Universidad Nacional Autónoma de México

Facultad de Ingeniería

EVALUACIÓN DE TÉCNICAS DE CONTROL VISUAL MONOCULAR PARA LA LOCOMOCIÓN DE ROBOTS HUMANOIDES NAO

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

Ingeniero Mecatrónico

PRESENTA:

Mario Alberto Zarco López

DIRECTOR DE TESIS:

Dr. Juan Mauricio Angeles Cervantes

México, D.F., 2015

Dedicatorias

A mis padres, gracias por las incontables formas de apoyo, son y serán imprescindibles en mi vida, inifinitas gracias por dejarme seguir mis sueños, los amo.

A mi abuela, si mañana muriera ten más que claro que fue más que un placer encontrarte en este mundo desolado. Mi ultimo pensamiento se lo dedicaría a los que amo y justo antes de cerrar los ojos sólo vería tu abrazo.

Al Dr. José Luis Urrutia Galicia, gracias por todas las lecciones, por compartir su conocimiento y valores.

Agradecimientos

Agradezco la oportunidad que me brindó el Centro de Investigación en Matemáticas (CIMAT), A.C.

De manera muy especial le doy las gracias al Dr. Héctor M. Becerra Fermín por su invaluable ayuda y paciencia en todo momento y en todos los aspectos.

Agradezco a la Facultad de Ingeniería y a la UNAM por todo de lo que me dejó disponer.

De manera particular le agradezco al Dr. Juan Mauricio Angeles Cervantes por apoyarme en la realización de la tesis.

Por último, le doy las gracias al Dr. Jesús Manuel Dorador González por los consejos y facilidades que me proporcionó para concluir este trabajo.

...el hombre ha podido humanizar en mucho a la naturaleza, apropiándose sus procesos y modificándolos con arreglo a sus propósitos. En cambio, el proceso de humanización del hombre se encuentra mucho menos avanzado y tropieza a cada paso con intereses de la minoría que ejerce el dominio sobre la sociedad. Este obstáculo hay que superarlo definitivamente, para dar libre curso a dicho proceso de humanización. Pero, es imposible esperar que la superación se logre sin esfuerzo y sin lucha denodados. Y, por esto la tarea indeclinable de la filosofía, como ciencia crítica de la realidad, consiste en extraer las consecuencias de esta trayectoria incontenible, para proyectarlas hacia el advenimiento del nuevo humanismo y propiciar así, a través de la actividad práctica eficaz, la realización de las mejores aspiraciones del hombre.

Elí de Gortari

Índice general

Índice de figuras

Lista de algoritmos

Índice de tablas

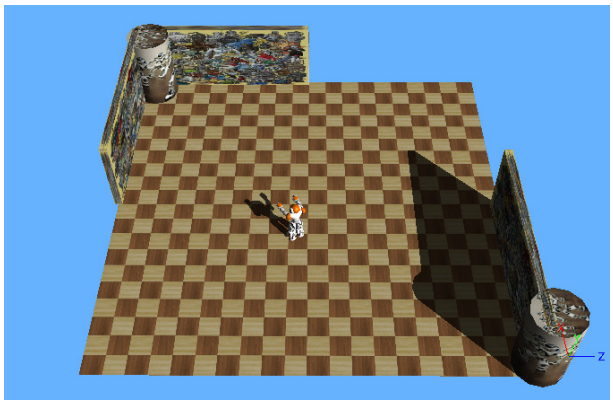
Capítulo 1

Introducción

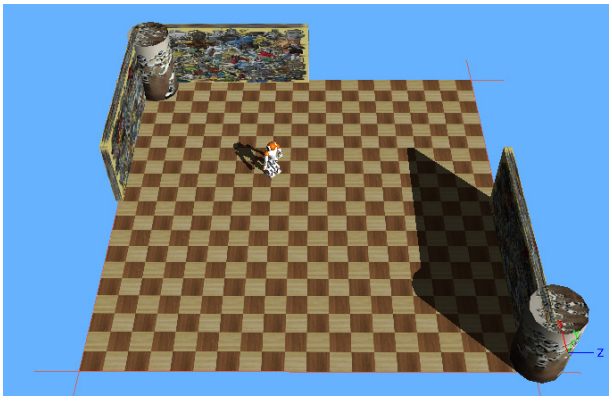
En este capítulo se explican los conceptos básicos del robot humanoide y su locomoción, así como del control visual. Posteriormente se presenta el estado del arte y las motivaciones para realizar este trabajo. Por último se describen los objetivos y se concluye describiendo la estructura del resto del trabajo.

1.1. Antecedentes

La investigación en robots humanoides es actualmente uno de los más populares e interesantes temas en el campo de la robótica. Estos robots están diseñados con el objetivo de desempeñarse en entornos humanos, definidos como ambientes no estructurados y dinámicos [1], donde los objetos se mueven sin que el robot deba intervenir. Entonces, para completar una tarea, el humanoide debe ser capaz de percibir y reaccionar a estos cambios. Con el objetivo de que se desplace a cierto punto del entorno es posible integrar una cámara al robot para identificar el ambiente. Por ejemplo, el robot en la Figura 1.1 debe llegar de la posición en la imagen (a) a la posición en la imagen (b). Por lo tanto, el control visual puede ayudar al robot a percibir su entorno y adaptar su comportamiento. Así, la tarea de posicionamiento, como un problema de control, se puede definir como [2]: percibido un error se requiere generar una demanda de velocidad que mueva el robot hacia un objetivo.



(a) Posición inicial



(b) Posición final

Figura 1.1: Ejemplo de la tarea de posicionamiento

1.2. Robot humanoide

El campo de estudio de los robots humanoides se enfoca en la creación de robots cuya morfología y capacidades son inspiradas en los humanos, de modo que traten de imitar la cinemática del cuerpo humano, la percepción y el comportamiento. En [3] se menciona que sus principales características son:

1. Trabajar en ambientes diseñados para humanos, es decir sin modificaciones.

2. Usar herramientas diseñadas para humanos.

3. Poseer estructura antropomórfica.

Los robots humanoides han sido desarrollados, por ejemplo, como trabajadores mecánicos de propósito general, para entretenimiento, y como banco de pruebas para teorías de neurociencias y psicología experimental [4].

La motivación para el desarrollo de estos robots varía ampliamente, por lo que es común que tengan diferentes tamaños, formas, grados de libertad y sensores, de acuerdo con la característica humana que se desea estudiar o la actividad que el robot ejecuta.

La locomoción bípeda es un tema de investigación clave en los robots humanoides, en especial en robots con dimensiones y una distribución de peso parecida a la humana.

1.2.1. Locomoción humanoide

La locomoción humanoide es un curso de acción cuidadosamente planeado del cuerpo entero que tiene lugar en tiempo real [5]. No obstante, al ser el cálculo de la dinámica del cuerpo entero de un costo computacional alto, en la prática se usan modelos de masa concentrada con el fin de lograr su implementación.

La locomoción bipeda está compuesta por dos fases, durante las cuales se debe asegurar que la suma de fuerzas actuando sobre el robot no resulten en una pérdida de balance [5], es decir debe permanecer en equilibrio dinámico (esto se trata en la sección 2.5). Por lo tanto, la marcha de humanoide consiste en la translación del robot en la dirección deseada mediante el conjunto de movimientos coordinados de las piernas y el tronco [6]. Durante este movimiento existe una transición, en cada paso, entre la pierna que se balancea y la pierna que se apoya.

1.2.2. Robot NAO

En esta tesis se trabajará con la plataforma humanoide NAO debido a las facilidades que ofrece para la implementación de experimentos, ya sean en simulación o reales. Este robot fue desarrollado por la compañía francesa *Aldebaran Robotics*, es autónomo, completamente programable y es usado

tanto con propósitos educativos como para investigación. El modelo V4-H25 tiene 25 grados de libertad y mide 58 centímetros de altura.

En la Figura 1.2 se muestra como evolucionó el robot NAO a través del tiempo, desde el prototipo hasta el producto final. La primera edición de producción fue utilizada en la competición Robocup del 2008 y actualmente se han desarrollado cinco generaciones, siendo la última NAO Evolution.

Figura 1.2: Desarrollo del robot NAO. Figura tomada de `http://robosavvy.com/forum/viewtopic.php?t=7670`

1.3. Control visual

El control visual, también conocido en inglés con el término genérico *visual servoing*, es el uso de información visual como realimentación en un lazo cerrado de control [7]. Esta información puede ser obtenida por una cámara montada en el robot (configuración *eye-in-hand*), o por una cámara fija en el espacio de trabajo de tal forma que observe al robot desde una configuración estacionaria (configuración *eye-to-hand*).

En el espacio imagen no se puede obtener una referencia absoluta, por lo que el valor deseado se establece dando una imagen de referencia u objetivo, la cual debe ser memorizada por el robot (estrategia *teach-by-showing*). Para extraer la información necesaria de ambas imágenes es necesario que compartan información, es decir, que tengan en común un conjunto de características visuales.

La configuración *eye-in-hand* es la más útil en el caso de robots móviles, por lo que el objetivo del control visual será regular la postura (posición y orientación) de la cámara, y en consecuencia del robot, de tal forma que la imagen de la vista actual de la cámara llegue a ser la misma que la imagen objetivo a través de minimizar una función de error. Esta función de error es referida también como una función de tarea. De acuerdo a su naturaleza, el

control visual puede clasificarse en tres grupos [7]:

1. Control visual basado en imagen (IBVS, por sus siglas en inglés) o 2D. El error es calculado a partir de un conjunto de características visuales que están directamente disponibles en el espacio imagen.

2. Control visual basado en posición (PBVS, por sus siglas en inglés) o 3D. El error es calculado en el espacio Cartesiano de la tarea a partir de un conjunto de parámetros 3D, los cuales son estimados mediante medidas visuales.

3. Control visual híbrido o 2-1/2D. La función de error es una combinación de medidas de imagen y Cartesianas.

1.4. Estado del arte

La visión artificial se ha convertido en un importante tema de investigación debido a que provee abundante información del ambiente con la ventaja de no tener contacto con éste, así como su bajo costo. De manera particular, la visión monocular proporciona buena precisión al medir el ángulo de orientación (*bearing angle*). Las características mencionadas son útiles para la regulación de la posición y orientación de un robot humanoide, es decir, el sistema de control debe indicar las velocidades para mover al robot a la ubicación deseada. No obstante, la visión monocular presenta desventajas, como la falta de información de profundidad y el relativamente alto tiempo de procesamiento de la imagen para la interpretación de datos.

En los últimos años se ha comenzado a investigar el control basado en visión de la marcha bípeda de los robots humanoides. Por ejemplo, en [1] se usa un control visual basado en posición para controlar la marcha dinámica del robot HRP-2. En [8] se propone un método para ignorar el movimiento de balanceo mientras la marcha de este robot es controlada con un esquema basado en imagen. Con relación al robot NAO, en [9] se utiliza un control visual basado en imagen para mover a un robot NAO en un laberinto de corredores. En [10] se aborda el problema de la generación de marcha humanoide con retroalimentación visual, y muestra resultados de un control visual basado en posición y otro basado en imagen, ambos resueltos con base en la estimación de la homografía. En [11] se propone una estrategia para que la navegación humanoide se realice a través de una ruta compuesta por varias

imágenes de referencia. Otros usos se pueden ver en [12] y en [13]. En el primero, el control visual es utilizado para para determinar la orientación de la cabeza del robot NAO en tareas de visibilidad, detección y manipulación de objetos. En el segundo, se suma la tarea de seguimiento de objetos y la autolocalización mientras camina en un entorno doméstico.

En este trabajo de tesis se usan dos esquemas de control visual, uno basado en posición y otro en imagen. Ambos se solucionan con base a la geometría epipolar. Las velocidades calculadas por el control visual son aplicadas al sistemas de referencia del robot a través de una transformación de la cámara al origen de dicho sistema. Este enfoque presenta el problema de que los esquemas usados no están realimentando la generación de la marcha humanoide, sólo le indica una velocidad de referencia a seguir y no existe garantía de ser seguida correctamente [14]. Sin embargo, el enfoque de introducir el control visual como referencia de velocidad para el sistema de locomoción tiene la ventaja de ser barato computacionalmente, a diferencia de introducir la información visual directamente en el problema de optimización del sistema de locomoción como en [14].

1.5. Motivaciones

La principal motivación fue evaluar y comparar diferentes esquemas de control visual para un robot humanoide debido al gran interés que existe sobre este tema en la comunidad de robótica. Estos esquemas se programaron para que, con algunos cambios, puedan ser implementados en un robot humanoide NAO y evaluar las diferencias que existen en un ambiente real.

En particular se propone comparar el desempeño de un esquema de control basado en posición contra un esquema basado en imagen; ambos formulados en términos de la geometría epipolar. De este modo sería posible comparar los resultados con trabajos que presenten pruebas similares, por ejemplo [10], en donde los esquemas de control se basan en el modelo de homografía.

1.6. Objetivos

El objetivo general de este trabajo es adaptar diferentes técnicas de control visual monocular para la realización de tareas de posicionamiento de robots humanoides NAO. Estas técnicas se evalúan a través de simulaciones en un ambiente virtual. Por lo anterior, se definen los siguientes objetivos particulares:

- Modificar el prototipo del robot de NAO en Webots para añadir un *GPS* y un *compass* al modelo del robot.
- Modificar el controlador *naoqisim* en Webots para poder adquirir los valores de los sensores añadidos cada vez que se requieran de manera externa al proceso.
- Programar en un lenguaje de alto nivel (C++) el control visual basado en posición propuesto en [15] con ayuda de un software de desarrollo (*naoqi-sdk*).
- Programar en un lenguaje de alto nivel (C++) el control visual basado en imagen propuesto en [16] con ayuda de un software de desarrollo (*naoqi-sdk*).
- Proponer una estructura modular para la ejecución de los controladores.
- Validar el funcionamiento de los esquemas de control a través de simulaciones en Webots y comparar su desempeño.

1.7. Contenido

El contenido del documento se divide en seis capítulos y un apéndice. El **Capítulo 2** introduce los conceptos y expresiones matemáticas necesarias para la realización de este trabajo. En el **Capítulo 3** se muestran los resultados de las simulaciones en Matlab del control visual basado en posición y del control visual basado en imagen.

El **Capítulo 4** describe como se implementaron las simulaciones en el ambiente de desarrollo Webots. El desarrollo de este capítulo se relaciona

con los temas del **Apéndice A**, el cual reseña los conceptos básicos para comprender la programación del robot NAO.

Las gráficas de los resultados de las pruebas hechas, con ambos esquemas de control visual, en el ambiente virtual de Webots se presentan en el **Capítulo 5**. Por último, el **Capítulo 6** presenta las conclusiones.

Capítulo 2

Marco teórico

En este capítulo se detallan los antecedentes necesarios para este trabajo, comenzando con el modelo de la cámara, necesario para describir la geometría epipolar. Se continúa con las características locales, tema que será útil para su posterior implementación. Le sigue el tema de control visual y el capítulo concluye con un panorama general de la marcha humanoide.

2.1. Modelo de cámara oscura (*Pinhole*)

Una cámara es un dispositivo que realiza un mapeo del mundo 3D a un plano, esto es, una imagen 2D, por lo cual se pierde una dimensión. El modelo matemático considera un sistema de coordenadas Euclideano cuyo origen está en el centro de la cámara, llamado *centro de proyección* o *centro óptico*. El eje z es considerado el *eje principal*, y a una distancia f sobre este eje se encuentra un plano paralelo al plano xy; la distancia f se conoce como *longitud focal*, el plano como *plano focal* o *plano imagen* y el punto localizado donde el eje z se encuentra con este plano es llamado *punto principal* de la imagen.

Considerando este modelo, un punto en el espacio con coordenadas $\widetilde{\mathbf{X}}_c = (X, Y, Z)^\top$ expresado con respecto del sistema de coordenadas de la cámara, es mapeado al punto donde la línea que une a $\widetilde{\mathbf{X}}_c$ y al centro de proyección intersecta el plano imagen. Usando triángulos semejantes, el punto mapeado tiene coordenadas $\widetilde{\mathbf{x}} = (fX/Z, fY/Z)$ en el plano, ver la Figura 2.1.

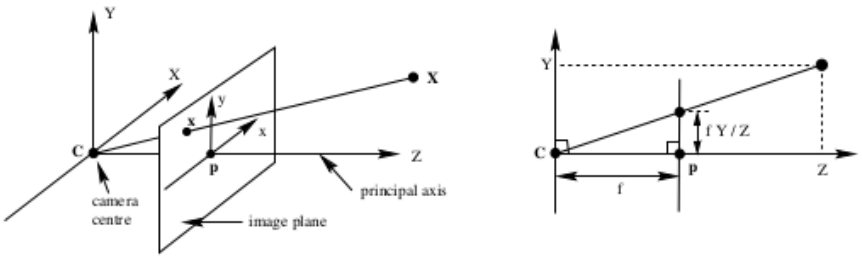


Figura 2.1: Geometría del modelo de cámara *pinhole*. Figura tomada de [17]

2.1.1. Modelo de proyección central

En la práctica el mapeo del espacio Euclideano $\mathbb{R}^3$ al espacio Euclideano $\mathbb{R}^2$ resulta en $(X, Y, Z) \mapsto (fX/Z + Zp_x, fY/Zp_y)$, donde $(p_x, p_y)^\top$ son las coordenadas del punto principal con respecto del sistema de coordenadas del plano imagen, ya que el origen de éste regularmente no se encuentra en el punto principal.

La proyección central se puede expresar como un mapeo entre coordenadas homogéneas, y a su vez en términos de la multiplicación de matrices como en la ecuación 2.1.

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{2.1}$$

La expresión anterior se puede reescribir de la siguiente forma:

$$\mathbf{x} = \mathbf{K}[\mathbf{I}|\mathbf{0}]\mathbf{X}_c, \tag{2.2}$$

donde $\mathbf{I}$ es la matriz identidad de 3×3, $\mathbf{0}$ es un vector columna de ceros de 3×1 y la matriz $\mathbf{K}$, en la ecuación 2.3, es llamada *matriz de calibración de la cámara*, definida como:

$$\mathbf{K} = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix}. \tag{2.3}$$

Es posible que los píxeles no sean cuadrados, como en el caso de las cámaras con sensores CCD[1], por lo que existe un factor de escala en cada dirección de las coordenadas de la imagen medida en píxeles. Si m_x y m_y representan el número de píxeles por unidad de distancia en las coordenadas x y y de la imagen, respectivamente, se puede expresar la ecuación 2.3 de la siguiente manera

$$\mathbf{K} = \begin{bmatrix} \alpha_x & & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix}, \tag{2.4}$$

donde $\alpha_x = fm_x$ y $\alpha_y = fm_y$ representan la distancia focal en términos de la dimensiones en píxeles. De forma similar $x_0 = m_x p_x$ y $y_0 = m_y p_y$ son las coordenadas del punto principal en términos de la dimensiones en píxeles.

En general, los puntos en el espacio son expresados con respecto de un sistema de referencia distinto al de la cámara, llamado sistema de referencia del mundo. Si $\widetilde{\mathbf{X}}_w$ y $\widetilde{\mathbf{X}}_c$ son puntos en coordenadas no homogéneas, expresados con respecto del sistema de referencia del mundo y de la cámara, respectivamente, se pueden relacionar a través de la expresión $\widetilde{\mathbf{X}}_c = \mathbf{R}\widetilde{\mathbf{X}}_w - \widetilde{\mathbf{C}}$, donde $\mathbf{R}$ es una matriz de rotación que alinea a los sistemas de referencia, y $\widetilde{\mathbf{C}}$ representa las coordenadas del centro de la cámara con respecto del sistema de referencia del mundo. Expresando la ecuación anterior en coordenadas homogéneas, como una multiplicación de matrices y considerando $\mathbf{t} = -\mathbf{R}\widetilde{\mathbf{C}}$ resulta en

$$\mathbf{X}_c = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \mathbf{X}_w. \tag{2.5}$$

De las ecuaciones2.2 y 2.5 obtenemos

$$\mathbf{x} = \mathbf{K}[\mathbf{R}|\mathbf{t}]\mathbf{X}_w. \tag{2.6}$$

Entonces se define la matriz de la cámara como

[1]Del inglés charge-coupled device, dispositivo de carga acoplada.

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]. \tag{2.7}$$

Los parámetros contenidos en **K** son llamados parámetros intrínsecos o internos, y los parámetros de **R** y **t** son llamados parámetros extrínsecos o externos.

2.2. Geometría epipolar

La geometría epipolar es la geometría proyectiva entre dos vistas, la cual sólo depende de los parámetros internos de las cámaras y su postura relativa. Es la geometría de la intersección del haz de planos que usan como eje la *línea base*, es decir, la línea que une los centros de cámara, etiquetados como **C** y **C'** en la Figura 2.2, con los planos imagen de cada vista. Los puntos generados por el cruce de la línea base con el plano imagen se llaman *epipolos*, **e** y **e'** en la figura.

El punto **X** en el espacio se proyecta al plano imagen, **x** y **x'** en la primera y segunda vista, respectivamente. Los centros de cámara, el punto en el espacio y sus proyecciones en los planos imagen son coplanares, localizados en el *plano epipolar* (π en la figura 2.2). El plano epipolar intersecta a los plano imagen en sus correpondientes *líneas epipolares*, **l** y **l'**.

El centro de cámara **C** y el punto **x** en la imagen definen un rayo en el espacio, cuyos puntos proyectados en el plano imagen de la segunda vista se localizan en la línea epipolar **l'**; como **X** es coplanar su proyección se debe localizar en esta línea epipolar, y de manera análoga en algún lugar de **l**.

2.2.1. Matriz fundamental

La matriz fundamental es la representación algebraica de la geometría epipolar. Esta matriz representa un mapeo proyectivo de puntos a líneas, es decir, a cada punto en la primera imagen le corresponde una línea epipolar en la otra imagen. Lo anterior se expresa en la siguiente ecuación:

$$\mathbf{l'} = \mathbf{F}\mathbf{x}. \tag{2.8}$$

Entonces F representa un mapeo de un espacio proyectivo de dimensión 1 a uno de dimensión 2, por lo que debe tener rango 2. De manera similar $\mathbf{l} = \mathbf{F}^\top \mathbf{x}$'.

La matriz fundamental satisface la condición que para cualquier par de puntos correspondientes $\mathbf{x} \longleftrightarrow \mathbf{x}$' en las imágenes

$$\mathbf{x}'^\top \mathbf{F} \mathbf{x} = 0, \tag{2.9}$$

ya que existe la propiedad de que $\mathbf{x}'^\top \mathbf{l}' = 0$, para cualquier $\mathbf{x}$', debido a que pertenece a la línea $\mathbf{l}$'. De forma similar, debido a que el epipolo $\mathbf{e}$' se encuentra sobre la línea se deduce lo siguiente:

$$\mathbf{e}'^\top \mathbf{F} = 0, \tag{2.10}$$

de donde se puede concluir que $\mathbf{e}$' es el vector nulo izquierdo de $\mathbf{F}$. Y de forma análoga para el epipolo en la primera imagen, es decir, $\mathbf{F}\mathbf{e} = 0$.

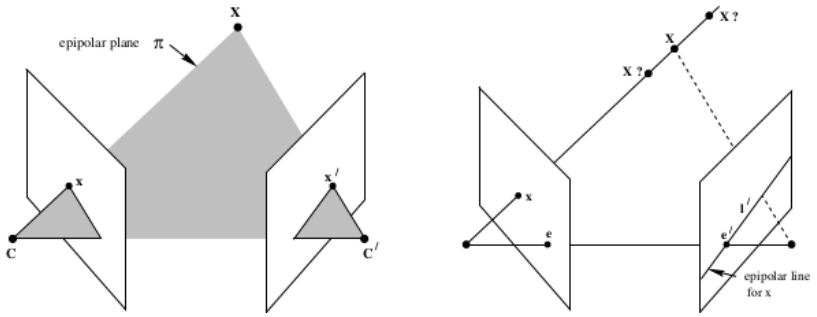


Figura 2.2: Geometría epipolar. Figura tomada de [17]

2.2.2. Matriz esencial

La matriz esencial es un caso especial de la matriz fundamental cuando se tienen coordenadas normalizadas en la imagen. Se puede expresar un punto $\hat{\mathbf{x}}$ en coordenadas normalizadas aplicando la inversa de la matriz $\mathbf{K}$ a la ecuación 2.6, de donde resulta $\hat{\mathbf{x}} = \mathbf{K}^{-1}\mathbf{x} = [\mathbf{R}|\mathbf{t}]\mathbf{X}_w$.

