, conocida como la matriz de interacción, encargada de relacionar los valores de **s** con la velocidad de la cámara de nuestro robot. Así, combinando las ecuaciones (2) y (3), podemos obtener la relación entre la velocidad de la cámara y la variación en el tiempo de nuestro error de la siguiente forma:

$$\dot{\mathbf{e}} = \mathbf{L}_e \, v_c \tag{4}$$

Donde $\mathbf{L}_e = \mathbf{L}_s$. Considerando las velocidades de la cámara como la entrada del controlador de nuestro robot, y si queremos asegurarnos de tener un decremento exponencial de nuestro error, podemos plantear la anterior formula de la siguiente manera:

(5)

$$v_c = -\lambda \mathbf{L}_e^+ \mathbf{e}$$

Donde $\mathbf{L}_e^+$ es la matriz pseudo inversa de Moore-Penrose de la matriz de interacción obtenida anteriormente, ya que la matriz de interacción no es siempre cuadrada, y en esos casos debemos de poder obtener su inversa de alguna forma. En los sistemas de control basados en visión reales, es imposible conocer perfectamente las matrices de interacción y sus inversas, por lo que se debe realizar aproximaciones sobre estas matrices. Conociendo estas aproximaciones, vemos que nuestra ley de control final quedaría de la siguiente forma finalmente:

$$v_c = -\lambda \widehat{\mathbf{L}_e^+} \mathbf{e} \quad (6)$$

Donde la matriz $\widehat{\mathbf{L}_e^+}$ sería nuestra aproximación de la matriz de interacción que deseamos utilizar [9]. Dicha aproximación se realiza debido a que la matriz depende de la distancia de la cámara al objeto seguido, considerando en muchas ocasiones que el parámetro de profundidad en la matriz es constante. Además, los parámetros intrínsecos no se calculan con exactitud, sino que también son aproximados [11]. Conociendo esto, podemos ver que se realizan tres aproximaciones principales normalmente, las cuales suelen ser:

- Suponer el conocimiento en cada iteración de la distancia entre la cámara y los puntos característicos ya sea por estimación o porque se miden en tiempo real, teniendo así que $\widehat{\mathbf{L}_s} = \mathbf{L}_s$. [9]
- Calcular la matriz de interacción para la posición final deseada, teniendo así que la matriz de interacción es constante, evitando la necesidad de estimar parámetros 3D durante la ejecución, y obteniendo que $\widehat{\mathbf{L}_s} = \mathbf{L}_{s*}$. [9]
- Realizar una combinación de ambas aproximaciones anteriores, requiriendo para ello la estimación o cálculo de la distancia cámara a objeto en cada iteración, y además dando buenos resultados en grandes desplazamientos, teniendo la aproximación la forma de $\widehat{\mathbf{L}_s} = \frac{1}{2}(\mathbf{L}_s + \mathbf{L}_{s*})$. [9]

Como podemos imaginar, cada aproximación dará unos resultados más o menos exactos, y además cada una supondrá unos problemas mayores o menores dependiendo de las variables que se deban computar en cada caso durante la ejecución del bucle de control de nuestro sistema.

Sabiendo todo esto, podemos pasar a ver la base de toda la teoría que hemos desarrollado, que es el desarrollo y forma de nuestra matriz de interacción, a partir de la cual podremos realizar los cálculos necesarios para estimar las velocidades que requerirán nuestras referencias, y finalmente nuestro robot dentro de su entorno de trabajo.

Para calcular nuestra matriz de interacción, podemos ver que para un punto en tres dimensiones con coordenadas X = (X, Y, Z) en la referencia de nuestra cámara, éste proyectará un punto en la imagen de dos dimensiones con coordenadas x = (*x*, *y*) con la siguiente forma según el modelo Pinhole:

$$\begin{cases} x = \dfrac{X}{Z} = (u - c_u)/f\alpha \\ y = \dfrac{Y}{Z} = (v - c_v)/f \end{cases} \tag{7}$$

Vemos que tenemos los valores de $m = (u, v)$ como las coordenadas del punto de la imagen expresadas en unidades de píxel, y los valores de $a = (c_u,\ c_v, f, \alpha)$, siendo los parámetros intrínsecos que necesitamos de nuestra cámara, representando c_u y c_v las coordenadas del punto focal, f la distancia focal, y α el ratio de las dimensiones de los píxeles. Para plantear la matriz asumiremos que las coordenadas del punto x que hemos tomado representarán las características necesarias para realizar nuestro controlador final.

Si realizamos la derivada sobre el tiempo de las ecuaciones que encontramos en (7), podemos ver que obtendremos las siguientes ecuaciones que representarán la velocidad de nuestro punto en la imagen:

$$\begin{cases} \dot{x} = \dfrac{\dot{X}}{Z} - \dfrac{X\dot{Z}}{Z^2} = \dfrac{(\dot{X} - x\dot{Z})}{Z} \\ \dot{y} = \dfrac{\dot{Y}}{Z} - \dfrac{Y\dot{Z}}{Z^2} = \dfrac{(\dot{Y} - y\dot{Z})}{Z} \end{cases} \tag{8}$$

Si realizamos la relación entre la velocidad del punto 3D con la de la velocidad espacial de la cámara de la siguiente forma:

$$\dot{X} = -v_c - \omega_c \times X \Leftrightarrow \begin{cases} \dot{X} = -v_x - \omega_y Z + \omega_z Y \\ \dot{Y} = -v_y - \omega_z X + \omega_x Z \\ \dot{Z} = -v_z - \omega_x Y + \omega_y X \end{cases} \tag{9}$$

Así, insertando la formula (9) en (8), agrupando términos, y usando la formula (7), podemos obtener lo siguiente [4]:

$$\begin{cases} \dot{x} = \dfrac{-v_x}{Z} + \dfrac{xv_z}{Z} + xy\omega_x - (1 + x^2)\omega_y + y\omega_z \\ \dot{y} = \dfrac{-v_y}{Z} + \dfrac{yv_z}{Z} + (1 + y^2)\omega_x - xy\omega_y - x\omega_z \end{cases} \tag{10}$$

Lo cual, como ya sabemos por la ecuación (3), es nuestra ley de control en la que nos basaremos. Sabiendo esto, podemos ver que la matriz de interacción para esta única característica tendrá la siguiente forma final [4]:

$$L_s = \begin{pmatrix} \frac{-1}{Z} & 0 & \frac{x}{Z} & xy & -(1+x^2) & y \\ 0 & \frac{-1}{Z} & \frac{y}{Z} & 1+y^2 & -xy & -x \end{pmatrix} \tag{11}$$

Cabe destacar que, aunque como veremos más adelante, en nuestro robot tan solo utilizaremos tres grados de libertad de todos los que tiene, el realizar una buena estimación de la cantidad de características que nos harán falta para un robot es importante. Por ejemplo, para un robot de seis grados de libertad, requeriríamos de al menos tres puntos para la matriz de interacción (es necesario que $k \geq 6$). Si simplemente realizamos nuestra matriz con estos tres puntos, veremos que esta puede tener algunas configuraciones en cuanto sus valores comiencen a cambiar que harán que encontremos configuraciones singulares de la matriz. Además, existirán posiciones distintas para las que el error final valga cero, siendo imposible diferenciar dichas posiciones [4]. Teniendo en cuenta este ejemplo, podemos ver que es importante el determinar el número de puntos que tendremos en cuenta a la hora de obtener nuestra matriz de interacción.

Conociendo ya todos estos datos sobre la base matemática de nuestro bucle de control, y además habiendo revisado previamente el modelo matemático que usaremos para nuestra cámara y la forma en la que detectaremos nuestros puntos de interés, tendremos ya un buen análisis sobre la metodología a nivel de visión que utilizaremos dentro de nuestro sistema, pudiendo pasar ya a hablar sobre nuestro robot y sus características, destacando las partes que más nos interesen de éste y cómo vamos a acoplar dichas partes al sistema que queremos plantear para este trabajo.

2.2. Robot NAO: especificaciones técnicas para nuestro sistema

En este apartado de nuestro trabajo, pasaremos a hablar sobre algunas de las características más destacables del robot con el que vamos a trabajar, viendo para ello en primer lugar los puntos más generales y destacables como pueden ser sus medidas, sus grados de libertad o sus cadenas cinemáticas, y luego pasando a puntos más específicos de éste, hablando en primer lugar sobre su método de locomoción bípeda y cómo se implementa, luego viendo las cámaras de este robot, su funcionamiento y las propiedades de éstas, y finalmente tratando de forma muy breve los sistemas de referencia con los que trabaja este robot, y que serán muy importantes a la hora de realizar las transformaciones que nos sean necesarias para hacer que la velocidad que obtengamos de nuestra cámara la tome el robot de forma adecuada.

2.2.1. Conceptos y datos genéricos de nuestro robot

El robot NAO (ver Fig. 27) es un robot humanoide totalmente programable. Se comenzó a fabricar en 2008 desarrollado por la empresa Softbank Robotics, y ya ha tenido seis versiones hasta llegar al modelo actual v6, que será con el que trabajemos en este proyecto [23].

Figura 27: Robot NAO en el entorno de trabajo

Nuestro robot tendrá unas medidas en milímetros de 547 mm de alto, 275 mm de ancho y de 311mm de profundidad con los brazos en paralelo al suelo en la versión del robot con la que vamos a trabajar (ver Fig. 28). Además, dicha máquina tendrá un peso de 5.4 kilogramos.

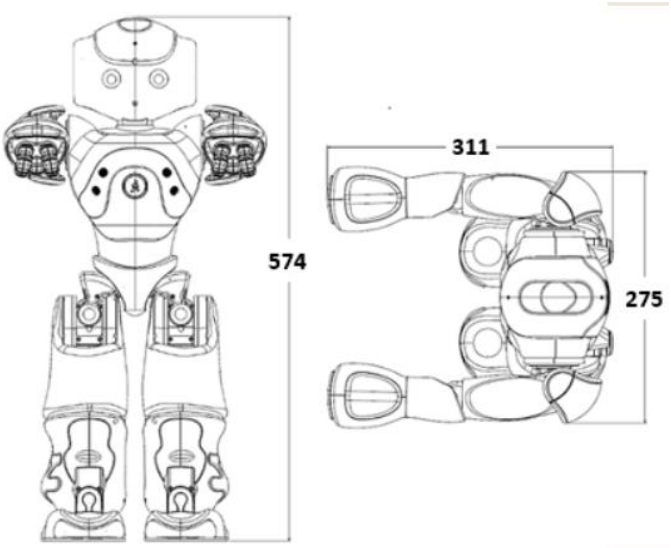

Figura 28: medidas generales de nuestro robot. Fuente: Aldebaran documentation. http://doc.aldebaran.com/2-1/family/robots/dimensions_robot.html (visitado el 16/05/2019)

El robot cuenta con dos cámaras, cuatro micrófonos, sensores táctiles en la cabeza, manos y pies, dos sensores de ultrasonidos en el pecho, ocho sensores de presión, un acelerómetro y un giroscopio, además de también incluir 53 Leds RGB distribuidos en el botón de su pecho y en sus ojos, y de un sintetizador de voz con dos altavoces [23]. Más adelante trataremos sobre los sensores que más nos interesen de los que hemos destacado, viendo cómo estos influirán en la forma de programar nuestro robot y la información de la que estos nos proveerán.

En cuanto al apartado computacional de nuestro robot, podemos ver que éste tiene incluidos en él una CPU ATOM Z530 de 1.6 GHz junto con un gigabyte de memoria RAM, dos de memoria flash, y una Micro SD de 8 GB [24]. Con todos estos componentes, en cuanto al sistema operativo podemos ver que este corre uno propio basado en la distribución de GNU/Linux de Gentoo, de nombre **OpenNao**. Dentro de este sistema, el software principal que se ejecuta es el **NAOqi**, siendo éste el encargado de crear comportamientos en nuestro robot, que no son más que módulos de programas y métodos gestionados por este software [25]. Cabe destacar que dicho software lo podremos utilizar además en nuestra computadora propia, tanto en forma de librería para programar en distintos lenguajes como pueden ser **C++** o **Python**, como haciendo uso del programa de interfaz gráfico de programación **Choreographe**, el cual nos ofrece un lenguaje de programación por bloques para nuestro robot basado en NAOqi y Python.

Finalmente, en cuanto a los datos genéricos de nuestro robot, cabe destacar sus cadenas cinemáticas, pues es lo que queremos controlar finalmente de éste. Así, en la Fig. 29 podemos ver que estas cadenas estarán distribuidas de la siguiente forma dentro de nuestro robot:

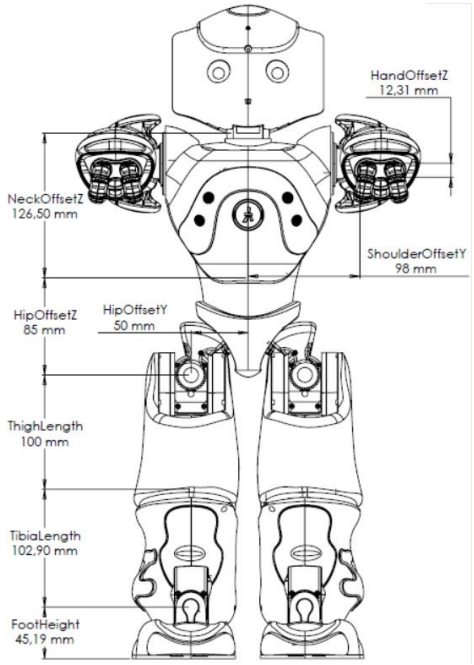

Figura 29: posiciones de las distintas cadenas cinemáticas que encontramos en nuestro robot. Fuente: Aldebaran Documentation. http://doc.aldebaran.com/2-1/family/robots/links_robot.html (visitado el 16/05/2019)

Viendo la Fig. 29, podemos deducir que nuestro robot se compone de seis partes diferenciadas, las cuales podremos mover, y que son la cabeza, los dos brazos, las dos piernas, y el torso. A excepción del torso, cada una de estas partes son consideradas cadenas cinemáticas, y podemos ver en la Fig. 30 que en éstas intervendrán las siguientes articulaciones, en función de cuál de ellas queramos mover:

	Cadenas cinemáticas del robot				
Articulaciones	Cabeza	Brazo izquierdo	Brazo derecho	Pierna izquierda	Pierna derecha
0	HeadYaw	LShoulderPitch	LHipYawPitch	RHipYawPitch	RShoulderPitch
1	HeadPitch	LShoulderRoll	LHipRoll	RHipRoll	RShoulderRoll
2		LElbowYaw	LHipPitch	RHipPitch	RElbowYaw
3		LElbowRoll	LKneePitch	RKneePitch	RElbowRoll
4		LWristYaw	LAnklePitch	RAnklePitch	RWristYaw
5		LHand	RAnkleRoll	LAnkleRoll	RHand

Figura 30: Articulaciones del NAO en sus respectivas cadenas cinemáticas. Fuente: Aldebaran Documentation. http://doc.aldebaran.com/2-1/family/robots/bodyparts.html (visitado el 16/05/2019)

Así, podemos observar que nuestro robot cuenta con 26 articulaciones, repartidas cada una en una cadena cinemática distinta que servirá para realizar distintos movimientos para nuestro mecanismo. Cabe destacar que en nuestro trabajo las cadenas que más destacarán serán las de las piernas, pues van a ser las que nos permitirán mover a nuestro robot.

Una vez conocemos todos estos datos iniciales sobre el robot con el que vamos a trabajar, podemos pasar a tratar algunos puntos más específicos sobre éste, empezando en primer lugar a tratar de explicar cómo realizamos la locomoción simple con nuestra máquina, y luego viendo algunas características de las cámaras que utilizaremos en nuestro robot.

2.2.2 Programación del robot: lenguaje, sistemas y librerías utilizados

Antes de pasar a hablar de puntos más concretos de nuestro robot, también caben destacar ciertos aspectos sobre en qué bases nos hemos basado en nuestro trabajo a la hora de programar este robot, explicando brevemente porque hemos escogido las herramientas que vamos a exponer, y viendo cuales son estas paso a paso.

En primer lugar, cabe destacar que la programación de nuestro controlador en el robot la realizaremos haciendo uso del lenguaje de programación orientado a objetos C++. Esto es debido a que este lenguaje nos ofrece un punto intermedio entre el suficiente bajo nivel para trabajar con las articulaciones de nuestro robot de forma efectiva y sin tener problemas de retardos debido a la interpretación de las ordenes que le damos como nos ocurrirá con otros lenguajes como por ejemplo Python, y también tiene el suficiente alto nivel como para que podamos realizar nuestra tarea de la forma más cómoda que nos sea posible sin tener que interpretar de manera excesiva.

En cuanto al método de comunicación que utilizaremos a la hora de tratar con los distintos datos que nos van a ofrecer los sensores de nuestro robot, podemos ver que haremos uso del **Sistema Operativo Robótico** (lo mencionaremos por sus siglas en ingles **ROS** a partir de aquí). Este método es un framework para el desarrollo en robots [26] el cual se basa en un sistema de nodos que publican tópicos a los que otros nodos pueden suscribirse para recibir información, siendo los nodos cualquier sensor que tenga nuestro robot, y los tópicos la información que podamos obtener de éstos. Como podemos imaginar, existen desarrollos de este método de comunicación específicos para el robot con el que vamos a trabajar, haciendo fácil su uso en nuestro entorno.

Cabe destacar que, aunque este trabajo lo podríamos realizar directamente con el sistema NAOqi, utilizaremos este método de comunicaciones ya que nos provee de más información sobre el robot en tiempo real que el método anteriormente mencionado, haciendo más fácil así el hecho de conocer las distintas informaciones que nos va ofreciendo nuestro sistema a lo largo de su ejecución. Además, también es destacable que este método de comunicación ofrece algunas herramientas útiles a la hora de realizar el control de un robot, y tiene una alta compatibilidad con el sistema que utilizaremos de visión para nuestra máquina.

Finalmente, en cuanto a las librerías destacables que utilizaremos, podemos ver que usaremos las herramientas que se nos ofrecen en **ViSP** (Visual Servoing Platform). Esta plataforma nos ofrece un entorno de programación especifico centrado en el control visual, pudiendo así con ésta realizar todas las tareas que se nos requieren, tanto a nivel de visión, como a nivel de control dentro de nuestro robot.

Como podemos imaginar, utilizaremos esta librería debido a que nos ofrece un espacio de trabajo combinado para los dos puntos más destacados de nuestro sistema, facilitando así nuestro trabajo de gran manera al poder disponer de funcionalidades ya implementadas en esta librería. Además, también cabe destacar que dicha librería nos ofrece una alta compatibilidad con el entorno de ROS, facilitando así la comunicación de datos que tengamos que realizar desde las cámaras a cualquiera de los sistemas de nuestro robot de gran manera.

2.2.2. Locomoción básica del NAO

Sabiendo ya algunos datos básicos sobre nuestro robot, y cómo vamos a realizar la programación de éste, podemos pasar a hablar sobre cómo vamos a hacer que nuestro robot se mueva en el entorno que planteamos, viendo para ello las bases teóricas en las que se basara su movimiento, y luego viendo qué será necesario para hacer que éste se mueva con las herramientas planteadas.

En primer lugar, cabe destacar que nuestro robot es capaz de estabilizarse en su marcha debido a que usa información propia de sus sensores en las articulaciones de las piernas, haciendo que su marcha sea robusta frente a pequeños movimientos inesperados y que además la máquina absorba las oscilaciones del tren superior del robot en los planos frontales y laterales, y haciendo al robot capaz de caminar en tipos de suelos muy distintos y variados, además pudiendo cambiar el tipo de suelo en su marcha. Pese a esto, nuestro robot puede caer debido a grandes obstáculos, ya que este asume en todo momento que el suelo es más o menos plano [27].

Pero toda esta información se basa en un modelo dinámico simple de péndulo invertido (ver Fig. 31 y Fig. 32), el cual deberemos de explicar para entender de mejor forma el funcionamiento de este tipo de marcha, viendo para ello las bases cinemáticas de este modelo simple, y luego tratando de extrapolar dichas bases al funcionamiento final de nuestro robot.

Al aguantar su cuerpo por una sola pierna, las dinámicas dominantes dentro de nuestro robot pueden ser representadas por un péndulo invertido simple que conecta el pie de soporte con el centro de masas de nuestro robot.

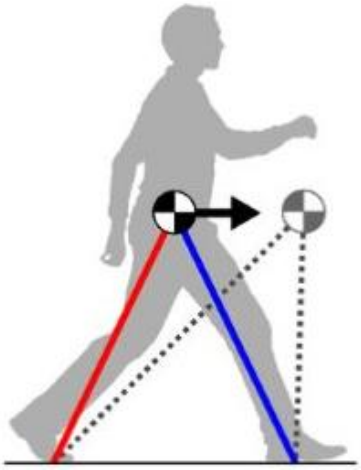

Figura 31: Péndulo invertido lineal bípedo. Fuente: van Dalen, S. J., A linear inverted pendulum walk implemented on TUlip, Student tesis, Eindhoven university of technology (2012)

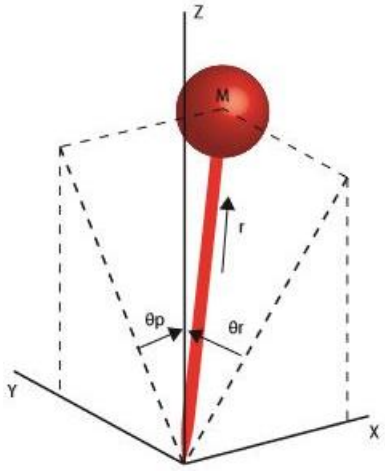

Figura 32: Péndulo invertido 3D. Fuente: van Dalen, S. J., A linear inverted pendulum walk implemented on TUlip, Student tesis, Eindhoven university of technology (2012)

La posición de dicho punto de masa $p = (x, y, z)$ es únicamente especificada por una serie de variables $q = (\theta_r, \theta_p, r)$.

$$x = r\, sen\theta_p \tag{12}$$

$$y = -r\, sen\theta_r \tag{13}$$

$$z = r\sqrt{1 - (sen\theta_r)^2 - \left(sen\theta_p\right)^2} \tag{14}$$

Siendo (τ_r, τ_p, f) el par y fuerza del actuador asociados con el estado de las variables de q, con estos datos, la ecuación de movimiento del péndulo invertido 3D en coordenadas cartesianas tendrá la siguiente forma:

$$m\begin{pmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{pmatrix} = (J^T)^{-1}\begin{pmatrix}\tau_r\\ \tau_p\\ f\end{pmatrix} + \begin{pmatrix}0\\ 0\\ -mg\end{pmatrix} \tag{15}$$

Donde m es la masa del péndulo y g es la aceleración de la gravedad. La estructura de la Jacobiana de dicho modelado tendrá la siguiente forma:

$$J = \frac{\partial p}{\partial q} = \begin{pmatrix}0 & rC_p & S_p\\ -rC_r & 0 & -S_r\\ -rC_rS_r/_D & -rC_pS_p/_D & D\end{pmatrix} \tag{16}$$

$$C_r \equiv \cos\theta_r,\ C_p \equiv \cos\theta_p,\ S_r \equiv \operatorname{sen}\theta_r,\ S_p \equiv \operatorname{sen}\theta_p,\ D \equiv \sqrt{1-(sen\theta_r)^2-\left(sen\theta_p\right)^2}$$

Para no tener que tratar con la Jacobiana inversa que aparece en (15), multiplicaremos la traspuesta de la Jacobiana desde la izquierda, de la siguiente forma:

$$m\begin{pmatrix}0 & -rC_r & -rC_rS_r/_D\\ rC_p & 0 & -rC_pS_p/_D\\ S_p & -S_r & D\end{pmatrix}\begin{pmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{pmatrix} = \begin{pmatrix}\tau_r\\ \tau_p\\ f\end{pmatrix} - mg\begin{pmatrix}-rC_rS_r\\ -rC_pS_p/_D\\ D\end{pmatrix} \tag{17}$$

Usando la primera línea de esta ecuación y multiplicando $D/_{C_r}$ obtenemos:

$$m(-rD\ddot{y} - rS_r\ddot{z}) = \frac{D}{C_r}\tau_r + rS_rmg \tag{18}$$

Si sustituimos las relaciones cinemáticas de las ecuaciones (13) y (14), obtenemos una buena ecuación que describe el comportamiento dinámico a través del eje y:

$$m(-z\ddot{y} + y\ddot{z}) = \frac{D}{C_r}\tau_r + mgy \tag{19}$$

Llevando a cabo un procedimiento similar para la segunda fila de la formula (17), podemos obtener la ecuación de la dinámica para el eje x:

$$m(z\ddot{x} - x\ddot{z}) = \frac{D}{C_p}\tau_p + mgx \tag{20}$$

Como podemos observar, con estas ecuaciones finalmente tendremos bien modelado tanto la dinámica como la cinemática de nuestro péndulo invertido simple, sirviendo éste como base para la realización de la marcha de nuestro robot bípedo [28].