Entonces, si se considera un par de matrices de cámara normalizadas $\mathbf{P} = [\mathbf{I}|\mathbf{0}]$ y $\mathbf{P}' = [\mathbf{R}|\mathbf{t}]$, y tomando en cuenta que la matriz fundamental para

aquéllas es $\mathbf{F} = \mathbf{K'}^{\top}\mathbf{R}\mathbf{K}^{\top}[\mathbf{K}\mathbf{R}^{\top}\mathbf{t}]_{\times}$[2], la matriz esencial sería

$$\mathbf{E} = [\mathbf{t}]_{\times}\mathbf{R} = \mathbf{R}[\mathbf{R}^{\top}\mathbf{t}]_{\times}, \tag{2.11}$$

tal que $\mathbf{R} \in SO(3)$ y $\mathbf{t} \in \mathbb{R}^3$. Análogamente con la matriz fundamental, se puede llegar a una ecuación similar a la ecuación 2.9, es decir

$$\hat{\mathbf{x}'}^{\top}\mathbf{E}\hat{\mathbf{x}} = 0. \tag{2.12}$$

Debido a que $\hat{\mathbf{x}} = \mathbf{K}^{-1}\mathbf{x}$, si se sustituye en la ecuación 2.12 se concluye que la matriz esencial se puede obtener de la siguiente multiplicación:

$$\mathbf{E} = \mathbf{K'}^{\top}\mathbf{F}\mathbf{K}. \tag{2.13}$$

2.2.3. Descomposición de la matriz esencial

Dada una matriz esencial $\mathbf{E}$ no nula, y su descomposición en valores singulares $\mathbf{E} = \mathbf{U}\Sigma\mathbf{V}^{\top}$ con $\mathbf{U}, \mathbf{V} \in SO(3)$ y $\Sigma = diag(\sigma, \sigma, 0)$ con $\sigma \in \mathbb{R}_+$ [18], entonces existen al menos cuatro posibles soluciones tales que se cumpla la ecuación 2.11. En la ecuación 2.14 se escriben las posibles soluciones, donde $\mathbf{R}_Z(\theta)$ es una rotación de θ radianes sobre el eje Z.

$$\begin{aligned}
([\mathbf{t}]_{\times}, \mathbf{R}) &= (\mathbf{U}\mathbf{R}_Z(+\tfrac{\pi}{2})\Sigma\mathbf{U}^{\top}, \mathbf{U}\mathbf{R}_Z^{\top}(+\tfrac{\pi}{2})\mathbf{V}^{\top}),\\
([\mathbf{t}]_{\times}, \mathbf{R}) &= (\mathbf{U}\mathbf{R}_Z(-\tfrac{\pi}{2})\Sigma\mathbf{U}^{\top}, \mathbf{U}\mathbf{R}_Z^{\top}(-\tfrac{\pi}{2})\mathbf{V}^{\top}),\\
([\mathbf{t}]_{\times}, \mathbf{R}) &= (\mathbf{U}\mathbf{R}_Z(+\tfrac{\pi}{2})\Sigma\mathbf{U}^{\top}, \mathbf{U}\mathbf{R}_Z^{\top}(-\tfrac{\pi}{2})\mathbf{V}^{\top}),\\
([\mathbf{t}]_{\times}, \mathbf{R}) &= (\mathbf{U}\mathbf{R}_Z(-\tfrac{\pi}{2})\Sigma\mathbf{U}^{\top}, \mathbf{U}\mathbf{R}_Z^{\top}(+\tfrac{\pi}{2})\mathbf{V}^{\top}).
\end{aligned} \tag{2.14}$$

La Figura 2.3 muestra la interpretación física de estas soluciones. La solución correcta (véase el inciso (a) de la figura) será aquella en donde un punto reconstruido esté frente a ambas cámaras. Los resultados difieren en la dirección del vector de traslación $\mathbf{t}$ y una rotación de 180º sobre el eje Z.

[2]La notación $[\cdot]_{\times}$ refiere a una matriz antisimétrica

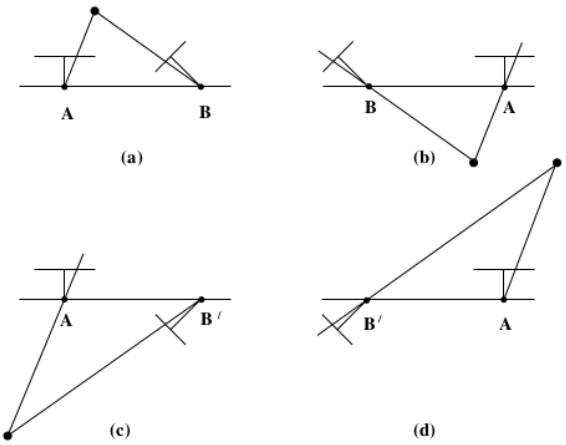


Figura 2.3: Las cuatro posibles soluciones para la reconstrucción a partir de la matriz esencial. Figura tomada de [17]

Para mayor referencia del modelo de cámara, geometría epipolar, algoritmos de estimación de la matriz fundamental y descomposición de la matriz esencial referirse a [17] e incluir [18].

2.3. Características locales

Una característica local es un patrón de imagen que difiere de su vecindad inmediata. Son usualmente asociadas a cambios en las propiedades de la imagen como intensidad, color y textura [19]. En la práctica son puntos, bordes o pequeñas regiones de la imagen.

2.3.1. Detectores de características

Un detector de características es un algoritmo usado para ubicar en una imagen puntos de interés con base en los atributos de ésta. En la literatura se encuentran muchos detectores, por ejemplo: Harris, Good features to track, SURF, SIFT, etc (la exposición de estos detectores se pueden encontrar, por ejemplo, en [20]).

Good Features to Track

El algoritmo conocido como *Good features to track* [21] es una variación del detector de esquinas Harris [22]. Este último se basa en el cálculo de los valores propios λ_1 y λ_2 de la matriz

$$\mathbf{M} = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_y I_x & I_y I_y \end{bmatrix}, \tag{2.15}$$

donde I_x e I_y son gradientes de la imagen en la dirección horizontal y vertical, respectivamente, y $w(x,y)$ es una función de pesos (o función ventana). Además, existe una función que define un criterio para saber si un punto es seleccionado como esquina. En el caso del detector *Good features to track* un punto es de interés si la funcion $R = min(\lambda_1, \lambda_2)$ es mayor a un umbral.

En la Figura 2.4 se muestra la detección de puntos con este algoritmo usando OpenCV[3]. En este algoritmo se ha seleccionado el uso de este tipo de detector debido a que los puntos que encuentra son convenientes para ser seguidos por un rastreador, como se explica más adelante.

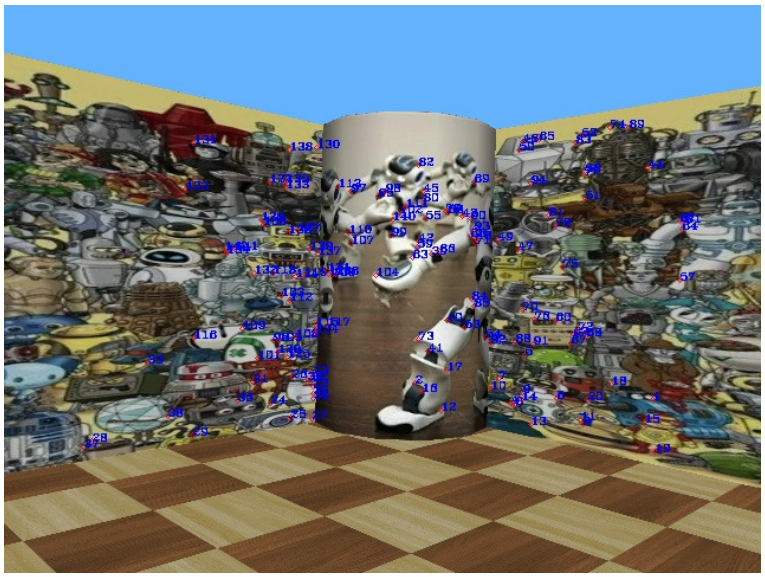


Figura 2.4: Ejemplo de detección con *Good features to track* (los puntos se identifican con números en azul)

[3]Del inglés Open Source Computer Vision

2.3.2. Descriptores de características

Una vez detectados los puntos clave es probable que se requiera identificar a cada uno de forma particular. Un descriptor de características es un algoritmo usado para asociar a cada punto detectado con cualidades que típicamente se encuentran en las variaciones de los píxeles a su alrededor. Existen muchos algoritmos descriptores en la literatura, por ejemplo: SURF, SIFT, BRIEF u ORB (la exposición de estos descriptores se encuentran, por ejemplo, en [20]). A continuación se describirá el tipo de descriptor usado en esta tesis.

Scale Invariant Features Transform: SIFT

Este algoritmo fue propuesto en [23]. La escala de los puntos detectados es usada para seleccionar una imagen con suavizado Gaussiano, $L(x, y)$. Se calcula la magnitud del gradiente ($m(x, y)$) y la orientación ($\theta(x, y)$), como lo indica la ecuación 2.16, para cada píxel en una región de 16 x 16 alrededor del punto de interés. Después se genera un histograma de la orientación, con 36 contenedores, el cual es ponderado con la magnitud del gradiente y una ventana circular Gaussiana (véase el lado izquierdo de la Figura 2.5). Los picos en el histograma corresponden a la orientación del punto, pero aquellas direcciones que tengan un 80 % de la magnitud máxima también serán consideradas representativas y se les asocian un punto de interés en la misma escala y posición.

$$\begin{aligned} m(x,y) &= \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}, \\ \theta(x,y) &= tan^{-1}\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}. \end{aligned} \tag{2.16}$$

Posteriormente, se subdivide en regiones de 4x4 y en cada una es calculado un histograma, con 8 contenedores, a partir de las orientaciones y magnitudes dominantes (véase el lado derecho de la Figura 2.5). Entonces, el descriptor resulta en un vector de 128 elementos, es decir, los valores de cada histograma en cada región (4x4x8).

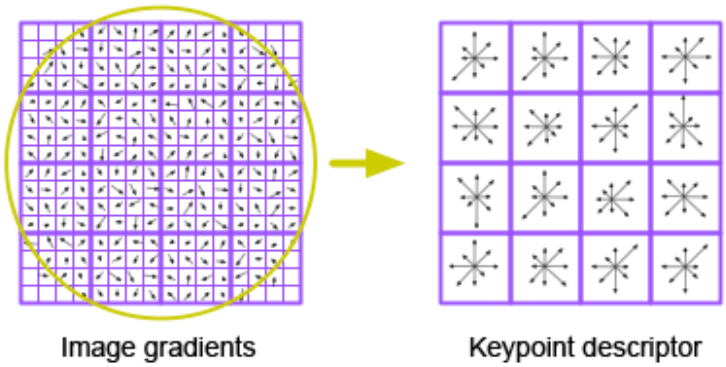


Figura 2.5: Representación esquemática del descriptor SIFT. Figura tomada de `https://www.eecs.tu-berlin.de/fileadmin/fg144/Courses/06WS/scanning/Jonas/html/index.html`

2.3.3. Emparejador de descriptores

Es posible que dos imágenes compartan información, por ejemplo, en una foto de la misma escena tomada de diferente ángulo y distancia. Un emparejador de descriptores es aquel algoritmo usado para buscar entre los descriptores de ambas imágenes candidatos probables a parear.

Emparejamiento por fuerza bruta (*Brute Force*)

Este algoritmo consiste en tomar un descriptor de la primera imagen y compararlo, con base en su distancia Euclideana, con cada descriptor en la segunda imagen. Entonces el emparejador encuentra el más cercano y repite el proceso para cada descriptor de la primera imagen. Un ejemplo se muestra en la Figura 2.6.

2.3.4. *Random Samples and Consensus*: RANSAC

Después del emparejamiento, es posible que haya puntos que no coincidan, estos puntos son llamados *outliers*. En [24] se definen como:

Un dato se considera un outlier si es que no se ajusta al modelo "correcto", instanciado por el conjunto "correcto"(inliers) de parámetros dentro de algún

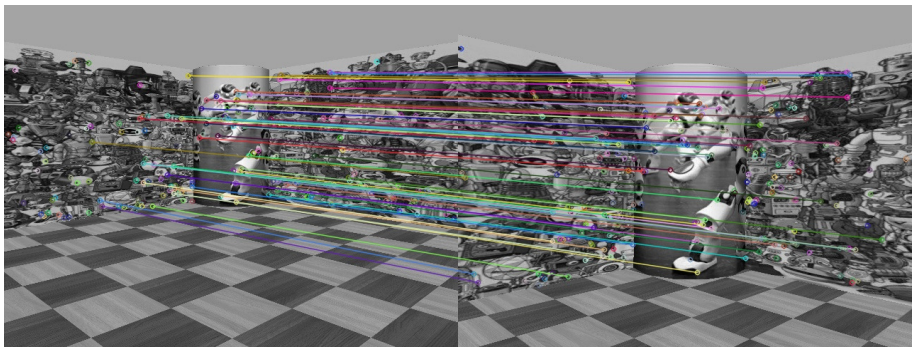


Figura 2.6: Ejemplo de emparejamiento con *Brute Force*

umbral de error que define la máxima desviación atribuible a los efectos del ruido.

De manera general, el algoritmo RANSAC selecciona aleatoriamente un *conjunto de muestras mínimas* del conjunto de datos de entrada, y los parámetros del modelo son calculados usando solamente estos elementos. El algoritmo usa el conjunto de datos más pequeño como sea factible y lo amplía con datos consistentes cuando es posible [17], por lo que esta primera cantidad de elementos es lo suficientemente pequeña sólo para instanciar el modelo.

Posteriormente, comprueba cuales elementos del conjunto total de datos son consistentes con este modelo, es decir, que se encuentran dentro de un umbral. El conjunto de estos elementos es llamado *conjunto consenso*. Si el tamaño del conjunto es menor a otro umbral se selecciona un nuevo subconjunto y se repite lo anterior. El algoritmo termina cuando la probabilidad de encontrar un mejor conjunto consenso cae por debajo de cierto umbral, y el modelo se vuelve a estimar usando los puntos de este último conjunto. Estos puntos se conocen como *inliers*.

En [17] también se encuentran detalles del algoritmo, así como su uso en el cálculo de la matriz fundamental.

2.3.5. Rastreador de Puntos

El detector y el descriptor proporcionan un conjunto limitado de puntos bien localizados e identificables individualmente. La localización de estos puntos, en un conjunto sucesivo de imágenes, puede ser determinada, de manera exacta y estable a través del tiempo, por medio de un algoritmo rastreador.

Rastreador Lucas-Kanade

Este algoritmo fue desarrollado con base en los trabajos [25], [26] y [21]. El rastreo de los puntos se realiza entre dos imágenes que comparten información, asumiendo que tienen brillo constante. Un punto de interés debe presentar un movimiento pequeño, así como una vecindad de puntos alrededor de éste. Entonces, el flujo óptico, es decir, el movimiento aparente de un punto $p(x, y)$ en el tiempo, se puede estimar a partir de la ecuación:

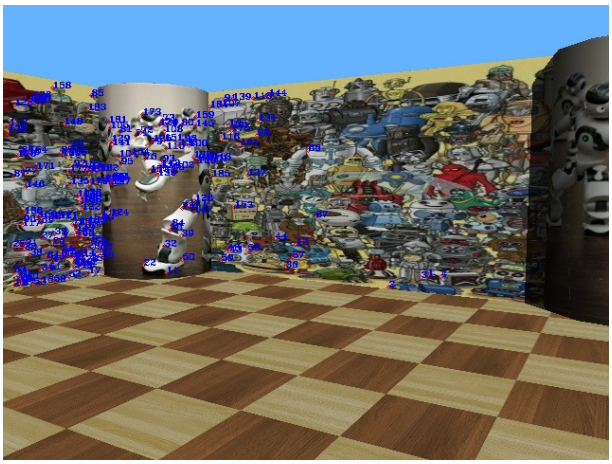
$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}, \tag{2.17}$$

donde I_x e I_y son los gradiente espaciales, e I_t el gradiente temporal y la sumatoria se realiza sobre todos los píxeles en una vecindad de $p(x, y)$.

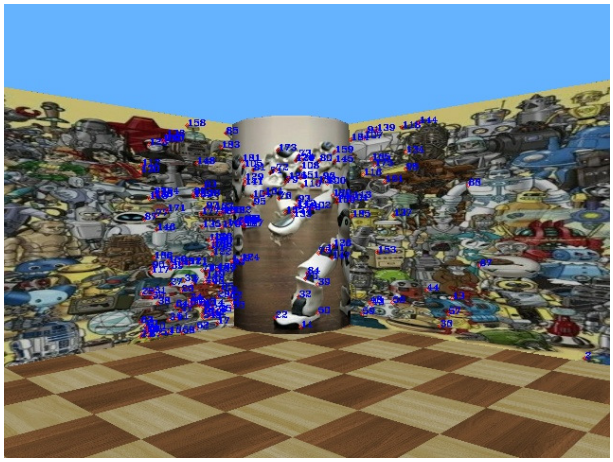
La matriz de la izquierda es la misma que la usada por el detector *Good Features to Track* (véase la ecuación 2.15). Ambas matrices están bien condicionadas para grandes valores propios, es decir, donde los gradientes son diferentes, por lo que los puntos encontrados por el detector son buenas para el rastreo.

Para resolver el problema de grandes desplazamientos se construye una pirámide Gaussiana para un par de imágenes, que pueden pertener a una secuencia de imágenes. Se comienza por las imágenes de más baja resolución, donde se ejecuta el rastreador hasta que converge. Se continua con la siguiente escala donde la posición de las puntos de intéres es la proporcionada por el rastreador en la escala anterior. Se ejecuta de nuevo el algoritmo de rastreo y se itera de esta forma hasta la original.

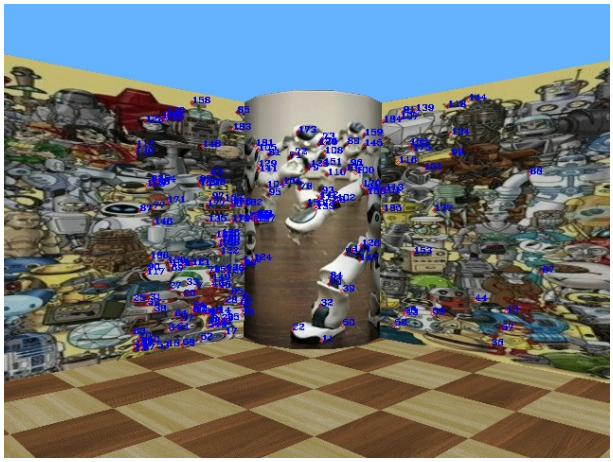
Por último, la Figura 2.7 muestra el uso del rastreador Lucas-Kanade con implementación piramidal de la biblioteca OpenCV. Esta versión fue propuesta en [27]). La Figura 2.7 ejemplifica el rastreo de puntos a través de tres imágenes tomadas de una secuencia mayor. De (a) a (b) de esta figura el rastreo se realizó durante un movimiento de rotación pura, y de (b) a (c) durante un movimiento de traslación pura. En ambos casos se aprecia un buen seguimiento de los puntos.



(a)



(b)



(c)

Figura 2.7: Ejemplo de rastreo de puntos con el método Lucas-Kanade

2.4. Control visual

El control visual, o VSC por sus siglas en inglés, se refiere a la utilización de datos de visión por computadora para controlar el movimiento de un robot. Se usa la referencia [15] para los siguientes apartados.

2.4.1. Componentes básicos del control visual

El objetivo de los esquemas de control visual es minimizar una función de error comunmente definida con la ecuación 2.18

$$\mathbf{e}(t) = \mathbf{s}(\mathbf{m}(t), \mathbf{a}) - \mathbf{s}^*, \tag{2.18}$$

donde $\mathbf{m}(t)$ es un conjunto de medidas de la imagen, $\mathbf{a}$ es un conjunto de parámetros que representan conocimiento potencial adicional, $\mathbf{s}(\mathbf{m}(t), \mathbf{a}) \in \mathbb{R}^{k\times 1}$ es un vector de k características visuales y $\mathbf{s}^*$ es el valor deseado de dichas características.

El esquema clásico de control visual están basados en la estimación del Jacobiano de la imagen o matriz de interacción, la cual relaciona el cambio de velocidades del robot con el cambio en las características visuales en el espacio imagen. La ecuación 2.19 expresa la relación entre el cambio con el tiempo de las características visuales y la velocidad de la cámara en el espacio.

$$\dot{\mathbf{s}} = \mathbf{L}_\mathbf{s}\mathbf{v}_c, \tag{2.19}$$

donde $\mathbf{L}_\mathbf{s} \in \mathbb{R}^{k\times 6}$ es la *matriz de interacción* y $\mathbf{v}_c = [\boldsymbol{v}_c, \boldsymbol{\omega}_c]^\top \in \mathbb{R}^{6\times 1}$, con $\boldsymbol{v}_c$ como la velocidad lineal instantánea del origen del sistema de coordenadas de la cámara y $\boldsymbol{\omega}_c$ la velocidad angular instántanea de este sistema de referencia. Considerando $\mathbf{s}^*$ constante, de las ecuaciones anteriores se obtiene

$$\dot{\mathbf{e}} = \mathbf{L}_\mathbf{e}\mathbf{v}_c, \tag{2.20}$$

donde $\mathbf{L}_\mathbf{e} = \mathbf{L}_\mathbf{s}$. Si se considera $\mathbf{v}_c$ como la entrada de control, y como desea asignar una disminución exponencial del error, es decir $\dot{\mathbf{e}} = -\lambda\mathbf{e}$, se concluye que

$$\mathbf{v}_c = -\lambda \widehat{\mathbf{L}_\mathbf{e}^+} \mathbf{e}, \tag{2.21}$$

donde $\widehat{\mathbf{L}_\mathbf{e}^+}$ es una estimación de la pseudo inversa de Moore-Penrose de la matriz de interacción, ya que en la práctica es imposible conocer perfectamente $\mathbf{L}_e$ o $\mathbf{L}_e^+$. La pseudoinversa se define como $\mathbf{L}_e^+ = (\mathbf{L}_e^\top \mathbf{L}_e)^{-1} \mathbf{L}_e^\top$.

2.4.2. Control visual basado en posición

En el caso del *control visual basado en posición*, el vector $\mathbf{s}$ es en un conjunto de parámetros tridimensionales estimados por medio de las mediciones de la imagen. Es usual definir el control en términos de la parametrización usada para representar la posición de la cámara con respecto a un sistema coordenado, y los parámetros $\mathbf{a}$, de los que se hace referencia en la sección 2.4.1, son los parámetros intrínsecos de la cámara.

Se pueden elegir dos formas distintas de definir $\mathbf{s} = (\mathbf{t}, \theta\mathbf{u})$, donde $\mathbf{t}$ es un vector de translación y $\theta\mathbf{u}$ es la parámetrización ángulo/eje para la rotación (véase [28]). En la primera de éstas, $\mathbf{t}$ define las coordenadas del origen del sistema de referencia del objeto con respecto al marco de referencia actual de la cámara y al deseado de la cámara, por lo que $\mathbf{s} = ({}^c\mathbf{t}_o, \theta\mathbf{u})$ y $\mathbf{s}^* = ({}^{c^*}\mathbf{t}_o, \mathbf{0})$, respectivamente. Entonces, el error es $\mathbf{e} = ({}^c\mathbf{t}_o - {}^{c^*}\mathbf{t}_o, \theta\mathbf{u})$.

La matriz de interacción es, en este caso, una matriz cuadrada y usando la ecuación 2.21 resulta [15]:

$$\widehat{\mathbf{L}_\mathbf{e}^{-1}} = \begin{bmatrix} -\mathbf{I}_3 & [{}^c\mathbf{t}_o]_\times \mathbf{L}_{\theta\mathbf{u}}^{-1} \\ \mathbf{0} & \mathbf{L}_{\theta\mathbf{u}}^{-1} \end{bmatrix}, \tag{2.22}$$

con

$$\mathbf{L}_{\theta\mathbf{u}} = \mathbf{I}_3 - \frac{\theta}{2}[\mathbf{u}]_\times + \left(1 - \frac{sinc(\theta)}{sinc^2(\theta/2)}\right)[\mathbf{u}]_\times^2. \tag{2.23}$$

Desarrollando, y ya que $\mathbf{L}_{\theta\mathbf{u}}^{-1}\theta\mathbf{u} = \theta\mathbf{u}$ [15], se obtiene

$$\begin{cases} \mathbf{v}_c = -\lambda(({}^{c^*}\mathbf{t}_o - {}^c\mathbf{t}_o) + [{}^c\mathbf{t}_o]_\times \theta\mathbf{u}) \\ \boldsymbol{\omega}_c = -\lambda\theta\mathbf{u} \end{cases} \tag{2.24}$$

En la otra elección de $\mathbf{s}$, ${}^{c^*}\mathbf{t}_c$ define las coordenadas del marco de referencia actual de la cámara con respecto al deseado, por lo que $\mathbf{s} = ({}^{c^*}\mathbf{t}_c, \theta\mathbf{u})$ y $\mathbf{s}^* = (\mathbf{0}, \mathbf{0})$. Entonces el error es $\mathbf{e} = \mathbf{s}$ y la matriz de interacción resulta

$$\mathbf{L}_e = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{L}_{\theta\mathbf{u}} \end{bmatrix} \tag{2.25}$$

Desarrollando mediante la ecuación 2.21 se obtiene un movimiento traslacional y rotacional desacoplados, como lo muestra la ecuación 2.26.

$$\begin{cases} \mathbf{v}_c = -\lambda \mathbf{R}^\top {}^{c^*}\mathbf{t}_c \\ \boldsymbol{\omega}_c = -\lambda\theta\mathbf{u} \end{cases} \tag{2.26}$$

Para este esquema en partícular se requiere una cámara calibrada y generalmente usa una técnica de estimación de su posición y orientación. Existe la desventaja de que cualquier error en la estimación de los parámetros requeridos por la matriz de interacción afectan la establidad del lazo de control [7].

2.4.3. Control visual basado en imagen

En el *control visual basado en imagen*, el vector $\mathbf{s}$ puede consistir en un conjunto de características disponibles de manera inmedianta en la imagen. En este caso las medidas de la imagen $\mathbf{m}$ son usualmene las coordenadas en píxeles de un conjunto de puntos en la imagen, y los parámetros $\mathbf{a}$, de los que se hace referencia en la ecuación 2.18, son los parámetros intrínsecos de la cámara.

De la proyección de un punto en el espacio $\mathbf{X} = (X, Y, Z)$ a un punto en la imagen $\mathbf{s} = (x, y)$ se tiene que $x = X/Z$ y $y = Y/Z$, de donde tomando las derivadas se obtienen las siguientes ecuaciones:

$$\begin{cases} \dot{x} = (\dot{X} - x\dot{Z})/Z \\ \dot{y} = (\dot{Y} - y\dot{Z})/Z \end{cases} \tag{2.27}$$

Se pueden relacionar las velocidades del punto en el espacio con las velocidades de la cámara del hecho de que $\dot{\mathbf{X}} = -\mathbf{v}_c - (\mathbf{w}_c \times \mathbf{X})$, de donde resulta en un sistema de ecuaciones tal que que puede ser escrito como

$$\dot{\mathbf{s}} = \mathbf{L_s}\mathbf{v}_c, \tag{2.28}$$

expresando la forma analítica de la matriz de interacción como

$$\mathbf{L_s} = \begin{bmatrix} -1/Z & 0 & x/Z & xy & -(1+x^2) & y \\ 0 & -1/Z & y/Z & 1+y^2 & -xy & -x \end{bmatrix}. \tag{2.29}$$

Se observa que se debe estimar o aproximar el valor de Z y los parámetros intrínsecos de la cámara están relacionados con el cálculo de los valores x y y, por lo que se debe usar una aproximación de esta matriz.

Este esquema de control visual es robusto a los parámetros intrínsecos de la cámara, no obstante presenta problemas de mínimos locales y su convergencia está restringida a una región alrededor de la pose deseada. Esto generalmente se debe a la matriz de interacción, la cual necesita una pseudoinversión durante la ejecución del control [7].

2.4.4. Control visual basado en geometría epipolar

En [16] se propone un esquema como el del apartado anterior y para resolverlo se basa en geometría epipolar. Usando la geometría epipolar, la mayoría de los parámetros que especifican la diferencia entre la posición y la orientación de la cámara entre la imagen actual y la imagen objetivo son recobrados, excepto por la profundidad [7].

Sean los puntos de la imagen $\mathbf{p}_{i/F_1}$ en la primera vista de la cámara y $\mathbf{D}_{i/F_2}$ las líneas epipolares correpondientes a estos puntos en la vista de referencia. El artículo propone que si usando el control visual basado en imagen se puede mover la cámara de tal forma que los puntos $\mathbf{p}_{i/F_k}$ en la k-ésima vista se encuentran con dichas líneas epipolares, entonces se puede controlar la orientación y la dirección de translación de la cámara.

Si los puntos se expresan en coordenadas homogéneas, es decir $\mathbf{p}_i(\mathbf{X}) = [x_i(X)\ \ y_i(X)\ \ 1]^\top$, las lineas epipolares se obtienen a partir de la matriz esencial como se indica a continuación:

$$\mathbf{D}_i(\mathbf{X}_2) : [a_i\ b_i\ c_i]^\top = \mathbf{E}\mathbf{p}_i(\mathbf{X}_1), \tag{2.30}$$

donde $\mathbf{X}_1$ es la primera vista y $\mathbf{X}_2$ la vista de referencia. Se utiliza el enfoque de *función de tarea* para regular a cero la función de salida $\mathbf{e}(\mathbf{X})$ en la ecuación 2.31, donde $\mathbf{e}_{1i}(\mathbf{X})$ representa la distancia con signo del punto $\mathbf{p}_i(\mathbf{X})$ a la linea epipolar en la ecuación 2.30.

$$\mathbf{e}_{1i}(\mathbf{X}) = \mathbf{p}_i^\top(\mathbf{X})(\mathbf{E}\mathbf{p}_i(\mathbf{X}_1)). \tag{2.31}$$

La función de tarea anterior se usa para converger a las lineas epipolares usando sólo la rotación, es decir se usan tareas descopladas. La segunda tarea permite que estos últimos puntos, ya sobre las lineas epipolares, se deslicen a lo largo de éstas hasta alcanzar los puntos objetivo. Entonces, $\mathbf{e}_1$ es la tarea prioritaria y $\mathbf{e}_2$ corresponde a una *función costo* h_s que debe minimizarse bajo la restricción $\mathbf{e}_1 = 0$. Estas condiciones dan como resultado la ecuación 2.32,

$$\mathbf{e} = \mathbf{W}^+\mathbf{e}_1 + \beta(\mathbf{I}_6 - \mathbf{W}^+\mathbf{W})\frac{\partial h_s}{\partial X}, \tag{2.32}$$

donde:

- β es un escalar positivo.
- $\mathbf{W} = [\mathbf{0}_{3\times3}\ \mathbf{I}_{3\times3}]^\top$.
- $\mathbf{W}^+$ es la pseudoinversa de $\mathbf{W}$.
- $h_s = ({}^k\mathbf{t}_1 - {}^2\,\mathbf{t}_1)^2$.

Derivando la ecuación 2.31 con respecto sólo a las variables de rotación de la cámara, se tiene la ecuación 2.33 donde se indica la forma analítica de la matriz de interacción, la cual ya no depende de la estructura 3D de la escena, es decir de la profundidad.

$$\begin{aligned} \dot{\mathbf{e}}_{1i} &= [a_i\ b_i]\mathbf{L}_{of_{rot}}^\top[\boldsymbol{\omega}_c] = \mathbf{L}_{rot}^\top[\omega_c], \\ \mathbf{L}_{of_{rot}}^\top &= \begin{bmatrix} x_iy_i & -(1+x_i^2) & y_i \\ (1+y_i^2) & -x_iy_i & x_i \end{bmatrix}. \end{aligned} \tag{2.33}$$

Ya que se desea asegurar una convergencia exponencial de $\mathbf{e}_1$, es decir $\dot{\mathbf{e}} = -\lambda\mathbf{e}$, y de las ecuaciones 2.32 y 2.33, la entrada de control resulta en

$$\mathbf{v}_c = -\lambda \mathbf{W}^{+} \mathbf{L}_{of_{rot}}^{\top +} \mathbf{e}_{1i}(X) - 2\beta(\mathbf{I}_6 - \mathbf{W}^{+}\mathbf{W})[^2\mathbf{t}_k \; \mathbf{0}], \tag{2.34}$$

donde $\mathbf{L}_{of_{rot}}^{\top +}$ denota la pseudoinvera de la matriz de interacción.

Entonces, el control está dividido en dos etapas. En la primera, la meta es alinear los puntos proyectados en la primera vista con las líneas epipolares en la vista de referencia. Por lo tanto, la rotación se mantiene funcionando hasta que $\|\mathbf{e}_{i1}(X)\| < \epsilon$, donde ϵ es un umbral que indica la máxima distancia de los puntos a las líneas epipolares, de tal forma que la velocidad angular sería muy cercana a cero. En la segunda etapa, el objetivo es mover estos puntos a lo largo de dichas líneas, sólo con la traslación, hasta donde se encuentran los puntos proyectados en la vista de referencia.

2.5. Marcha humanoide

En la marcha del robot cada paso consiste en una fase de soporte simple y una fase de soporte doble. La fase de soporte simple es aquella donde el contacto con el suelo se realiza sólo con una pierna, mientras que la otra se balancea hacia el punto de apoyo siguiente. La fase de soporte doble se presenta cuando la pierna que se encontraba en el aire toca el suelo, por lo que ambas se mantienen en contacto con el suelo, y se realiza el cambio de soporte inclinando el cuerpo hacia la pierna que permanecerá apoyada [6], ver la Figura 2.8.

Figura 2.8: Marcha humanoide. Figura tomada de
`http://www.cs.ubc.ca/~van/papers/Simbicon.htm`

El polígono de soporte está formado por el envolvente convexo alrededor de los puntos de soporte en el suelo. En el caso de la fase de soporte simple es el área que incluye todos los puntos de contacto del pie de apoyo, como en (a) de la Figura 2.9. En la fase de soporte doble, es el área de contacto

de ambos pies más el espacio que existe entre ellos, como en (b) de la misma figura.

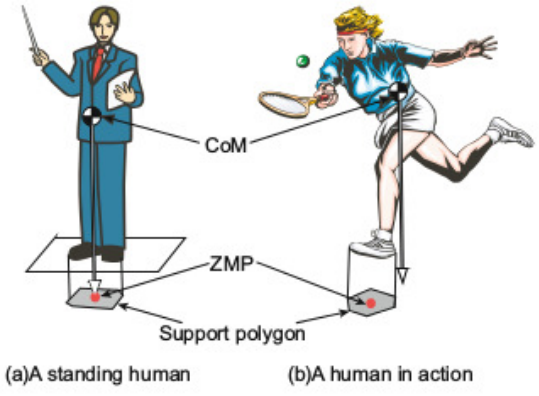


Figura 2.9: Punto de momento nulo, polígono de soporte y centro de masa. Figura tomada de [3]

Si el humanoide no tiene movimiento o un movimiento muy lento, experimenta sólo fuerzas gravitacionales, las cuales se pueden remplazar por una fuerza virtual resultante que actúa sobre su centro de masa, CoM por sus siglas en inglés. Si esta fuerza se proyecta en el suelo tendrá las mismas coordenadas horizontales del CoM, como se observa en la Figura 2.9.

El Punto de Momento Nulo, o ZMP por sus siglas en inglés, se define como el punto en el suelo donde el momento neto de las fuerzas de inercia y gravedad no tienen componentes a lo largo de los ejes horizontales [3].

Para velocidades con las cuales se pueda considerar una marcha en equilibrio estático, es necesario que el ZMP y la proyección del CoM se encuentren dentro del polígono de soporte (véase (a) de la Figura 2.9). En el caso del criterio de estabilidad dinámica sólo es necesario que el ZMP se encuentre dentro del polígono de soporte (véase (b) de la Figura 2.9), lo que significa que los momentos están siendo totalmente contrarrestados por una fuerza de reacción vertical aplicada al suelo.

Los trabajos [3], [5] y [6] presentan con detalle el tratamiento de los temas de esta sección, los cuales no se desarrollan con más amplitud ya que están fuera de los alcances de esta tesis.

2.6. Control de caminata del robot NAO

De acuerdo con la documentación en línea de Aldebaran[4], NAO usa un modelo dinámico simple basado en [29] y se resuelve usando programación cuadrática, como se propone en [30].

En [31] se describe el algoritmo de caminata implementado en NAO, el cual se muestra en la Figura 2.10.

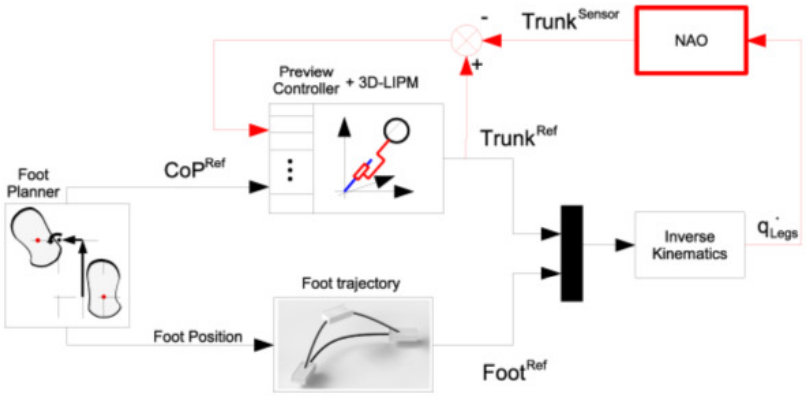


Figura 2.10: Lazo cerrado de caminata. Figura tomada de [31]

El controlador tiene un planificador local de pasos, el cual considera que el robot permanece en balance por lo que el ZMP es igual al centro de presión, CoP por sus siglas en inglés. El CoP esta en el centro el área de soporte del pie de apoyo durante la fase de soporte simple y se mueve al siguiente pie de apoyo durante la fase de soporte doble.

La posición del pie se deduce del planificador de pasos y su posición se decribe con una matriz homogénea. La trayectoria del pie es calculada a través de la generalización de un *spline*[5] cúbico, el cual garantiza una trayectoria suave con respecto a las velocidades lineales y angulares.

El modelo dinámico usado es el péndulo invertido lineal de la Figura 2.11. Este modelo no toma en cuenta los efectos de la inercia debido a los movimientos de las diferentes partes del cuerpo. Así, como supone que las piernas no tienen masa ni inercia, también considera que el tronco equivale

[4] `http://doc.aldebaran.com/1-14/naoqi/motion/control-walk.html`
[5] Un *spline* es una curva suave de seguimiento.

al CoM, ya que aquí se concentra la mayor cantidad de masa, y supone que el CoM no se mueve verticalmente, por lo que la posición longitudinal p del CoP se define por

$$p = x - \ddot{x} z_c / g, \tag{2.35}$$

donde x es la posición horizontal del CoM, $\ddot{x}$ la aceleración horizontal, z_c la altitud y g el valor de la gravedad.

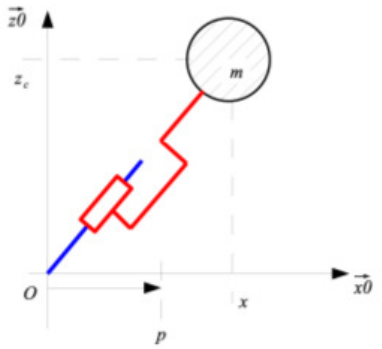


Figura 2.11: Modelo Lineal del Péndulo Invertido. Figura tomada de [31]

Este modelo desacopla los movimientos longitudinales (de avance) y laterales, por lo que este último se modela de la misma forma.

Por último, como se observa en la Figura 2.10, el lazo cerrado se basa en la realimentación de la posición real del CoM, la cual es obtenida a través de la cinemática directa con los sensores de las articulaciones. Finalmente, el control de locomoción, el cual usa un esquema de control predictivo [30], genera una velocidad deseada del CoM del robot mientras mantiene el balance del mismo (estabilidad dinámica).

El resto del desarrollo matemático se encuentra en los artículos citados en esta sección y no se considera necesario tratarlo en este trabajo.

Capítulo 3

Control visual en Matlab

En este capítulo se exponen los resultados de simulaciones realizadas en Matlab de los dos esquemas de control visual expuestos en las secciones 2.4.2 y 2.4.4. Estas simulaciones se realizaron con ayuda del *toolbox* descrita en [32] y del *toolbox* en [33].

En cada sección se indican los valores de los parámetros usados y se muestran las gráficas de las velocidades lineales y angulares de la cámara, la evolución del error promedio de píxeles, la ubicación de los puntos en el plano imagen desde la posición actual hasta la posición objetivo, y la posición y orientación de la cámara en el espacio.

3.1. Control visual basado en posición

A continuación se presentan los detalles de la simulación del esquema basado en posición descrito en la sección 2.4.2. La ubicación inicial de la cámara es tal que, para llegar a la ubicación deseada, tiene que describir un movimiento general. La ubicación deseada está en el origen del sistema de referencia mundo y no tiene rotación. Se simulan puntos en el espacio, cuya posición está dada de forma aleatoria con la condición de que la cámara pueda observarlos.

Por lo tanto, la imagen objetivo está definida por la proyección de estos puntos en el plano imagen de la cámara en la localización objetivo. Para generar las imágenes sintéticas tanto de la imagen objetivo como de la imagen

actual se aplicó el modelo de cámara descrito en la sección 2.1, con unos parámetros de cámara:

$$K = \begin{pmatrix} 400 & 0 & 320 \\ 0 & 400 & 240 \\ 0 & 0 & 1 \end{pmatrix}. \tag{3.1}$$

Estos parámetros se propusieron de tal forma que se simule una cámara real. El lazo de control se detiene con la condición de que el error promedio de píxeles, ecuación 3.2, sea menor a un umbral.

$$error = \frac{1}{N_P} \sum_{i=1}^{N_P} ||query_i - train_i||, \tag{3.2}$$

donde N_P es el número de puntos, *query* los puntos en la imagen actual y *train* los puntos en la imagen objetivo.

Para obtener las soluciones correctas del método de descomposición de la matriz esencial (véase 2.2.3) a partir de la función `fmatrix` de la *vision toolbox* [32], la cual regresa dos soluciones para $\mathbf{R}$ y dos soluciones para $\mathbf{t}$, se considero lo siguiente:

1. La rotación en cualquier eje no será mayor a $\pi/2$ radianes, por lo que los valores de la diagonal principal de la matriz de rotación $\mathbf{R}$ son siempre positivos. Esta consideración es válida dado que el campo de vista de la cámara convencional no permite rotaciones mayores sin que la escena salga del campo de vista.

2. El vector de traslación $\mathbf{t}$ sólo indica la dirección, no la magnitud, por lo que debe estar normalizado y la elección de su sentido debe ser acorde al cálculo de la matriz fundamental. Esto es, si la matriz fundamental se calcula de tal forma que el sistema de referencia está en la ubicación objetivo, entonces el vector $\mathbf{t}$ con componentes en z negativa es el correcto.

Entonces, sean $\mathbf{R}^1$, $\mathbf{R}^2$, $\mathbf{t}^1$ y $\mathbf{t}^2$ las soluciones de la descomposición, el Algoritmo 1 explica la elección correcta, donde R_{ij} indica el elemento ij de la matriz y t_k la componente k del vector.

Algoritmo 1 Elección de $\mathbf{R}$ y $\mathbf{t}$ de la descomposición de la matriz esencial

1: **Función** findR
2: **Si** $R^1_{11} > 0$ && $R^1_{22} > 0$ && $R^1_{33} > 0$ **entonces**
3: $\mathbf{R} = \mathbf{R}^1$
4: **Si no**
5: $\mathbf{R} = \mathbf{R}^2$

6: **Función** findt
7: **Si** $t^1_3 < 0$ **entonces**
8: $\mathbf{t} = \mathbf{t}^1$
9: **Si no**
10: $\mathbf{t} = \mathbf{t}^2$

Debido al punto 2, la constante $error/error\_inicial$, donde $error\_inicial$ es el valor del error al comienzo de las iteraciones, premultiplica a $\mathbf{t}$ para que la velocidad pueda disminuir, ya que se desea una convergencia exponencial del error.

En la Figura 3.1 se presenta la posición de los puntos proyectados en el plano imagen al inicio (rojo), durante el movimiento de la cámara (azul) y en la posición objetivo (verde). De manera análoga, se observa la posición inicial de la cámara, la final y la objetivo en color rojo, azul y verde, respectivamente. La posición inicial es $(5, 0, -1)$ y la orientación inicial es $(0, -10, 5)$, la cual indica la rotación en grados sobre el eje x, y y z respectivamente. La trayectoria de la cámara durante la ejecución del control se ilustra en rosa. En ambos casos se aprecia que la cámara llega a la posición y orientación deseadas.

La Figura 3.2 muestra el comportamiento de las entradas de control y el error pomedio. Tanto las velocidades lineales y angulares como el error tienen una convergencia exponencial. Se consideró un umbral de 5 píxeles de error promedio para detener el lazo de control. En estas simulaciones es posible reducir este umbral para obtener una mayor precisión del control, sin embargo esto no es factible en experimentos más realistas, como se verá en el siguiente capítulo, y aquí se ha optodo por dejarlo en un valor no tan bajo.

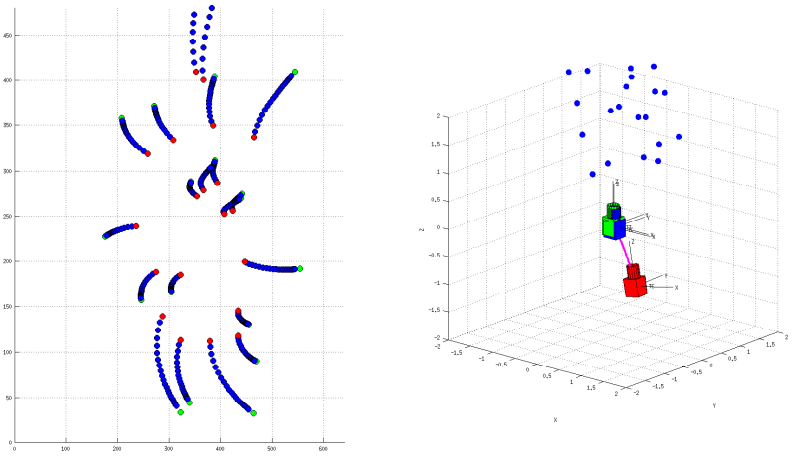


Figura 3.1: Puntos en el plano imagen y movimiento de la cámara

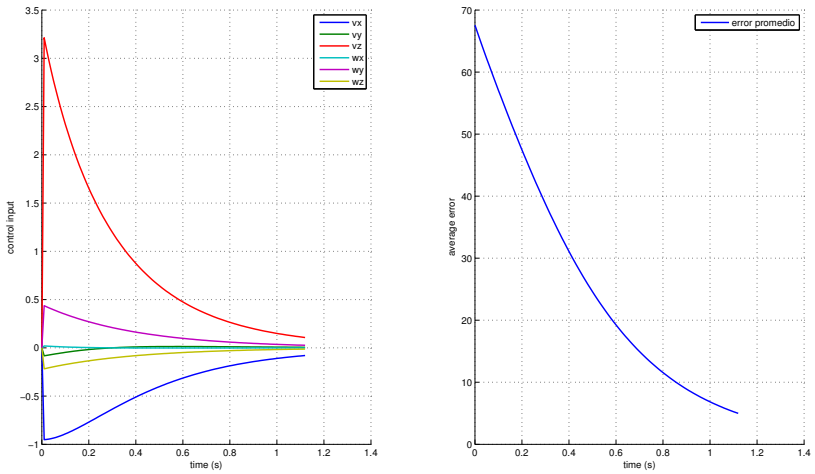


Figura 3.2: Entradas de control y error promedio

El ruido puede ocasionar problemas de convergencia, como se observa en el plano imagen de la Figura 3.3, donde los puntos quedan desviados del objetivo. No obstante, la cámara llega a la orientación deseada.

En la Figura 3.4 se muestra otra simulación de control 3D para la misma condición inicial, pero incluyendo ruido de imagen. Este ruido se introduce

como un ruido Gaussiano que se suma a la ubicación real de los puntos en píxeles. Las gráficas muestran como el ruido afecta a las velocidades y al cálculo del error promedio. El umbral de paro tiene el mismo valor que en el caso anterior.