Una vez conocemos todos estos datos sobre la dinámica y la cinemática del péndulo invertido en el que se va a basar la locomoción de nuestro robot, de ésta cabe destacar que, para la marcha, cada paso estará compuesto por dos fases de soporte, siendo una de éstas con las dos piernas y la otra con tan solo una (momento del modelado del péndulo invertido). La fase de soporte a dos, pues, tan solo usará un tercio del tiempo empleado en realizar el paso, teniendo una duración tanto al principio como al final del inicio de la marcha de 0.6 segundos. También cabe destacar que el robot utiliza un controlador previo para estimar los valores necesarios a la hora de calcular como va a realizar el paso, teniendo una duración éste de 0.8 segundos, y además realizando una interpolación cúbica para calcular la trayectoria que quiere que sigan las articulaciones de nuestro robot en su movimiento a la hora de dar el paso con el modelado de péndulo invertido [27].

Una vez conocidos todos estos datos, podemos pasar a hablar brevemente sobre cómo podemos mandarle la orden a nuestro robot para que inicie un comportamiento de marcha. Con nuestros paquetes de ROS específicos del robot, podemos enviarle a éste comandos específicos de marcha para que comience a moverse, teniendo dos principales destacables, que son los comandos de posición y velocidad. Como es de esperar, el primero de estos lanzará un mensaje al topic del nodo de la posición de nuestro robot para que éste la actualice a la que le pasemos por comando, iniciando para ello la macha, mientras que el segundo hará lo propio con el topic de la velocidad, cambiando ésta del reposo a la que nosotros le indiquemos hasta que cualquier otro mensaje actualice dicha velocidad.

Además, haciendo uso de los paquetes de ViSP y ROS combinados, también podemos indicarle a nuestro robot comandos de velocidad en función a vectores de referencia de velocidades que creemos en nuestro código, teniendo el código necesario para realizar estas acciones la siguiente forma:

```
vpROSRobot robot;
robot.setCmdVelTopic("/cmd_vel");
robot.init();

vpColVector v;
robot.setVelocity(vpRobot::REFERENCE_FRAME, v);
```

Figura 33: Código ViSP para mandar un comando de velocidad a nuestro robot

Como podemos observar, el código es muy simple, realizando en primer lugar una declaración del robot con el que vamos a trabajar, y luego dándole un topic al que enviar sus mensajes y otro comando para que se inicie las transferencias de mensajes al robot.

Tras esto, simplemente crearemos nuestro vector de valores de velocidad, le daremos las velocidades que deseemos pasarle a nuestro robot, y con la última función le enviaremos la velocidad deseada a nuestro robot en el topic indicado anteriormente, y además señalándole en qué sistema de referencia queremos que se envíe dicha velocidad, pudiendo enviar distintas velocidades en función de qué queremos que se mueva de nuestro robot. Como podemos observar, el método de enviar mensajes con ViSP sobre la velocidad para que nuestro robot se mueva también es muy simple, empleando esta forma de envío de datos en nuestro código final, como veremos más adelante.

Sabiendo todo esto, ya tenemos un conocimiento bastante extenso sobre el movimiento de nuestro robot y cómo ejecutarlo con las herramientas que tenemos, pudiendo pasar así a ver las características de la cámara con la que vamos a trabajar con nuestro robot, para más tarde ver sus sistemas de referencia y así tener un conocimiento conjunto lo bastante amplio para que finalmente podamos exponer cómo vamos a realizar nuestra experimentación del controlador visual en nuestro robot.

2.2.3. Sistema de cámaras del NAO y obtención de imágenes

Una vez hemos visto el funcionamiento básico del método de locomoción de nuestro robot, podemos pasar a ver cómo vamos a hacer que éste trabaje con las imágenes que le vayan llegando desde su entorno de trabajo. Para ello, en primer lugar, debemos hablar de las cámaras de nuestro robot. Estas cámaras están dispuestas en la cabeza del humanoide, estando una en la frente y otra en la boca (ver Fig. 34), y ambas tienen una resolución de 1280x960 a 30 cuadros por segundo.

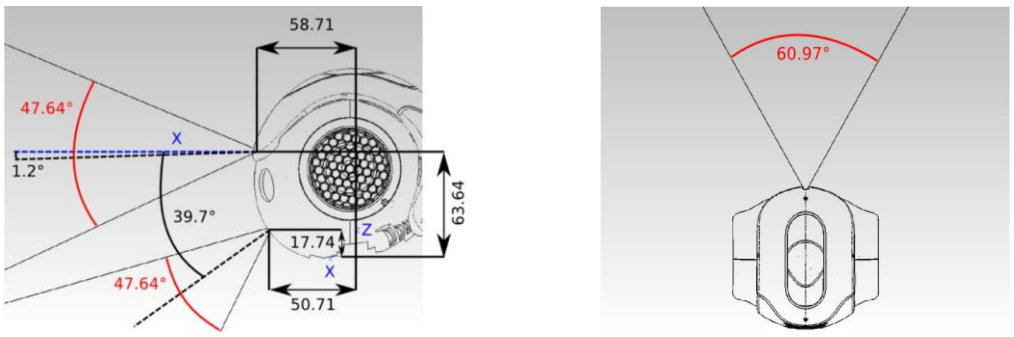

Figura 34: Ilustración en lateral y en planta de la disposición y rango de las cámaras. Fuente: Aldebaran Documentation. http://doc.aldebaran.com/2-1/family/robots/video_robot.html (visitado el 16/05/2019)

Sabiendo esto, cabe destacar que, en nuestro trabajo, tan solo utilizaremos la cámara superior de nuestro robot, pues es la que nos ofrece una visión más directa del entorno en el que vamos a trabajar al estar en una posición más ventajosa, estando esta, más elevada y adelantada que la cámara inferior.

En cuanto al tratamiento y obtención de la imagen, podemos ver que para realizar estas acciones utilizaremos los métodos de obtención que nos provee ViSP para sistemas con base en ROS, pudiendo ver un ejemplo de cómo realizar esta tarea de la siguiente forma:

```
vpROSGrabber g;
g.setImageTopic("/nao_robot/camera/top/camera/image_raw");
g.setCameraInfoTopic("/nao_robot/camera/top/camera/camera_info");
g.setRectify(true);
g.open(argc, argv);
g.acquire(I);
```

Figura 35: Código básico para la obtención de imagen mediante ViSP y ROS

Como podemos observar, con las dos primeras funciones nos encargaremos de obtener el topic de ROS a partir del cual obtendremos las imágenes e información de nuestra cámara, para más tarde en las tres siguientes funciones rectificar nuestra imagen y luego abrirla. Como podemos ver, tenemos así un método de tratamiento de imágenes muy simple y efectivo con el que podremos trabajar.

2.2.4. Sistemas de referencia de nuestro robot

Finalmente, una vez conocemos las propiedades de la cámara de nuestro robot, y cómo va a caminar, junto con sus datos más genéricos, podemos pasar a ver los sistemas de referencia con los que nuestro robot va a trabajar para más tarde saber cómo vamos a utilizar éstos en nuestro sistema. En primer lugar, podemos ver en la Fig. 36 todos los sistemas de referencia que existen dentro de nuestro robot referidos a sus articulaciones.

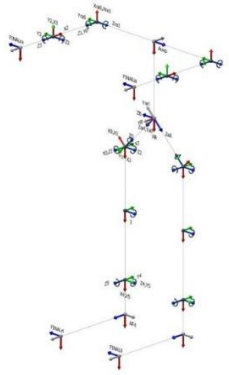

Figura 36: Sistemas de referencia de nuestro robot para cada una de sus articulaciones. Fuente: ResearchGate. https://www.researchgate.net/figure/Figura-2-Diagrama-Cinematico-Robot-NAO_fig2_303592455 (visitado el 16/05/2019)

Aunque para otros trabajos dichas referencias puedan resultar interesantes, en nuestro caso no lo serán tanto, pues nuestro trabajo se basará más en otras respecto a las cuales nuestro robot tendrá que moverse. Tres de estas referencias que sí que nos proporcionan más datos en nuestro trabajo son las referidas al torso, el espacio de trabajo externo, y al mismo robot. Dichas referencias tienen las siguientes características:

- La del torso, como su nombre indica, se encuentra en esta zona del robot, cambiando su posición cuando el robot camina y su orientación cuando este se inclina. Esta referencia es útil a la hora de realizar tareas muy locales, que estén en concordancia con la orientación del torso.
- La del espacio de trabajo es un origen fijado que nunca se altera, se va dejando atrás conforme el robot camina y tendrá su orientación en el eje Z distinta al robot en cuanto este gire. Este espacio es útil para cálculos que requieren un espacio externo de referencia.
- Finalmente, el sistema de referencia del robot se encuentra entre los dos pies de éste (ver Fig. 37), siendo su eje Z la proyección del eje Z del sistema de referencia del torso en el suelo. Este espacio es útil ya que nos da una referencia egocéntrica natural, la cual nos facilita ciertas acciones como la del caminar del robot.

Como podemos imaginar, la referencia que más útil nos será de las tres destacadas anteriormente será la del robot, pues es la que toman como referencia las librerías del robot a la hora de pasarle comandos de locomoción a éste.

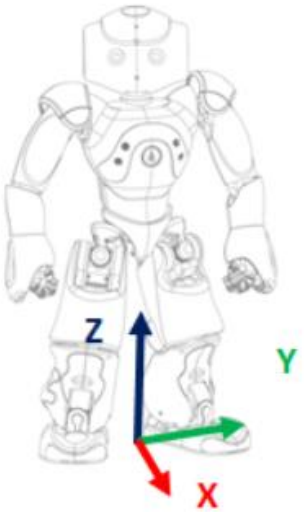

Figura 37: Frame de referencia del robot. Fuente: Aldebaran Documentation. http://doc.aldebaran.com/2-1/naoqi/motion/control-cartesian.html (visitado el 16/05/2019)

Finalmente, también deberemos de hablar sobre los sistemas de referencia de nuestro robot situados en las cámaras de éste. Dichos sistemas de referencia estarán orientados de forma distinta al sistema anterior, teniendo unas rotaciones sobre los ejes Z e Y que afectarán a la orientación de dichos sistemas de referencia, además de estar localizados en lo más alto de nuestro robot y, además, más adelantados que el anterior, el cual se encontraba a la profundidad del centro de gravedad de nuestro robot. La Fig. 38 muestra dicho sistema de referencia para la cámara superior, que será la que utilizaremos.

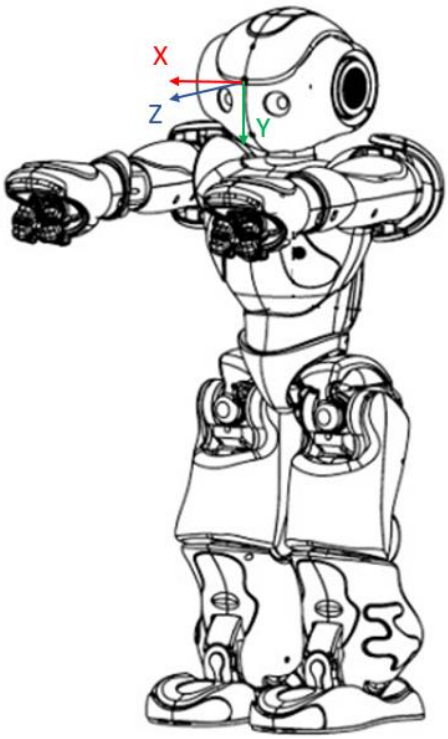

Figura 38: Dibujo del robot con el sistema de referencia utilizado en nuestra cámara a la hora de trabajar. Fuente: SciELO. http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0123-77992014000200007 (visitado el 17/05/2019)

Así, conociendo estos sistemas de referencia, más adelante veremos cómo vamos a trabajar con ellos, viendo para ello qué transformaciones entre estos nos serán necesario realizar a la hora de obtener los resultados deseados con la velocidad que obtengamos para nuestra cámara en el bucle de control.

Sabiendo todo esto, ya podemos pasar a explicar cómo hemos realizado la implementación de nuestro sistema de control visual basado en imagen basándonos en toda la teoría vista previamente dentro del sistema robótico que hemos visto en este punto, utilizando para ello todas las herramientas que hemos destacado, y todos los puntos importantes de nuestro robot de los que hemos hablado.

2.3. Control visual en nuestro sistema robótico.

En este último apartado de metodología pasaremos a ver cómo hemos realizado la implementación de nuestro controlador visual en el robot con el que estamos trabajando. Para ello, nos basaremos en toda la información vista en los dos puntos anteriores para conocer cómo vamos a realizar el reconocimiento de nuestros patrones, cómo vamos a interpretar dichos patrones, y las transformaciones que nos harán falta a la hora de transformar nuestra velocidad del controlador al sistema con el que trabajamos.

Cabe destacar antes de empezar a explicar las implementaciones técnicas realizadas que, para nuestro caso, a la hora de implementar los bucles de control que vamos a explicar, lo hemos hecho con una configuración de cámara en mano y pasando comandos de forma indirecta, pues como veremos más adelante, la cámara realizará el movimiento que deba hacer junto con el robot, y además dicho movimiento se realizará indicando sólo velocidades o posiciones a nuestro robot, dejando que sea su controlador interno el que decida cómo va a mover las articulaciones que le hagan falta al sistema para moverse.

Además de todo esto, debemos saber que todos los datos que tratemos dentro de este punto estarán referidos a cómo se ha realizado el control visual en función de los programas que podemos ver destacados en los anexos A y B. En ellos, veremos en el primero el control de ver y mover, y en el segundo el control visual basado en imagen. En los siguientes puntos, trataremos de explicar lo que se ha realizado en dichos programas, y el porqué de los pasos que hemos ido siguiendo para obtener nuestros controladores principales.

Una vez hayamos explicado todos los desarrollos que hemos realizado en nuestro robot, podremos pasar a tratar sobre las experimentaciones que hemos realizado en nuestro laboratorio, teniendo en cuenta todos los datos que hemos ido destacando a lo largo de estos puntos de metodología para interpretar los resultados que iremos obteniendo paso a paso para las distintas pruebas que realicemos sobre nuestro robot con los controladores visuales que hemos implementado, tratando de realizar algunos cambios puntuales en éstos para ver cómo afectan dichos cambios a toda la teoría que vayamos destacando.

2.3.1. Transformaciones realizadas para los sistemas de referencia.

Como hemos mencionado anteriormente, la configuración de nuestro robot de cámara en mano y además de control indirecto, hará que las velocidades que obtengamos de nuestro controlador sean referidas directamente a la cámara de nuestro robot. Si le pasamos esta velocidad directamente a nuestro robot, puede causarnos problemas, pues como hemos visto en el punto de detalles sobre el NAO, éste recibe los comandos de velocidad en referencia a un espacio de trabajo propio llamado "*FRAME_ROBOT*", el cual se encuentra en todo momento entre las piernas de nuestro robot, siendo éste la proyección del centro de gravedad de nuestra máquina en el suelo (ver Fig. 39 y Fig. 40).

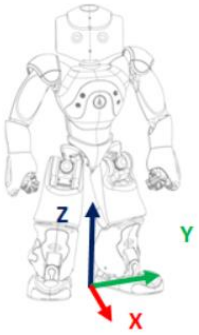

Figura 39: Referencia para los comandos de velocidad. Fuente: Aldebaran Documentation. http://doc.aldebaran.com/2-1/naoqi/motion/control-cartesian.html (visitado el 16/05/2019)

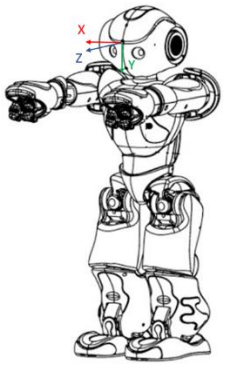

Figura 40: Referencia para la cámara. Fuente: SciELO. http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0123-77992014000200007 (visitado el 17/05/2019)

Como podemos observar, requeriremos de ciertas transformaciones a la hora de transmitir posición o velocidad desde la cámara hasta nuestro espacio de trabajo para el movimiento del robot, pues los sistemas de referencia de ambos espacios están girados y trasladados en posición, haciendo que en principio las velocidades que obtenemos no se correspondan con las que deseamos realmente.

Sabiendo esto, en primer lugar, podemos tratar las transformaciones más básicas de rotación y traslación necesarias para obtener la matriz de transformación para las posiciones de un punto, pudiendo cambiar éste de las coordenadas de la cámara a los del sistema de referencia de nuestro movimiento. Como podemos observar, en lo relativo a las rotaciones tenemos dos, habiendo una en el eje X, y otra en el eje Z, siendo ambas de noventa grados en cada caso con sentido negativo en él giro. Para realizar dichas rotaciones, deberemos de tener en cuenta las matrices de rotación ya conocidas para cada eje, las cuales tiene la siguiente forma en cada caso:

$$ {}^{C}R_{B_X} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(-90) & -sen\,(-90) \\ 0 & sen\,(-90) & cos(-90) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix} \quad (21) $$

$$ {}^{C}R_{B_Z} = \begin{pmatrix} \cos(-90) & -sen\,(-90) & 0 \\ sen\,(-90) & \cos(-90) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (22) $$

Sabiendo que ${}^{C}R_{B_X}$ es la matriz de giro necesaria para realizar la transformación de la cámara a la base de referencia en X, y ${}^{C}R_{B_Z}$ es lo mismo en Z. Si queremos obtener la matriz de giro final como la composición de estos dos giros, lo que haremos será pre multiplicar, obteniendo así los siguientes resultados:

$$ {}^{C}R_{B} = {}^{C}R_{B_Z} \times {}^{C}R_{B_X} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix} \quad (23) $$

Así, obtenemos la matriz de rotación final de la cámara a la base ${}^{C}R_{B}$. Conociendo este dato, y sabiendo que, si medimos desde la referencia de la cámara, habrá una distancia de una referencia a otra de 0.544 metros en Y, además de otra de 0.055 metros en Z, podemos obtener la siguiente matriz de transformación para la localización de puntos en el espacio:

$$ {}^{C}T_{B} = \begin{pmatrix} {}^{C}R_{B} & {}^{C}t_{B} \\ 0 \quad 0 \quad 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0{,}544 \\ 0 & -1 & 0 & -0{,}055 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (24) $$

Como podemos observar, con esta matriz podremos cambiar puntos de sistema de referencia al multiplicarlos por ella, siendo útil para sistemas de control como el de ver y avanzar, pero requiriendo de otras informaciones para controles basados en velocidad.

Para realizar las transformaciones en velocidad, vemos que requeriremos de la siguiente matriz de transformación en velocidad basándonos en todas las transformaciones que hemos visto previamente:

$$ {}^{C}V_{B} = \begin{pmatrix} {}^{C}R_{B} & [{}^{C}t_{B}]_{\times}\,{}^{C}R_{B} \\ 0 & {}^{C}R_{B} \end{pmatrix} \quad (25) $$

Como podemos observar, tendremos el término $[{}^{C}t_{B}]_{\times}$, el cual esta referido a la matriz de traslación de nuestros sistemas transformado a su forma de matriz de *skew* para que se pueda multiplicar con la rotación y dar un resultado de matriz cuadrada. Así, dicha operación daría el siguiente resultado:

$$ [{}^{C}t_{B}]_{\times}\,{}^{C}R_{B} = \begin{pmatrix} 0 & 0.055 & 0.544 \\ -0.055 & 0 & 0 \\ -0.544 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} -0.055 & -0.544 & 0 \\ 0 & 0 & -0.055 \\ 0 & 0 & -0.544 \end{pmatrix} \quad (26) $$

Sabiendo esto, podemos ver que la matriz de transformación de velocidades de nuestra cámara a la base del movimiento de nuestro robot tendrá finalmente la siguiente forma:

$$ {}^{c}V_{B} = \begin{pmatrix} 0 & 0 & 1 & -0{,}055 & -0{,}544 & 0 \\ -1 & 0 & 0 & 0 & 0 & -0{,}055 \\ 0 & -1 & 0 & 0 & 0 & -0{,}544 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \end{pmatrix} \quad (27) $$

Una vez conocemos todas estas matrices, podemos ver que podremos realizar cualquier tipo de transformación que nos sea necesaria para nuestros controladores, teniendo así que, aunque las velocidades o posiciones con las que vayamos a trabajar estén referidas a la cámara de nuestro robot, podremos transformar éstas a la base del NAO para tener siempre unos datos validos con los que trabajar a la hora de pasarle comandos a nuestro sistema.

Otro punto que se debe tener en cuenta en nuestro sistema será el de las transformaciones que tendremos que realizar entre los puntos que utilizaremos para detectar en el sistema de coordenadas de la misma imagen, los cuales estarán referidos a la esquina de ésta, y medidos como píxeles, y su transformación a puntos genéricos 3D, los cuales estarán referidos al centro de la imagen, y estarán expresados en metros. Estos cálculos los realizaremos debido a cómo interpreta ViSP las características que detecta en la imagen, pues lo hace con un tipo llamado *vpImagePoint*, el cual está referido en píxeles como hemos dicho anteriormente, mientras que nosotros tendremos que utilizar otro tipo de puntos que se expresen en metros, ya que el controlador espera los datos en estas medidas, y además referidos al centro de la imagen.

Conociendo estos datos, podemos observar que nuestros ejes de referencia se encontrarán en sentidos opuestos en los dos casos (ver Fig. 41), y que además distarán de una distancia de 0.045 metros en ambas coordenadas, teniendo así una disposición del siguiente tipo en la imagen para los ejes con los que vamos a trabajar:

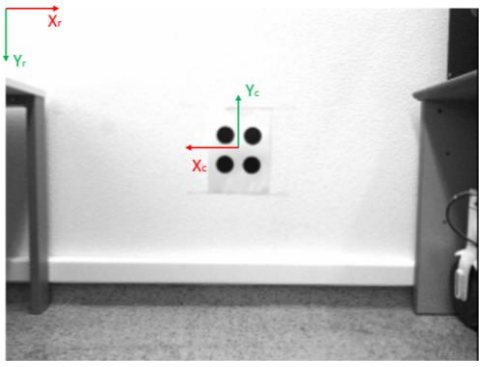

Figura 41: Diferencia entre las referencias dentro de la imagen

Sabiendo esto, y que el factor de conversión de píxel a metro equivaldrá a que un metro es igual a 3779 píxeles, vemos que para realizar nuestra transformación 2D tan solo tendremos que multiplicar el valor que obtengamos por la referencia de la cámara por la conversión de píxel a metros, y luego restarle la distancia en ambas coordenadas de un sistema a otro para conocer la posición de un punto referido al centro de nuestra imagen.