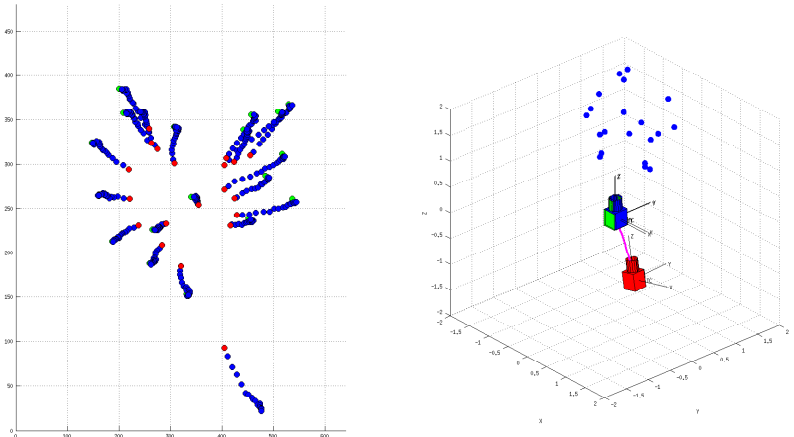


Figura 3.3: Puntos en el plano imagen y movimiento de la cámara simulados con ruido

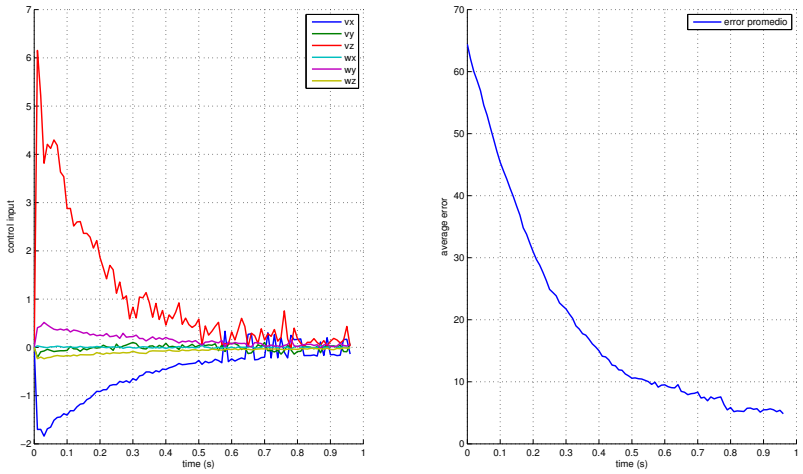


Figura 3.4: Entradas de control y error promedio simulados con ruido

3.2. Control visual basado en imagen

En el caso de la simulación del esquema basado en imagen descrito en la sección 2.4.2 se siguieron las mismas condiciones que en la anterior simulación, es decir la cámara describe un movimiento general, los puntos en el espacio se calculan de manera aleatoria y la localización objetivo está en el origen del sistema de referencia y no presenta rotación. Los parámetros de cámara son los mismos que en la simulación anterior (véase la ecuación 3.1). La posición inicial es $(0.25, 0.0, -1)$ y la orientación inicial es $(0, 0, 15)$, la cual indica la rotación en grados sobre el eje x, y y z respectivamente.

La programación esta basada en un ejemplo del *toolbox* descrita en [33]. Se recurre al mismo método del Algoritmo 1 y al mismo cálculo del error en la ecuación 3.2.

De igual forma, la Figura 3.5 muestra en rojo la posición inicial de los puntos y de la cámara, y en verde la posición objetivo de los puntos y de la cámara. También se muestran en rojo las líneas epipolares. En azul se muestra la posición de los puntos, primero llegando a la línea epipolar y posteriormente deslizandose sobre ésta para finalmente alcanzar los puntos objetivo. La trayectoria de la cámara se ilustra en rosa y su posición final en azul. El umbral para detener el movimiento es de 5 píxeles de error promedio.

En las velocidades de la Figura 3.6 se observa que durante un tiempo sólo existe velocidad angular, es decir la cámara sólo rota, hasta que aquélla es muy cercana a cero (donde los puntos están sobre las líneas epipolares). La norma de la función de tarea en la ecuación 2.32 también descae exponencialmente durante este tiempo. En seguida la cámara se comienza a desplazar en los tres ejes (los puntos se deslizan por las líneas epipolares) hasta que es detenida por el umbral del error promedio en píxeles. Tanto las velocidades lineales como la norma de la función de tarea convergen exponencialmente.

Si se afecta con ruido a la simulación se observa en la Figura 3.7 que los puntos presentan más problemas durante la traslación, es decir mientras de deslizan sobre las líneas epipolares. Esto se debe a que, durante esta fase, la velocidad angular no es nula o lo suficientemente pequeña, afectando el desempeño (ver la Figura 3.8). Este remanente afecta principalmente antes de que termine el desplazamiento. No obstante, la cámara logra llegar a la posición deseada y la norma de la función de tarea converge exponencialmente en ambas fases.

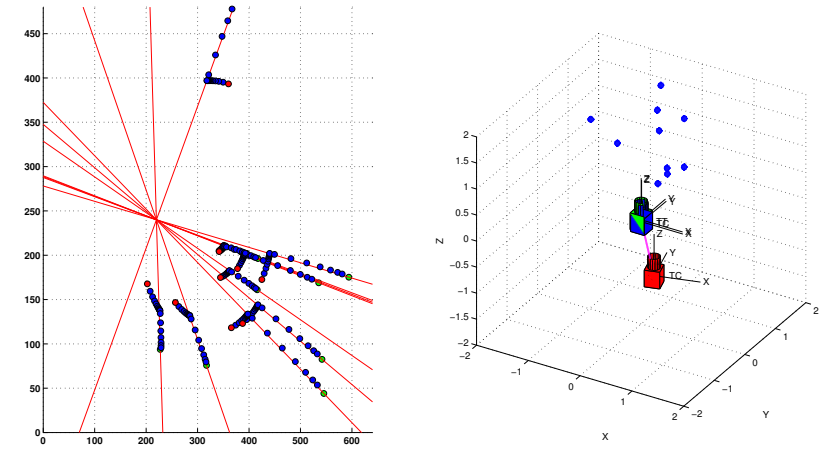


Figura 3.5: Puntos en el plano imagen y movimiento de la cámara

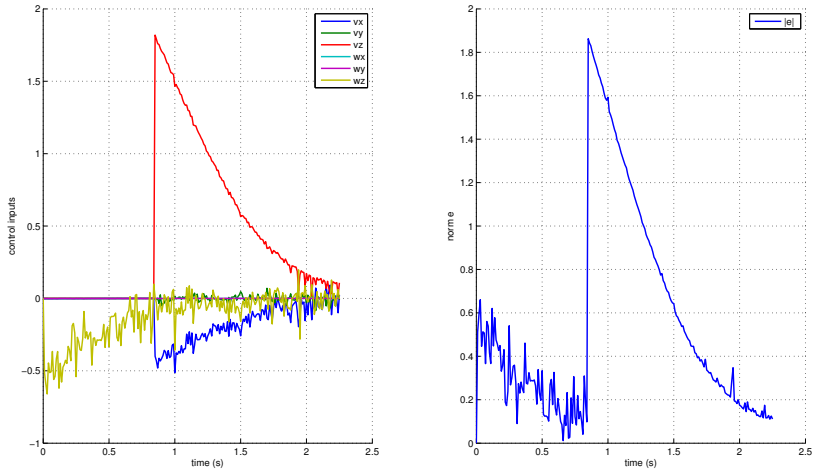


Figura 3.8: Entradas de control y norma de la función de tarea simulados con ruido

En conclusión, el esquema basado en posición corrige simultáneamente todos los grados de libertad, mientras que el esquema basado en imagen corrige primero los tres grados de libertad que permiten rotar a la cámara y posteriormente los otros tres que permiten su desplazamiento.

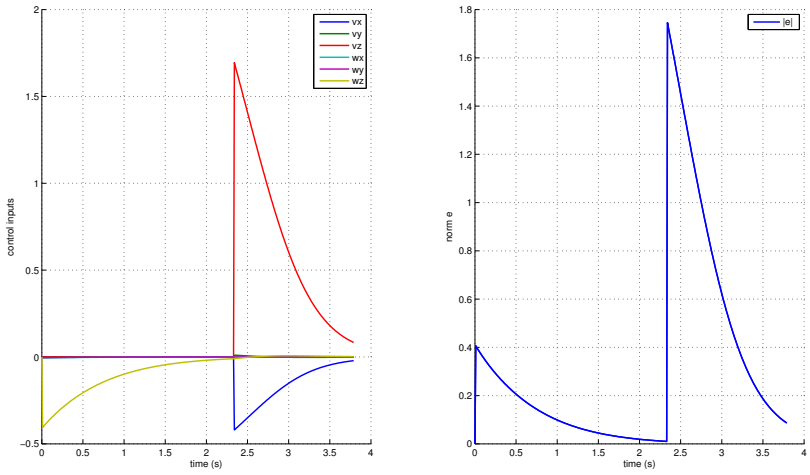


Figura 3.6: Entradas de control y norma de la función de tarea

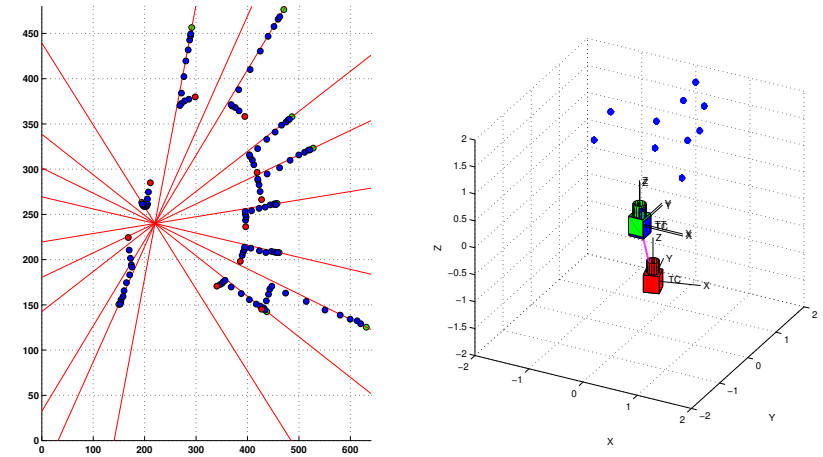


Figura 3.7: Puntos en el plano imagen y movimiento de la cámara simulados con ruido

Capítulo 4

Implementación de algoritmos de control visual

En este capítulo se describe la forma en que se implementó la simulación para evaluar los esquemas de control visual en un entorno más realista que las simulaciones previas de Matlab, comenzando por los detalles en el mundo virtual y modificaciones hechas al modelado del robot. Posteriormente, se da un panorama de la programación del robot en el mundo virtual y se explican los cambios hechos a dicho programa. Por último se desarrollan, a manera de pseudocódigo, los algoritmos de control implementados.

4.1. Webots

Webots, software creado por la empresa *Cyberbotics*, es un ambiente de desarrollo usado para modelar, programar y simular robots móviles en ambientes virtuales 3D físicamente realistas.

La interfaz gráfica de Webots está compuesta por lo siguiente [34] (Véase la Figura 4.1):

- Ventana 3D (a): permite interactuar con la simulación 3D.
- Árbol de escena (b): representación jerárquica del mundo.
- Editor de texto (c): permite editar código fuente.

- Consola (d): muestra los mensajes de compilación y salidas de ejecución.

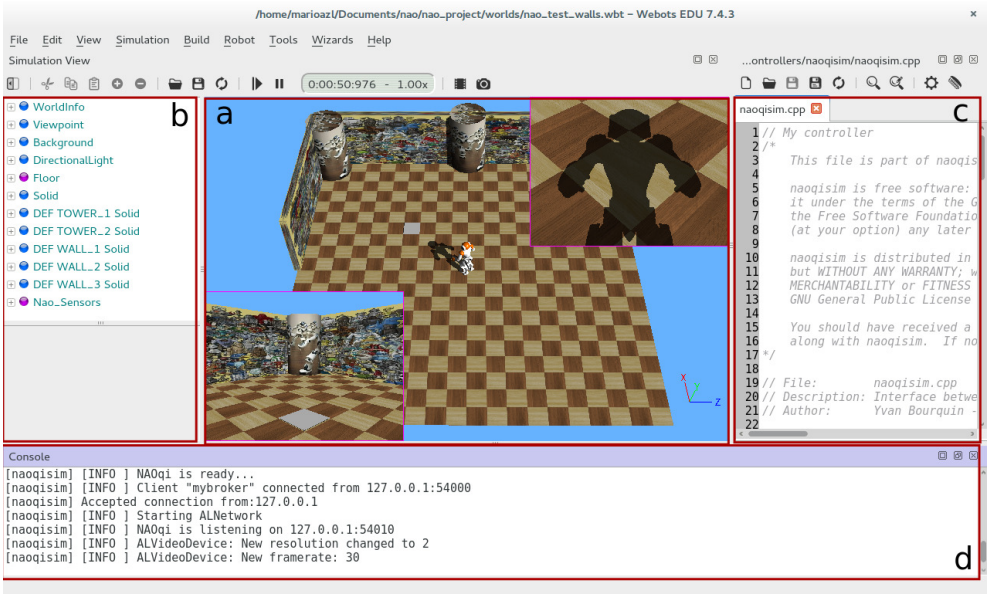


Figura 4.1: Interfaz gráfica de usuario. Ver descripción en el texto

Una simulación en este paquete requiere de un archivo mundo y al menos un programa controlador. Opcionalmente se pueden usar complementos para modificar el comportamiento físico regular de Webots o para extender la funcionalidad del controlador.

4.1.1. Mundo

Un mundo, archivo con extensión .wbt, es una descripción 3D de las propiedades de los robots y de cada objeto de su entorno [34]. Este archivo puede depender de archivos *proto* externos y de texturas.

El árbol de escena contiene la información que describe al mundo simulado a tráves de una lista de nodos, en los cuales se puede definir la representación gráfica del robot y su ambiente, y a su vez contener otros nodos con campos de valores numéricos o cadenas de texto para modificar las propiedades del mundo.

En mundo creado para este trabajo está constituido por los nodos básicos

WolrdInfo, *Viewpoint*, *Background*, *Directionallight* y *Floor*, y de manera partícular por:

- Nodos sólidos, los cuales definen las paredes y las columnas cilíndricas.
- Un nodo robot, el cual simula al robot NAO.

Para la simulación implementada, el origen del sistema de referencia mundo de Webots está en el centro del cuadro (véase en (a) de la Figura 4.1) que define al piso.

Un nodo sólido contiene otros nodos que permiten definir sus dimensiones y textura, así como su posición y orientación con respecto al sistemas de referencia del mundo. La textura es dada por las imágenes que se ven en la Figura 4.2. Estas figuras fueron elegidas para que se puedan detectar una gran cantidad de características locales en ellas (Véase la sección 2.3).

Figura 4.2: Texturas de los sólidos (vista de la cámara superior de NAO)

La localización de los objetos definidos por los nodos se realizó de tal forma que el robot observara al menos tres planos, es decir las dos paredes y la columna de la Figura 4.2. De esta forma es posible tener un buen conocimiento del cálculo de la geometría epipolar (Véase la sección 2.2).

El nodo del robot se modificó a través del archivo proto y del controlador. Estas modificacione se explican en los siguientes dos apartados.

4.1.2. Proto

El archivo .proto es aquel que define los nodos con los que cuenta el prototipo de un robot, por ejemplo elementos sólidos, textura, sensores, etc. La estructura de este archivo es [34]:

$$PROTO\ nombre\ [interfaz]\ cuerpo$$

El nombre debe coincidir con el nombre del archivo. La interfaz es una secuencia de declaraciones de campo, los cuales equivalen a los nodos en el árbol de escena; cada uno especifica el tipo, nombre y valores predeterminados de ese campo usando la siguiente sintaxis [34]:

$$field\ tipo\ nombre\ valor$$

El robot NAO se define a través del archivo Nao.proto que proporciona Webots. El modelo se diseñó de acuerdo a la documentación en línea de *Aldebaran Robotics*, por lo que el robot sólo cuenta con los sensores predeterminados para el robot. Para este trabajo es útil conocer su posición y orientación real en el mundo virtual con el objetivo de evaluar el desempeño de los esquemas de control. En consecuencia se definió un *GPS* y un *compass* agregando los siguiente campos a la interfaz:

```
field SFFloat gpsAccuracy 0.0

field SFBool compassXAxis TRUE

field SFBool compassYAxis TRUE

field SFBool compassZAxis TRUE

field SFFloat compassResolution -1
```

Y sus respectivas definiciones en el cuerpo del proto:

```
DEF GPS GPS {
    accuracy IS gpsAccuracy
}
DEF COMPASS Compass {
```

xAxis IS compassXAxis

yAxis IS compassYAxis

zAxis IS compassZAxis

resolution IS compassResolution

}

4.1.3. Controlador

Un controlador es un programa de computadora que determina el comportamiento de un robot especificado en un archivo mundo [34]. El controlador es ejecutado como un proceso secundario del proceso de Webots y es vinculado con la biblioteca dinámica **libController** en el arranque, como se observa en la Figura 4.3. Esta biblioteca se encarga de la comunicación entre el controlador y Webots. La comunicación entre procesos (IPC por sus siglas en inglés) entre esta biblioteca y Webots es a través de un *socket* de dominio local.

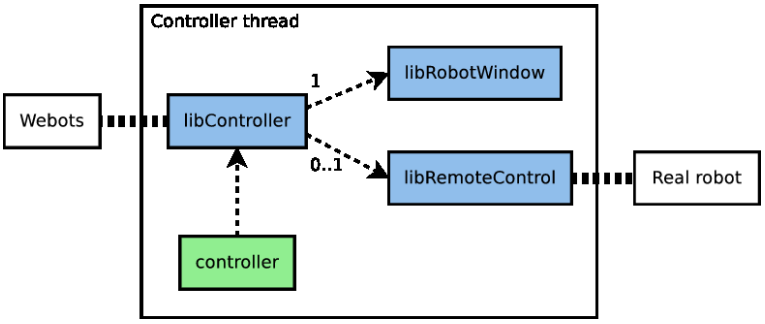


Figura 4.3: Esquema para un complemento del controlador. Figura tomada de `http://www.cyberbotics.com/dvd/common/doc/webots/guide/section6.6.html`

La funcionalidad del controlador puede ser extendida con un complemento diseñado por el usuario. Un complemento es una biblioteca compartida cargada dinámicamente (en tiempo de ejecución) después de un evento específico en función de su tipo y se ejecuta en el hilo principal del proceso. Por lo tanto, si alguna entrada del complemento esta bloqueda (*blocking*), el controlador también estará bloqueado, y si el complemento falla el controlador

también fallará.

Naoqisim

Naoqisim es el controlador del robot NAO para generar la interfaz entre Webots y el simulador de NAOqi (véase el Apéndice A). Los módulos programados con el software de desarrollo **naoqi-sdk** (véase la sección A.1) se pueden probar de forma remota a través de este simulador y observar el comportamiento en el mundo virtual de Webots.

El controlador *naoqisim* es un software libre que se puede redistribuir y/o modificar bajo los términos de la Licencia Pública General de GNU. Está compuesto por varias bibliotecas, cuyas clases definen principalmente sus actuadores, sensores y procesos de comunicación.

Simulador-SDK

El simulador del proceso de NAOqi tiene la estructura mostrada en la Figura 4.4.

Primero se tiene que crear una instancia de un modelo proporcionando un archivo *xml* a la biblioteca *ALRobotModel*. Usando el API del simulador del mundo virtual, se deben transmitir los valores físicos medidos a través de los sensores del robot simulado. Para esto se requiere de la capa de abstracción de hardware (HAL por sus siglas en inglés) y de la biblioteca *libalnaosim*, los cuales transmiten los valores al proceso de NAOqi a través del HAL. De manera análoga, NAOqi comunica los valores de los actuadores a través del HAL y es responsabilidad del API del simulador del ambiente virtual proporcionar los comandos al robot simulado [35].

Entonces, para hacer compatible al controlador con las modificaciones hechas al proto, es necesario un complemento que habilite el *GPS* y el *compass* de manera independiente, ya que estos no necesitarán comunicación con el proceso de NAOqi. Además, sus valores serán requeridos en cada ciclo del lazo de control. En consecuencia, se diseñó una biblioteca servidor para indicarle al robot el momento exacto en que se requiere la lectura de estos sensores. La comunicación se realiza a través del protocolo TCP/IP y los valores sensados se guardan en un archivo de texto.

El Algoritmo 2 muestra los dos métodos más importante de la biblioteca

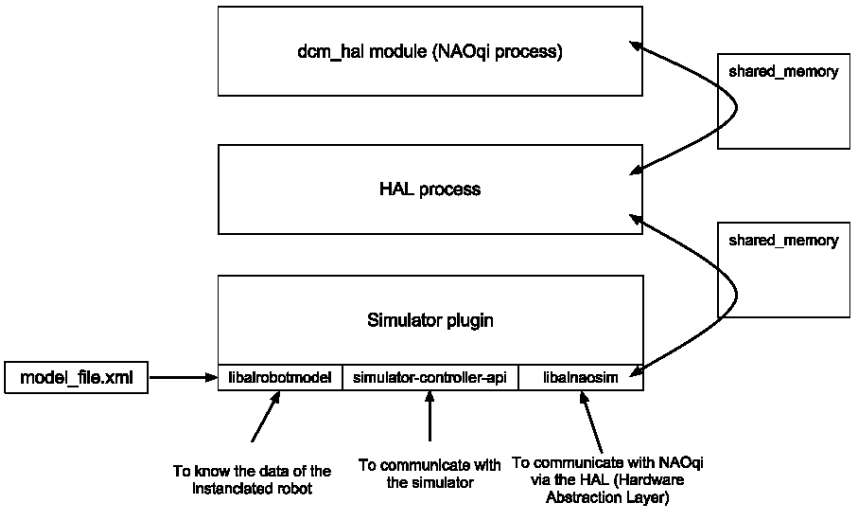


Figura 4.4: Esquema del simulador de NAOqi. Figura tomada de `http://doc.aldebaran.com/1-14/ref/simulator_sdk.html`

servidor. En la primera función se habilitan los sensores a través del API de Webots y, como se indicó anteriormente, el complemento espera el evento de la conexión con el cliente para desbloquear el controlador (línea 4). La segunda función espera recibir un arreglo de caracteres determinado (bandera) para obtener los valores de los sensores y guardarlos. No obstante, esta función no está bloqueda (línea 8 a 10), ya que si aún no ha recibido la bandera sale de la función.

Típicamente un controlador *naoqisim* tiene dos métodos principales: `update()` y `run()`. El primero está diseñado para actualizar los valores de los sensores y actuadores una vez. El segundo está diseñado como un ciclo infinto que contiene a `update()` y está condicionado por la simulación, es decir mientras la simulación está activa se llamará al método `update` en función del paso de simulación definido por el usuario. Luego, los valores actualizados en los sensores son enviados a través del HAL al NAOqi en cada llamada a `update()`, y los comandos de los actuadores son enviados del NAOqi a través del HAL para se actualizados también en cada llamada.

El Algoritmo 3 describe la modificación del método `run()`. Primero, se

Algoritmo 2 Biblioteca servidor para naoqisim

```
1: Función initialize
2:     Habilitar GPS.
3:     Habilitar compass.
4:     Esperar por una conexión en algún puerto (blocking mode).
5:     Abrir el fichero donde se guardarán los datos de los sensores.
6:     return

7: Función getValues
8:     Ver el descriptor del fichero para saber si hay datos (non-blocking
   mode)
9:     Si no hay bytes en el socket entonces
10:        return
11:    Leer el buffer
12:    Si buffer = "S" entonces
13:        Obtener valores del GPS.
14:        Obtener valores del compass.
15:        Escribir valores del GPS y compass en el fichero.
16:    Si no
17:        Si buffer = "exit" entonces
18:            Cerrar la conexión
       return
```

inicia el servidor, donde se observa que el controlador permanece bloqueado. Una vez conectado con el cliente, ingresa al ciclo infinito, donde se llama a la función `update()` y al método del servidor, el cual revisa en cada iteración si el *buffer* tiene la bandera.

Algoritmo 3 Modificación de la clase Nao de naoqisim

Requiere: server.hpp, server.cpp

```
1: Función Nao::run
2:     server.initialize
3:     Loop
4:         sever.getValues
5:         Nao::update
```

4.1.4. Módulos

NAOqi tiene una estructura modular, donde cada módulo tiene una funcionalidad y se pueden comunicar entre ellos.

Una función definida en un módulo puede ser agregada al API del robot con la instrucción `BIND_METHOD`. Esto le permite llamar a la función de manera local o remota desde una computadora u otro robot a través de un *proxy* (véase la sección A.1.2 del apéndice).

Para comenzar un módulo se debe de instanciar un *broker* (véase la sección A.1.1 del apéndice). En este trabajo es necesario un módulo remoto, ya que se ejecutará en el simulador de NAOqi y se usará la biblioteca *highgui* de OpenCV como interfaz para evaluar las métodos de visión. Además, es necesario otro módulo que se encargue sólo de la comunicación con el complemento de Webots.

La estructura del módulo programado está ejemplificado en la Figura 4.5. Al *broker* se ligaron dos módulos: webotsModule y opencvModule. El primero esta encargado del cliente para las peticiones al servidor en el controlador de Webots, y el segundo de la ejecución de los algoritmos de los esquemas de control visual.

Un módulo tiene dos métodos básicos: el constructor e `init`. En el constructor se definen, por medio de la instrucción `BIND_METHOD`, aquellos métodos disponibles fuera del proceso. El método `init` es llamado justo después del constructor, por lo que ejecuta las primeras acciones requeridas para la actividades del robot.

webotsModule

El Algoritmo 4 muestra las tres principales funciones de este módulo, para el cual se diseñó una biblioteca cliente que usa el protocolo TCP/IP. El constructor, líneas 1 y 2, habilita la función `sendFlagSensor()` para ser llamada de manera local por un *proxy* instanciado en el módulo opencvModule. Esta función es la encargada de enviar la bandera al complemento del controlador de Webots por medio del cliente (línea 6), el cual se inicia con el método init.

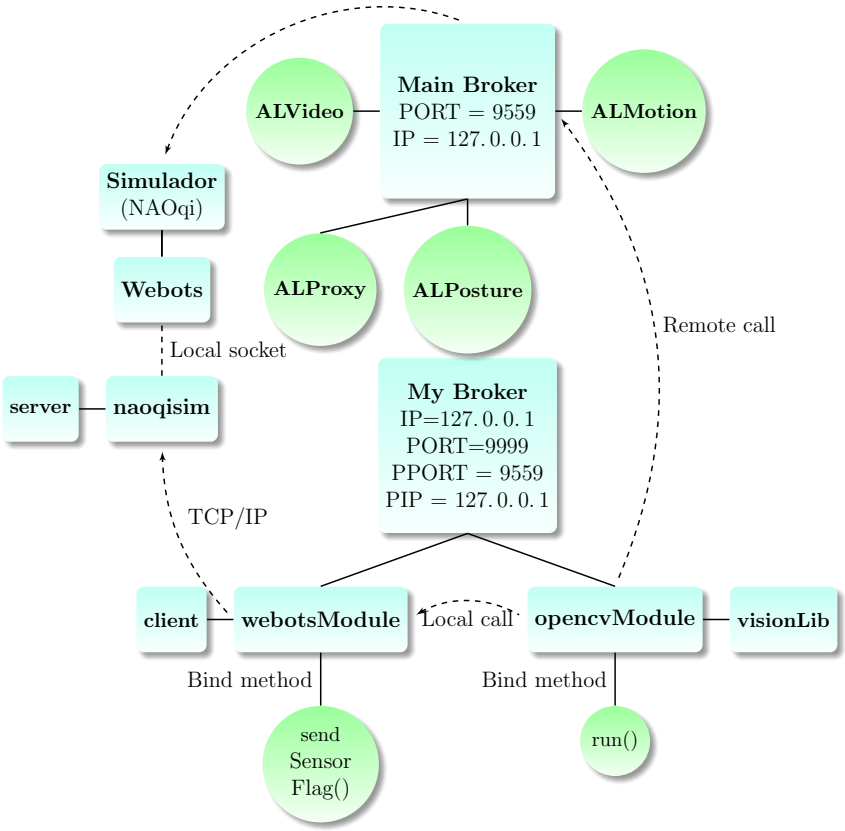


Figura 4.5: Estructura de los módulos

opencvModule

El constructor de este módulo habilita el método `run()`, el cual será llamado fuera del proceso por un *proxy* instanciado en Python que le indique cuando iniciar el algoritmo de control (líneas 1 y 2 en el Algoritmo 5).

El método `init` coloca al robot en una correcta posición de inicio (cuerpo erguido y con los brazos abajo), líneas 4 a 7 del Algoritmo 5, y habilita la cámara para su uso definiendo un nombre, índice, resolución, espacio de color y los cuadros por segundo. Para acceder a los métodos que permitan realizar estas tareas es necesario instanciar un proxy por cada módulo del que se requiere un método. Por lo tanto, también es necesario para webotsModule (línea 10).

A continuación se muestran los algoritmos de los esquemas de control visual. El Algoritmo 6 es el control visual basado en posición descrito en la

Algoritmo 4 Funciones del módulo webotsModule

Requiere: client.hpp, client.cpp

1: **Función** webotsModule
2: BIND_METHOD(sendSensorFlag)

3: **Función** init
4: client.initialize

5: **Función** sendFlagSensor
6: client.readSensorValues

Algoritmo 5 Constructor y función Init de opencvModule

1: **Función** opencvModule
2: BIND_METHOD(run)

3: **Función** init
4: Instanciar *proxy* al módulo ALMotion.
5: Interpolar sus articulaciones a una rigidez específica.
6: Instanciar *proxy* al módulo ALPosture.
7: Ajustar las articulaciones del robot en la configuración de la postura predefinida *StandInit*.
8: Instanciar *proxy* al módulo ALVideoDevice.
9: Subscribir a ALVideoDevice.
10: Instanciar *proxy* al módulo webotsModule.

sección 2.4.2, el Algoritmo 7 modifica este esquema al ejecutarlo por fases. En un movimiento general, primero sólo la velocidad angular es ingresada al robot para girar hasta que un umbral le indica que se encuentra alineado y detiene su movimiento. A continuación, el robot se desplaza longitudinalmente hasta llegar al objetivo. Por último, el Algoritmo 8 es el control visuado basado en imagen descrito en la sección 2.4.4.

En cada algoritmo, el detector de características locales (línea 4 y 9) se implementó con la función de opencv `GoodFeaturesToTrackDetector` (véase la sección 2.3.1) y el descriptor (línea 5 y 10) con la función `SiftDescriptorExtractor` (véase la sección 2.3.2). Para el emparejador (línea 11) se uso el tipo *BruteForce-L2* (véase la sección 2.3.3), es decir de distancia Euclidiana, y la función `knnMatch` que encuentra las mejores k coincidencias en la segunda imagen

para cada descriptor del conjunto de la primera imagen con base en la distancia Euclideana entre los descriptores [36]. En el caso del rastreador se usó la función `calcOpticalFlowPyrLK` (véase la sección 2.3.5) con base en un ejemplo en [37].

Para filtrar los *outliers* del emparejamiento (línea 12) se usaron dos métodos propuestos en [38]. Primero, el método de emparejamiento solicita las dos mejores coincidencias ($k = 2$) en la segunda imagen de cada punto de la primera imagen. Después se encuentran las dos mejores coincidencias en la primera imagen de cada punto de la segunda imagen.

En ambos casos, si la distancia es más pequeña para la primera coincidencia y muy grande para la segunda, se acepta la primera. Por otro lado, si las distancias son parecidas puede resultar en un falso *inlier*, por lo que se evalúa la razón entre las distancias de la primera y la segunda coincidencia. Si esta razón es cercana a la unidad entonces no se toma en cuenta, pero si es mayor a un umbral (denominado *ratio* para referencias futuras) entonces se mantienen las coincidencias en un vector, entendiendo que la primera es la correcta. A esta prueba le conoce como *ratioTest*.

Ahora se tienen los mejores emparejamientos de los puntos de la primera imagen con los puntos de la segunda imagen y viceversa. El siguiente paso es extraer las coincidencias tales que, para que un par emparejado sea aceptado, ambos puntos deben ser la mejor coincidencia del otro. Por último se utiliza el método RANSAC (véase la sección 2.3.4) para asegurar el filtrado.

En la instrucción "Salvar valores de *GPS* y *compass*" de los algoritmos (líneas 8 y 28 en el Algoritmo 6; 8, 28 y 47 en el Algoritmo 7; y 8 y 32 en el Algoritmo 8) se usa un *proxy* para llamar a la función `sendSensorFlag()` del Algoritmo 4 y así se guarden los valores de los sensores sólo cuando son solicitados.

El cálculo del error promedio de píxeles (línea 15 en los Algoritmos 6 y 8, y líneas 15 y 34 en el Algoritmo 7) se realiza con la ecuación 3.2, así como el cálculo de la orientación relativa (línea 20 en el Algoritmo 6, 25 en el Algoritmo 8, y líneas 20 y 39 en el Algoritmo 7) usa el Algoritmo 1 del capítulo anterior.

El método que pertenece a NAOqi usado para comandar la velocidad instántanea del robot es `move()`, cuyos parámetros son las velocidades lineales y angulares expresadas en el sistema de referencia del robot (véase FRAME_ROBOT en la Figura 4.6). La función `move()` convierte las velocidades

dadas por el usuario en un control a bajo nivel de las articulaciones del robot para lograr una caminata dinámica estable que alcance dichas velocidades.

Las velocidades comandadas por el controlador visual están expresadas en el sistema de referencia de la cámara, por lo que es necesario realizar una transformación entre sistemas de referencia. Por lo tanto, para la instrucción "Transformar velocidades al sistema de referencia del robot" (línea 23 en el Algoritmo 6 y líneas 23 y 42 en el Algoritmo 7) se usó la ecuación 4.1. Ésta está basada en la transformación dada en [10], la cual es válida considerando la cabeza fija durante la caminata.

$$\mathbf{v}_{rf} = {}^{m}\mathbf{T}_{rf}\mathbf{v}_{c}, \tag{4.1}$$

donde $\mathbf{v}_{rf}$ es la velocidad del sistema de referencia del robot, $\mathbf{v}_c$ la velocidad de la cámara y

$${}^{m}\mathbf{T}_{rf} = \begin{bmatrix} {}^{m}\mathbf{R}_{rf}^{\top} & -{}^{m}\mathbf{R}_{rf}^{\top}[\mathbf{r}]_{\times} \\ \mathbf{0} & {}^{m}\mathbf{R}_{rf}^{\top} \end{bmatrix}, \tag{4.2}$$

donde $[\mathbf{r}]_{\times} \in \mathbb{R}$ es una matriz antisimétrica y ${}^{m}\mathbf{R}_{rf}^{\top} \in SO(3)$. La matriz antisimétrica sería

$$r = \begin{pmatrix} 0 & 0.05871 & 0.52323 \\ -0.05871 & 0 & 0 \\ -0.52323 & 0 & 0 \end{pmatrix} \tag{4.3}$$

donde el valor 0.52323 y -0.05871 son las distancias en y y z, respectivamente, del origen del sistema de referencia de la cámara al origen de sistema de referencia del robot, medidas en el sistema de referencia de la cámara.

La matriz ${}^{m}\mathbf{R}_{c}$ es la multiplicación de dos matrices de rotación, de tal forma que el sistema de referencia de la cámara rota primero $\frac{\pi}{2}$ sobre el eje z y después $\frac{-\pi}{2}$ sobre el eje y.

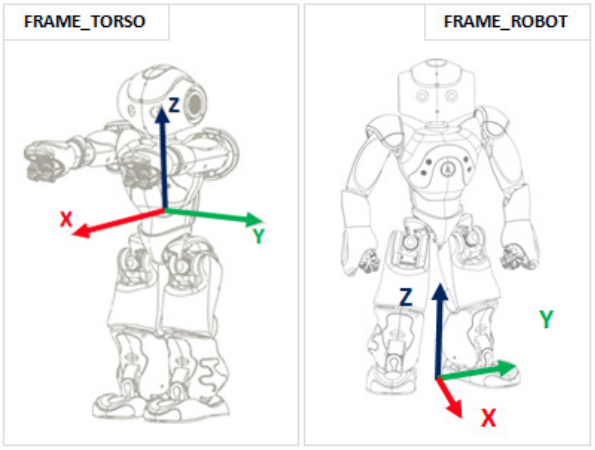


Figura 4.6: Sistemas de referencia del robot NAO. Figura tomada de `http://doc.aldebaran.com/2-1/naoqi/motion/control-cartesian.html`

Una vez transformado el vector de velocidades, sólo se toman tres de sus seis componentes: las velocidades lineales en los ejes x y y, y la velocidad angular en el eje z. Las otras tres velocidades son prácticamente nulas todo el tiempo.

En comparación, en el caso del control visual basado en imagen, la entrada de control son las velocidades de la cámara corregidas en eje y dirección (línea 27 en el Algoritmo 8) , de tal forma que coincidan con el sistema de referencia del robot. Es decir, las velocidades lineales en los ejes z y x de la cámara son las velocidades en los ejes x y $-y$ del robot, respectivamente. La velocidad angular en el eje y de la cámara es la velocidad en el eje z del robot, pero en sentido contrario.

En el caso particular del Algoritmo 7, es necesaria una condición para detener la rotación. Esta condición define la desviación de la matriz de rotación $\mathbf{R}$ de la matriz de identidad $\mathbf{I}$. Se calcula de acuerdo con [39], donde se emplea la métrica de la ecuación 4.4 como una medida de la distancia entre dos desplazamientos de un cuerpo rígido

$$d(\mathbf{R}_1, \mathbf{R}_2) = ||\mathbf{I} - \mathbf{R}_1\mathbf{R}_2^\top||_F, \tag{4.4}$$

donde $\mathbf{R}_1, \mathbf{R}_2 \in SO(3)$ y $||\bullet||_F$ denota la norma de Frobenius. En el caso de una secuencia de matrices $\mathbf{R}_n$ tal que $\mathbf{R}_n \to \mathbf{R}$ cuando $n \to \infty$, entonces $\mathbf{R}_n^\top \to \mathbf{R}^\top$ y $\mathbf{R}\mathbf{R}_n^\top \to \mathbf{R}\mathbf{R}^\top = \mathbf{I}$, por lo que se puede demostrar que [39]

$$d(\mathbf{R}_n, \mathbf{R}) = ||\mathbf{I} - \mathbf{R}_n\mathbf{R}^\top||_F \to ||\mathbf{I} - \mathbf{R}\mathbf{R}^\top||_F = 0. \tag{4.5}$$

Por último, la matriz de parámetros intrínsecos de la cámara del NAO en Webots es:

$$K = \begin{pmatrix} 544 & 0 & 320 \\ 0 & 544 & 240 \\ 0 & 0 & 1 \end{pmatrix}. \tag{4.6}$$

En general, el módulo cuenta con una biblioteca donde se implementaron todos los métodos usados para la correcta ejecución del lazo de control. En los algoritmos se indica, de ser necesario, la referencia de la ecuación usada.

Algoritmo 6 Control visual basado en posición

1: **Función** run
2: Leer imagen objetivo (*target*).
3: Convertir a escala de grises.
4: Instanciar el detector de características.
5: Instanciar el descriptor de características.
6: Inicializar el proceso de movimiento del robot.
7: Tomar foto de la imagen actual de la cámara (*current*).
8: Salvar valores de *GPS* y *compass*.
9: Detectar características locales en *current* y *target*.
10: Describir características locales en *current* y *target*.
11: Emparejar características locales en *current* y *target*.
12: Filtrar los *outliers*
13: Salvar los *inliers* de *current* (*query*) y de *target* (*train*).
14: **Mientras** contador < máximo número de iteraciones **hacer**
15: Medir el error promedio de píxeles (Ec. 3.2).
16: **Si** error promedio de píxeles < umbral **entonces**
17: **break**
18: Estimar la matriz fundamental (Ec. 2.9).
19: Calcular la matriz esencial (Ec. 2.13).
20: Calcular la orientación y posición relativa (Ec. 2.14).
21: Calcular la parametrización ángulo/eje para la rotación.
22: Calcular la entrada de control (Ec. 2.26).
23: Transformar las velocidades al sistema de referencia del robot (Ec. 4.2).
24: Aplicar las velocidades al robot.
25: Tomar foto de la imagen actual de la cámara (*next*)
26: Convertir a escala de grises.
27: Rastrear los puntos buenos de *current* a *next*.
28: Salvar valores de *GPS* y *compass*.
29: Asignar *next* a *current*.
30: Asignar los puntos rastreados a *query*.
31: Sumar 1 al contador.
32: Detener el movimiento del robot.

Algoritmo 7 Control visual basado en posición por fases

1: **Función** run
2: Leer imagen objetivo (*target*).
3: Convertir a escala de grises.
4: Instanciar el detector de características.
5: Instanciar el descriptor de características.
6: Inicializar el proceso de movimiento del robot.
7: Tomar foto de la imagen actual de la cámara (*current*).
8: Salvar valores de *GPS* y *compass*.
9: Detectar características locales en *current* y *target*.
10: Describir características locales en *current* y *target*.
11: Emparejar características locales en *current* y *target*.
12: Filtrar los *outliers*
13: Salvar los *inliers* de *current* (*query*) y de *target* (*train*).
14: **Mientras** contador < máximo número de iteraciones **hacer**
15: Medir el error promedio de píxeles (Ec. 3.2).
16: **Si** matriz de Rotación = matriz Identidad (Ec. 4.4) **entonces**
17: **break**
18: Estimar la matriz fundamental (Ec. 2.9).
19: Calcular la matriz esencial (Ec. 2.13).
20: Calcular la orientación y posición relativa (Ec. 2.14).
21: Calcular la parametrización ángulo/eje para la rotación.
22: Calcular la entrada de control (Ec. 2.26).
23: Transformar las velocidades al sistema de referencia del robot (4.2).
24: Aplicar la **velocidad angular** al robot.
25: Tomar foto de la imagen actual de la cámara (*next*).
26: Convertir a escala de grises.
27: Rastrear los puntos buenos de *current* a *next*.
28: Salvar valores de *GPS* y *compass*.
29: Asignar *next* a *current*.
30: Asignar los puntos rastreados a *query*.
31: Sumar 1 al contador.
32: Detener el movimiento del robot.

33: **Mientras** contador < máximo número de iteraciones **hacer**
34: Medir el error promedio de píxeles (Ec. 3.2).
35: **Si** error promedio de píxeles < umbral **entonces**
36: **break**
37: Estimar la matriz fundamental (Ec. 2.9).
38: Calcular la matriz esencial (Ec. 2.13).
39: Calcular la orientación y posición relativa (Ec. 2.14).
40: Calcular la parametrización ángulo/eje para la rotación.
41: Calcular la entrada de control (Ec. 2.26).
42: Transformar las velocidades al sistema de referencia del robot (Ec. 4.2).
43: Aplicar las **velocidades lineales** al robot.
44: Tomar foto de la imagen actual de la cámara (*next*).
45: Convertir a escala de grises.
46: Rastrear los puntos buenos de *current* a *next*.
47: Salvar valores de *GPS* y *compass*.
48: Asignar *next* a *current*.
49: Asignar los puntos rastreados a *query*.
50: Sumar 1 al contador.
51: Detener el movimiento del robot.

Algoritmo 8 Control visual basado en imagen

1: **Función** run
2: Leer imagen objetivo (*target*).
3: Convertir a escala de grises.
4: Instanciar el detector de características.
5: Instanciar el descriptor de características.
6: Inicializar el proceso de movimiento del robot.
7: Tomar foto de la imagen actual de la cámara (*current*).
8: Salvar valores de *GPS* y *compass*.
9: Detectar características locales en *current* y *target*.
10: Describir características locales en *current* y *target*.
11: Emparejar características locales en *current* y *target*.
12: Filtrar los *outliers*
13: Salvar los *inliers* de *current* (*query*) y de *target* (*train*).
14: **Mientras** contador $<$ máximo número de iteraciones **hacer**
15: Medir el error promedio de píxeles.
16: **Si** error promedio de píxeles $<$ umbral **entonces**
17: **break**
18: **Si** los *inliers* estan en las líneas epipolares (Ec. 2.31) **entonces**
19: Detener el movimiento del robot.
20: Dormir el robot 1 segundo.
21: Inicializar el proceso de movimiento del robot ($\beta = 1$).
22: Estimar la matriz fundamental (Ec. 2.9).
23: Calcular la matriz esencial (Ec. 2.13).
24: **Si** $\beta = 1$ **entonces**
25: Calcular la posición relativa (Ec. 2.14).
26: Calcular la entrada de control (Ec. 2.34).
27: Ajustar las velocidades al centro de masa del robot.
28: Aplicar las velocidades al robot.
29: Tomar foto de la imagen actual de la cámara (*next*).
30: Convertir a escala de grises.
31: Rastrear los puntos buenos de *current* a *next*.
32: Salvar valores de *GPS* y *compass*.
33: Asignar *next* a *current*.
34: Asignar los puntos rastreados a *query*.
35: Sumar 1 al contador.
36: Detener el movimiento del robot.

Capítulo 5

Resultados

En este capítulo se presentan las tablas y gráficas de las pruebas hechas para cada simulación de los dos esquemas de control.

Las tablas de cada apartado contienen los valores de los parámetros usados para esa prueba en particular.

En cada apartado, las gráficas de las simulaciones muestran la posición y orientación del robot con respecto al sistema de referencia de la imagen objetivo, la evolución del error promedio de píxeles y las velocidades lineales y angular del robot. De acuerdo al controlador, se muestran otras gráficas y figuras pertinentes para su análisis.

En el último apartado de cada sección se muestran los resultados de las pruebas para evaluar la incidencia en el error de localización del robot, con respecto a la posición deseada, en función del número de puntos de interés detectados.

5.1. Preparación de las simulaciones

La programación de los controladores se realizó en un solo módulo ya que ambos esquemas comparten varios métodos, como se observa en los algoritmos de la sección 4.1.4. La Figura 5.1 muestra la imagen de referencia memorizada por el robot, la cual fue usada para todas las pruebas.

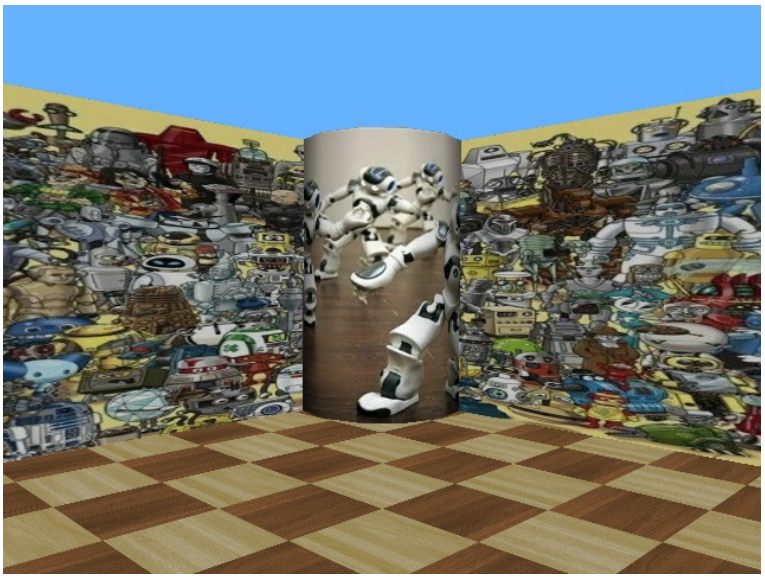


Figura 5.1: Imagen objetivo (640 × 480 píxeles)

El método usado para detectar puntos de interés divide la imagen en cuatro secciones y en cada una realiza la búsqueda de puntos. Después, todos los puntos encontrados se ingresan en un vector, el cual es usado por el descriptor. Esto se realiza con la meta de que exista una distribución uniforme de *inliers* en la imagen, lo cual favorece a un cálculo estable de la geometría epipolar.

Los métodos para filtrar *outliers*, *ratioTest* y RANSAC, requieren de los valores mostrados en la tabla siguiente.

Parámetro	Valor
Ratio	0.99
Confidence	0.99
Distance	3.0

Tabla 5.1: Parámetros para el filtrado de *outliers*

El valor de *ratio* es el umbral con el que se compara la relación entre las distancias de la primera y la segunda coincidencia de los dos mejores emparejamientos entre ambas imágenes en el método `ratioTest` (véase **opencvModule** en la sección 4.1.4). *Confidence* y *Distance* son los parámetros usados por RANSAC, los cuales deben ser proporcionados a la función `findFundamentalMat` de OpenCV.

El sistema de coordenadas del mundo de Webots se muestra en la esquina inferior derecha de la Figura 5.2. El centro del piso se colocó en el origen del sistema de referencia del mundo. La figura también muestra el sistema de referencia del robot NAO: eje x en rojo, eje y en verde y eje z en azul.

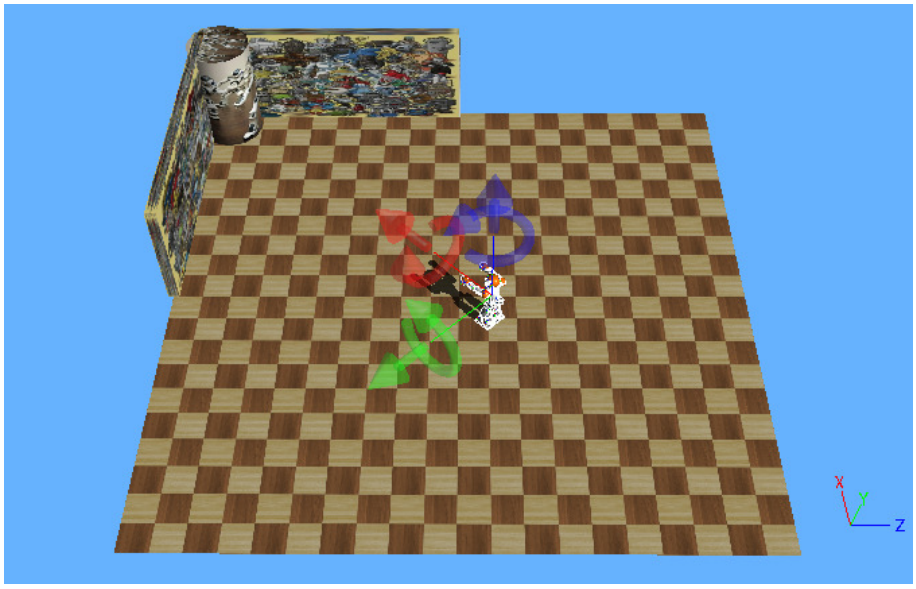


Figura 5.2: Sistemas de referencia en el mundo de Webots

Para cada esquema de control se realizó una simulación de traslación pura y cuatro de movimiento general. En las pruebas se denominó **posición cero** al punto donde se colocó al robot NAO para la prueba de traslación pura, ver (0) en la Figura 5.3 (sistema de referencia en azul). La imagen objetivo se tomó en el punto (T) de la Figura 5.3, considerando que el robot debía desplazarse al menos un metro. El sistema de referencia (W) es el sistema de referencia mundo de Webots. Estas localizaciones se definieron con base en varios experimentos previos de tal forma que la imagen objetivo estuviera lo suficientemente alejada de las paredes y en todo momento el robot observará tres planos, así como que éste se mantuviera a una distancia adecuada de la escena observada para que procesará correctamente las imágenes, es decir para que la detección, descripción, emparejamiento y rastreo de puntos de interés se realizara correctamente.