Una vez conocida esta transformación y la anterior, ahora si tenemos un tratamiento de las referencias ya valido con las cuales podremos estimar bien posición de un punto dentro de nuestra imagen, y luego además sabremos como transformar dicho punto dentro de las referencias de nuestro robot, ya sea en posición o en velocidades.

2.3.2. Detección de referencias con nuestro robot NAO

Una vez conocemos como estimar las posibles transformaciones que nos van a ser necesarias para guiar a nuestro robot, debemos pasar a plantearnos cómo vamos a hacer para que nuestro robot estime la posición de los puntos de interés con los que vamos a trabajar para que obtengamos las referencias necesarias dentro de nuestro controlador.

En primer lugar, es interesante hablar sobre la detección de código Qr para el controlador ver y mover. En este caso, nos basaremos en la teoría desarrollada y vista en la Fig. 25, siendo dicho método implementado por el paquete de ROS en combinación con ViSP de *visp_auto_tracker*. Dicho paquete tiene varias implementaciones de seguimiento de objetos, teniendo éste un nodo especial de seguimiento de códigos de barra y Qr, el cual se encarga de publicar los distintos datos que vamos obteniendo de la imagen con el seguimiento del objeto en cuestión del cual queremos sacar su información (ver Fig. 42).

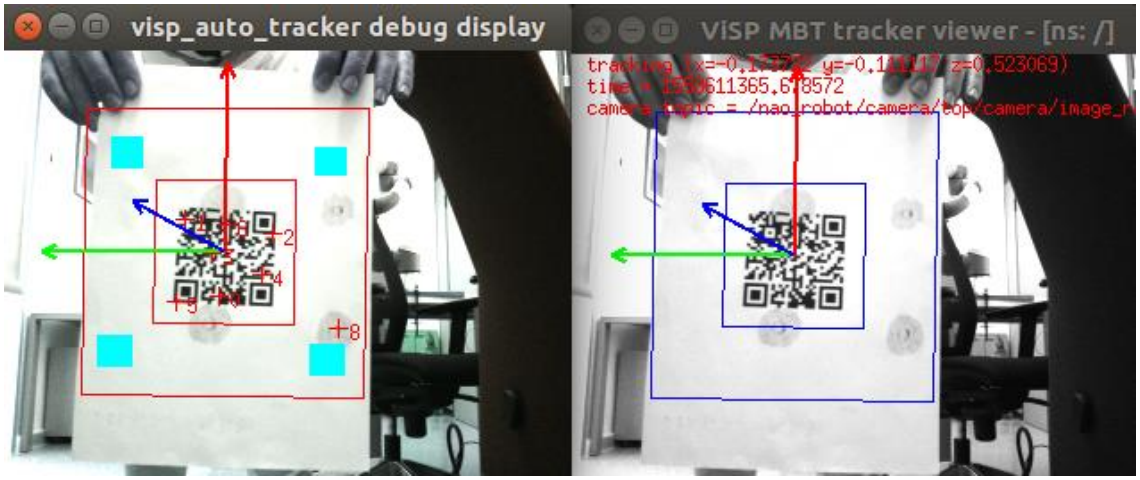

Figura 42: Fotografías de la detección del código Qr que realizamos con nuestro robot con información espacial extraída de la imagen

Como podemos imaginar, uno de los tópicos de nuestro nodo nos dará información sobre la posición del código Qr dentro de la imagen. Sabiendo esto, simplemente tendremos que realizar una transformación haciendo uso de la matriz expuesta en la Ecuación (24) para obtener la posición real que queremos mandarle a nuestro robot para que éste realice su avance, comunicando todos estos datos con los métodos de suscripción y publicación que nos ofrece el NAO.

En cuanto a la detección de puntos necesaria para obtener los patrones de control visual basado en imagen, vemos que aquí tenemos más teoría que tratar sobre cómo el robot realiza la estimación de la posición de los puntos necesaria para utilizarla en el controlador de nuestro sistema. En primer lugar, debemos de ver que la estimación inicial que realizamos para obtener los puntos que queremos destacar en la imagen, se basan en el código mostrado en la Fig. 43.

```
vpDot2 dot_final[4] ;
vpImagePoint cog_final;
vpPoint point[4];

for (int i=0 ; i < 4 ; i++) {
  dot_final[i].setGraphics(true);
  dot_final[i].setComputeMoments(true);
  dot_final[i].setEllipsoidShapePrecision(0.);
  dot_final[i].setGrayLevelPrecision(0.9);
  dot_final[i].setEllipsoidBadPointsPercentage(0.5);
  dot_final[i].initTracking(I) ;
  cog_final = dot_final[i].getCog();
  vpDisplay::displayCross(I, cog_final, 10, vpColor::blue) ;
  vpDisplay::flush(I);
}
```

Figura 43: Código realizado para la estimación inicial de los puntos dentro de la imagen

Dentro de este código, podemos ver que la función encargada de obtener el punto a seguir será la de inicio de seguimiento (*initTracking*), la cual nos mostrará la imagen y nos pedirá que seleccionemos un punto de ésta para realizar el seguimiento de la región que hayamos seleccionado. Así, al seleccionar un punto con el ratón, el programa realizará una estimación de la región a seguir como hemos indicado en el punto de visión artificial. Además, encontramos otras funciones que también son interesantes para este seguimiento, como bien pueden ser la de aceptación de porcentaje de error para que no perdamos la referencia que hayamos obtenido al tener algún movimiento brusco (*setEllipsoidBadPointsPercentage*), la de precisión para ajustar la región que queremos detectar lo más que podamos (*setEllipsoidShapePrecision*), o el computo de momentos (*setComputeMoments*).

De todas estas funciones que hemos destacado, la que nos será más importante es la que realiza el computo del centro de masas de nuestra referencia (*getCog*), pues con esta función obtendremos las coordenadas del punto que utilizaremos para realizar el seguimiento con nuestro controlador. También cabe destacar de este código que realiza la misma operación cuatro veces dentro del mismo bucle iterativo, siendo esto debido a que nuestra referencia tendrá cuatro puntos a estimar, y tendremos que marcar todos éstos.

Otro punto destacable de esta detección es que la repetiremos dos veces en posiciones distintas de nuestro robot. Así, tomaremos en primer lugar la información en la posición de referencia que queremos que tenga nuestro robot, el cual nos dará los puntos a alcanzar dentro de nuestra imagen con el movimiento a realizar, y luego tomaremos los puntos a partir de la posición inicial desde la cual queremos que nuestro robot comience el movimiento, teniendo así la información de las referencias y del inicio para que nuestro controlador pueda realizar con ambos datos los cálculos que sean pertinentes.

Una vez conocemos todo esto, ya sabremos cómo vamos a realizar la estimación de la posición de nuestros puntos en la imagen, pudiendo pasar a ver la forma de realizar el controlador en el que nos vamos a basar para el movimiento en nuestro robot, tratando varios temas destacables sobre cómo hemos desarrollado éste.

2.3.3. Implementación del controlador: estimación de características, ganancia adaptativa, y propiedades del controlador.

Una vez hemos visto cómo detectar los puntos que vamos a seguir, pasamos a hablar sobre el controlador más complejo que hemos realizado, que es el de control visual basado en imagen, pues el controlador de ver y seguir no requiere de ninguna estimación de control compleja, y además no tiene una realimentación por la que se requiera un envío de información de los datos que se van midiendo en ejecución, pues este simplemente envía la información de posición a nuestro robot y hace que éste se mueva a dicha posición, sin estimar ninguna información adicional.

Así, sabiendo que vamos a hablar de nuestro segundo controlador, cabe destacar en primer lugar cómo vamos a realizar la creación de todos los puntos que hemos ido obteniendo como puntos de características para nuestro controlador, siguiendo para ello el código mostrado en la Fig. 44.

```
vpFeaturePoint p[4] ;
for (int i=0; i < 4 ; i++)
    vpFeatureBuilder::create(p[i], cam, dot[i]);
```

Figura 44: Código requerido para la creación de puntos de características con nuestro controlador

Como podemos observar, en este caso también repetiremos la operación cuatro veces debido a los cuatro puntos de nuestro patrón. Sabiendo esto, podemos ver como la función de creación simplemente requiere del punto anteriormente destacado en nuestra imagen y de un punto de características definido por un tipo ya creado en ViSP, por lo que la operación de estimar nuestros puntos característicos es bastante simple. Cabe destacar que, en la estimación de las características, a la hora de realizar esta operación para los puntos deseados finales, deberemos de tener en cuenta también la información de profundidad de éstos para obtener la matriz de interacción con la que trabajaremos, por lo que deberemos de indicar dicho valor a parte mediante una función específica para dar estos valores.

Una vez que hemos obtenido los puntos de valores de características, deberemos de estimar la posición de nuestro robot en su punto de inicio del movimiento, para obtener con éste la matriz de transformación que definirá el cambio que iremos obteniendo sobre la profundidad de nuestra máquina conforme ésta va avanzando. Esta acción se muestra en la Fig. 45.

```
vpHomogeneousMatrix cMo;
vpPose pose;
pose.clearPoint();

for (int i=0 ; i < 4 ; i++)
{
      double x,y;
      vpPixelMeterConversion::convertPoint(cam,dot[i].getCog(),x,y)  ;
      point[i].set_x(x) ;
      point[i].set_y(y) ;
      pose.addPoint(point[i]) ;
}
pose.computePose(vpPose::LAGRANGE, cMo) ;
pose.computePose(vpPose::VIRTUAL_VS, cMo) ;
```

Figura 45: Código para la estimación de posición de nuestro robot

Como podemos observar, simplemente transformaremos la información que tenemos de píxeles a metros, pasando luego estos valores a la estimación de la pose dentro del punto que utilizamos para ello. Tras esto, utilizaremos la función de computación de la posición (*computePose*) para conocer finalmente la matriz de transformación con la que tendremos nuestro punto inicial. Esta función destacada será importante durante toda la ejecución de nuestro programa, pues será la encargada también de obtener el valor de profundidad que vayamos obteniendo paso a paso por nuestro robot para que este realice una estimación de la matriz de interacción en cada ciclo que va avanzando nuestra máquina.

Para realizar dicha estimación, podemos observar que mediante la función mencionada, llamamos a dos métodos distintos, pues en primer lugar llamaremos a la función mediante el método *vpPose::LAGRANGE,* que es la aproximación lagrangiana para la estimación inicial de la pose con el cual estimaremos la primera profundidad que tendremos con nuestra primera pose en él movimiento, y luego haremos uso del método *vpPose::VIRTUAL_VS*, el cual será un método ya inicializado por el anterior con el cual podremos ir estimando en cada vuelta del bucle más adelante las actualizaciones de la pose de nuestro robot en cada momento, y que se trata de la aproximación de control visual virtual.

En cuanto a la aproximación de Lagrange, sabiendo la teoría que hemos desarrollado en la ecuación (7), y conociendo la posición de la cámara con respecto al espacio de trabajo de nuestras referencias ${}^{C}M_{O}$, podemos ver que tenemos las siguientes ecuaciones:

$$x_i = \frac{r_{11}{}^0X_i + r_{12}{}^0Y_i + r_{13}{}^0Z_i + t_x}{r_{31}{}^0X_i + r_{32}{}^0Y_i + r_{33}{}^0Z_i + t_z} \tag{28}$$

$$y_i = \frac{r_{21}{}^0X_i + r_{22}{}^0Y_i + r_{23}{}^0Z_i + t_y}{r_{31}{}^0X_i + r_{32}{}^0Y_i + r_{33}{}^0Z_i + t_z} \tag{29}$$

Si desarrollamos estas ecuaciones, podemos ver que estas pueden llegar a dar el siguiente resultado si agrupamos todos los componentes en el mismo lado de la ecuación:

$$\begin{cases} r_{31}{}^0X_i x_i + r_{32}{}^0Y_i x_i + r_{33}{}^0Z_i x_i + x_i t_x - (r_{11}{}^0X_i + r_{12}{}^0Y_i + r_{13}{}^0Z_i + t_x) = 0 & (30) \\ r_{31}{}^0X_i y_i + r_{32}{}^0Y_i y_i + r_{33}{}^0Z_i y_i + y_i t_y - \left(r_{21}{}^0X_i + r_{22}{}^0Y_i + r_{23}{}^0Z_i + t_y\right) = 0 & (31) \end{cases}$$

Vemos que obtenemos un sistema homogéneo con doce parámetros desconocidos: AI = 0, donde A depende de la información que obtenemos de nuestra imagen e I es función de los parámetros a estimar. Así, cada punto de la ecuación nos da las siguientes ecuaciones:

$$A_i I = [0\ 0]^T \tag{32}$$

$$A_i = \begin{pmatrix} -{}^0X_i & -{}^0Y_i & -{}^0Z_i & 0 & 0 & 0 & x_i{}^0X_i & x_i{}^0X_i & x_i{}^0X_i & -1 & 0 & x_i \\ 0 & 0 & 0 & -{}^0X_i & -{}^0Y_i & -{}^0Z_i & y_i{}^0X_i & y_i{}^0Y_i & y_i{}^0Z_i & 0 & -1 & y_i \end{pmatrix}$$

Si descomponemos dichas ecuaciones en partes de la matriz A, podemos ver que obtenemos el siguiente sistema:

$$AX_1 + BX_2 = 0 \tag{33}$$

Dichas partes de la ecuación tendrán el siguiente significado:

$$\begin{cases} X_1 = (r_{31}\ r_{32}\ r_{33})^T\ y\ \|X_1\| = 1 & (34) \\ X_2 = (r_{11}\ r_{12}\ r_{13}\ r_{21}\ r_{22}\ r_{23}\ t_x\ t_y\ t_z)^T & (35) \end{cases}$$

$$A_i = \begin{pmatrix} x_i\,{}^0X_i & x_i\,{}^0Y_i & x_i\,{}^0Z_i \\ y_i\,{}^0X_i & y_i\,{}^0Y_i & y_i\,{}^0Z_i \end{pmatrix} \quad (36)$$

$$B_i = \begin{pmatrix} -{}^0X_i & -{}^0Y_i & -{}^0Z_i & 0 & 0 & 0 & -1 & 0 & x_i \\ 0 & 0 & 0 & -{}^0X_i & -{}^0Y_i & -{}^0Z_i & 0 & -1 & y_i \end{pmatrix} \quad (37)$$

Como una solución directa es imposible (I = 0 no se puede dar), tendremos que considerar una minimización Lagrangiana, por lo que minimizaremos el siguiente criterio:

$$C = \|A \cdot X_1 + B \cdot X_2\|^2 + \lambda(1 - \|X_1\|^2) \quad (38)$$

Con esto, si hacemos nulas las siguientes derivadas parciales:

$$\frac{1}{2}\frac{\partial C}{\partial X_1} = A^T A \cdot X_1 + A^T C \cdot X_2 - \lambda X_1 = 0 \quad (39)$$

$$\frac{1}{2}\frac{\partial C}{\partial X_2} = B^T A \cdot X_1 + B^T C \cdot X_2 = 0 \quad (40)$$

Con esto, obtenemos lo siguiente:

$$X_2 = -(B^T B)^{-1} B^T A \cdot X_1 \quad (41)$$

$$E \cdot X_1 = \lambda X_1\ con\ E = A^T A - A^T B (B^T B)^{-1} B^T A \quad (42)$$

Cabe destacar que X_1 será un vector de valores propios unitario de E y correspondiendo al valor propio λ tenemos que C = λ. Así, X_1 es el vector normal de valores propios correspondiente a el mínimo valor propio de E. Con esto, obtenemos X_1 y X_2, y para obtener I, tan solo tendremos que ver que $I_{12} = Z_0 > 0$.

En cuanto a la aproximación de control visual, podemos ver que nos basaremos en todo lo destacado para el método Lagrangiano con el que obtenemos la matriz ${}^CM_O$ más las velocidades que vamos obteniendo de la realimentación de nuestro robot, planteando para este método la siguiente ecuación [29]:

$${}^cM_O^{k+1} = {}^cM_O^k \exp(v) \quad (43)$$

Donde k será el ciclo de control en el que estamos. Con estas ecuaciones, podremos estimar la pose de nuestro robot tanto en su posición inicial con el método de Lagrange como en la ejecución del bucle con el método de control visual, para así poder actualizar nuestra matriz de interacción de manera adecuada con la profundidad correspondiente.

Sabiendo esto, debemos de destacar cómo vamos a interpretar el controlador desde nuestro código. Para ello, nos basaremos en el tipo *vpServo*, el cual lo utilizaremos para crear variables que sean objetos de clases que nos sirvan como el controlador que queremos implementar. Con esta clase, tendremos una serie de valores y funciones que nos servirán para definir nuestro sistema, pudiendo definir así todos los detalles que queremos realizar para nuestro controlador.

Sabiendo esto, podemos ver que ya podemos empezar a definir las características que nos servirán como puntos de referencia para nuestro controlador, utilizando para ello el fragmento de código mostrado en la Fig. 46.

```
for (int i=0 ; i < 4 ; i++)
    task.addFeature(p[i], pd[i]) ;
```

Figura 46: Método para añadir las características que hemos calculado dentro de nuestro controlador

Como podemos observar, con una simple función podremos añadir la información que hemos obtenido de todos los puntos que hemos computado (puntos iniciales *p* y deseados *pd*) con lo que podemos pasar a ver cómo vamos a preparar a nuestro controlador para empezar a trabajar.

En primer lugar, antes de empezar a matizar nada más, debemos de dar un valor a la ganancia con la que queremos que nuestro controlador trabaje. Para ello, veremos que podemos definir dicha ganancia tal y como se muestra en la Fig. 47.

```
vpAdaptiveGain  lambda;
lambda.initStandard(4.10, 0.4, 40);

task.setLambda(lambda) ;
```

Figura 47: Funciones y valores para nuestra ganancia de controlador adaptativa

Como podemos observar, para el control que vamos a implementar, haremos uso de una **ganancia adaptativa**. Este tipo de ganancia lo que hará será realizar la estimación de su propio valor a partir de unas cotas de máximo y mínimo a poder tener, estimando nuestra ganancia final de la siguiente forma:

$$\lambda(error) = a \times e^{(-b \times error)} + c \tag{44}$$

Con estos valores definiremos la ganancia que iremos utilizando en cada momento del proceso de nuestro controlador, viendo así que el valor de nuestra variable del controlador ira adaptándose a lo que le indiquemos con dichos valores. Para estimar estos valores, podemos ver que utilizaremos las siguientes funciones:

$$\begin{cases} a = \lambda_0 - \lambda_\infty & (45) \\ b = \dfrac{\lambda'_0}{a} & (46) \\ c = \lambda_\infty & (47) \end{cases}$$

Los distintos valores de lambda que observamos son los que definiremos dentro de nuestra función, y estos tendrán el siguiente significado:

- λ_0: Representa la ganancia cuando tenemos un error equivalente a cero. Esta ganancia influirá en los movimientos en los que nuestro robot se empiece a acercar a su objetivo.
- λ_∞: Representa la ganancia en el infinito. Esta ganancia es la que más afectara cuanto más lejos estemos de nuestro objetivo.
- λ'_0: Representa la pendiente de la ganancia cuando nos encontramos en un error igual a cero. Se utiliza para definir cómo vamos a querer que evolucione nuestra ganancia en los movimientos finales.

Modificando dichos valores, podremos ver cómo va variando la evolución de la señal de nuestro robot durante la ejecución del sistema, teniendo así un método para hacer que nuestra ganancia se adapte a las circunstancias que vayamos encontrando en el entorno en el que vamos a trabajar [30].

Una vez que sabemos cómo va a funcionar la ganancia que vamos a utilizar en nuestro sistema, podemos ver cómo vamos a iniciar nuestro controlador finalmente, viendo para ello que el código que utilizaremos para ello será el mostrado en la Fig. 48.

```
task.setServo(vpServo::EYEINHAND_CAMERA) ;
task.setInteractionMatrixType(vpServo::CURRENT, vpServo::PSEUDO_INVERSE) ;
```

Figura 48: Código para iniciar el controlador que vamos a utilizar

Como podemos observar, en esta parte de nuestro código tendremos que indicar la configuración que tendremos para nuestra cámara, y además, qué tipo de aproximación vamos a realizar para nuestra matriz de interacción.

Tras esto, comenzaremos el bucle de control, en el que comenzaremos a ir actualizando los valores de referencia que vamos obteniendo, modificando también la posición que vamos calculando para ir actualizando el valor de profundidad que tiene nuestro robot con respecto a su referencia. Tras esto, veremos cómo estimaremos en cada final de nuestro bucle la velocidad que vamos a tener para la cámara de nuestro robot, la cual la estimaremos con el código mostrado en la Fig. 49.:

```
vpColVector v ;
v = task.computeControlLaw() ;
```

Figura 49: Código necesario para calcular la velocidad que estimaremos para la cámara de nuestro robot

Así, ya tenemos toda la implementación de nuestro bucle de control final, viendo que finalmente tendríamos que multiplicar el valor de las velocidades que obtenemos de nuestro controlador por la matriz de transformación de velocidad que estimamos previamente, teniendo así la velocidad final que mandaremos al tópico de nuestro robot para que realice su movimiento hacia la referencia.

Finalmente, cabe destacar que emplearemos un método de control para saber cuándo hemos alcanzado un error aceptable para nuestro controlador (ver Fig. 50).

```
if (task.getError().sumSquare() < 0.0005)
    break;
```

Figura 50: Código para estimar el final del bucle de control

Como podemos observar, con la función de obtención de error (*getError*) obtendremos el vector de los errores de nuestras referencias, y con la función de suma cuadrada (*sumSquare*) obtendremos el valor del módulo de nuestro error, estimando así que nuestro controlador parará en el momento en el que dicho valor del módulo de nuestro error se encuentre por debajo de 0.0005, y finalizando así el bucle de control.

Sabiendo todos estos datos, podemos observar que ya tenemos todos los conocimientos tanto teóricos como técnicos del modelo de control que queremos implementar y del robot con el que vamos a trabajar, por lo que ya podríamos pasar a ver la parte experimental de nuestro trabajo, comparando para ello los distintos valores que hemos ido obteniendo en cada una de las pruebas que hemos ido realizando al modificar tanto la posición de nuestro robot como algunos de los valores de control de los que hemos ido definiendo a lo largo de la memoria.

3. Experimentación

Finalmente, podemos pasar a hablar sobre la experimentación que hemos realizado con el sistema que hemos implementado en nuestro robot, viendo la preparación que hemos realizado para ello, los valores de los resultados que hemos obtenido de cada experimentación, y finalmente comparando cada uno de los resultados que hemos obtenido para tener una idea más general de las conclusiones que podemos obtener sobre el funcionamiento de este tipo de control para el control de guiado de la marcha de robots humanoides como el nuestro.

3.1. Planteamiento inicial: posicionamiento del entorno y experimentos a realizar

En primer lugar, veremos qué experimentos vamos a analizar, destacando las diferencias entre todos éstos, y viendo para ello cómo cambiará el posicionamiento de nuestro robot en cada caso en función de qué desarrollo vayamos a realizar para el estudio de los resultados.