Las pruebas de movimiento general se pueden resumir de la siguiente forma (véase los sistemas de referencia (1), (2), (3) y (4) de la Figura 5.3):

1. Se colocó al robot en la posición cero y se giró $-\frac{\pi}{12}$ sobre el eje z del

robot.

2. Se colocó al robot en la posición cero y se giró $\frac{\pi}{12}$ sobre el eje z del robot.

3. Se colocó al NAO como en la prueba 1 y, en esta orientación, se desplazó 0.5 metros en la dirección positiva del eje y del robot.

4. Se colocó al NAO como en la prueba 2 y, en esta orientación, se desplazó 0.5 metros en la dirección negativa del eje y del robot.

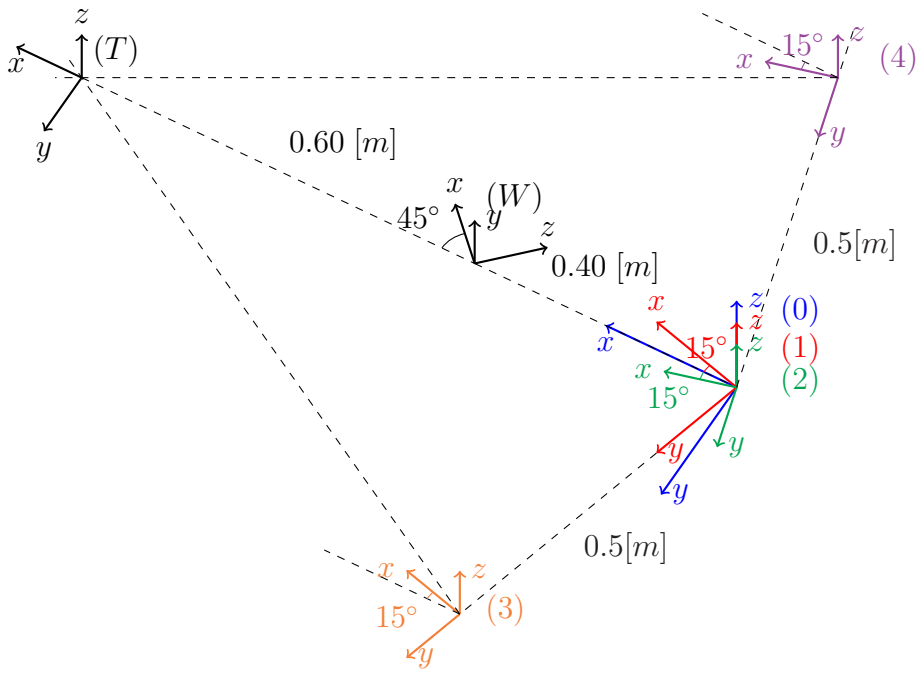


Figura 5.3: Localización del robot NAO en las pruebas

5.2. Control Visual Basado en Posición

Este esquema de control se programó con base en el algoritmo 6. Los parámetros usados para la simulación se muestran en la Tabla 5.2.

Parámetro	Valor
Puntos detectados	1000
Iteraciones	2500
Ganancia v	0.05
Ganancia ω	0.7
Umbral de píxeles	10

Tabla 5.2: Parámetros de la simulación

El parámetro puntos detectados se refiere a los puntos solicitados al detector (250 puntos por sección), no a los *inliers*. Se define un número máximo de iteraciones para que el robot se detenga aunque no haya alcanzado la posición deseada, para que la simulación no falle debido a las limitaciones de memoria de la computadora. La ganancia v y la ganancia ω se refieren a las ganancias del control para las velocidades lineas y angulares, respectivamente. Por último, el umbral de píxeles es el valor máximo del error promedio de píxeles que el control puede presentar para que el robot se detenga.

Los resultados de las simulaciones descritas en la sección anterior se muestra en los siguiente dos apartados.

Las Figuras 5.4, 5.6, 5.8, 5.10 y 5.12 describen la posición y orientación del robot con respecto al sistema de referencia de la imagen objetivo, usando los valores guardados del *GPS* y el *compass*, respectivamente. El *GPS* se colocó en el torso del robot, por lo que las gráficas de posición tiene esa tendencia debido a que, cuando el robot apoya un pie durante la caminata, mueve su centro de masa para que la proyección de éste se encuentre dentro del polígono de soporte. En las gráficas de posición se observa cómo y cuánto se desplazó, así como el error final con respecto a la localización deseada.

Las flechas en las figuras ejemplifican la orientación del robot en esa iteración, por lo que en las pruebas de movimiento general se observa como el controlador corrige la orientación. Las flechas no están dibujadas en cada iteración, pero en todos los casos se dibujan en la primera y en la última iteración. De esta forma se observa el error final de la orientación.

Las Figuras 5.5, 5.7, 5.9, 5.11 y 5.13 muestran las gráficas de error promedio de píxeles y velocidades lineales y angulares. En todos los casos el comportamiento es el esperado, como se observa si se compara con las gráficas de las simulaciones de Matlab en el capítulo 3. Es decir, tanto el error

como las velocidades convergen a cero o muy cercano a cero debido al ruido.

En algunos casos, por ejemplo las Figuras 5.10 y 5.11, se observa la dificultad de orientarse correctamente al acercarse al objetivo, así como picos de velocidad antes de que el robot finalice su ejecución. En la Figura 5.11, la gráfica de error muestra que este comportamiento comienza cuando el error está por debajo de los 20 píxeles. Este tipo de problemas se observó durante la realización de más experimentos.

5.2.1. Traslación pura

Prueba 0

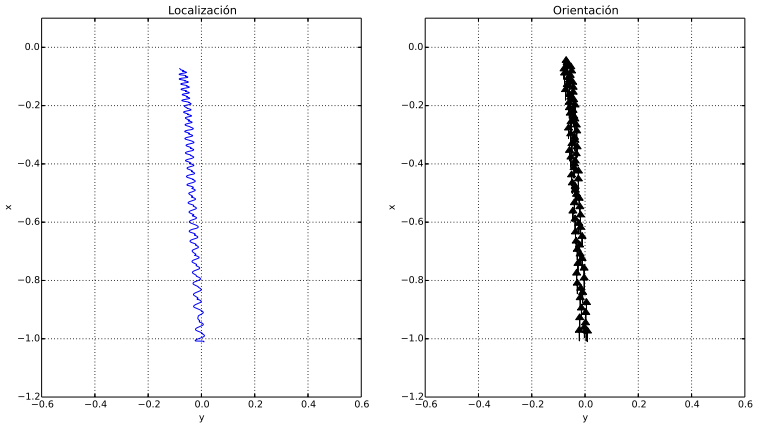


Figura 5.4: Posición y orientación del robot con respecto al sistema de referencia de la imagen objetivo

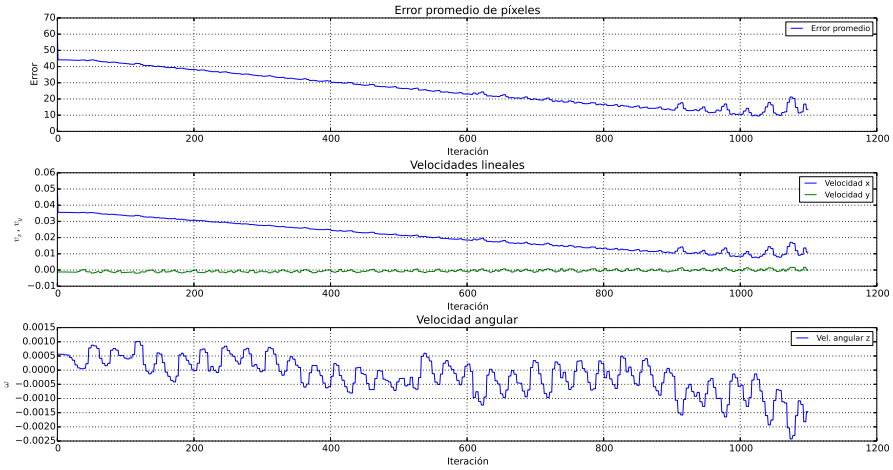


Figura 5.5: Error promedio de píxeles y entradas de control

5.2.2. Movimiento general

Movimiento general desde la posición cero

Prueba 1

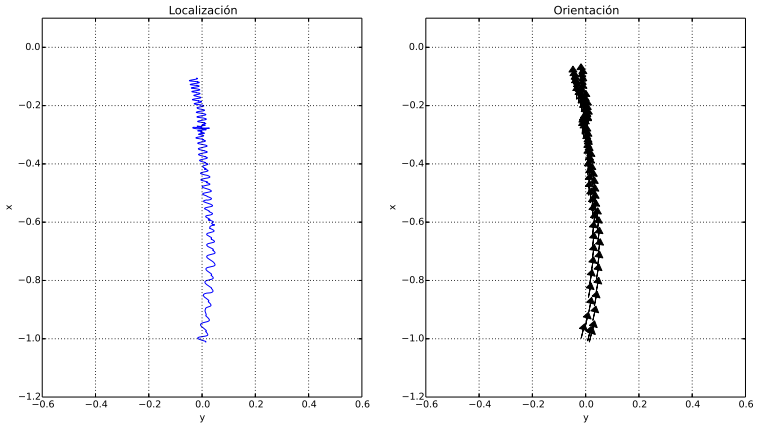


Figura 5.6: Posición y orientación del robot con respecto al sistema de referencia de la imagen objetivo

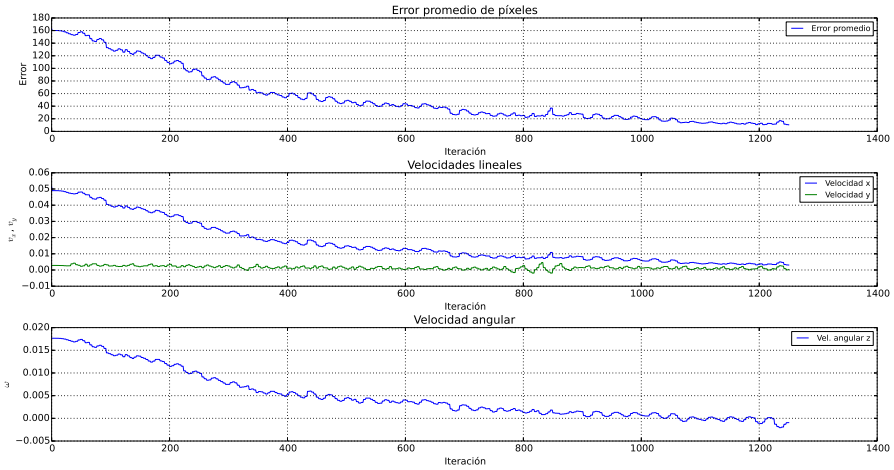


Figura 5.7: Error promedio de píxeles y entradas de control

Prueba 2

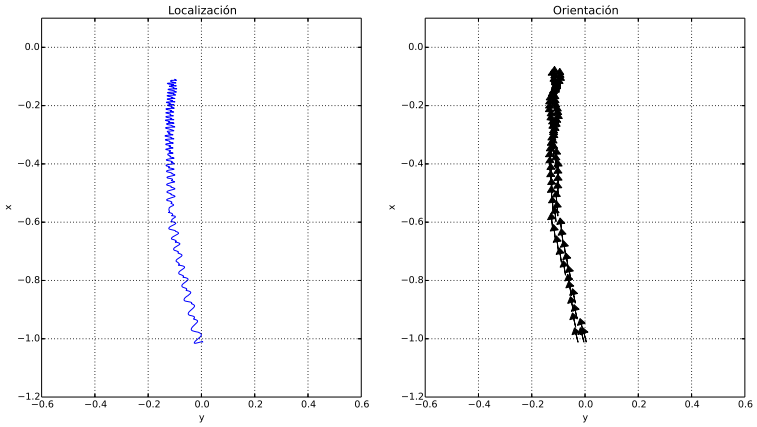


Figura 5.8: Posición y orientación del robot con respecto al sistema de referencia de la imagen objetivo

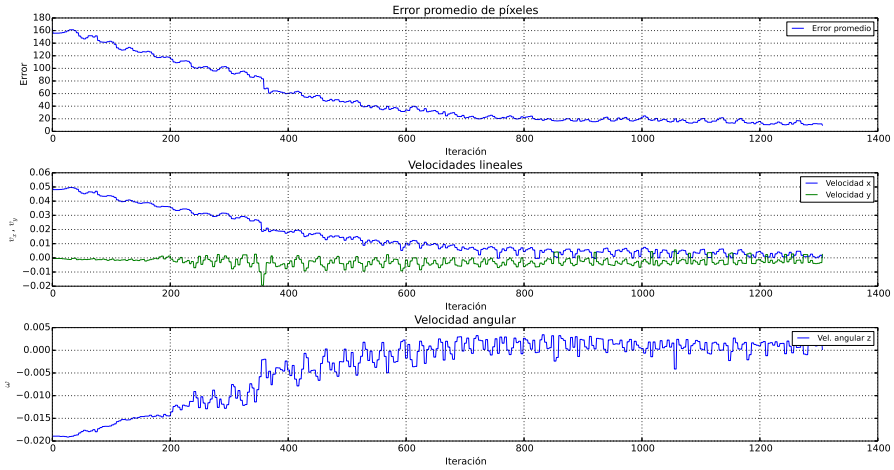


Figura 5.9: Error promedio de píxeles y entradas de control

Movimiento general fuera de la posición cero

Prueba 3

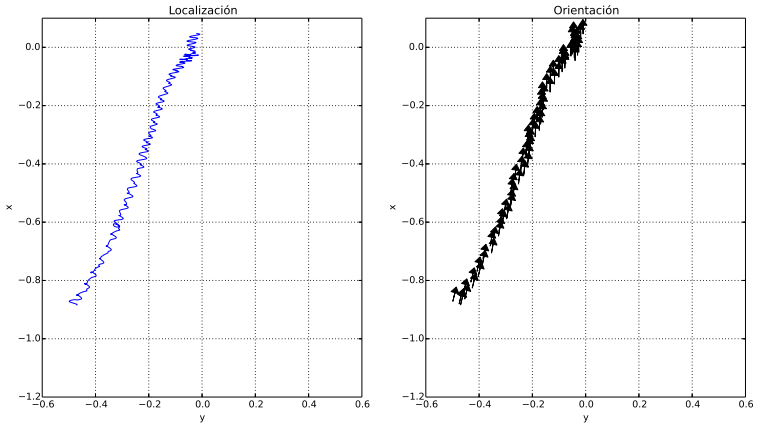


Figura 5.10: Posición y orientación del robot con respecto al sistema de referencia de la imagen objetivo

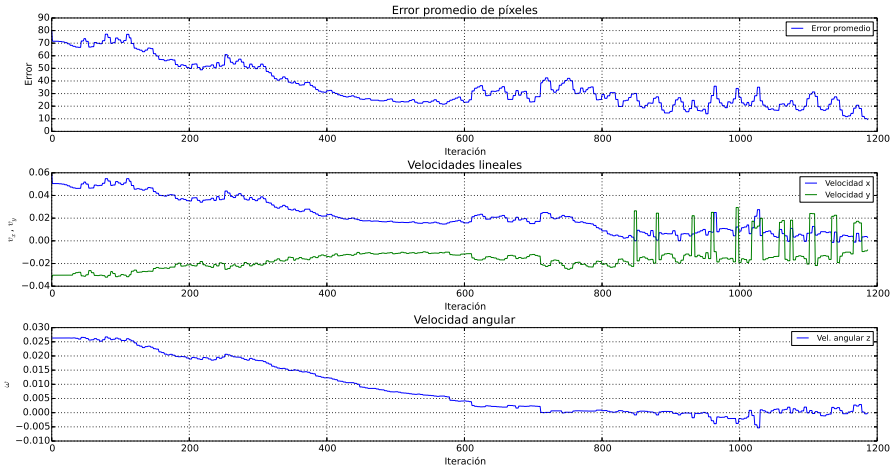


Figura 5.11: Error promedio de píxeles y entradas de control

Prueba 4

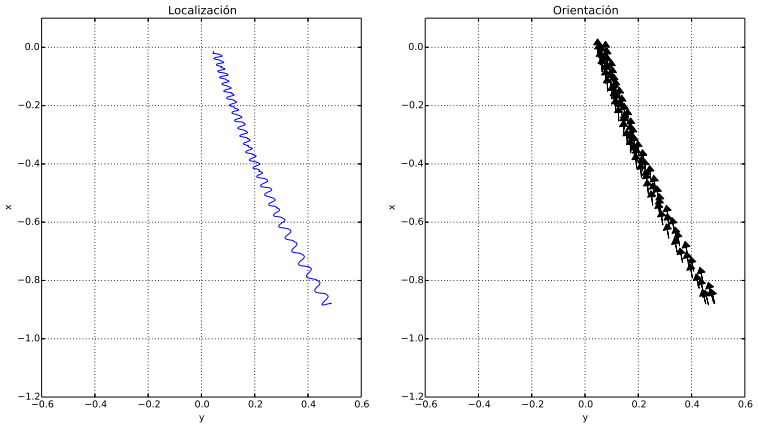


Figura 5.12: Posición y orientación del robot con respecto al sistema de referencia de la imagen objetivo

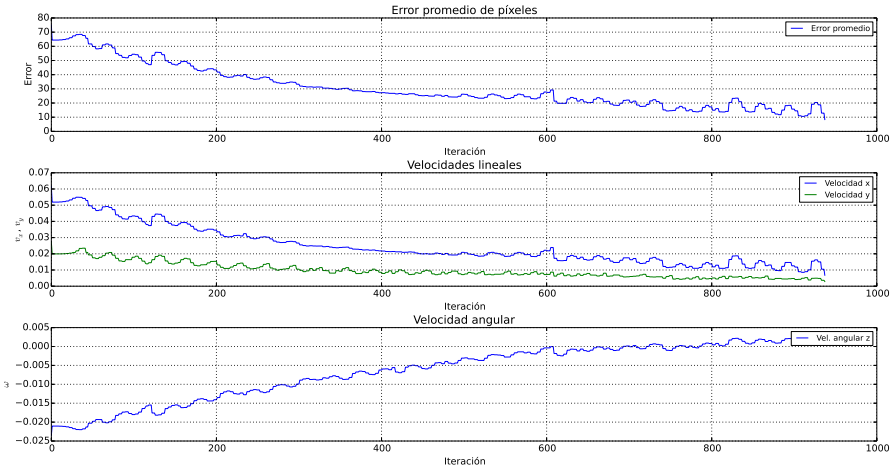


Figura 5.13: Error promedio de píxeles y entradas de control

5.2.3. Control visual basado en posición por fases

A diferencia del control de la sección anterior, donde el control corrige los tres grados de libertad del posicionamiento del robot simultáneamente, en esta sección se presenta un caso particular del control visual basado en posición, en el cual primero se le permite al robot unicamente rotar y posteriormente sólo trasladarse (véase el algoritmo 7). Los parámetros usados son los mismos mostrados en la Tabla 5.2. Para este algoritmo es necesario establecer un umbral para la condición de rotación mostrada en la ecuación 4.4. El valor elegido se muestra en la Tabla 5.3 y se eligió de acuerdo con la ecuación 4.5.

Parámetro	Valor
Umbral de rotación ($\|\|\mathbf{I} - \mathbf{R}_1\mathbf{R}_2^\top\|\|_F$)	0.1

Tabla 5.3: Parámetro extra de la simulación

Se realizaron las mismas pruebas para el movimiento general que el caso anterior, descritas en la sección 5.1. Las gráficas de posición y orientación en las Figuras 5.14, 5.16, 5.18 y 5.21 tienen un comportamiento similar que en el esquema sin fases, excepto por el inicio en el que se presenta un desplazamiento sobre el eje y del robot y se observa una acumulación de flechas, las cuales indican como primero se corrigió la orientación y después se trasladó.

Las gráficas 5.15, 5.17, 5.19 y 5.22 muestran las entradas de control, pero a continuación se aclara cómo se ingresan al robot. Durante la primera fase, sólo se ingresa la velocidad angular hasta que la condición para detener la rotación se cumple (Tabla 5.3). Lo anterior se observa en la tercera y cuarta gráfica de todas las figuras. En las gráficas de las velocidades lineales se presentan valores diferentes de cero que no son ingresadas al robot durante la primera fase.

La segunda fase, es decir la traslación, comienza cuando en la segunda gráfica se observa un incremento de la velocidad lineal en x en lugar de un decremento. En esta fase la velocidad angular es nula o muy cercana a cero durante todo el tiempo que tardan en converger las velocidades lineales y el error.

5.2.4. Movimiento general

Rotación pura y traslación desde la posición cero

Prueba 1

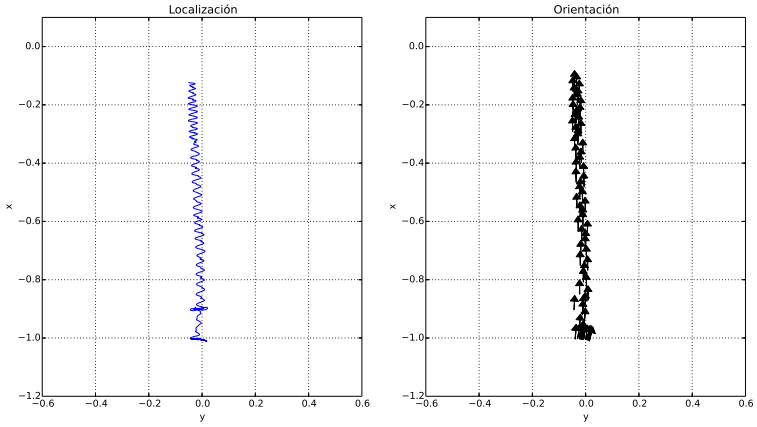


Figura 5.14: Posición y orientación del robot con respecto al sistema de referencia de la imagen objetivo

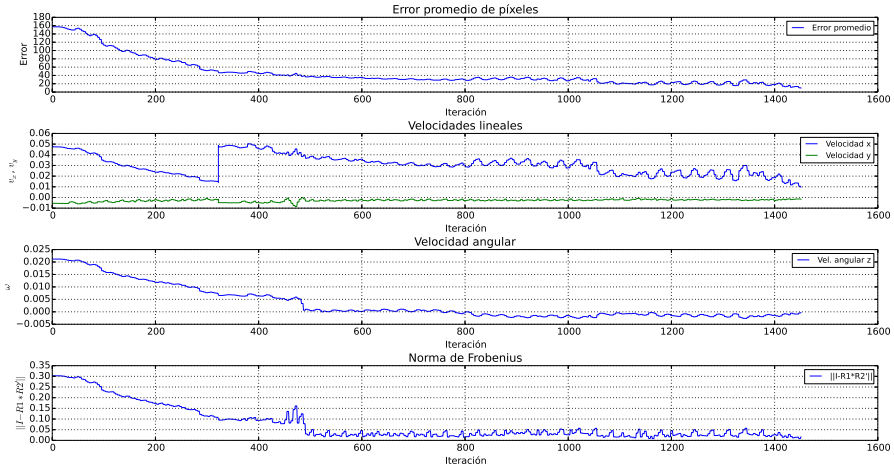


Figura 5.15: Error promedio de píxeles, entradas de control y condición de rotación

Prueba 2

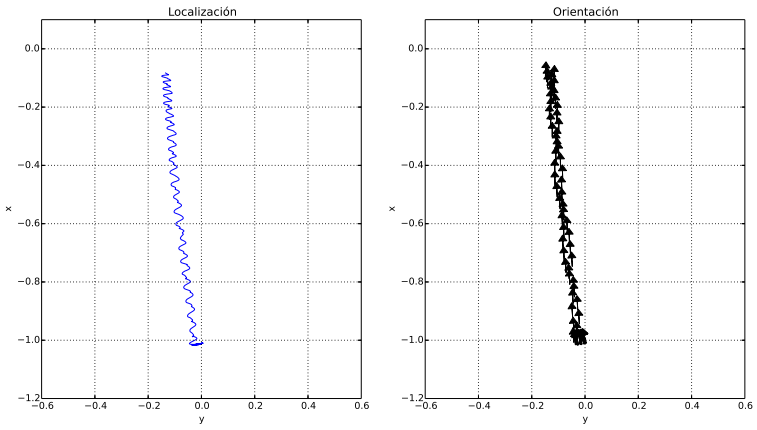


Figura 5.16: Posición y orientación del robot con respecto al sistema de referencia de la imagen objetivo