3.1.1. Planteamiento para los experimentos de control visual basado en imagen

Como primer punto importante sobre el que reflexionaremos, es sobre el posicionamiento en el que tomaremos los puntos de referencia a los que debe llegar nuestro robot, viendo así que, en dichos puntos, tendremos el reparto de cada componente de nuestro desarrollo tal y como se muestra en la Fig. 51.

Figura 51: Posicionamiento de referencia en perspectiva, planta, y desde el punto de vista del robot de los elementos del experimento para tomar los datos de referencia.

En esta posición que hemos destacado, veremos que la referencia en el eje Y estará completamente en cero, al igual que la referencia en el eje Z. Será la referencia de X la que varía de la posición del patrón de referencia que hemos usado, teniendo un valor de profundidad del robot a nuestro patrón de 0.542 metros.

Una vez conocemos la posición objetivo y desde la que tomaremos las referencias a las cuales queremos llegar, deberemos de pasar a plantear dos posicionamientos distintos que vamos a plantear para nuestro robot para ir variándolos en distintos experimentos, los cuales podemos explicar de la siguiente forma:

- Posicionamiento en línea recta (ver Fig. 52), creando así una trayectoria recta completamente para nuestro robot con la referencia a la que queremos que llegue, para ver cómo va a afectar el movimiento más simple que podemos realizar al método de control de nuestro robot. En esta posición, además, será en la que más alejemos a nuestro robot, viendo cómo va a afectar la distancia a nuestro bucle de control.
- Posicionamiento doblado respecto a la referencia (ver Fig. 53), creando así una trayectoria curva para nuestro robot que nos dejara ver cómo van a variar las variables de control en este caso para nuestro robot, forzando a éste a realizar una curva con los valores de las velocidades que puede modificar.

Figura 52: Posicionamiento inicial en trayectoria lineal en perspectiva, planta, y desde el punto de vista del robot de los elementos del experimento.

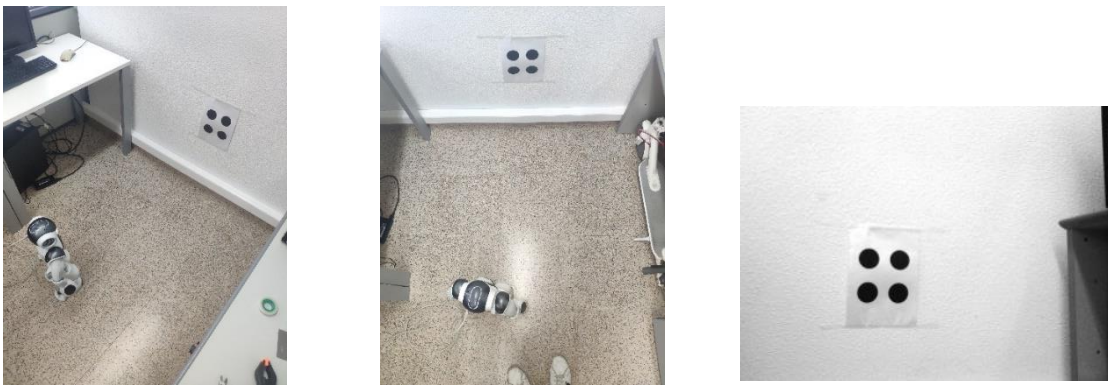

Figura 53:Posicionamiento inicial en trayectoria curvada en perspectiva, planta, y desde el punto de vista del robot de los elementos del experimento.

Una vez que conocemos todas las posiciones que vamos a tomar de referencia y desde las cuales vamos a salir, podemos pasar a analizar los cambios que haremos a nivel de control. Para ello, lo que haremos será variar los valores de la ganancia proporcional para ambas trayectorias, viendo cómo afecta esto al movimiento de nuestro robot y a cómo vamos obteniendo y siguiendo nuestras referencias. Así, podemos ver que plantearemos los siguientes casos:

- λ_0 = 1, λ_∞= 0.1 y λ'_0 = 10, pretendiendo con este caso que nuestro robot no sea capaz de alcanzar su referencia final debido a que sus ganancias tan pequeñas harán que el bucle de control llegue un momento en el que no sea capaz de hacer converger todas las velocidades a cero.
- λ_0 = 4.10, λ_∞= 0.4 y λ'_0 = 40, viendo en este caso que nuestro bucle de control sí que podrá hacer que el robot llegue a su referencia final, y viendo cómo esta será la forma más lenta, pero precisa, de realizar esta acción.
- λ_0 = 5.1, λ_∞= 0.5 y λ'_0 = 50, teniendo con esta prueba un controlador con una ganancia intermedia, que ira más rápido que el caso que hemos destacado anteriormente, y será más preciso que los que destaquemos a continuación.
- λ_0 = 6.1, λ_∞= 0.6 y λ'_0 = 60, obteniendo en esta ocasión el controlador más rápido en ser capaz de alcanzar las referencias que le planteamos, pero perdiendo precisión para ello debido a la elevada velocidad.
- λ_0 = 10, λ_∞= 1 y λ'_0 = 100, tratando finalmente en este caso de ver cómo, al tener ganancias demasiado elevadas, nuestro robot acaba por realizar movimientos demasiado bruscos, los cuales harán que el sistema de visión pierda las referencias en las que se basa para avanzar, y forzando así al sistema a acabar su ejecución antes de llegar al objetivo.

Cabe destacar que, en los que sepamos que el robot no podrá alcanzar de ninguna forma su posición final, tan solo realizaremos pruebas con un movimiento lineal, pues nos interesa ver cómo será el comportamiento del error y de la velocidad en ambos casos, pero no tiene ningún objetivo repetir experimentos con estas ganancias, pues sabemos de antemano que no conseguirán llegar al objetivo, aunque cambiemos la posición.

Finalmente, como prueba final, realizaremos un experimento de seguimiento, haciendo que nuestras referencias se muevan para ver cómo esto puede afectar al controlador final de nuestro robot, y viendo cómo variará el comportamiento si la visión detecta acercamientos o alejamientos muy repentinos de los puntos a seguir.

Una vez conocemos todos estos datos, podemos pasar a ver los resultados que hemos obtenido en cada caso, para más tarde poder comparar éstos y ver que se cumplen las predicciones que hemos realizado previamente para realizar los experimentos.

3.1.2. Planteamiento para los experimentos de control de ver y mover

En este segundo caso, el planteamiento del entorno que realizaremos será mucho más simple, pues simplemente requeriremos posicionar el robot frente a nuestro código Qr para que éste lo detecte y con la posición que obtenga de él, se pueda mover hacia su posición.

Así, en la Fig. 54 podemos ver el posicionamiento entre robot y referencia final para este experimento.

Figura 54: Posicionamiento inicial del robot y la referencia en perspectiva y planta para el control de tipo ver y mover

En este caso, no requeriremos de una posición fija de nuestra referencia para estimar nada, pudiendo dejar éste a la distancia que queramos en todo momento. Esto en parte será debido a que tan solo la utilizaremos al inicio, tomando de ella sus datos de posición y pasándoselos a nuestro robot, sin plantearnos nada más.

3.2. Resultados de los experimentos realizados

Una vez conocidos todos los planteamientos iniciales que hemos realizado, podemos pasar a ver los resultados que hemos obtenido en cada uno de los casos que hemos planteado. Para ello, iremos dividiendo los resultados que vamos a mostrar en función de las ganancias que hemos planteado, para ver así la evolución que van sufriendo los valores que vamos tratando en cada caso. Cabe destacar que en los datos que observaremos en gráficas, los veremos duplicados dos veces, pues trataremos de ver la señal sin filtrar para ver claramente el inicio y el final de ésta, y la señal filtrada, para poder observar cómo evolucionan los datos que vamos a ir analizando. Esto lo haremos debido a que el movimiento del NAO en marcha genera mucho ruido a nuestras velocidades y datos, por lo que requeriremos de este filtro para observar cómo varia cada característica.

3.2.1. Resultados para λ_0= 1, λ_∞ = 0.1 y λ'_0 = 10

En este caso, podemos observar, como ya habíamos predicho, que nuestro controlador no será capaz de realizar el cálculo de las velocidades finales requeridas para alcanzar a su objetivo, acabando así por perder las referencias debido al movimiento continuo de nuestro robot. Así, la Fig. 55 muestra la evolución de la velocidad y la Fig. 56 el error en este caso.

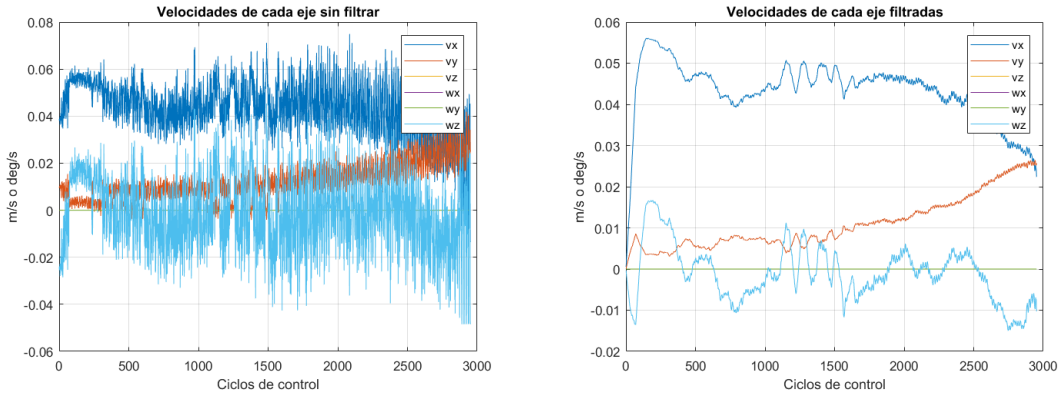

Figura 55: Evolución de la señal de velocidad para el caso λ_0 = 1, λ_∞ = 0.1 y λ'_0 = 10

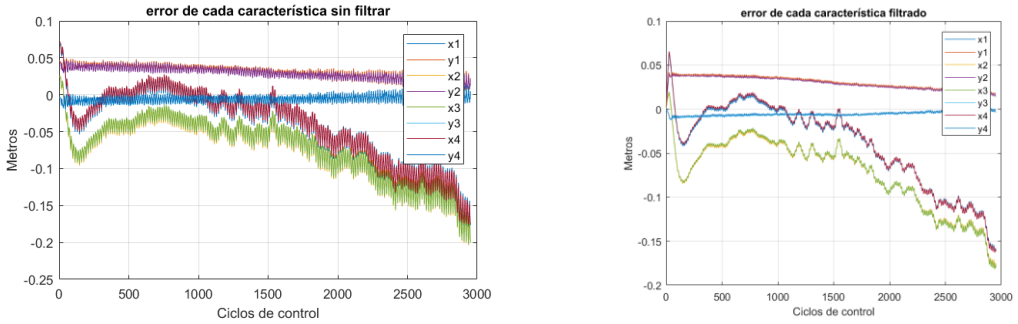

Figura 56: Evolución de la señal de error para el caso λ_0 = 1, λ_∞ = 0.1 y λ'_0 = 10

Como podemos observar, ni el error ni la velocidad acabarán en cero, y no sólo eso, sino que además veremos que al final podemos ver que la velocidad acaba por divergir, lo cual nos termina de indicar que el controlador ha perdido la referencia y no ha conseguido ajustarla al final. La Fig. 57 muestra el seguimiento de las referencias en imagen.

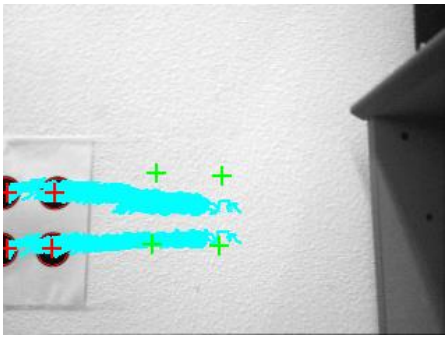

Figura 57: Seguimiento de las referencias visuales en imagen del robot

Con esta imagen, vemos cómo el robot finalmente ha realizado un movimiento brusco doblando, por el cual ha perdido la referencia, y por el que no ha sido capaz de situarse en la localización final que le correspondía. En la Fig. 58 se muestra la localización final alcanzada por el robot al perder las referencias.

Figura 58: Pose final en perspectiva y planta de nuestro robot para $\lambda_0 = 1$, $\lambda_\infty = 0.1$ y $\lambda'_0 = 10$

Como podemos observar si vemos la Fig. 50, nuestro robot ha quedado desplazado a la derecha de la posición que debería haber obtenido, y además girado, sin la capacidad de poder ver las referencias visuales a las cuales debería de seguir.

Finalmente, de este caso podemos observar en la Fig. 59 la matriz de interacción final que hemos obtenido, y el error final de cada par de coordenadas de las características que íbamos siguiendo.

```
Interaction Matrix Ls
-4.939198309  0  -1.536029011  0.003877051174  -1.096713235  -0.01246690294
0  -4.939198309  -0.06157650594  1.000155424  -0.003877051174  0.3109875156
-5.022281695  0  -1.277246028  0.002869711967  -1.064676571  -0.01128404517
0  -5.022281695  -0.05667165352  1.00012733  -0.002869711967  0.2543158878
-5.035626836  0  -1.299386433  -0.01432638162  -1.066583953  0.0555202901
0  -5.035626836  0.2795794627  1.003082503  0.01432638162  0.2580386664
-4.951557308  0  -1.547259233  -0.0174121179  -1.097643322  0.05572246579
0  -4.951557308  0.2759129827  1.003104993  0.0174121179  0.3124793144
Error vector (s-s*)
-0.1770090942  0.02239946672  -0.1990959999  0.02103526204  -0.1998653433  0.003141932882  -0.1748489959  0.004957525711
Gain : Zero= 1.1        Inf= 0.1        Slope= 10
```

Figura 59: Matriz de interacción y errores para el caso $\lambda_0 = 1$, $\lambda_\infty = 0.1$ y $\lambda'_0 = 10$

Como podemos observar, en muchos casos el error será muy elevado, viendo así que esta ganancia no es válida, ya que con ella no somos capaces de alcanzar las referencias que habíamos marcado en un principio.

3.2.2. Resultados para λ_0= 4.10, λ_∞= 0.4 y λ'_0= 40

En este caso, como ya habíamos avecinado previamente, sí que alcanzaremos la referencia final. Así, tendremos que dividir nuestros datos obtenidos entre la prueba con la trayectoria lineal y la de la trayectoria curva, para así, más adelante, poder comparar éstas con sus respectivas iguales de otras ganancias.

Trayectoria lineal

En primer lugar, podemos observar en la Fig. 60 la evolución la velocidad.

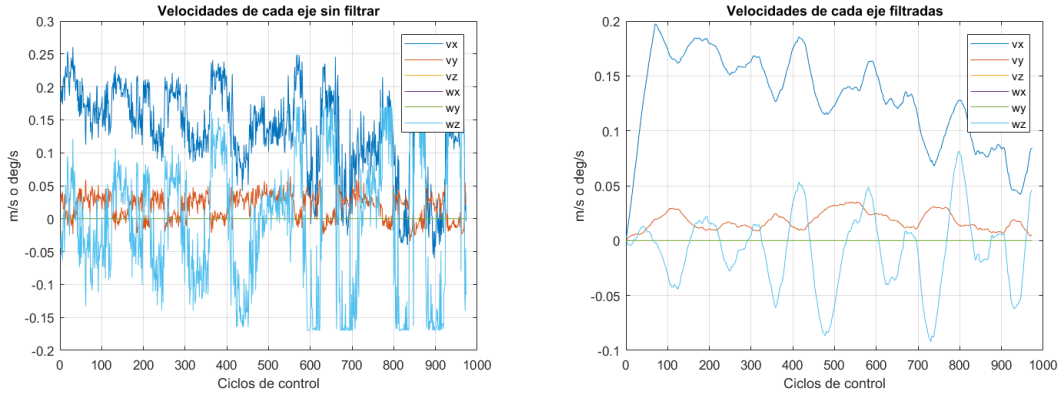

Figura 60: Evolución de la señal de velocidad para λ_0 = 4.1, λ_∞ = 0.4 y λ'_0= 40 con trayectoria lineal

En la Fig. 61 se muestra la evolución del error.

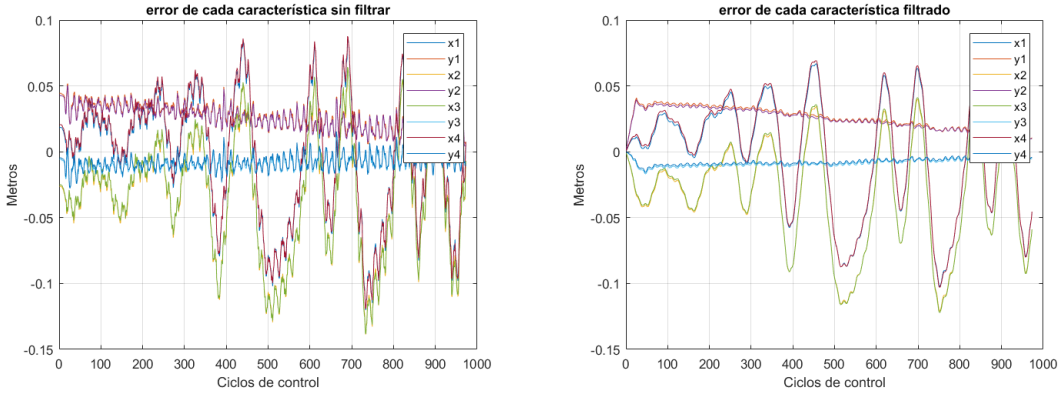

Figura 61: Evolución de la señal de error para λ_0 = 4.1, λ_∞ = 0.4 y λ'_0= 40 con trayectoria lineal

Como podremos observar más adelante, estos serán los datos de todas las ganancias que hemos realizado que más lentos son, viendo que son los que más ciclos tarda en converger a cero. Pese a esto, podemos ver que aun así será el controlador más exacto, dejando nuestro robot lo más cercano que hemos podido a las referencias reales que habíamos tomado. Esto lo podremos observar desde la imagen del seguimiento de la trayectoria desde la perspectiva del robot, la cual se muestra en la Fig. 62.

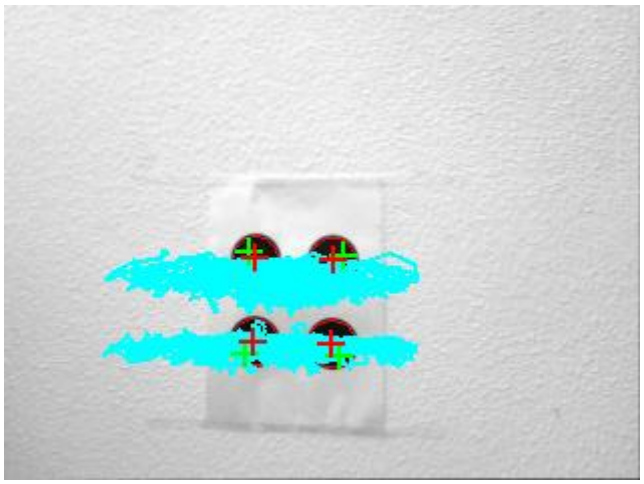

Figura 62: Seguimiento de las referencias visuales en imagen del robot para $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria lineal

En cuanto vayamos observando el resto de los resultados, podremos ver que este conjunto de movimientos es el más acotado que tendremos, viendo así que los movimientos bruscos que ha realizado nuestro robot han sido menos que en otros casos que estudiaremos más adelante, y considerando así que esta ganancia será la que más precisión nos dé a la hora de ajustar las referencias de nuestro robot con las referencias finales que queremos ajustar.

Además, podemos ver en la Fig. 63 la posición final a la que ha llegado nuestro robot tras realizar todo el movimiento con el control.

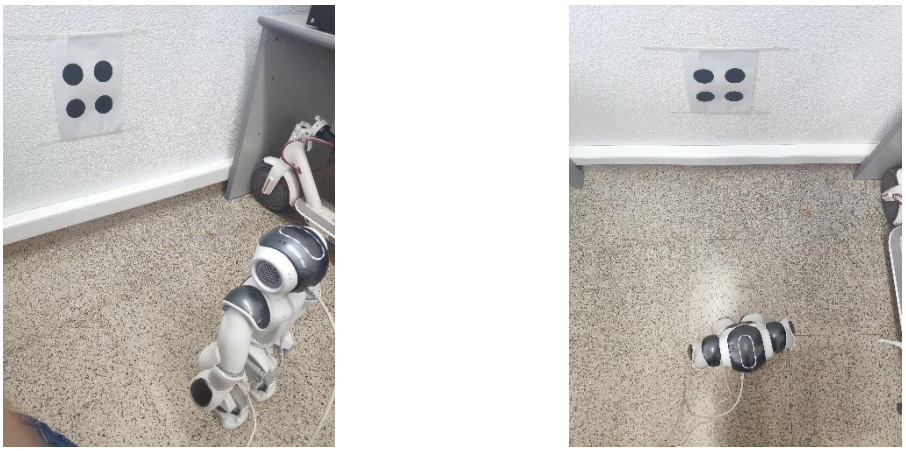

Figura 63: Pose final en perspectiva y planta de nuestro robot para $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria lineal

Como veremos más adelante, esta será la pose que logremos más aproximada a la posición inicial con la cual habíamos tomado las referencias de nuestros patrones en primer lugar. Finalmente, la Fig. 64 muestra los resultados de matriz de interacción y errores obtenidos en este experimento.

```
Interaction Matrix Ls
-5.204428347  0  -0.5636168356  0.002862598661  -1.011727943  -0.02643318771
0  -5.204428347  -0.1375696314  1.000698713  -0.002862598661  0.1082956279
-5.22366042  0  -0.2299361355  0.001107004746  -1.001937602  -0.02514879562
0  -5.22366042  -0.1313687683  1.000632462  -0.001107004746  0.04401820123
-5.179710751  0  -0.2309842344  -0.001980252452  -1.001988629  0.04440621216
0  -5.179710751  0.2300113345  1.001971912  0.001980252452  0.04459404115
-5.160333473  0  -0.5634081162  -0.004683240778  -1.011920395  0.04289445511
0  -5.160333473  0.2213496925  1.001839934  0.004683240778  0.1091805635
Error vector (s-s*)
0.005136185607  0.005055790964  -0.008770130402  0.003925876033  -0.006953217141  -0.01124360443  0.007450650319  -0.01085754479
Gain : Zero= 4.1        Inf= 0.4        Slope= 40
```

Figura 64: Matriz de interacción y errores para el caso $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria lineal

Trayectoria curvada

En este segundo tipo de trayectoria para la ganancia planteada, observaremos que tendremos la evolución en la velocidad que muestra la Fig. 65, así como la evolución del error que muestra la Fig. 66.

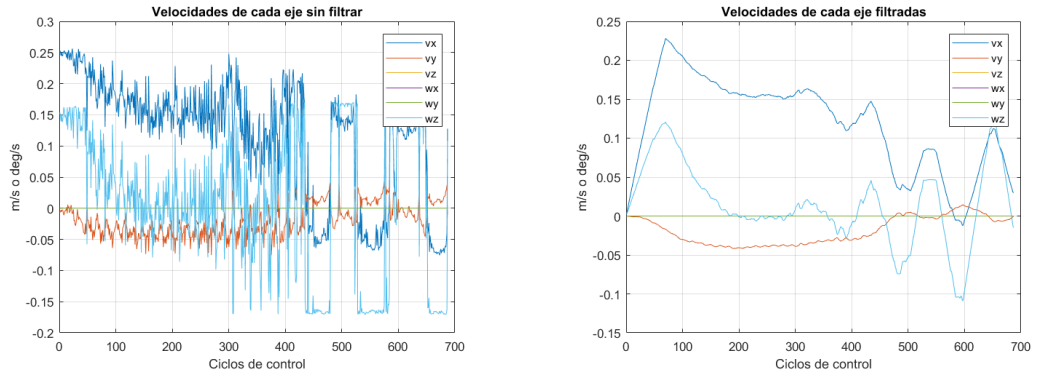

Figura 65: Evolución de la señal de velocidad para $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria curva

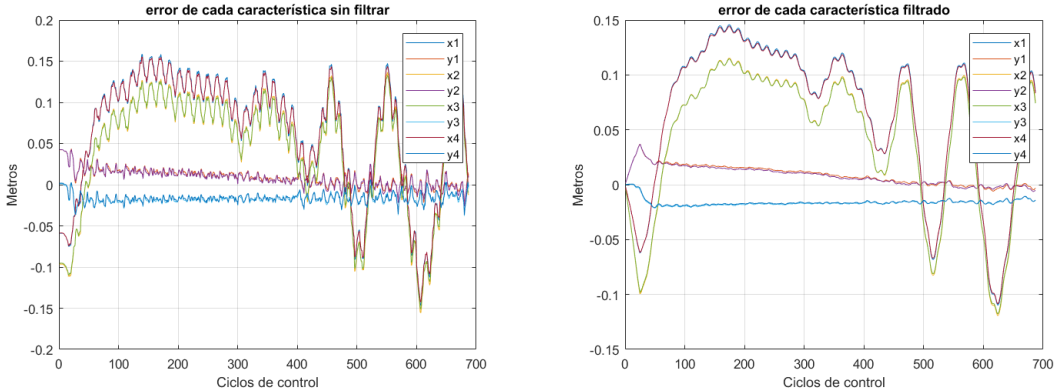

Figura 66: Evolución de la señal de error para $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria curvada

Como podemos observar, encontramos una diferencia notable tanto en velocidad como en error. En primer lugar, en la velocidad, vemos que nuestra velocidad angular para el eje z es bastante más elevada que en el caso de la trayectoria lineal, siendo esto debido a cómo tendrá que corregir nuestro robot su comportamiento de movimiento para ajustar en giro las características que queremos seguir dentro de la imagen. En cuanto al error, vemos que también varía bastante, en las coordenadas x de la imagen, siendo la variación mayor que en el caso lineal, debido esto a que el ajuste de error en este eje será más complicado debido a los movimientos de giro para ajustarse que tendrá que hacer nuestro robot hasta llegar a una referencia válida.

Conociendo estos datos, en cuanto a la referencia de la imagen que tendremos para este caso, podemos ver que tendrá la forma final que se muestra en la Fig. 67.

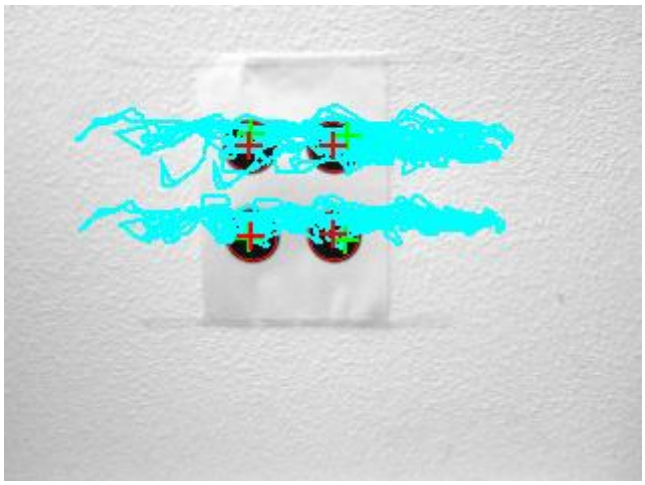

Figura 67: Seguimiento de las referencias visuales en imagen del robot para $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria curvada

En este caso, podemos notar una diferencia notable en las zonas donde más ha realizado la trayectoria nuestro robot, viendo cómo, debido a la forma del giro, el robot habrá realizado una trayectoria en la que se enfoca más la parte derecha de nuestras referencias que la parte izquierda hacia la que el robot ha debido de orientarse poco a poco. Otro punto destacable, el cual podremos ver reflejado más adelante, es que en este caso también tendremos las referencias más acotadas en la imagen que con ganancias más grandes, deduciendo así también para este caso que el robot no ha realizado movimientos muy bruscos, y que así éste ha podido estimar las referencias a las que quería llegar utilizando una actualización de los comandos de velocidad que no es tan agresiva como en otros casos con ganancias más elevadas.

En cuanto a la configuración de posición final a la que llegará nuestro robot, podemos observar que esta tendrá la siguiente forma al acabar de ejecutarse nuestro bucle de control al llegar a la referencia deseada (ver Fig. 68).

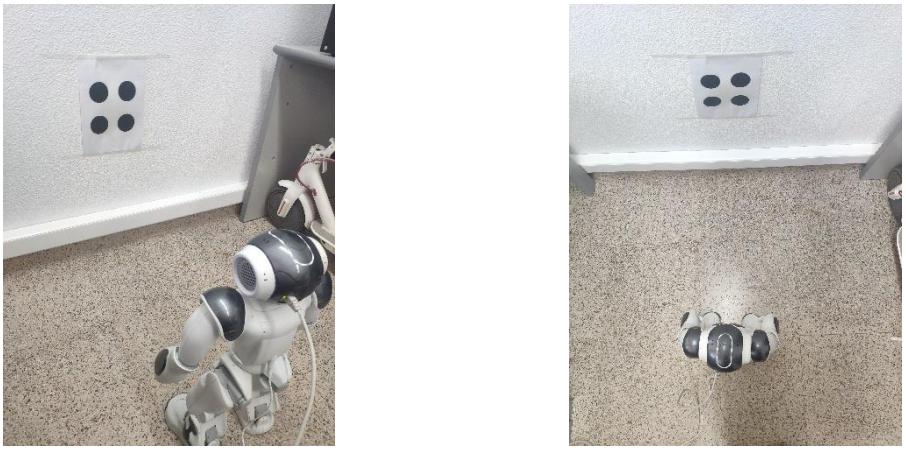

Figura 68: Pose final en perspectiva y planta de nuestro robot para $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria curvada

Como podemos observar, la posición alcanzada será muy parecida a la que hemos llegado previamente, teniendo en ésta simplemente un ligero desviamiento al estar el robot un poco doblado con respecto a las referencias que debía de seguir.

Finalmente, podemos ver que la matriz de interacción, con el error asociado del controlador que hemos realizado con esta ganancia, tendrá la forma final mostrada en la Fig. 69 tras realizar todos los movimientos necesarios.

```
Interaction Matrix Ls
-5.3880791  0  -0.6276837832  0.01432420267  -1.013571064  -0.1229599029
0  -5.3880791  -0.6625176831  1.015119138  -0.01432420267  0.1164949088
-5.363511061  0  -0.2515878433  0.005765998746  -1.002200295  -0.1229232607
0  -5.363511061  -0.6593002684  1.015110128  -0.005765998746  0.0469073039
-5.388117143  0  -0.2507839486  0.002292536771  -1.002166334  -0.04925537199
0  -5.388117143  -0.2653937142  1.002426092  -0.002292536771  0.04654389316
-5.412533761  0  -0.621093891  0.005633670232  -1.013167802  -0.04909471945
0  -5.412533761  -0.2657268265  1.002410291  -0.005633670232  0.1147510424
Error vector (s-s*)
-0.0007884331828  0.01087021199  -0.0123445852  0.008682771401  -0.009561073423  -0.00421172895  0.001566980338  -0.002511002374
Gain : Zero= 4.1        Inf= 0.4        Slope= 40
```

Figura 69: Matriz de interacción y errores para el caso $\lambda_0 = 4.1$, $\lambda_\infty = 0.4$ y $\lambda'_0 = 40$ con trayectoria curvada

Como podemos observar, los valores de la matriz cambian levemente con respecto de los valores de la matriz del caso anterior, siendo esto debido a los cambios en las posiciones que hemos realizado para nuestro robot para que este realizara la trayectoria curvada que deseábamos en un principio.

Viendo que con la ganancia adecuada podemos realizar las dos trayectorias que hemos planteado para nuestro robot, podemos pasar a ver algunas ganancias más donde se llega a nuestro objetivo final con estos movimientos, observando cómo estos valores harán que cambien las propiedades de nuestro controlador e insertando significativos cambios a la evolución de su error y de su velocidad.

3.2.3. Resultados para λ_0 = 5.10, λ_∞ 0.5 y λ'_0 = 50

Como podemos imaginar, en este caso también llegaremos a nuestra referencia final, por lo que podremos analizar para esta ganancia las diferencias que encontramos para la trayectoria lineal y la curvada que hemos planteado inicialmente. Veremos así, cómo encontraremos en este caso una ganancia que hará que nuestro controlador sea más rápido que los anteriores vistos, pero que también sea más estable que los controladores que veremos más adelante, encontrando con esta ganancia el valor intermedio más valido para realizar otras implementaciones.

Trayectoria lineal

En primer lugar, podemos observar la evolución de la velocidad (ver Fig. 70) y del error (ver Fig. 71).

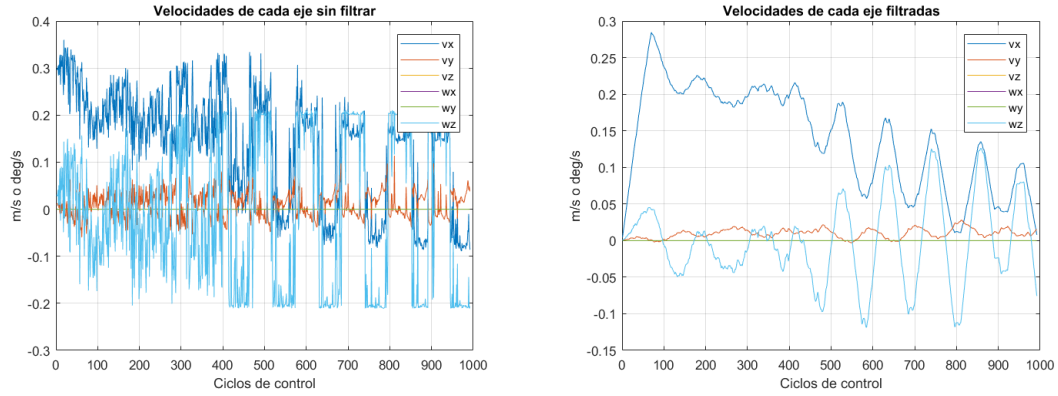

Figura 70: Evolución de la señal de velocidad para λ_0 = 5.1, λ_∞= 0.5 y λ'_0 = 50 con trayectoria lineal

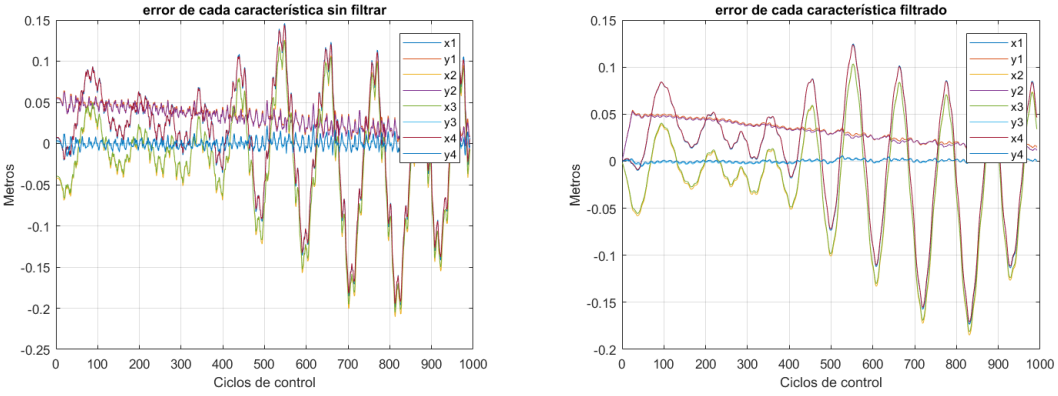

Figura 71: Evolución de la señal de error para λ_0 = 5.1, λ_∞ = 0.5 y λ'_0 = 50 con trayectoria lineal

Como podemos observar, en este caso tendremos que ambas señales evolucionarán más rápido a la referencia en cero pero con más ruido en ambas de ellas y más movimientos bruscos, viendo así lo que queríamos confirmar sobre que la señal de nuestro controlador en este caso avanzará de forma más rápida que en los casos anteriores.

La Fig. 72 muestra la evolución de la trayectoria en la imagen que toma nuestro robot.

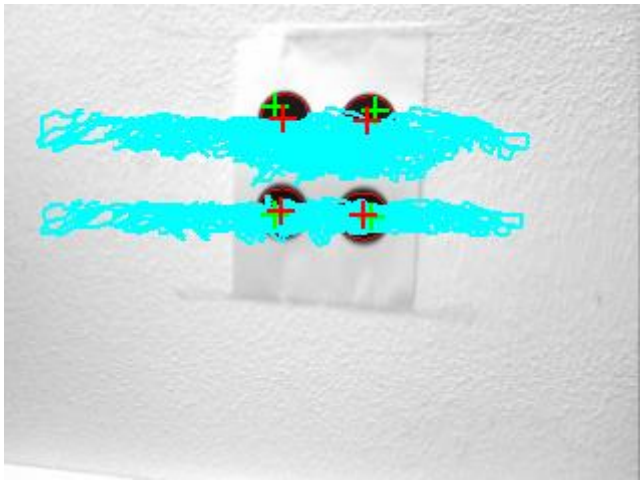

Figura 72: Seguimiento de las referencias visuales en imagen del robot para $\lambda_0 = 5.1$, $\lambda_\infty = 0.5$ y $\lambda'_0 = 50$ con trayectoria lineal

Como podemos observar, en este caso tendremos una dispersión de la trayectoria que seguirán nuestras características en la imagen algo mayor que en el caso anterior, siendo esto debido a los movimientos más bruscos que se obtienen al subir el valor de nuestras ganancias notablemente. Finalmente, la Fig. 73 muestra la posición final a la que llegará nuestro robot tras ser sometido al control.

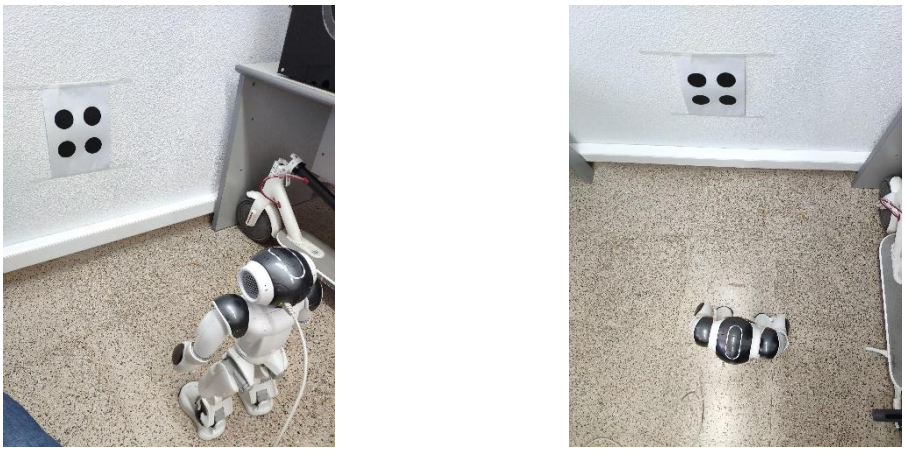

Figura 73: Pose final en perspectiva y planta de nuestro robot para $\lambda_0 = 5.1$, $\lambda_\infty = 0.5$ y $\lambda'_0 = 50$ con trayectoria lineal

Como podemos observar, en este caso tendremos a nuestro robot más acercado a la referencia inicial de la que habíamos partido, siendo esto debido a que, al evolucionar las velocidades de forma distinta, estas crecen más rápido en posiciones finales, haciendo que el robot termine por bajar las velocidades a cero de forma más lenta que con las ganancias estudiadas en los casos anteriores.

En cuanto a la matriz de interacción conseguida y los errores que se ha obtenido sobre las referencias que hemos cogido, podemos ver que esta tendrá la siguiente forma (ver Fig. 74).

```
Interaction Matrix Ls
-5.461450268  0  -0.4598428601  0.01211948448  -1.007089293  -0.1439403924
0  -5.461450268  -0.7861232947  1.020718837  -0.01211948448  0.08419793966
-5.557594176  0  -0.08223470762  0.002099716913  -1.000218946  -0.1419032769
0  -5.557594176  -0.7886408252  1.02013654  -0.002099716913  0.01479681765
-5.523698738  0  -0.105253552  0.001203574985  -1.000363089  -0.06316352747
0  -5.523698738  -0.348896297  1.003989631  -0.001203574985  0.01905490451
-5.431340249  0  -0.4694305678  0.005759574428  -1.007470139  -0.06663862679
0  -5.431340249  -0.3619370559  1.004440707  -0.005759574428  0.0864299687
Error vector (s-s*)
0.006796983438  0.008756986687  -0.006941812461  0.008555885165  -0.007810674773  -0.001632053447  0.00553818857  -0.003205773398
Gain : Zero= 5.1        Inf= 0.5        Slope= 50
```

Figura 74: Matriz de interacción y errores para el caso $\lambda_0 = 5.1$, $\lambda_\infty = 0.5$ y $\lambda'_0 = 50$ con trayectoria lineal

De estos datos debemos destacar que, sobre los errores, podremos ver que estos serán algo mayores que los que teníamos con la ganancia anterior, siendo esto debido a que este controlador será algo menos exacto debido a la subida de la ganancia que hará que nuestro robot se mueva de manera más acelerada que en el anterior experimento.

Trayectoria curvada

En este segundo experimento para la ganancia que hemos planteado, podemos observar que la evolución de las señales que nos interesan tendrá la siguiente forma con respecto al movimiento que hemos realizado (ver Fig. 75 y Fig. 76).

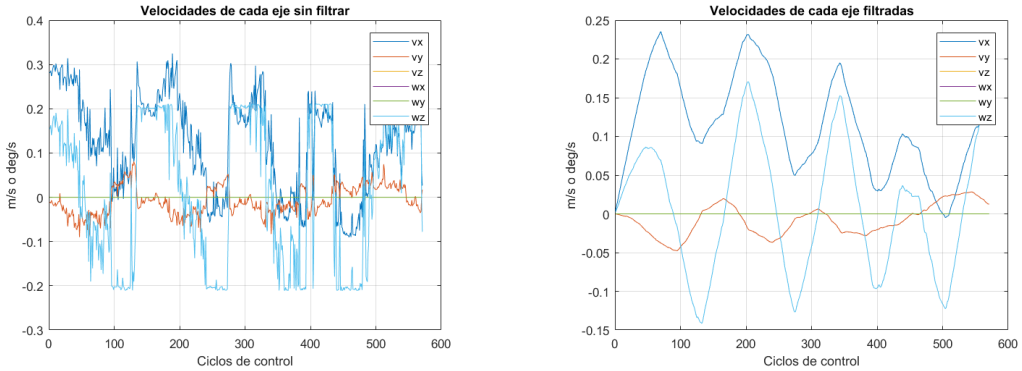

Figura 75: Evolución de la señal de velocidad para $\lambda_0 = 5.1$, $\lambda\infty = 0.5$ y $\lambda'_0 = 50$ con trayectoria curvada

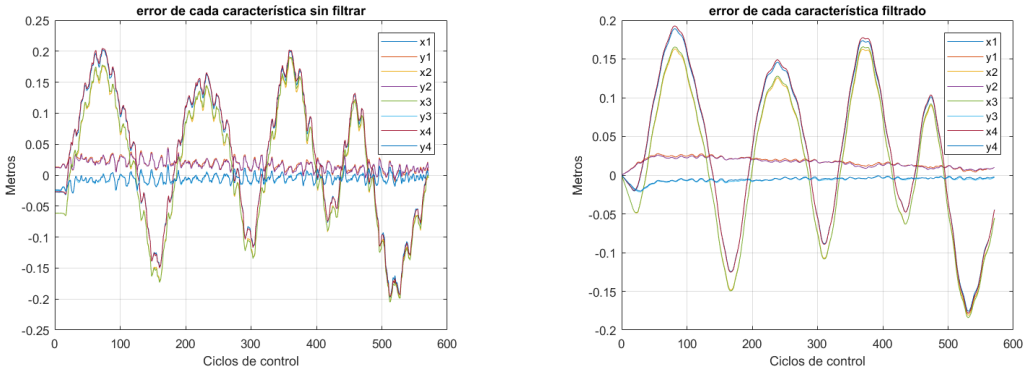

Figura 76: Evolución de la señal de error para $\lambda_0 = 5.1$, $\lambda_\infty = 0.5$ y $\lambda'_0 = 50$ con trayectoria curvada

Como podemos observar, en este caso otra vez tendremos una variación mayor en ambas señales, siendo esto debido tanto al tipo de trayectoria y cómo deberemos de corregir ésta, y además a las ganancias más elevadas que hemos empleado. De igual manera que en el caso lineal, también podemos ver que nuestras señales evolucionarán de forma más rápida, pues ésta será la influencia que la ganancia nueva causará en nuestro controlador.

En cuanto a las referencias en imagen, podemos ver su evolución en la Fig. 77 con la ganancia descrita y la trayectoria curvada.

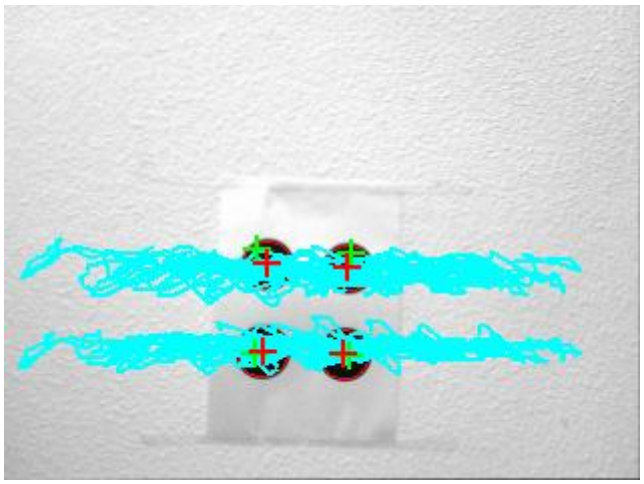

Figura 77: Seguimiento de las referencias visuales en imagen del robot para $\lambda_0 = 5.1$, $\lambda_\infty = 0.5$ y $\lambda'_0 = 50$ con trayectoria curvada

Igual que en la anterior trayectoria curvada, podemos ver cómo las referencias se moverán más por una zona de nuestra imagen que por otra, siendo esto debido a la trayectoria que hemos realizado. Pese a esto, cabe destacar que se ve que en la zona donde la trayectoria se ha movido menos, ahora vemos más movimientos, siendo esto debido a la evolución más rápida obtenida al subir las ganancias con las que trabajamos.

La posición del robot que se obtiene finalmente se muestra en la Fig. 78.