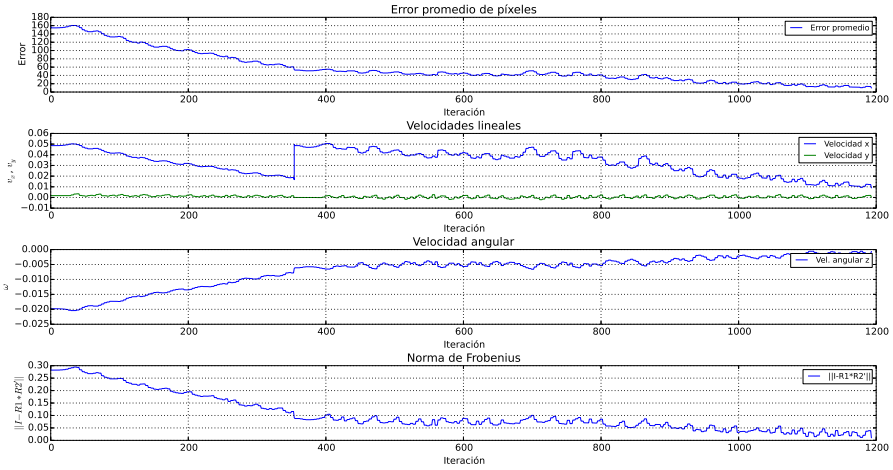


Figura 5.17: Error promedio de píxeles, entradas de control y condición de rotación

Rotación pura y traslación pura fuera de la posición cero

Prueba 3

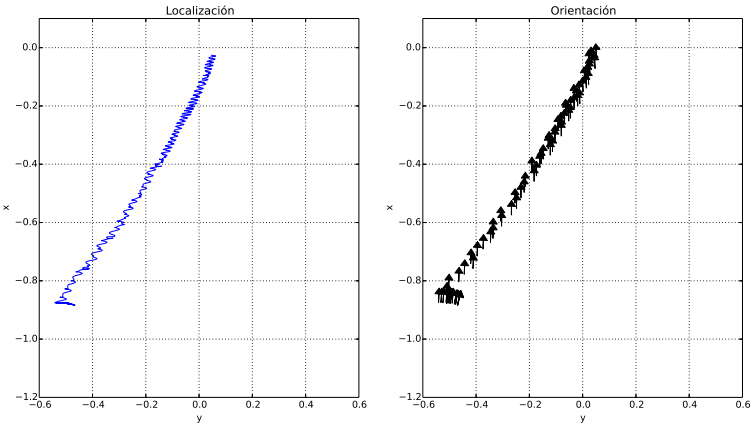


Figura 5.18: Posición y orientación del robot con respecto al sistema de referencia de la imagen objetivo

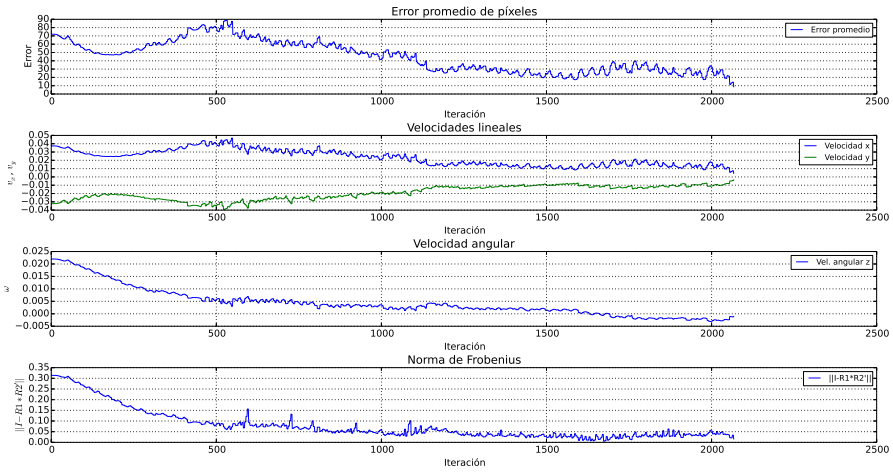


Figura 5.19: Error promedio de píxeles, entradas de control y condición de rotación

En la Figura 5.20 se muestra un ejemplo en imágenes durante la caminata del robot NAO para llegar a la posición objetivo. La imagen (a) indica la posición deseada y la imagen (l) indica la posición inicial.

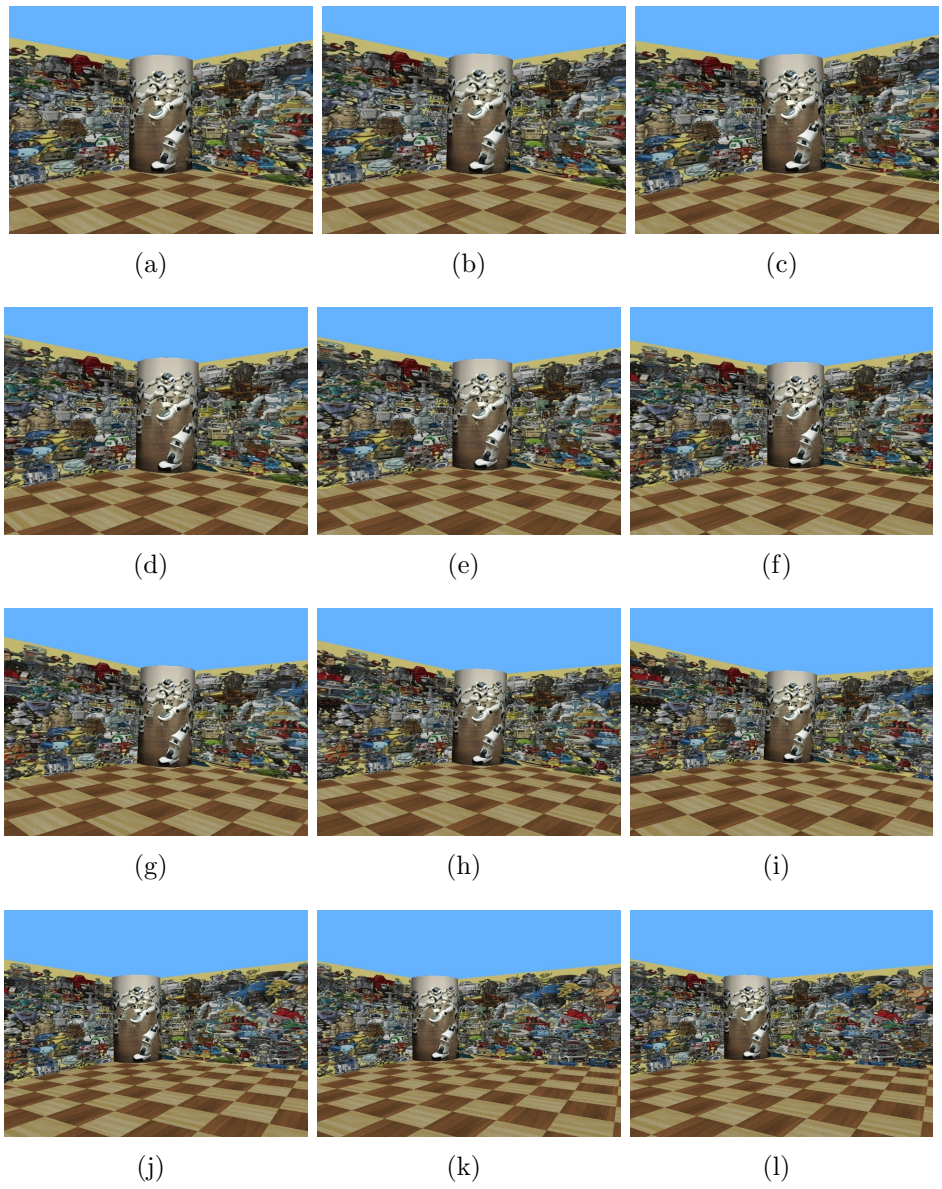


Figura 5.20: Ejemplo de las imágenes durante la caminata del robot NAO

Prueba 4

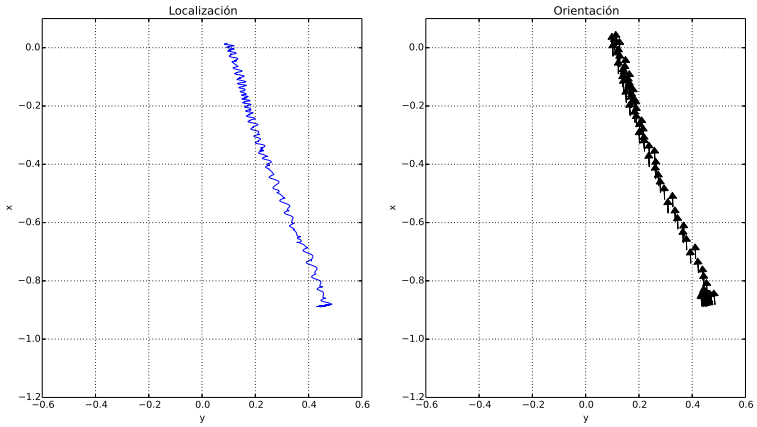


Figura 5.21: Posición y orientación del robot con respecto al sistema de referencia de la imagen objetivo

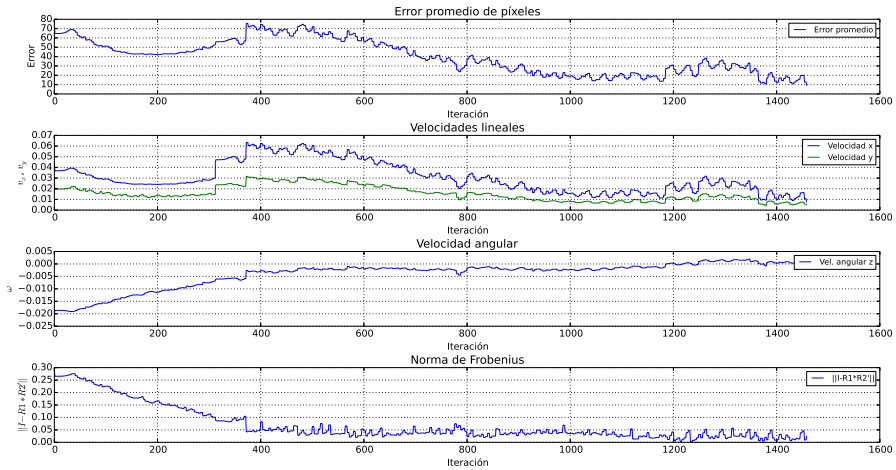


Figura 5.22: Error promedio de píxeles, entradas de control y condición de rotación

5.2.5. Error de localización

Por último, se muestran las tablas de las pruebas de traslación pura (robot alineado y con el objetivo al frente) hechas para conocer la incidencia de la

cantidad de *inliers* en el error promedio de píxeles y en el error final de posición.

Prueba	Puntos por sección	*Inliers*	Error [píxeles]	Error(dist. Euc.) [m]
1	50	79	24.4037	0.1912
2	70	108	20.0524	0.1186
3	90	128	20.5654	0.1490
4	110	160	16.0982	0.1713
5	130	182	27.9186	0.0688
6	150	202	15.8295	0.1803
7	170	213	9.9546	0.0994
8	190	234	15.8397	0.1499
9	210	248	7.9955	0.0928
10	230	271	15.7363	0.1411
11	250	315	9.8607	0.1635

Tabla 5.4: Error promedio de píxeles y error de localización en metros

En la Tabla 5.9 se observa que no existe una tendencia clara de los errores en función del número de puntos, por lo que se eligió utilizar un número grande de puntos detectados para que los falsos *inliers* que aún quedan después del filtro no afecten en la determinación de la matriz de rotación y el vector de traslación.

La Tabla 5.5 presenta los resultados de la ejecución en 10 ocasiones del control para una traslación pura desde la misma condición inicial y el mismo número de puntos. Se observa que el número de *inliers* puede variar entre una prueba y otra, y que los errores tanto en medidas visuales (píxeles) como en medidas geométricas también tienen una variación entre pruebas. Sin embargo, en todos los casos el robot tiende a alcanzar la ubicación deseada.

Prueba	Puntos por sección	*Inliers*	Error [píxeles]	Error(dist. Euc.) [m]
1	250	313	15.0164	0.1566
2	250	316	6.1911	0.1760
3	250	297	15.6496	0.0475
4	250	283	14.8877	0.1849
5	250	299	17.2641	0.0913
6	250	305	8.0481	0.1386
7	250	291	19.5855	0.2384
8	250	294	11.5850	0.1560
9	250	303	14.2714	0.0701
10	250	285	12.0320	0.0358

Tabla 5.5: Errores para un número constante de características locales

En las Tablas 5.6 y 5.7 se presentan los errores en píxeles y los errores geométricos para las diferentes pruebas presentadas en las subsecciones previas. Los errores finales son equiparables entre el control que corrige simultáneamente los tres grados de libertad del robot y el que lo hace por fases.

Prueba	*inliers*	Error [píxeles]	Error(dist. Euc.) [m]
0	294	9.8958	0.1092
1	329	10.3449	0.1078
2	293	9.7933	0.1486
3	369	9.41036	0.0513
4	331	8.3199	0.0486

Tabla 5.6: Error final de las pruebas del control visual basado en posición

Prueba	*inliers*	Error [píxeles]	Error(dist. Euc.) [m]
1	305	9.54282	0.1335
2	305	9.98525	0.1583
3	348	8.6736	0.0532
4	338	9.3762	0.0951

Tabla 5.7: Error final de las pruebas del control visual basado en posición por fases

5.3. Control visual basado en imagen

Este controlador se programó con base en el algoritmo 8. De forma análoga al esquema anterior, se definen los parámetros de la simulación en la Tabla 5.8. En este esquema el valor de umbral para detener la rotación se define con base en la ecuación 2.31.

Parámetro	Valor
Puntos detectados	1000
Iteraciones	2500
Ganancia v	0.05
Ganancia ω	5
Umbral de rotación ($\|\mathbf{e}_1\|$)	0.1
Umbral de píxeles	10

Tabla 5.8: Parámetros de la simulación

Las gráficas de las pruebas descritas en la sección 5.1 de este capítulo se muestran en los siguientes apartados.

Para la prueba de traslación pura, este esquema tiene un comportamiento similar al del control basado en posición, como se observa en la Figura 5.23.

En el caso de las pruebas de movimiento general el comportamiento se parece al control visual basado en posición por fases. Por ejemplo, en las Figuras 5.25, 5.27, 5.29 y 5.31 se observa el desplazamiento sobre el eje y del robot durante la rotación y la acumulación de flechas que indica la corrección de la orientación.

En las gráficas de la Figura 5.24, el valor de la velocidad angular es prácticamente nulo, debido a que sólo se está trasladando. Las velocidades lineales convergen a cero y el error promedio de píxeles está por debajo del umbral impuesto.

En las Figuras 5.24, 5.26, 5.28, 5.30 y 5.32 se muestra en la cuarta gráfica la norma de $\mathbf{e}$ dada por la ecuación 2.32. De esta forma se concluye que el controlador se está comportando como se esperaba con relación a la simulaciones de Matlab (véase el capítulo 3). En las figuras de las pruebas de movimiento general se observa como esta norma primero decrece hasta que la velocidad angular es nula o muy cercana a cero. En el cambio a la fase de

traslación el valor de la norma se incrementa y comienza a decrecer de nuevo hasta que las velocidades lineales convergen. Durante la traslación, la velocidad angular no cambia su valor de cero. Se puede comparar con el esquema basado en posición por fases donde esta velocidad puede seguir decreciendo pero ya no es proporcionada al robot. Por último, se observa que el error promedio de píxeles llega a un valor por debajo del umbral establecido.

5.3.1. Traslación pura

Prueba 0

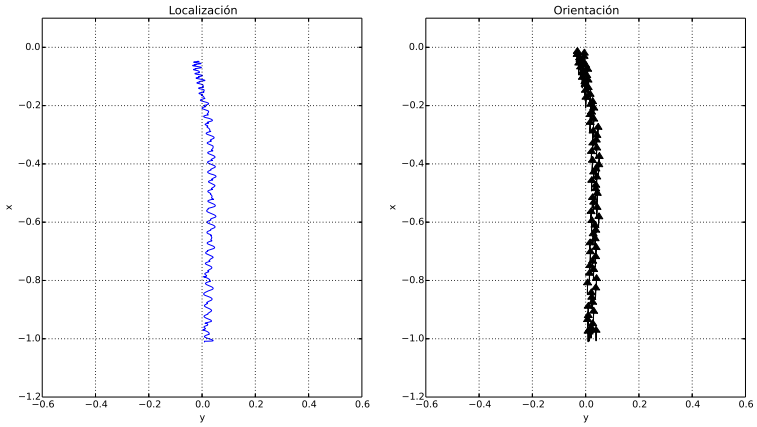


Figura 5.23: Posición y orientación del robot con respecto al sistema de referencia de la imagen objetivo

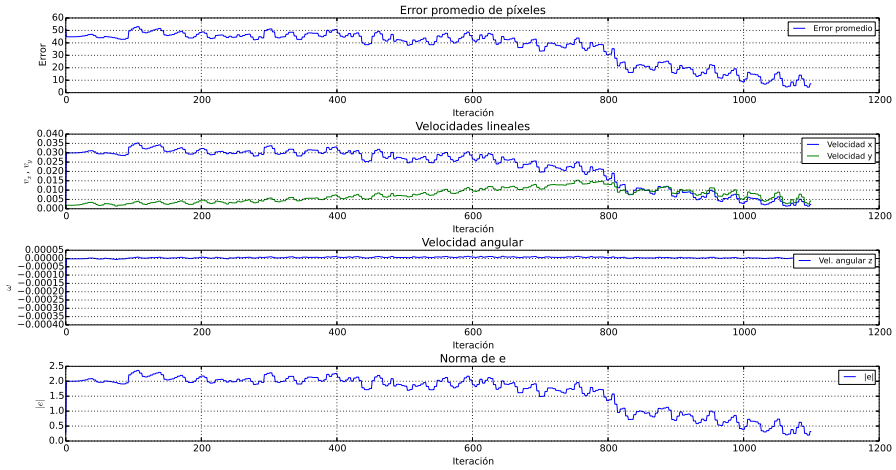


Figura 5.24: Error promedio de píxeles, entradas de control y condición de rotación

5.3.2. Movimiento general

Movimiento general desde la posición cero

Prueba 1

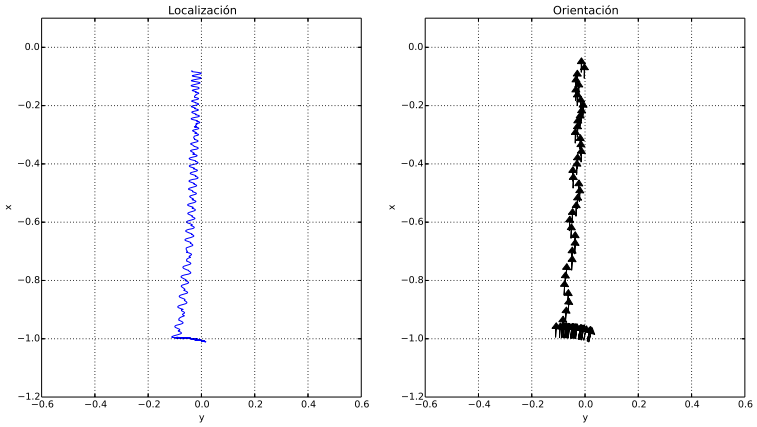


Figura 5.25: Posición y orientación del robot con respecto al sistema de referencia de la imagen objetivo

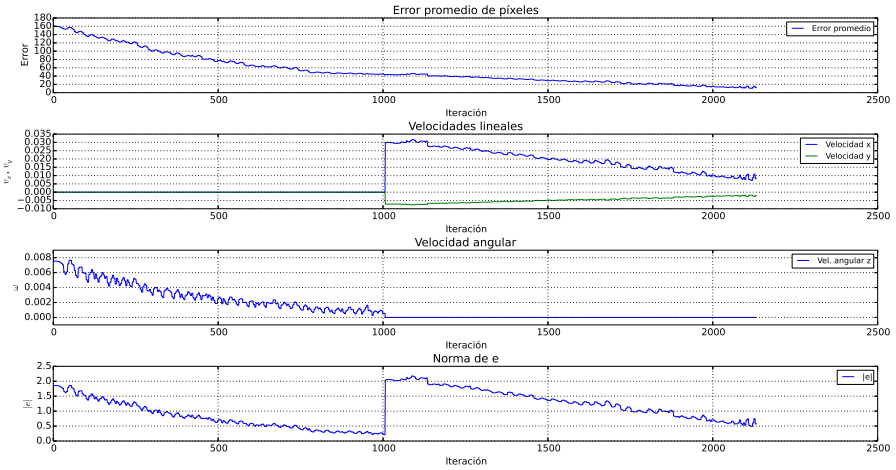


Figura 5.26: Error promedio de píxeles, entradas de control y condición de rotación

Prueba 2

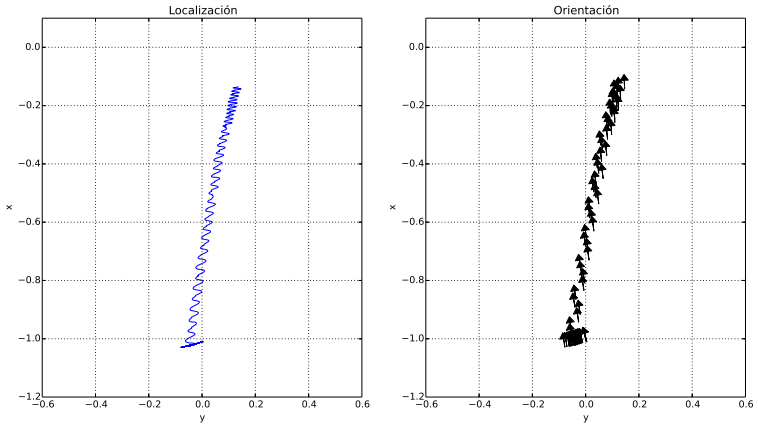


Figura 5.27: Posición y orientación del robot con respecto al sistema de referencia de la imagen objetivo

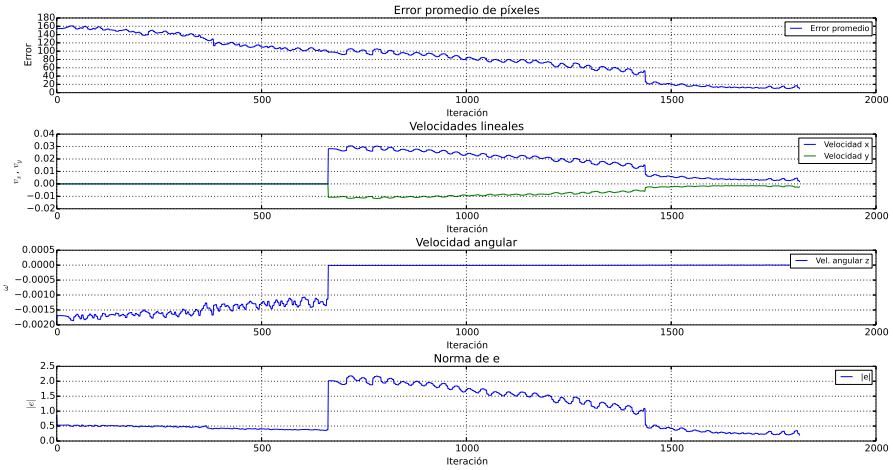


Figura 5.28: Error promedio de píxeles, entradas de control y condición de rotación

Movimiento general fuera de la posición cero

Prueba 3

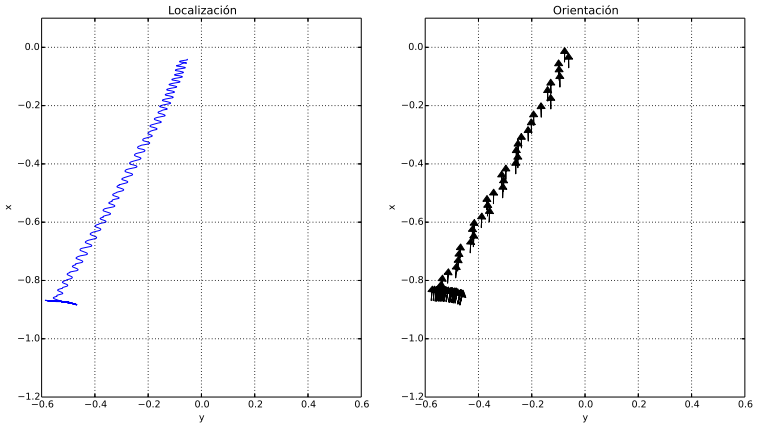


Figura 5.29: Posición y orientación del robot con respecto al sistema de referencia de la imagen objetivo

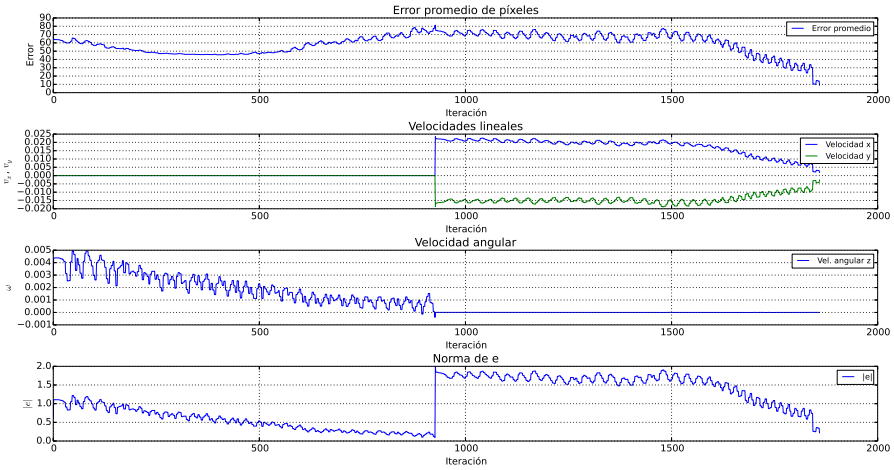


Figura 5.30: Error promedio de píxeles, entradas de control y condición de rotación

Prueba 4

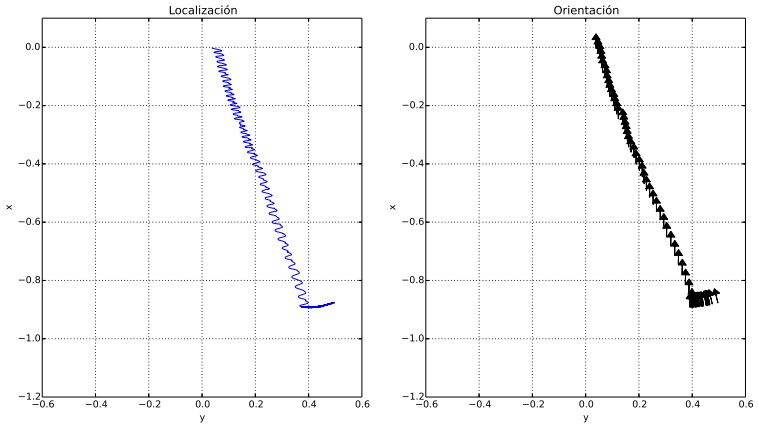


Figura 5.31: Posición y orientación del robot con respecto al sistema de referencia de la imagen objetivo

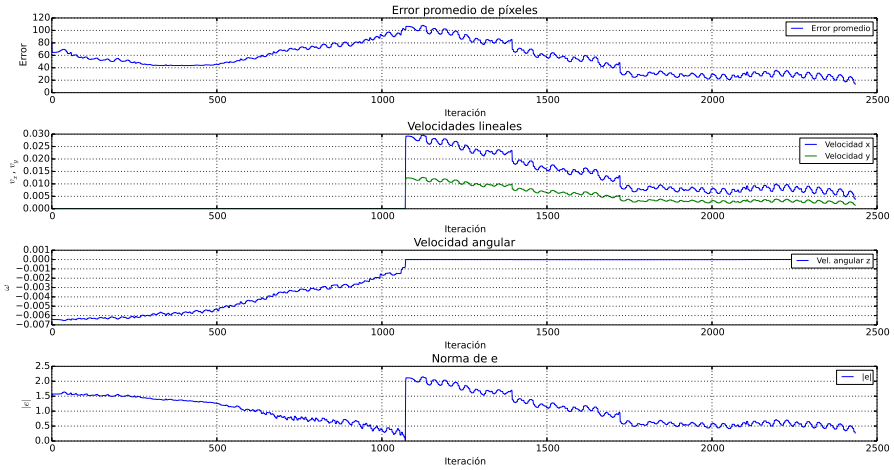


Figura 5.32: Error promedio de píxeles, entradas de control y condición de rotación

5.3.3. Error de localización

De manera análoga a las pruebas del esquema basado en posición, se evaluó la incidencia del número de *inliers* en el error de píxeles y de posición para una ejecución del control en traslación pura. No obstante tampoco existe una tendencia clara, como se puede apreciar en la Tabla 5.9. Por lo que, para poder compararlo con el caso anterior, se eligió utilizar el mismo número de puntos detectados (250 por sección), aunado a las razones ya expuestas.

Prueba	Puntos por sección	*Inliers*	Error [píxeles]	Error(dist. Euc.) [m]
1	50	83	22.5696	0.2959
2	70	96	7.8544	0.0593
3	90	123	21.718	0.1316
4	110	171	18.3022	0.0658
5	130	175	14.5471	0.0190
6	150	197	25.5694	0.0222
7	170	226	9.84847	0.0490
8	190	241	9.17521	0.0293
9	210	256	13.8479	0.0870
10	230	277	11.0129	0.0503
11	250	305	7.1643	0.0506

Tabla 5.9: Error promedio de píxeles y error de localización en metros

En la Tabla 5.10 se presentan los resultados de 10 pruebas del control para una traslación pura desde la misma condición inicial y con el mismo número de puntos por sección. Así como en las pruebas del esquema basado en posición, el número de *inliers*, el error en medidas visuales (píxeles) y el error en medidas geométricas varia entre pruebas. No obstante, en todos los casos el robot tiende a alcazar la ubicación deseada.

Prueba	Puntos por sección	*Inliers*	Error [píxeles]	Error(dist. Euc.) [m]
1	250	300	16.3584	0.1470
2	250	294	23.1000	0.1502
3	250	303	17.1602	0.1733
4	250	307	18.4291	0.1588
5	250	294	18.2790	0.2236
6	250	284	23.9569	0.1278
7	250	302	10.5630	0.1674
8	250	286	23.1098	0.1537
9	250	295	11.8883	0.0837
10	250	296	18.7998	0.2053

Tabla 5.10: Errores para un número constante de características locales

Para terminar, en la Tabla 5.11 se presentan los errores en píxeles y

geométricos para las pruebas presentadas en las subsecciones previas. Estos errores finales son equiparables con los obtenidos en el control basado en posición por fases. El robot tiene un comportamiento similar con ambos esquemas. No obstante, no se puede señalar que uno sea mejor que el otro con base sólo en estos errores.

Prueba	*inliers*	Error [píxeles]	Error(dist. Euc.) [m]
0	305	7.1643	0.0506
1	296	12.1763	0.0880
2	297	9.6326	0.1916
3	353	8.9468	0.0753
4	331	13.5464	0.0535

Tabla 5.11: Error final de las pruebas del control visual basado en imagen

Capítulo 6

Conclusiones

Con la realización de este trabajo se comprobó que, tanto el control visual basado en posición como el control visual basado en imagen, funcionan para situar al robot humanoide NAO en una vecindad de la localización deseada. Esto se debe a que las entradas de control convergen exponencialmente y se necesitaría un tiempo infito para alcanzar la posición de forma exacta. Aunado a esto, el robot tiene una velocidad mínima con la cual se genera movimiento, por lo que al alcanzar velocidades inferiores a este valor el robot no avanzara más, o avanzará muy lento y necesitaría un número muy grande de iteraciones para mejorar la precisión de su posicionamiento.

En algunos casos, el robot presentó problemas para converger por debajo del umbral establecido, terminando por alejarse de la posición deseada. Estas dificultades se pueden atribuir a una mala estimación de la matriz fundamental como consecuencia de una mala distribución del número de *inliers* o a la existencia de falsos *inliers* después del filtrado. También es posible que se relacione con la cercanía de la imagen de referencia a la escena 3D. Por lo tanto, hay un cálculo erróneo de las soluciones de la descomposición de la matriz esencial ($\mathbf{R}$ y $\mathbf{t}$). Esto aumenta la posibilidad de picos indeseables de velocidad (angular y/o lineal) cambiando la orientación y/o posición del robot de manera abrupta, lo que conlleva a un aumento en el error y en consecuencia a una posible falla del control al perder de vista los puntos de la imagen objetivo. Cabe resaltar que la geometría epipolar presenta el problema denominado *short baseline*, que consiste en un mal condicionamiento de la matriz fundamental para imágenes cercanas entre si, como es el caso de cuando el robot ya está cerca de la posición objetivo.

Hay que resaltar que el aumento en la velocidad angular implica un aumento en la velocidad lineal lateral, ya que no están completamente desacopladas. Entonces, aún sin aumentos súbitos en la velocidad, el error puede aumentar en el caso de un movimiento general. El aumento en el error promedio de píxeles, así como el desplazamiento lateral con respecto al sistema de referencia de la imagen objetivo, se observan claramente en las pruebas de movimiento general fuera de la posición cero en ambos esquemas de control. Además, el robot tiene una deriva que dificulta la corrección completa del error lateral.

También hay que destacar que el vector $\mathbf{t}$ indica sólo el sentido de la velocidad frontal, con base en el cual se define el sistema de referencia para el control. Por lo tanto, al rebasar la posición objetivo, la velocidad no puede cambiar de sentido y el robot no es capaz de retroceder. Si el cálculo del error promedio le indica que está por debajo del umbral de píxeles, el robot se detendrá. En el caso contrario, aumentará la velocidad frontal ya que el error comenzará a crecer. Este es un aspecto mejorable de la implementación de los controladores evaluados.

En el caso de la traslación pura o las fases de translación de los esquemas que presentan etapas, el robot tiene una deriva al caminar en línea recta con una baja velocidad. Por otro lado, es posible que el número de iteraciones limite del lazo de control termine y el robot se encuentre lejos de la posición objetivo. Una forma de tratar con estos problemas es aumentar las ganancias, no obstante existe un fuerte compromiso en la elección de estos valores ya que, como se ha descrito anteriormente, un aumento en las velocidades podría tener un aumento en el error.

De acuerdo a los resultados, los dos esquemas de control visual implementados cumplen satisfactoriamente el objetivo de posicionar el robot. Sin embargo, en ambos su desempeño es altamente dependiente del buen emparejamiento de los puntos. La implementación del control basado en imagen resultó más sencilla, así como la sintonización de ganancias es más sencilla para el control basado en posición. Cabe resaltar que las evaluaciones se hicieron considerando una cámara calibrada, con parámetros intrínsecos conocidos, sin embargo se sabe que el control basado en posición no es robusto a errores de calibración, aunque sería un trabajo a futuro realizar una evaluación en este sentido. Por otro lado, las modificaciones hechas al controlador *naoqisim* y el módulo de comunicación con Webots podrán integrarse en el diseño de otros módulos que se deseen validar en un mundo virtual.

Para finalizar, este trabajo se podría expandir posteriormente progra-

mando un control visual híbrido, que combine el control basado en posición con el control basado en imagen, para su comparación y proponiendo estrategias para realizar tareas de posicionamiento en un tiempo deseado. Esto último implicaría el uso de técnicas de control con convergencia en tiempo finito. También se podrían probar los esquemas de control visual aquí tratados sobre un robot NAO en un ambiente real y, posteriormente, en tareas de navegación donde la ruta esté compuesta por varias imágenes de referencia.

Bibliografía

[1] C. Dune, A. Herdt, E. Marchand, O. Stasse, P.-B. Wieber, and E. Yoshida, "Vision based control for humanoid robots," *IROS Workshop on Visual Control of Mobile Robots*, pp. 19–26, 2011.

[2] P. Corke, *Robotics Research: The Seventh International Symposium*, ch. Dynamic Issues in Robot Visual-Servo Systems, pp. 488–498. Springer-Verlag Berlin Heidelberg, 1996.

[3] S. Kajita, H. Hirukawa, K. Harada, and K. Yokoi, *Introduction to Humanoid Robotics*. Springer-Verlag Berlin Heidelberg, 2014.

[4] C. Kemp, P. Fitzpatrick, H. Hirukawa, K. Yokoi, K. Harada, and Y. Matsumoto, *Springer Handbook of Robotics*, ch. Humanoids, pp. 1307–1333. Springer-Verlag Berlin Heidelberg, 2008.

[5] S. Piperakis, "Predictive control for stable dynamic locomotion of real humanoid robots," Master's thesis, Technical University of Crete, Greece. School of Electronic and Computer Engineering, 2014.

[6] A. L. López Moreno, "Generador de marcha reactiva humanoide," Master's thesis, Centro de Investigación y Estudios Avanzados del Instituto Politécnico Nacional. Unidad Saltillo, 2013.

[7] H. M. Becerra and C. Sagües, *Visual Control of Wheeled Mobile Robots*. Springer-Verlag Berlin Heidelberg, 2014.

[8] C. Dune, A. Herdt, O. Stasse, P.-B. Wieber, K. Yokoi, and E. Yoshida, "Cancelling the sway motion of dynamic walking in visual servoing," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3175 – 3180, 2010.

[9] A. Faragasso, G. Oriolo, A. Paolillo, and M. Vendittelli, "Vision-based corridor navigation for humanoid robots," in *IEEE International Conference on Robotics and Automation*, pp. 3190–3195, 2013.

[10] J. Delfín, G. Arechavaleta, and H. M. Becerra, "Locomoción humanoides basada en estrategias de control servo visual," in *Memorias del XVI Congreso Latinoamericano de Control Automático*, pp. 438–444, 2014.

[11] J. Delfín, H. M. Becerra, and G. Arechavaleta, "Visual path following using a sequence of target images and smooth robot velocities for humanoid navigation," in *International Conference on Humanoid Robots*, pp. 354–359, 2014.

[12] A. Abou Moughlbay, E. Cervera, and P. Martinet, "Error regulation strategies for model based visual servoing tasks. application to autonomous object grasping with nao robot," in *International Conference on Control, Automation, Robotics & Vision*, pp. 1311 – 1316, 2012.

[13] A. Abou Moughlbay, E. Cervera, and P. Martinet, *Intelligent Autonomous Systems 12*, ch. Real-time Model Based Visual Servoing Tasks on a Humanoid Robot, pp. 321–333. Springer-Verlag Berlin Heidelberg, 2013.

[14] M. Garcia, O. Stasse, and J.-B. Hayet, "Vision-driven walking pattern generation for humanoid reactive walking," in *IEEE International Conference on Robotics and Automataion*, pp. 216–221, 2014.

[15] F. Chaumette and S. Hutchinson, "Visual servo control part i: Basic approaches," *IEEE Robotics & Automation Magazine*, pp. 82–90, 2006.

[16] P. Rives, "Visual servoing based on epipolar geometry," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 1, pp. 602–607, 2000.

[17] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision.* Cambridge University Press, 2004.

[18] Y. Ma, J. Kosecká, S. Soatto, and S. Sastry, *An invitation to 3-D Vision.* Springer-Verlag New York, 2011.

[19] T. Tuytelaars and K. , "Local invariant feature detectors: A survey," *Foundation And Trends in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177–280, 2008.

[20] R. Szeliski, *Computer Vision: Algorithms and Applications.* Springer-Verlag London, 2011.

[21] J. Shi and C. Tomasi, "Good feature to track," in *Computer Vision and Pattern Recognition*, pp. 593–600, 1994.

[22] C. Harris and M. Stephens, "A combined corner and edge detector," in *Fourth Alvey Vision Conference*, pp. 147–151, 1988.

[23] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, 2004.

[24] M. Zuliani, "Ransac for dummies." 2014.

[25] B. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *International Joint Conference on Artificial Intelligence*, pp. 674–679, 1981.

[26] C. Tomasi and T. Kanade, "Detection and tracking of point features," tech. rep., Carnegie Mellon University, 1991.

[27] J.-y. Bouguet, "Pyramidal implementation of the lucas kanade feature tracker description of the algorithm," *Intel Corporation, Microprocessor Research Labs*, 2000.

[28] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control.* Jonh Wiley and Sons, 2005.

[29] S. Kajita and T. K., "Experimental study of biped dynamic walking in the linear inverted pendulum mode.," in *IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2885 – 2891, 1995.

[30] P.-B. Wieber, "Trajectory-free linear model predictive control for stable walking in the presence of strong perturbation," in *IEEE International Conference on Humanoid Robots*, pp. 137–142, 2006.

[31] K. Erbatur and U. Seven, "An inverted pendulum based approach to biped trajectory generation with swing leg dynamics," in *2007 IEEE-RAS International Conference on Humanoid Robots*, pp. 216–221, 2007.

[32] P. Corke, *Robotics, Vision and Control. Fundamental Algorithms in Matlab.* Springer-Verlag Berlin Heidelberg, 2013.

[33] G. Mariottini and D. Prattichizzo, "Egt: a toolbox for multiple view geometry and visual servoing," *IEEE Robotics and Automation Magazine*, pp. 26–39, 2005.

[34] Cyberbotics, *Webots User Guide. Release 8.0.5*, 2015.

[35] Aldebaran, "Nao software 1.14.5 documentación."

[36] "Opencv 2.4.11.0 documentation."

[37] D. Lélis Baggio, S. Emami, D. Millán Escrivá, K. Ievgen, N. Mahmood, J. Saragih, and R. Shilkrot, *Mastering OpenCV with Practical Computer Vision Projects*. Packt Publishing, 2012.

[38] R. Laganière, *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Publishing, 2011.

[39] D. Q. Huynh, "Metrics for 3d rotations: Comparision and analysis," *Journal of Mathematical Imaging and Vision*, pp. 155–164, 2009.

Apéndices

Apéndice A

NAOqi Framework

Aquí se explican los conceptos básicos sobre la infraestructura de programación.

A.1. NAOqi

El robot NAO cuenta con el siguiente software embebido [35] (véase el lado izquierdo de la Figura A.1):

- OpenNAO: es el sistema operativo del robot. Es una distribución embebida de GNU/Linux basada en Gentoo
- NAOqi: es el principal programa que se ejecuta en el robot. Se usa para definir el comportamiento del robot.

Para programar el comportamiento del robot, *Aldebran* [35] proporciona varias herramientas de desarrollo, como las mostradas del lado derecho de la Figura A.1. Aquí se destaca C++ SDK, el cual permite escribir código para su ejecución en tiempo real, y Python SDK que facilita la programación. Ambos se encuentran en el software de desarrollo **naoqi-sdk**.

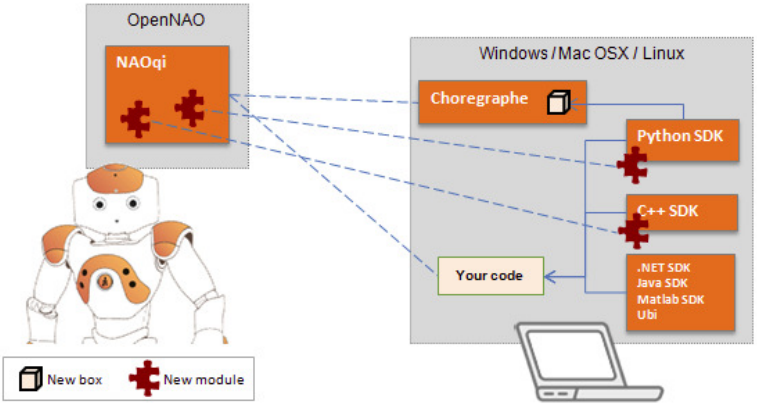


Figura A.1: Herramientas de Desarrollo. Figura tomada de `http://doc.aldebaran.com/1-14/getting_started/software_in_and_out.html`

A.1.1. *Broker*

NAOqi define el comportamiento del robot a través de la llamada a módulos y la ejecución de métodos. Para esto se necesita un intermediario de petición de objetos denominado *broker*. En la Figura A.2 se muestra de manera esquemática la relación entre éste, los módulos y los métodos.

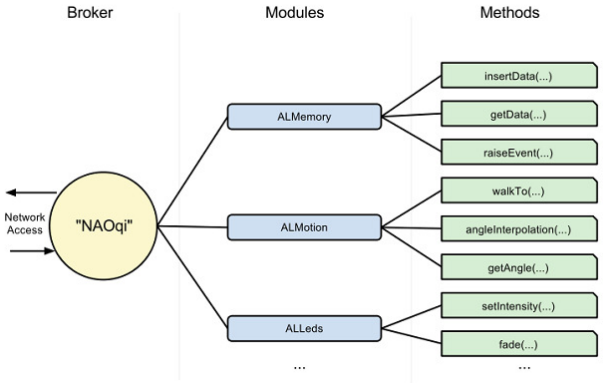


Figura A.2: Esquema del proceso del NAOqi. Figura tomada de `http://doc.aldebaran.com/1-14/dev/naoqi/index.html#naoqi-framework-overview`

Entonces, las funciones principales del *broker* son [35]:

- Servicio de directorio: permite encontrar módulos y métodos.

- Acceso de Red: permite a los métodos de los diferentes módulos ser llamados desde fuera del proceso.

El *broker* es un archivo binario que se ejecuta de forma independiente y se le asigna una dirección IP y un puerto. Puede ser ejecutado en el robot o en la computadora.

La funcionalidad de cada *broker* está dada por sus módulos. El robot NAO tiene un *broker* principal, al cual están ligados módulos predeterminados por Aldebaran [35] (véase el lado derecho de la Figura A.3). En el *broker* principal se puede conectar al menos un *broker*, el cual contiene uno o mas módulos diseñados por el usuario, como muestra el lado izquierda de la Figura A.3.

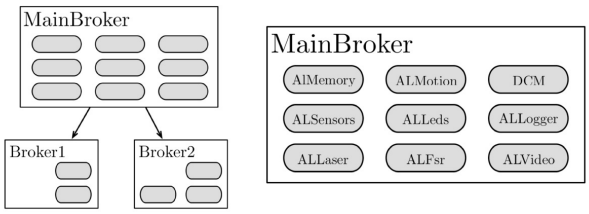


Figura A.3: Broker

A.1.2. *Proxy*

El *proxy* es un objeto de un determinado módulo, el cual permite la comunicación entre los módulos en diversos *brokers* para poder acceder a sus métodos. El *proxy* puede realizar dos tipos de llamadas [35]:

- Local: Si el código que se está ejecutando y el módulo al que se desea conectar el *proxy* están en el mismo *broker*.
- Remota: Si los módulos que se desean conectar no se encuentran en el mismo *broker*.

Lo anterior se puede ver de manera gráfica en la Figura A.4.

A su vez, los métodos pueden ser llamados de dos formas distintas [35]:

- *Blocking call*: la siguiente intrucción será ejecutada después del fin de la llamada previa.

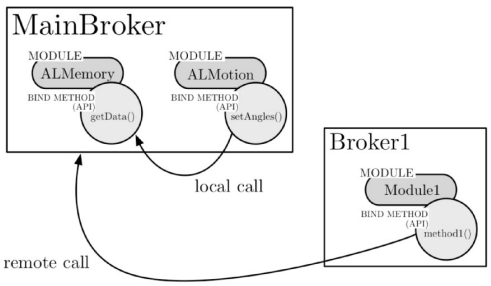


Figura A.4: Proxy

- *Non-blocking call*: una tarea es creada en un hilo paralelo con ayuda de la instrucción *post* de un *proxy*.

A.1.3. Módulo

El autoload.ini es el archivo que contiene todos las bibliotecas y ejecutables a lanzar cuando NAOqi inicia, véase la Figura A.5.

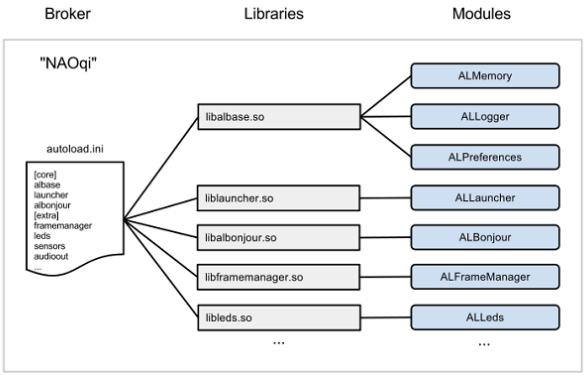


Figura A.5: Proceso del NAOqi. Figura tomada de
`http://doc.aldebaran.com/1-14/dev/naoqi/index.html#naoqi-framework-overview`

Por lo tanto, un módulo es una clase dentro de una biblioteca y tiene métodos diseñados para su funcionamiento o para ser ejecutados fuera del proceso. Se puede programar de dos formas distintas [35]:

- Módulo local: es compilado como una biblioteca y sólo puede ser ejecutada dentro del robot. Son dos o más módulos lanzados en el mismo

proceso y se comunican sólo usando el *broker* al que están conectados.

- Módulo remoto: es compilado como un archivo ejecutable y puede ser usado fuera del robot. Los módulos se comunican a través de una red y es necesario un *broker*, que será responsable de la comunicación, para poder llamar a otro módulos a través de un *proxy*.

En consecuencia, una conexión *broker* a *broker* abre una comunicación mutua entre los módulos de ambos *brokers*. En el otro caso, una conexión *proxy* a *broker* abre una comunicación unidireccional, donde un *proxy* instanciado en un módulo puede acceder a los métodos de los módulos registrados en el *broker*, pero estos últimos no pueden acceder a los métodos del módulo al cual pertenece el *proxy*.