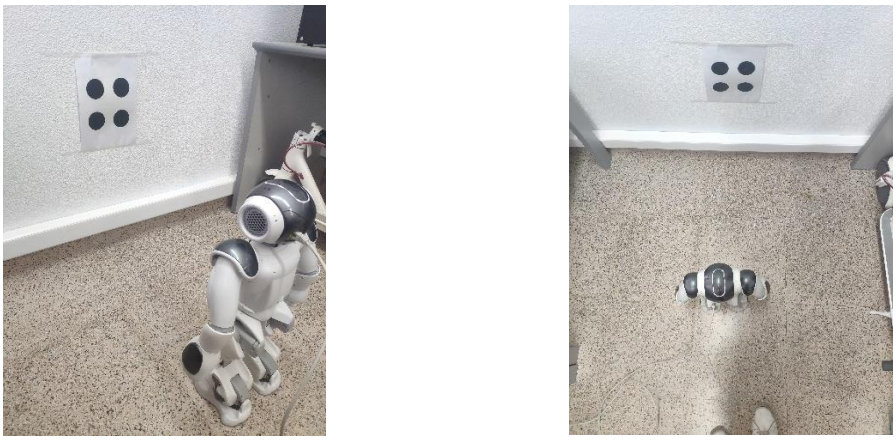

Figura 78: Pose final en perspectiva y planta de nuestro robot para $\lambda_0 = 5.1$, $\lambda_\infty = 0.5$ y $\lambda'_0 = 50$ con trayectoria curvada

En este caso, para la trayectoria distinta, podemos ver que, aun así, igualmente conseguimos con ésta que nuestro robot se aproxime más que en el caso de la ganancia anteriormente analizada, viendo con esta ganancia que tenemos una muy buena aproximación para tener un controlador fiable, que sea rápido y a la vez preciso tanto en posición final como en referencias de la imagen.

La Fig. 79 muestra la matriz de interacción y los errores que obtendremos finalmente.

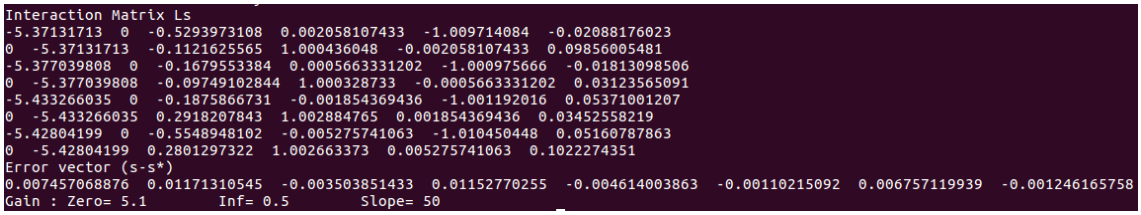

Figura 79: Matriz de interacción y errores para el caso λ_0 = 5.1, λ_∞ = 0.5 y λ'_0 = 50 con trayectoria curvada

Como podemos ver en este caso también, los valores de error sobre imagen serán algo más grandes que en el caso de la ganancia anterior, siendo esto debido a la misma razón que con el movimiento anteriormente definido, pues la evolución más rápida de nuestras señales hará que sea más difícil que las referencias acaben confluyendo en el punto preciso al que queremos llegar.

3.2.4. Resultados para λ_0 = 6.10, λ_∞= 0.6 y λ'_0 = 60

En este caso, podemos observar que la evolución de las características que analicemos en cada caso será más rápida debido a la subida de la ganancia que nos dará un aumento en la velocidad del controlador, pero esto lo realizaremos a costa de sacrificar más exactitud en el error final que tenemos de nuestro robot, viendo así además cómo la evolución de ambas señales serán más abruptas que en los casos anteriores debido a la evolución más rápida de los datos con los que vamos a trabajar.

Cabe destacar que este será el último caso que estudiemos de evoluciones de ganancia con las cuales conseguimos que nuestro controlador llegue a su referencia final en estático, pudiendo ver en las experimentaciones que tendremos más adelante, cómo si obtenemos una ganancia demasiado elevada, no conseguiremos que el robot llegue a su objetivo debido a que los movimientos que nuestro sistema realiza son tan bruscos que perderemos las referencias con las cuales se supone que tenemos que guiar a nuestro robot.

Trayectoria lineal

En primer lugar, para este caso con la trayectoria lineal, la Fig. 80 muestra la evolución de la señal de velocidad, así como la Fig. 81 muestra la evolución del error en imagen para el experimento realizado.

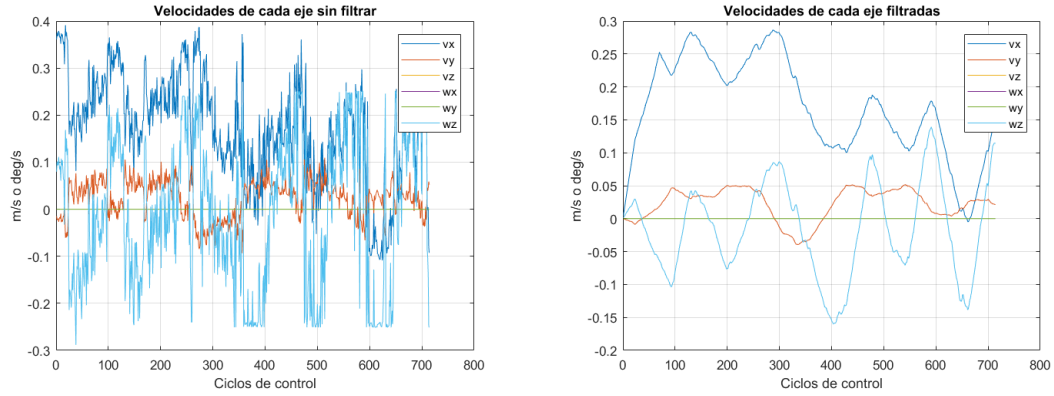

Figura 80: Evolución de la señal de velocidad para λ_0 = 6.1, λ_∞= 0.6 y λ'_0 = 60 con trayectoria lineal

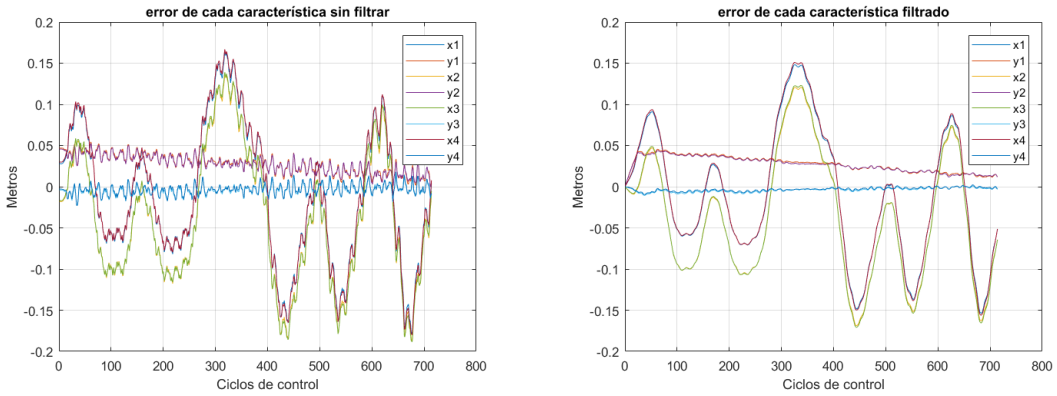

Figura 81: Evolución de la señal de error para λ_0 = 6.1, λ_∞ = 0.6 y λ'_0 = 60 con trayectoria lineal

En cuanto a la señal de velocidad cabe destacar que, aunque fuese esperable que ésta evolucionase de manera más rápida que en el caso anterior, vemos que el hecho de que nuestro robot se mueva más rápido, con movimientos más bruscos, hace que finalmente la compensación que tenemos que ir realizando de ésta será más abrupta, consiguiendo una señal así que hace que las velocidades que tenemos vayan variando mucho.

En cuanto a la señal de error, en este caso sí que podemos ver que evoluciona de manera aún más rápida, viendo así que la ejecución de nuestro bucle de control pese a todo terminará siendo la más rápida que tengamos en nuestra experimentación para los casos en estático donde nuestro robot alcanza la referencia. Probaremos ganancias más adelante que en principio deberían hacer que nuestro controlador avanzase de manera más rápida aún, pero como hemos dicho anteriormente, el hecho de tener movimientos tan bruscos acabará haciendo que finalmente perdamos la referencia de las señales que queríamos seguir y no podamos continuar con nuestro bucle de control.

En cuanto a la evolución de las referencias que vamos a obtener desde nuestro robot, podemos observar que los resultados que obtendremos serán los mostrados en la Fig. 82.

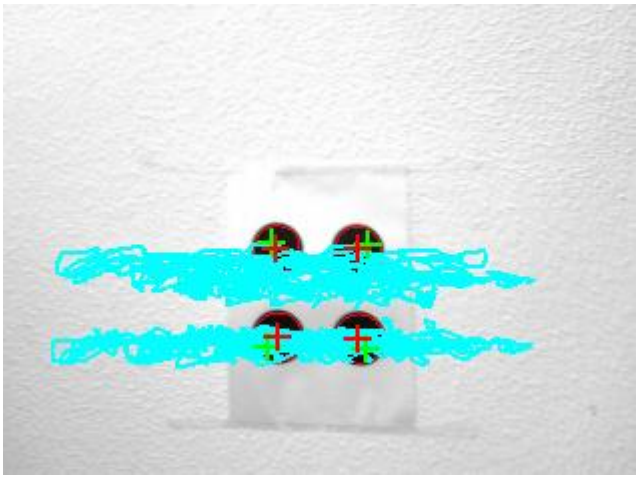

Figura 82: Seguimiento de las referencias visuales en imagen del robot para $\lambda_0 = 6.1$, $\lambda_\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria lineal

Como podemos observar, en este caso la dispersión que tendremos en la trayectoria será la más elevada que vamos a conseguir en todos los casos en los que nuestro robot alcance la referencia estática, viendo así de forma más gráfica el efecto que tendrá el hecho de que nos encontremos con velocidades más bruscas que las que teníamos en casos anteriores.

En cuanto a cómo afectará este movimiento más brusco a la posición final de nuestro robot, podemos ver en la Fig. 83 que este hará que nuestra máquina llegue a la siguiente posición.

Figura 83: Pose final en perspectiva y planta de nuestro robot para $\lambda_0 = 6.1$, $\lambda_\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria lineal

Como podemos observar, la pose final no será tan cercana como en el caso de la ganancia revisada anteriormente, siendo esto debido a que el cambio brusco de velocidades hará que no seamos capaces de obtener referencias más finas, y además hará que nuestro robot acabe por realizar movimientos hacia atrás por la velocidad de actualización de comandos que posicionarán a nuestro sistema más alejado de lo que habíamos planteado como objetivo inicialmente.

En cuanto a la matriz de interacción y los errores que hemos obtenido finalmente, se muestran en la Fig. 84.

```
Interaction Matrix Ls
-5.41939331  0  -0.4940639035  0.003078830087  -1.008311222  -0.03377172681
0  -5.41939331  -0.1830222703  1.00114053  -0.003078830087  0.09116590645
-5.482945782  0  -0.130926632  0.0007776874364  -1.000570201  -0.03256799616
0  -5.482945782  -0.1785685572  1.001060674  -0.0007776874364  0.02387888505
-5.423790375  0  -0.1230105849  -0.0009300000007  -1.000514374  0.04100561798
0  -5.423790375  0.2224058761  1.001681461  0.0009300000007  0.02267981917
-5.36093739  0  -0.482634914  -0.003570439574  -1.008105056  0.03965917603
0  -5.36093739  0.2126103596  1.00157285  0.003570439574  0.09002808257
Error vector (s-s*)
0.003942948768  0.004912397224  -0.008595240893  0.00397096396  -0.005483567971  -0.008663129451  0.007641306289  -0.008091179942
Gain : Zero= 6.1        Inf= 0.6        Slope= 60
```

Figura 84: Matriz de interacción y errores para el caso $\lambda_0 = 6.1$, $\lambda_\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria lineal

Como podemos observar, el error otra vez habrá aumentado en casi todos los casos, y tendremos así la razón por la que la trayectoria es más movida y el comportamiento de la gráfica con la que hemos estudiado los valores finales de error.

Trayectoria curvada

En cuanto a la trayectoria curvada que hemos planteado para esta ganancia, podemos observar que nos dará la siguiente forma para la evolución de las señales de velocidad y error en cada caso (ver Fig. 85 y Fig. 86).

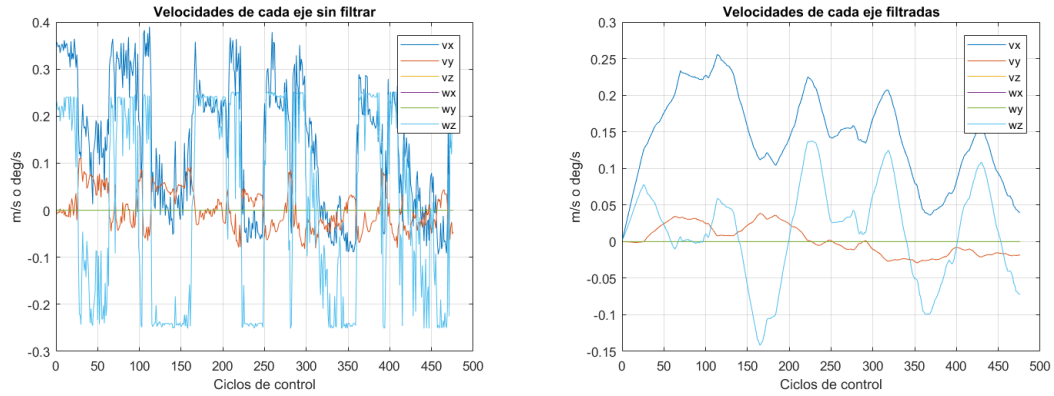

Figura 85: Evolución de la señal de velocidad para $\lambda_0 = 6.1$, $\lambda_\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria curvada

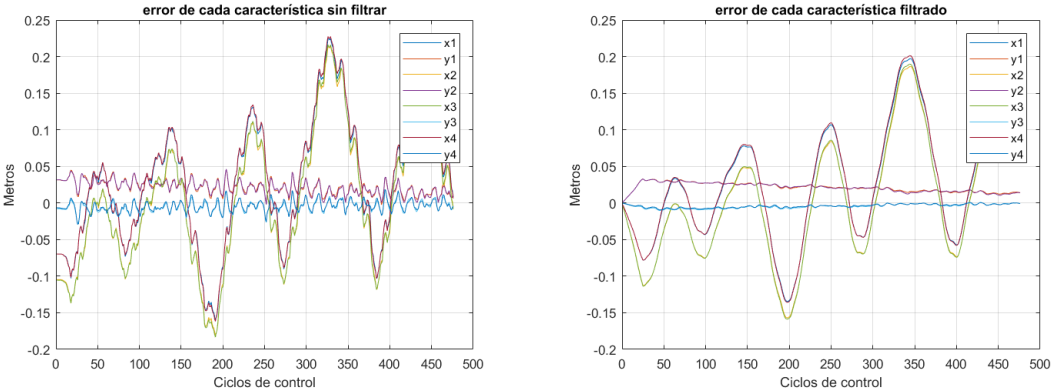

Figura 86: Evolución de la señal de error para $\lambda_0 = 6.1$, $\lambda_\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria lineal

Como podemos observar, otra vez nos encontramos con una mayor variación en ambas señales, siendo esto debido a la suma de la distorsión de las velocidades más elevadas más el hecho de estar haciendo una trayectoria en la cual no tenemos a primer golpe de vista las referencias que queremos que nuestro robot siga. En la Fig. 87 podemos observar la evolución de las características visuales en el plano imagen.

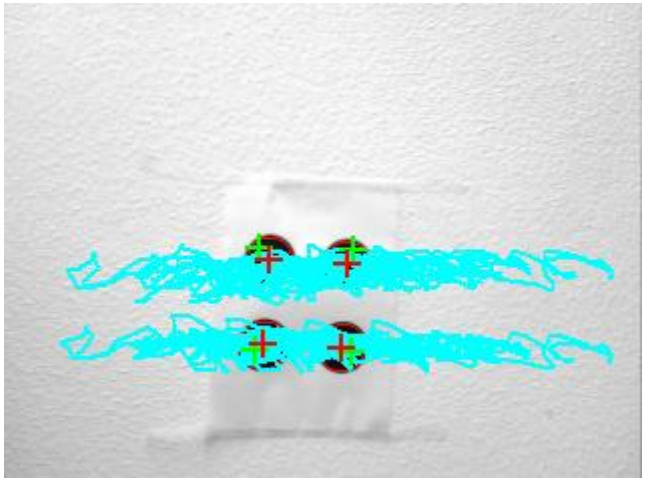

Figura 87:Seguimiento de las referencias visuales en imagen del robot para $\lambda_0 = 6.1$, $\lambda_\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria curvada

Como podemos observar, otra vez tendremos la trayectoria más desviada de un lado de la imagen que del otro, siendo esto debido otra vez al tipo de trayectoria que realiza nuestro robot. Otro punto destacable es cómo la dispersión de las trayectorias en este caso llega a un máximo, viendo cómo la línea azul que indica la trayectoria seguida por las referencias casi va de lado a lado de la imagen, y siendo esto debido al tipo de movimiento que hemos realizado y además a la ganancia más elevada con la que trabajamos.

En cuanto a la posición final que alcanzaremos de nuestro robot con dicha ganancia y la trayectoria definida, podemos ver que nuestro robot llegara a la pose final mostrada en la Fig. 88.

Figura 88: Pose final en perspectiva y planta de nuestro robot para $\lambda_0 = 6.1$, $\lambda_\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria curvada

Como podemos observar, este será el caso en el que nuestro robot más girado quede, debido esto a las altas evoluciones de las velocidades que tenemos con las ganancias propuestas, y al tipo de trayectoria que realizamos, haciendo así que el robot no sea capaz de colocarse frente a las características debido al movimiento constante de actualización que éste realiza por la ganancia que hemos implementado.

Finalmente, la Fig. 89 muestra la matriz de interacción final y el vector de error de las características con las que trabajamos.

```
Interaction Matrix Ls
-5.269084494  0  -0.51849807  0.002487217299  -1.009683313  -0.02527561599
0  -5.269084494  -0.1331793563  1.000638857  -0.002487217299  0.09840382529
-5.192814874  0  -0.174924197  0.0007460445063  -1.001134734  -0.02214714188
0  -5.192814874  -0.1150060078  1.000490496  -0.0007460445063  0.03368581419
-5.151929308  0  -0.1969293496  -0.001816291225  -1.001461104  0.04751655359
0  -5.151929308  0.244801925  1.002257823  0.001816291225  0.03822438892
-5.227656761  0  -0.539284973  -0.00468830013  -1.010641982  0.04544688819
0  -5.227656761  0.2375807323  1.00206542  0.00468830013  0.1031599812
Error vector (s-s*)
0.008262819822  0.01055769848  -0.005804898179  0.01167682848  -0.008483674102  -0.004132252953  0.00628076181  -0.00460339735
Gain : Zero= 6.1        Inf= 0.6        Slope= 60
```

Figura 89: Matriz de interacción y errores para el caso $\lambda_0 = 6.1$, $\lambda_\infty = 0.6$ y $\lambda'_0 = 60$ con trayectoria curvada

Como podemos ver, este será el caso en el que encontramos errores más grandes dentro de nuestro vector, siendo esto debido a todas las características que hemos ido destacando en este punto sobre la trayectoria curvada y el tipo de evolución que realizan las características debido a la ganancia que hemos utilizado en este caso.

3.2.5. Resultados para $\lambda_0 = 10.1$, $\lambda_\infty = 1$ y $\lambda'_0 = 100$

En este caso, observaremos cómo al utilizar ganancias demasiado elevadas, el comportamiento de nuestro sistema pasa a ser excesivamente inestable, haciendo que nuestro robot se mueva en demasía y causando así que éste acabe por perder las referencias que debería de seguir en todo momento. Puesto que con esta ganancia no podremos realizar un seguimiento de ninguna forma, en este caso tan solo estudiaremos el intento de realizar un movimiento lineal, obteniendo con este las siguientes gráficas de evolución de la velocidad (Fig. 90) y el error (Fig. 91).

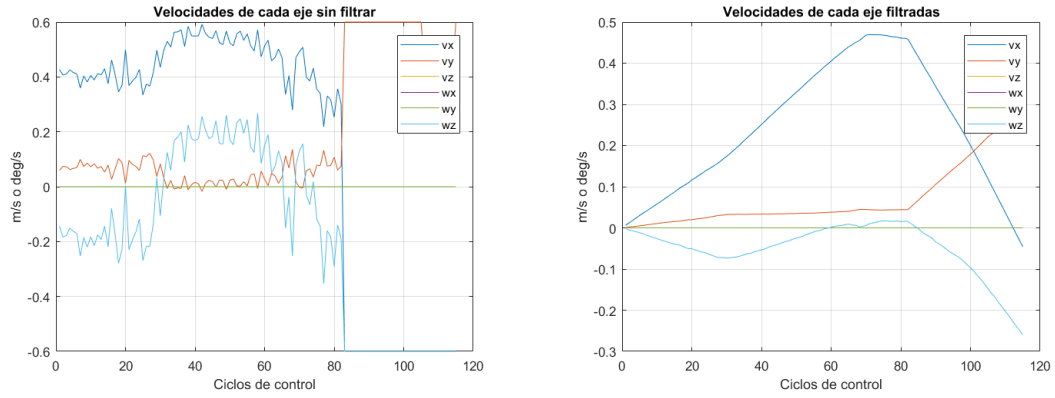

Figura 90: Evolución de la señal de velocidad para el caso $\lambda_0 = 10$, $\lambda_\infty = 1$ y $\lambda'_0 = 100$

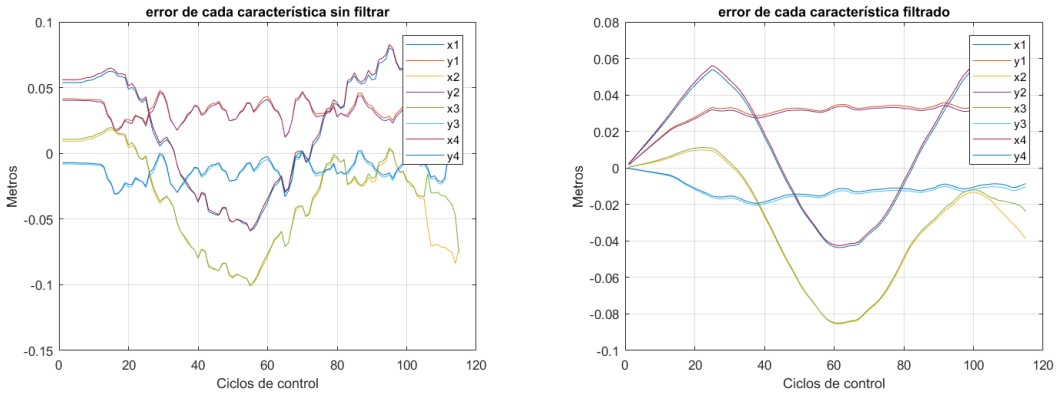

Figura 91: Evolución de la señal de error para el caso $\lambda_0 = 10$, $\lambda_\infty = 1$ y $\lambda'_0 = 100$

Como podemos ver, en este caso los ciclos de control serán pocos, ya que acabaremos rápido nuestro bucle de control al acabar perdiendo las referencias que queríamos seguir. También es destacable el hecho de cómo las velocidades y los errores acaban por irse al infinito en el momento en el que perdemos nuestra referencia, indicando que no seremos capaces de ninguna forma de llegar a las referencias que habíamos marcado inicialmente.

En cuanto a la evolución de las características en la imagen de la cámara de nuestro robot, podemos verla en la Fig. 92.

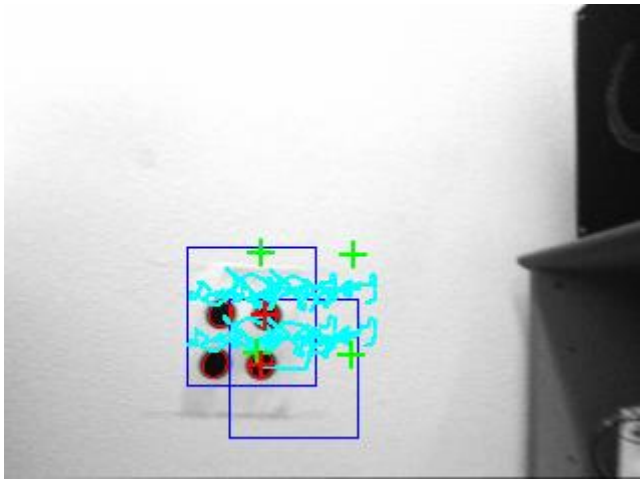

Figura 92: Seguimiento de las referencias visuales en imagen del robot para $\lambda_0 = 10$, $\lambda_\infty = 1$ y $\lambda'_0 = 100$

Como podemos observar, la trayectoria no será realizada, ya que no podremos ejecutar mucho el bucle de control de nuestro robot al perder las referencias. Otro punto destacable de esta fotografía es el poder ver cómo las referencias que mantenemos se quedan marcadas con una cruz en rojo, mientras que las que hemos perdido dejan de tener dicha cruz, indicando que nuestro controlador ha perdido el seguimiento de dichos puntos antes de tiempo y que ya no puede estimar los cálculos necesarios para llegar a dicha referencia.

La Fig. 93 muestra claramente que la posición final del robot está muy alejada de la deseada.

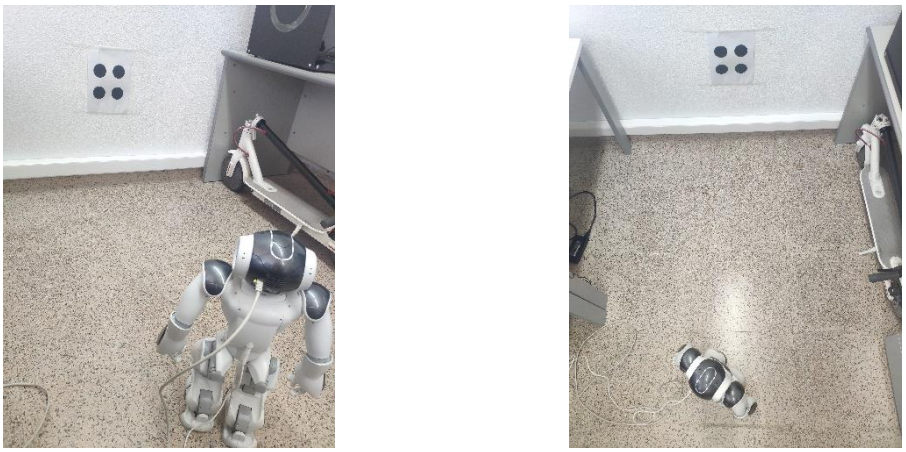

Figura 93: Pose final en perspectiva y planta de nuestro robot para $\lambda_0 = 10$, $\lambda_\infty = 1$ y $\lambda'_0 = 100$

Como podemos observar, al acabar tan pronto nuestro bucle de control debido a la perdida de referencias, nuestro robot quedará en una posición mucho más atrasada con respecto a nuestra referencia de lo que debería, y además más girado también de lo que debería en un principio, pues al perder las referencias el robot continúa girando hasta que es capaz de frenar las altas velocidades que tiene por las ganancias tan elevadas que hemos utilizado.

La Fig. 94 muestra la matriz de interacción y los errores finales que obtenemos en este caso.

```
Interaction Matrix Ls
-2.974477558  0  -0.3000038466  -0.002241903508  -1.010172607  0.02222802056
0  -2.974477558  0.06611674832  1.000494085  0.002241903508  0.1008593411
-3.087357295  0  -0.3113888225  -0.002241903508  -1.010172607  0.02222802056
0  -3.087357295  0.06862584145  1.000494085  0.002241903508  0.1008593411
-3.093880024  0  -0.3258030134  -0.00661700185  -1.011089278  0.06283615867
0  -3.093880024  0.1944075361  1.003948383  0.00661700185  0.1053056391
-2.979679917  0  -0.3137770981  -0.00661700185  -1.011089278  0.06283615867
0  -2.979679917  0.1872316401  1.003948383  0.00661700185  0.1053056391
Error vector (s-s*)
0.004557303654  0.05300100787  -0.07331537087  0.05094209701  -0.07557282634  0.007141433064  0.002937837033  0.008822298085
Gain : Zero= 10.1       Inf= 1  Slope= 100
```

Figura 94: Matriz de interacción y errores para el caso $\lambda_0 = 10$, $\lambda_\infty = 1$ y $\lambda'_0 = 100$

Como podemos observar en primer lugar, la matriz de interacción que obtendremos en este caso será muy distinta a la que hemos ido obteniendo en otras pruebas, siendo esto debido a la distancia tan grande que habrá entre nuestro robot y las referencias finalmente. En cuanto al error, también podemos destacar que terminará siendo excesivamente elevado, como hemos podido ver en las figuras destacadas anteriormente.

3.2.6. Experimento con referencia en movimiento

Finalizando con los experimentos realizados con el controlador basado en imagen, podemos pasar a destacar la prueba hecha moviendo las referencias a las que nuestro robot debe llegar en tiempo real, viendo cómo esto afectará a nuestro controlador en todo momento. Para realizar este experimento, vemos que seguiremos los siguientes pasos con el movimiento de las referencias con respecto a nuestro robot:

- En primer lugar, alejaremos mucho las referencias con las que trabajamos, viendo cómo la velocidad de nuestro robot aumentará para poder acercarse a estas poco a poco, e irá bajando las velocidades conforme se vaya acercando.
- Una vez hayamos comprobado las velocidades elevadas y cómo estas disminuyen en el robot conforme se acercan a la referencia alejada, lo que haremos será acercar la referencia justo enfrente de nuestro robot, haciendo que los puntos de referencia sean visualizados demasiado adelante para el robot, y forzando a este a hacer marcha atrás para adaptar los puntos que quiere que alcance las referencias.
- Finalmente, volveremos a alejar los puntos de referencia un poco de nuestro robot, y los dejaremos estáticos, esperando a que este ajuste los puntos y finalmente termine su bucle de control.

Para realizar este comportamiento, utilizaremos las ganancias destacadas en el punto 3.2.2, pues serán con las que mejor podamos ajustar al final la suma de la velocidad del robot con los movimientos de nuestro robot para que éste no pierda las referencias de las que parte. Sabiendo esto, podemos ver que los resultados que hemos obtenido de respuesta de velocidad (Fig. 95) y error (Fig. 96) tendrán la siguiente forma para este caso.

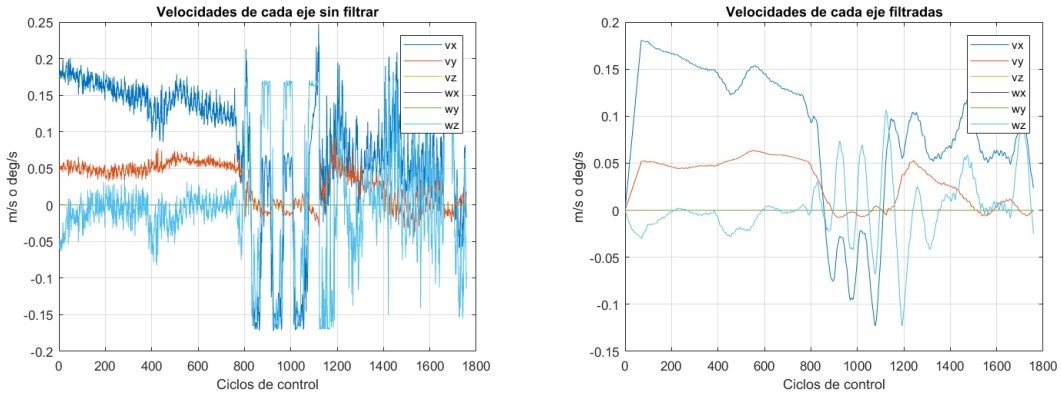

Figura 95: Evolución de la señal de velocidad para el caso de la referencia móvil

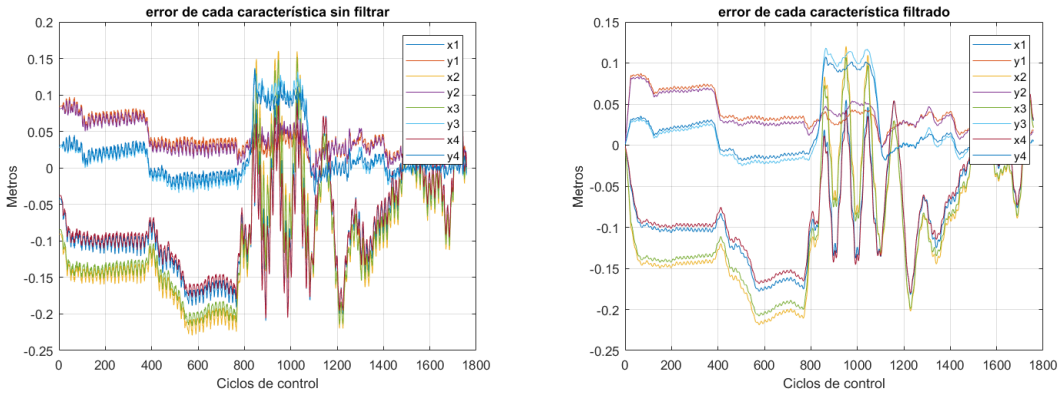

Figura 96: Evolución de la señal de error para el caso de la referencia móvil

Como vemos en las gráficas, observaremos cambios de los ciclos 0 a 800, de 800 a 1200, y de 1200 a 1800 que sería el final de la ejecución cuando el robot alcanza sus referencias. Así, podemos ver que en el primer tramo, tanto las velocidades como el error son muy elevados en principio, pero que van disminuyendo a una buena velocidad para ir acercándose cada vez más a cero, hasta que entramos en el segundo tramo cuando adelantamos nuestra referencia, donde vemos algo curioso, pues mientras que en las velocidades, el movimiento más significativo lo tendremos linealmente en el eje X, lo que significa que nuestro robot se encuentra haciendo marcha atrás para alcanzar las referencias visuales, mientras que en la evolución del error, vemos cómo el error vuelve a aumentar al acercar las referencias, al volver a tener una diferencia grande entre el punto a mover y el fijo de referencia. Finalmente, podremos ver en el último tramo cómo volvemos a un comportamiento marcha adelante de nuestro robot, pero con la aceleración del error y las velocidades a cero mucho más disminuidas debido a que ahora la referencia se encuentra mucho más cercana.

En cuanto a la evolución de la trayectoria de las características visuales en nuestra imagen, se muestra en la Fig. 97.

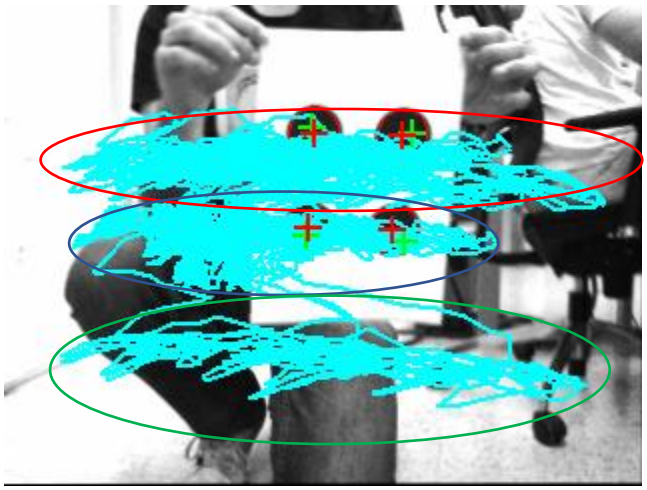

Figura 97: Seguimiento de las referencias visuales en imagen del robot para el caso de la referencia móvil

Como podemos observar, encontramos tres cambios muy bruscos en las trayectorias marcados con elipses de distintos colores, siendo esto debido al acercamiento y el alejamiento de nuestro patrón. Así, cada trayectoria rodeada de cada elipse representa unos ciclos de control, siendo la trayectoria en la elipse azul los primeros ciclos de control, la referencia en verde los ciclos en los que tenemos a nuestro robot pegado a la referencia, y finalmente las trayectorias marcadas en rojo serían nuestros ciclos finales. Vemos así, cómo la distancia de profundidad de nuestros patrones afecta significativamente a cómo vamos a detectar éstos en nuestro robot, pues harán que, en el sistema de referencia de nuestro robot, estos puntos se vean en posiciones distintas de altura en función de la profundidad.

Finalmente, la Fig. 98 muestra la matriz de interacción final obtenida y los valores de error que hemos conseguido con este experimento.

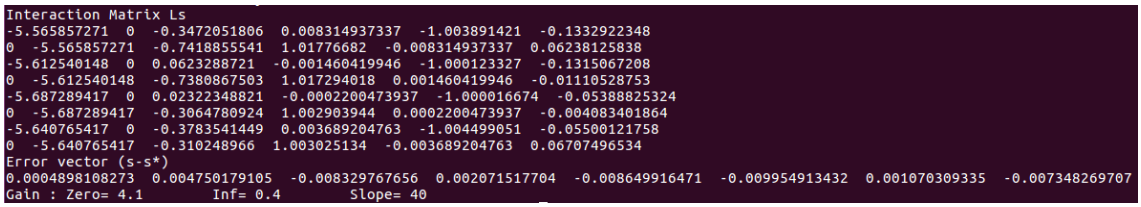

Figura 98: Matriz de interacción y errores para el caso de la referencia móvil

Como podemos observar, en este caso no distaremos mucho de lo visto en el punto 3.2.2, pues la ganancia es la misma, y prácticamente nuestro robot acabará en una pose muy parecida a la que acabó en este experimento destacado.

3.2.7. Resultados obtenidos para el controlador ver y mover

Finalmente, podemos pasar a ver los resultados que hemos obtenido con el controlador ver y mover, partiendo de que éstos son más simples y no tendremos tanta información como hemos tenido en los casos anteriores. Así, podemos ver que si ejecutamos nuestro código con la configuración inicial que habíamos planteado en el punto 3.1.2, veremos que obtendremos los valores de posición a pasarle a nuestro robot desde el programa que se muestran en la Fig. 99.

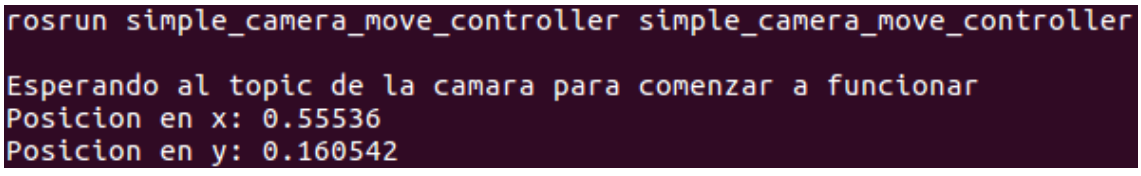

Figura 99: Valores de posición finales obtenidos con el controlador ver y mover

Esta será la posición que le mandemos a nuestro robot para que la alcance lo mejor que pueda y se posicione frente a la referencia que estábamos siguiendo. Así, una vez enviamos estos valores a nuestro robot y éste se mueve a la referencia, podemos ver su posición final con respecto a esta referencia en la Fig. 100.

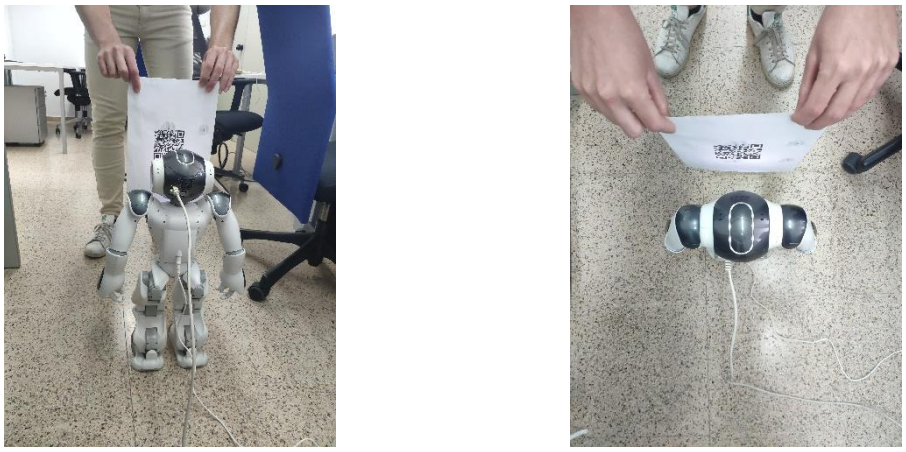

Figura 100: Posición final alcanzada con el controlador de ver y mover

Aunque en el experimento mostrado vemos que se alcanza una posición válida, cabe destacar que analizando bien este método, podemos ver que cualquier aspecto externo al método de control que introduzcamos en nuestro entorno de trabajo hará que este falle, pudiendo ser esto un cambio en la posición de referencia, objetos que se interpongan en la trayectoria y que varíen levemente la posición de nuestro robot, o cualquier caso que desvíe a este de su trayectoria inicial, pues como ya sabemos de este controlador, tan solo utiliza las referencias iniciales que coge de la detección de su destino, por lo que no es capaz de reestimar con realimentación unos nuevos comandos que corrijan la trayectoria de nuestro robot.

3.3. Comparativa de resultados y conclusiones sobre estos

Finalmente, podemos pasar a hablar sobre las conclusiones a las que podemos llegar en función de los resultados en los experimentos y la comparativa que tenemos entre los distintos resultados que hemos ido obteniendo en cada resultado.

En primer lugar, cabe destacar las notables diferencias de resultados entre el controlador simple sin realimentación de ver y mover, y el controlador visual basado en imagen, viendo cómo mientras que en el primero el hecho de no tener una realimentación y la estimación de Qr con pequeñas inexactitudes hace que nuestro robot no pueda terminar de centrarse en el punto óptimo al que queríamos que nuestro sistema llegase, mientras que con el segundo tenemos un método de realimentación que nos hace estimar de forma mucho más exacta la posición final que vamos a obtener en función del movimiento de las referencias visuales con las que estamos trabajando.

Sabiendo esto, vemos cómo hemos centrado más nuestros esfuerzos en tomar datos de este tipo de control, del cual podemos obtener más datos significativos los cuales podemos comprar, y de los cuales podemos sacar conclusiones. Así, de toda la información obtenida de los experimentos de control basado en imagen con la referencia en una localización fija, podemos observar las siguientes comparativas de las cuales podemos obtener ya algunas conclusiones sobre los datos que hemos ido consiguiendo de la experimentación:

- Si comparamos los valores de ganancia que hemos ido utilizando en los puntos 3.2.2., 3.2.3. y 3.2.3., observaremos cómo el aumento de la ganancia hará que nuestros resultados vayan teniendo un comportamiento en velocidad y error con el cual veremos cómo llegamos más rápido desde donde partimos a un valor cero tanto en error como en velocidad, viendo así que conforme subimos la ganancia, tendremos una mayor velocidad a la hora de llegar a nuestras referencias en el control.
- En cuanto a la precisión de los resultados obtenidos en los casos de éxito que hemos mencionado en el anterior punto, podemos ver que aquí tendremos un caso inverso, viendo como nuestro error se acercará más a cero cuanto más pequeña sea nuestra ganancia, viendo esto reflejado tanto en los datos de error como en las imágenes de evolución de la trayectoria de las referencias de nuestros puntos, donde podemos observar cómo la evolución de ésta será más dispersa conforme más ganancia utilicemos en nuestro controlador.
- Viendo los casos de error, también podemos observar unas diferencias notables entre unos casos y otros, pues como podemos ver en estos, si nos quedamos en ganancias muy pequeñas lo que tendremos finalmente es que nuestro robot entrará en un bucle infinito al no poder estimar las velocidades tan pequeñas que le pedimos a nuestra máquina, y finalmente perdiendo las referencias, mientras que con una ganancia demasiado elevada, lo que observamos es que el movimiento de nuestro robot será excesivamente inestable, causando esto que el robot al final acabe por perder las referencias debido a movimientos muy bruscos de éste, lo cual tampoco nos interesa.
- El posicionamiento del robot con respecto a las referencias nos dará resultados muy distintos, pudiendo ver cómo en los casos en los que nuestro robot comenzaba algo más curvado, la trayectoria que éste hacía finalmente era curvada, influyendo esto en el comportamiento de las velocidades y los errores finalmente, y pudiendo ver en las referencias finalmente cómo nuestras trayectorias se movían más en unas zonas de la imagen que en otras, mientras que situábamos nuestro robot de forma completamente recta con respecto a las referencias finales, vemos que la trayectoria final era más lineal, y que las trayectorias de las referencias se quedaban acotadas en una zona más centrada de nuestra imagen.

Una vez hechas estas comparativas, también cabe destacar los resultados que hemos obtenido en el experimento con nuestras referencias en movimiento, viendo en éste claramente como la posición de cada uno de los componentes de nuestro sistema influye en su comportamiento final, y viendo cómo nuestro robot ve afectado sus velocidades en función de si acercábamos o alejábamos nuestras referencias.

Así, teniendo en cuenta todas estas comparativas y datos destacados, podemos ver que algunas de las conclusiones a las que podemos llegar con respecto a las experimentaciones realizadas y los resultados obtenidos, serán las siguientes:

- La estimación de la ganancia en nuestro método de control influirá enormemente en el comportamiento que tengamos finalmente, viendo así que en función de qué valores utilicemos para ésta, tendremos más exactitud o más velocidad, teniendo que estimar así con esto qué nos premia más en el control que queremos realizar.
- La posición inicial desde la cual comencemos nuestro bucle de control y cómo se muevan nuestras referencias en nuestro espacio de trabajo influirán notablemente en nuestro sistema final, pues esto hará que la estimación de la profundidad que hacemos sea distinta en cada caso, y que por lo tanto nuestra matriz de interacción cambie en función de dicha profundidad y cómo vayamos variando ésta en cada caso.

Una vez destacados todos estos puntos sobre la experimentación realizada en nuestro trabajo, ya podemos pasar a hablar en el siguiente punto de las conclusiones que hemos extraído de éste, para así finalmente poder finalizar todo lo que tenemos que añadir sobre las investigaciones realizadas.

4. Conclusiones

Una vez hemos visto ya todo el estado del arte sobre la investigación que hemos realizado, hemos repasado toda la base teórica y técnica de los pasos que hemos realizado para realizar nuestro sistema de control, y hemos visto las experimentaciones y los resultados finales que hemos obtenido con éstas al final con los métodos de control finales que hemos implementado, podemos pasar a hablar de las conclusiones a las que hemos llegado con todos estos datos que hemos recogido para este trabajo de investigación.

En primer lugar, deberemos de destacar la importancia de los desarrollos vistos sobre el control de guiado de marcha de robots humanoides en el estado del arte que hemos investigado, viendo así cómo este método de control supone una base desde la cual partir muy útil a la hora de plantear un sistema robótico de este tipo, y pudiendo realizar control no sólo de la marcha, sino además de otras partes del cuerpo de nuestro robot, viendo así cómo el sistema de control visual para este campo de la robótica supone una línea de investigación muy interesante para un futuro, y planteando grandes retos interesantes para futuras investigaciones.

Otro punto interesante destacable es la importancia de acotar bien un sistema de control visual que se adecue a la situación que queremos plantear, pues este supondrá muchos datos distintos que tendremos que tener en cuenta para nuestro robot, y tendremos que plantear distintos escenarios en función de qué tipo de control queremos realizar en cada caso para obtener las distintas informaciones que queremos extraer.

También podemos ver que otra conclusión extraíble a partir del entorno en el que hemos realizado nuestras experimentaciones es que el robot NAO nos ofrece unas posibilidades a la hora de implementar desarrollos de este tipo muy elevadas, siendo esta máquina muy útil a la hora de realizar implementaciones de métodos de control para distintas partes del robot a partir de las cuales podamos plantear distintos comportamientos en nuestro robot muy variados gracias a su versatilidad.

Finalmente, basándonos en los resultados de las experimentaciones realizadas, podemos concluir que en un sistema de control como el que hemos implementado, es de vital importancia el hecho de elegir bien unos buenos valores para cada una de las partes de los métodos de control, habiendo visto en nuestra implementación cómo la profundidad de nuestro robot respecto a las referencias o la estimación de la ganancia necesaria para nuestro método de control será decisiva a la hora de obtener un comportamiento válido con respecto a las acciones que queremos que nuestro robot realice.

4.1. Trabajos futuros

A partir de las conclusiones obtenidas, podemos pasar a destacar algunos trabajos futuros que podemos plantear a raíz de éstas y de los datos que hemos obtenido a partir de la experimentación realizada, viendo que algunos interesantes podrían ser los siguientes:

- Implementación de un método de control visual basado en posición con realimentación en lugar de en imagen pudiendo tener una comparativa así entre ambos métodos de control con la que podríamos estudiar las diferencias que encontramos en cada caso a la hora de estimar las referencias de nuestro robot.
- Implementación de sistemas de control visual para otras tareas de nuestro robot, como bien pueden ser la manipulación de objetos con sus brazos, o el movimiento de la cabeza para el seguimiento de referencias con esta.
- Métodos de detección de referencias más precisos o alguna mejora de las cámaras disponibles, teniendo así un sistema más preciso que fuese capaz de realizar un seguimiento de las referencias que tenemos más exacto y que no pierda éstas con velocidades más elevadas que las que hemos utilizado en nuestro trabajo.
- Implementación de algún método de estimación y cancelación del ruido producido por el movimiento bípedo del robot, pudiendo así obtener señales más limpias con las que trabajar y realizar los desarrollos de control para el guiado que deseemos implementar en cualquier robot humanoide.

Referencias

1. *Belzec*. https://blog.belzec.net/historia-de-los-robots-el-primer-robot/ (visitado el 06/05/2019)
2. *Wikipedia*, https://es.wikipedia.org/wiki/Punto_de_Momento_Cero (visitado el 07/05/2019)
3. Xue, Feng; Chen, Xiaoping; Liu, Jinsu; Nardi, Daniele. *Real Time Biped Walking Gait Pattern Generator for a Real Robot*, RoboCup 2011: Robot Soccer World Cup XV. RoboCup 2011 Lecture Notes in Computer Science, Ed. Springer, vol 7416 (2012).
4. Siciliano, Bruno; Oussama, Khatib. *Springer Handbook of Robotics*, Springer, 1, 1312-1315 (2008).
5. Chik, David; Dundas, Jitesh. *Introduction to the Research of Humanoid Robots*, iConcept Journal of Human-Level Intelligence, 4, 2014
6. *Wikipedia*, https://es.wikipedia.org/wiki/Honda_P_series_(Robots) (visitado el 08/05/2019)
7. Fitzpatrick, P.M.; Metta, G.; Natale, L.; Rao, A.; Sandini, G. *Learning about objects through action -initial steps towards artificial cognition*. ICRA (2003).
8. Alaerts, K; Senot, P; Swinnen, S; Craighero, L; Wenderoth, N; Fadiga, L. *Force requirements of observed object lifting are encoded by the observer's motor system: A TMS-study*. European Journal of Neuroscience, 31(6) (2010).
9. Chaumette, F., Hutchinson, S. *Visual servo control. I. Basic approaches*, IEEE Robotics and Automation Magazine, 13(4), 82-90 (2006).
10. Nigri, Ilana; Meggiolaro, Marco A.; Queiroz, Raul, *Comparison Between look-and-move and visual servo control using sift transform in eye-in-hand manipulator system*, 20th International Congress of Mechanical Engineering (2009).
11. Pomares, P., García, G. J., Pérez, J., Gil, P., Torres, F. *"Control Visual", Conceptos y Métodos en Visión por Computador*, Ed. CEA, (2016) pp. 303-319.
12. Hutchinson, Seth; Hager, Gregory D.; Corke, Peter I., *A tutorial on visual Servo Control*, IEEE Transactions on Robotics and Automation, 12(5) (1996).
13. Vahrenkamp, Nikolaus; Böge, Christian; Welke, Kai; Asfour, Tamim; Walter, Jürgen; Dillman, Rüdiger, *Visual Servoing for Dual Arm Motions on a Humanoid Robot*, 9th IEEE-RAS International Conference on Humanoid Robots, Humanoids09 (2009).
14. Fantacci, Claudio; Vezzani, Giula; Pattacini, Ugo; Tikhanoff, Vadim; Natale, Lorenzo, *Markerless visual servoing on unknown objects for humanoid robot platforms*, 2018 IEEE International Conference on Robotics and Automation (ICRA) (2018): 3099-3106
15. Ardón, Paola; Dragone, Mauro; Erden, Mustafa, *Reaching and Grasping of Objects by Humanoid Robots Through Visual Servoing*, Lecture Notes in Computer Science, 10894 (2018)
16. Dune, C.; Stasse, O.; Yoshida, E; Herdt, A; Wieber, P-B.; Marchand, E, *Visual Servoing for Dynamic Walking*, IEEE/RSJ International Conference on Intelligent Robots and System (2010)
17. Delfin, Josafat; Becerra, Hector M.; Arechavaleta, Gustavo, *Visual Servo Walking Control for Humanoids with Finite-time Convergence and Smooth Robot Velocities*, International Journal of Control, 89(7) (2016): 1342-1358
18. Bombile, M., *Visual servo control on a humanoid robot*, University of Cape Town (2015)
19. Agravante, Don; Claudio, Giovanni; Spindler, Fabien; Chaumette, François, *Visual servoing in an optimization framework for the whole-body control of humanoid robots*, IEEE Robotics and Automation Letters, 2(2) (2017): 608-615
20. *ZBar bar code reader*, http://zbar.sourceforge.net/about.html (visitado el 13/05/2019)
21. Monasse, Pascal; Morel, Jean-Michel; Tang, Zhongwei, *Epipolar rectification*, (2019)

22. *Sightations, A Computer Vision Blog*. http://ksimek.github.io/2013/08/13/intrinsic/ (visitado el 13/05/2019)
23. *AliveRobots*. https://aliverobots.com/nao/ (visitado el 16/05/2019)
24. *Aldebaran Documentation, Motherboard*. http://doc.aldebaran.com/2-1/family/robots/motherboard_robot.html (visitado el 16/05/2019)
25. *Aldebaran Documentation, Software in and out of the robot.* http://doc.aldebaran.com/1-14/getting_started/software_in_and_out.html *(visitado el 16/05/2019)*
26. *Wikipedia.* https://es.wikipedia.org/wiki/Sistema_Operativo_Robótico (visitado el 16/05/2019)
27. *Aldebaran Documentation, Locomotion control.* http://doc.aldebaran.com/1-14/naoqi/motion/control-walk.html (visitado el 17/05/2019)
28. Kajita, Shuuji; Kanehiro, Fumio; Kaneko, Kenji; Yokoi, Kazuhito; Hirukawa, Hirohisa, *The 3D linear inverted pendulum model: A simple modeling for a biped walking pattern generation*, IEEE/RSJ International Conference on Intelligent Robots and Systems, Maui, Hawaii, USA, 1 (2001)
29. Chaumette, François; Marchand, Eric; Spindler, Fabien; Tallonneau, Romain, *ViSP 2.6.2: Visual Servoing Platform*, Aurélien Yol Contents, Lagadic Project (2012).
30. *ViSP, vpAdaptativeGain Class Reference*. https://visp-doc.inria.fr/doxygen/visp-2.8.0/classvpAdaptativeGain.html (visitado el 23/05/2019).

Anexos

Anexo A: Programa de controlador ver y mover

```
#include <ros/ros.h>
#include <geometry_msgs/PoseStamped.h>
#include <geometry_msgs/Pose2D.h>
#include <stdlib.h>

using namespace std;
int first_sub = 0;
int first_pub = 0;
float x = 0;
float y = 0;
float z = 0;
float xFin = 0;
float yFin = 0;
float zFin = 0;

void visionCallback(const geometry_msgs::PoseStamped& msg)
{
    x = msg.pose.position.x;
    y = msg.pose.position.y;
    z = msg.pose.position.z;
}

int main(int argc, char **argv)
{

       ros::init(argc, argv, "simple_camera_move_controller");
       ros::NodeHandle n;
       cout << "Esperando al topic de la camara para comenzar a funcionar" <<
endl;
       ros::Subscriber sub =
n.subscribe("/visp_auto_tracker/object_position", 1000, visionCallback);
       ros::Publisher pub =
n.advertise<geometry_msgs::Pose2D>("/cmd_pose",1000);

       ros::Rate rate(10.0);

       while((x == 0) && (y == 0))
       {
              ros::spinOnce();
              rate.sleep();
       }

       //Resultado de multiplicar por la matriz de transformacion
       xFin = z;
       yFin = -x;
       zFin = -y;

       cout << "Posicion en x: " << xFin << endl;
    cout << "Posicion en y: " << yFin << endl;

       geometry_msgs::Pose2D myFinishPose;
       myFinishPose.x = xFin;
       myFinishPose.y = yFin;
       myFinishPose.theta = 0;
       pub.publish(myFinishPose);
       ros::spinOnce();
       rate.sleep();

       return 0;
}
```

Anexo B: Programa del controlador visual basado en imagen

```
#include <visp/vpConfig.h>
#include <visp/vp1394TwoGrabber.h>
#include <visp/vpImage.h>
#include <visp/vpDisplay.h>
#include <visp/vpDisplayX.h>
#include <visp/vpMath.h>
#include <visp/vpHomogeneousMatrix.h>
#include <visp/vpFeaturePoint.h>
#include <visp/vpPoint.h>
#include <visp/vpServo.h>
#include <visp/vpFeatureBuilder.h>
#include <visp/vpRobotViper850.h>
#include <visp/vpException.h>
#include <visp/vpMatrixException.h>
#include <visp/vpServoDisplay.h>
#include <visp/vpDot2.h>
#include <visp/vpPose.h>
#include <visp/vpAdaptiveGain.h>

#include <iostream>
#include <fstream>

#include <visp/vpCameraParameters.h>
#include <visp/vpDisplayX.h>
#include <visp/vpDot2.h>
#include <visp/vpFeatureBuilder.h>
#include <visp/vpFeatureDepth.h>
#include <visp/vpFeaturePoint.h>
#include <visp/vpHomogeneousMatrix.h>
#include <visp/vpImage.h>
#include <visp/vpImageConvert.h>
#include <visp/vpServo.h>
#include <visp/vpVelocityTwistMatrix.h>

#include <visp_ros/vpROSGrabber.h>
#include <visp_ros/vpROSRobot.h>
#include <visp3/gui/vpPlot.h>

using namespace std;

int main(int argc, char **argv)
{

  vpROSRobot robot ;
  robot.setCmdVelTopic("/cmd_vel");
  robot.init();

  // Wait 3 sec to be sure that the low level Aria thread used to control
  // the robot is started. Without this delay we experienced a delay
(arround 2.2 sec)
  // between the velocity send to the robot and the velocity that is really
applied
  // to the wheels.
  vpTime::sleepMs(3000);

  vpServo task ;

  vpImage<unsigned char> I ;

  // Camera parameters. In this experiment we don't need a precise
calibration of the camera
  vpCameraParameters cam;
```

```
  // Create a grabber based on libdc1394-2.x third party lib (for firewire
cameras under Linux)
  vpROSGrabber g;
  g.setImageTopic("/nao_robot/camera/top/camera/image_raw");
  g.setCameraInfoTopic("/nao_robot/camera/top/camera/camera_info");
  g.setRectify(true);

  g.open(argc, argv);
  g.acquire(I);

  // Create an image viewer
  vpDisplayX d(I, 10, 10, "Frame de seleccion");
  vpDisplay::display(I);
  vpDisplay::flush(I);

  vpDot2 dot_final[4] ;
  vpImagePoint cog_final;
  vpPoint point[4];

  std::cout << "Click on the 4 dots clockwise starting from upper/left dot
for final points"
      << std::endl;
  std::cout << std::endl;

  for (int i=0 ; i < 4 ; i++) {
    dot_final[i].setGraphics(true);
    dot_final[i].setComputeMoments(true);
    dot_final[i].setEllipsoidShapePrecision(0.);  // to track a blob without
any constraint on the shape
    dot_final[i].setGrayLevelPrecision(0.9);  // to set the blob gray level
bounds for binarisation
    dot_final[i].setEllipsoidBadPointsPercentage(0.5); // to be accept 50%
of bad inner and outside points with bad gray level
    dot_final[i].initTracking(I) ;
    cog_final = dot_final[i].getCog();
    vpDisplay::displayCross(I, cog_final, 10, vpColor::blue) ;
    vpDisplay::flush(I);
    point[i].setWorldCoordinates(cog_final.get_j()*0.0002645833-0.045,
cog_final.get_i()*0.0002645833-0.045, 0);
  }

  int question = 0;

  if(question != 1){
    std::cout << "Realocate the robot for the initial position" <<
std::endl;
    std::cout << "Press 1 when the robot is in his initial position" <<
std::endl;
    std::cin >> question;
    std::cout << std::endl;
  }

  g.acquire(I);
  vpDisplay::display(I) ;
  vpDisplay::flush(I);

  vpDot2 dot[4];
  vpImagePoint cog;

  std::cout << "Click on the 4 dots clockwise starting from upper/left dot
for initial points"
          << std::endl;
  std::cout << std::endl;

  for (int i=0 ; i < 4 ; i++) {
    dot[i].setGraphics(true);
    dot[i].setComputeMoments(true);
```

```
    dot[i].setEllipsoidShapePrecision(0.);  // to track a blob without any
constraint on the shape
    dot[i].setGrayLevelPrecision(0.9);  // to set the blob gray level bounds
for binarisation
    dot[i].setEllipsoidBadPointsPercentage(0.5); // to be accept 50% of bad
inner and outside points with bad gray level
    dot[i].initTracking(I) ;
    cog = dot[i].getCog();
    vpDisplay::displayCross(I, cog, 10, vpColor::blue) ;
    vpDisplay::flush(I);
  }

  // Sets the current position of the visual feature
  vpFeaturePoint p[4] ;
  for (int i=0 ; i < 4 ; i++)
    vpFeatureBuilder::create(p[i], cam, dot[i]);

  // Sets the desired position of the visual feature
  vpFeaturePoint pd[4];
  for (int i = 0; i < 4; i++){
    vpFeatureBuilder::create(pd[i], cam, dot_final[i]);
    pd[i].set_Z(0.542);
  }

  //Compute the initial pose
  vpHomogeneousMatrix cMo;
  vpPose pose;
  pose.clearPoint();

  for (int i=0 ; i < 4 ; i++)
    {
      double x,y;
      vpPixelMeterConversion::convertPoint(cam,dot[i].getCog(),x,y)  ;
      point[i].set_x(x) ;
      point[i].set_y(y) ;
      pose.addPoint(point[i]) ;
    }
  pose.computePose(vpPose::LAGRANGE, cMo) ;
  pose.computePose(vpPose::VIRTUAL_VS, cMo) ;

  // We want to see a point on a point
  for (int i=0 ; i < 4 ; i++)
    task.addFeature(p[i], pd[i]) ;

  // Set the proportional gain
  vpAdaptiveGain  lambda;
  lambda.initStandard(4.10, 0.4, 40);

  task.setLambda(lambda) ;

  // Define the task
  task.setServo(vpServo::EYEINHAND_CAMERA) ;
  task.setInteractionMatrixType(vpServo::CURRENT, vpServo::PSEUDO_INVERSE) ;

  vpList<vpImagePoint> list[4];

  // Create a window (800 by 500) at position (400, 10) with 2 graphics
  vpPlot graph(2, 700, 700, -1, -1, "Curves...");
  // Init the curve plotter
  graph.initGraph(0, 6);
  graph.initGraph(1, 8);
  graph.setTitle(0, "Velocities");
  graph.setTitle(1, "Error s-s*");
  graph.setLegend(0, 0, "vx");
  graph.setLegend(0, 1, "vy");
  graph.setLegend(0, 2, "vz");
  graph.setLegend(0, 3, "wx");
  graph.setLegend(0, 4, "wy");
```

```
  graph.setLegend(0, 5, "wz");
  graph.setLegend(1, 0, "x1");
  graph.setLegend(1, 1, "y1");
  graph.setLegend(1, 2, "x2");
  graph.setLegend(1, 3, "y2");
  graph.setLegend(1, 4, "x3");
  graph.setLegend(1, 5, "y3");
  graph.setLegend(1, 6, "x4");
  graph.setLegend(1, 7, "y4");


  int iter = 0;
  bool lost = false;
  ofstream file_velocity("vel.txt");
  ofstream file_error("err.txt");

  file_velocity << "v = [";
  file_error << "e = [";

  // Servo loop
  try{
    while (1) {
      g.acquire(I) ;

      vpDisplay::display(I) ;

      // For each point achieve the tracking of the dot in the image
      for (int i=0 ; i < 4 ; i++)  {
           dot[i].track(I) ;
           // save the cog position
           list[i].addRight(dot[i].getCog());
      }
      // Display the dot cog trajectories
      for (int i=0 ; i < 4 ; i++) {
           if (list[i].nbElements() >= 2) {
             list[i].front();
             for (int j = 0; j < list[i].nbElements()-1;j++) {
               vpDisplay::displayLine(I, list[i].value(),
                             list[i].nextValue(),
                             vpColor::cyan, 2);
               list[i].next();
             }
           }
      }

        // Compute the pose in order to update the Z coordinate of the
points
      pose.clearPoint();
      for (int i=0 ; i < 4 ; i++) {
           double x,y;
           vpPixelMeterConversion::convertPoint(cam, dot[i].getCog(), x, y)
;
           point[i].set_x(x) ;
           point[i].set_y(y) ;
           pose.addPoint(point[i]) ;
      }
      pose.computePose(vpPose::VIRTUAL_VS, cMo) ;


      // Update the point feature with the dot distance (Z)
      for (int i=0 ; i < 4 ; i++) {
           vpFeatureBuilder::create(p[i], cam, dot[i]);
           // Set the feature Z coordinate from the pose
           vpColVector cP;
           point[i].changeFrame(cMo, cP) ;

           p[i].set_Z(cP[2]);
      }
```

```
// Compute the visual servoing skew vector
vpColVector v ;
v = task.computeControlLaw() ;

vpMatrix cVe;
cVe.resize(6, 6, true);

cVe[0][0] = 0;
cVe[0][1] = 0;
cVe[0][2] = 1;
cVe[0][3] = -0.055;
cVe[0][4] = -0.544;
cVe[0][5] = 0;

cVe[1][0] = -1;
cVe[1][1] = 0;
cVe[1][2] = 0;
cVe[1][3] = 0;
cVe[1][4] = 0;
cVe[1][5] = -0.055;

cVe[2][0] = 0;
cVe[2][1] = -1;
cVe[2][2] = 0;
cVe[2][3] = 0;
cVe[2][4] = 0;
cVe[2][5] = -0.544;

cVe[3][0] = 0;
cVe[3][1] = 0;
cVe[3][2] = 0;
cVe[3][3] = 0;
cVe[3][4] = 0;
cVe[3][5] = 1;

cVe[4][0] = 0;
cVe[4][1] = 0;
cVe[4][2] = 0;
cVe[4][3] = -1;
cVe[4][4] = 0;
cVe[4][5] = 0;

cVe[5][0] = 0;
cVe[5][1] = 0;
cVe[5][2] = 0;
cVe[5][3] = 0;
cVe[5][4] = -1;
cVe[5][5] = 0;


v = cVe*v;

for(int i = 0; i<6; i++)
{
  if(v[i]>=0.61)
    v[i] = 0.60;
  if (v[i]<=-0.61)
    v[i] = -0.60;
  if (i == 2 || i == 3 || i == 4)
    v[i] = 0;
}

std::cout << "vx = " << v[0] << "m/s" << std::endl;
std::cout << "vy = " << v[1] << "m/s" <<std::endl;
std::cout << "vz = " << v[2] << "m/s" <<std::endl;
std::cout << "wx = " << v[3] << "deg/s" <<std::endl;
std::cout << "wy = " << v[4] << "deg/s" <<std::endl;
```

```
      std::cout << "wz = " << v[5] << "deg/s" <<std::endl;
      std::cout << "err_x1 = " << task.getError()[0] << std::endl;
      std::cout << "err_y1 = " << task.getError()[1] << std::endl;
      std::cout << "err_x2 = " << task.getError()[2] << std::endl;
      std::cout << "err_y2 = " << task.getError()[3] << std::endl;
      std::cout << "err_x3 = " << task.getError()[4] << std::endl;
      std::cout << "err_y3 = " << task.getError()[5] << std::endl;
      std::cout << "err_x4 = " << task.getError()[6] << std::endl;
      std::cout << "err_y4 = " << task.getError()[7] << std::endl;
      std::cout << std::endl;


      // Display the current and desired feature points in the image display
      vpServoDisplay::display(task, cam, I, vpColor::red, vpColor::green, 2)
;

      // Apply the computed joint velocities to the robot
      robot.setVelocity(vpRobot::REFERENCE_FRAME, v);

      graph.plot(0, iter, v);
      graph.plot(1, iter, task.getError()); // plot error vector


      file_velocity << v[0] << ", "<<v[1] << ", "<<v[2] << ", "<<v[3] << ",
"<<v[4] << ", "<<v[5] << "; ";
      file_error << task.getError()[0] << ", "<<task.getError()[1] << ",
"<<task.getError()[2] << ", "<<task.getError()[3] << ",
"<<task.getError()[4] << ", "<<task.getError()[5] << ",
"<<task.getError()[6] << ", "<<task.getError()[7] << "; ";

      // Flush the display
      vpDisplay::flush(I);

      iter++;
      if (task.getError().sumSquare() < 0.0005) {
        std::cout << "Reached a small error. We stop the loop... " <<
std::endl;
        break;
      }

    }
  }
  catch(...) {
    lost = true;
    robot.stopMotion();
    file_velocity << "]";
    file_error << "]";
    const char *legend = "Click to quit...";
    vpDisplay::displayText(graph.I, (int)graph.I.getHeight() - 60,
(int)graph.I.getWidth() - 150, legend, vpColor::red);
    vpDisplay::flush(graph.I);
    vpDisplay::getClick(graph.I);
    task.print();
    task.kill();
  }

  if (lost == false){
    robot.stopMotion();
    file_velocity << "]";
    file_error << "]";
    const char *legend = "Click to quit...";
    vpDisplay::displayText(graph.I, (int)graph.I.getHeight() - 60,
(int)graph.I.getWidth() - 150, legend, vpColor::red);
    vpDisplay::flush(graph.I);
    vpDisplay::getClick(graph.I);
    task.print();
    task.kill();
  }
```

```
  return 0;

}
```