

# Desarrollo de un sistema de navegación basado en visión estéreo utilizando Lego<sup>®</sup> Mindstorms<sup>®</sup> NXT

Alejandro Sartori

Facultad de Ingeniería y Ciencias Hídricas  
Universidad Nacional del Litoral

**Resumen** En este proyecto se utiliza una cámara estereoscópica con la que se capturan pares de imágenes de la misma escena, para producir mapas de disparidad que sirvan al esquivar obstáculos por parte de un robot Lego<sup>®</sup> Mindstorms<sup>®</sup> NXT. Con tal capacidad, dicho robot debe ser capaz de aproximarse sin riesgo al destino de manera autónoma. Dado que se extrae información métrica a partir de imágenes bidimensionales, primero se realiza un módulo de calibración de cámaras, para corregir distorsiones producidas por las mismas. Posteriormente se utiliza un método basado en áreas para generar mapas de disparidad, los cuales se interpretan con el fin de detectar obstáculos frente al robot. Las imágenes obtenidas son rectificadas, para aprovechar las ventajas que brinda la geometría epipolar. El manejo del robot se realiza según el paradigma reactivo en el marco de la arquitectura de control. La ejecución se lleva a cabo en una mini computadora denominada Raspberry Pi. Dicho sistema es capaz de responder en tiempo real a un entorno estático. Los experimentos de odometría y trayectoria llevados a cabo muestran que el sistema es capaz de alcanzar un punto objetivo con precisión satisfactoria aún en la presencia de obstáculos dispuestos de formas diversas.

## 1. Introducción

La visión estéreo se consigue observando la misma escena desde dos perspectivas diferentes. Esto es posible capturando el mismo punto en las dos visualizaciones (correspondencias), para calcular diferencias de coordenadas. Tales diferencias se denominan disparidades y al conjunto total de diferencias mapa de disparidad, con el cual se infiere distancias relativas.

Para que la búsqueda de correspondencias gane en precisión y eficiencia, se debe: por un lado obtener lentes libres de distorsión y por el otro utilizar la geometría asociada al sistema estéreo (epipolar), para encontrar las relaciones espaciales entre ambos lentes. Estas relaciones permiten la rectificación de imágenes, haciendo que las búsquedas sean unidimensionales [2].

En este proyecto se desarrolla un sistema de navegación y esquivar obstáculos utilizando un algoritmo basado en la interpretación tridimensional de la escena. Para alcanzar el objetivo deben resolverse tres problemas: primero obtener

imágenes libres de distorsión, segundo interpretar el mapa generado y el tercero estimar el movimiento del robot mediante un módulo de navegación en tiempo real.

## 2. Marco Teórico

### 2.1. Modelo de cámara

La formación de imágenes se representa mediante proyección central, en la cual se dibuja un rayo desde un punto del espacio hasta un punto fijo denominado centro de proyección. El rayo intersecta un plano que se elige como plano de imagen. Esa intersección representa la imagen del punto.

Este modelo concuerda con el modelo de cámara pinhole. En éste, el centro de proyección es el origen de un sistema de coordenadas Euclideo.

Para llevar el modelo pinhole a coordenadas de píxel, se introducen los parámetros  $c_x$  y  $c_y$ , que representan los desplazamientos en el sensor CCD con respecto al eje óptico y los parámetros  $f_x$  y  $f_y$  que representan las longitudes focales. Estos parámetros se organizan en una matriz que lleva puntos del espacio al plano de imagen. Dicha matriz es la matriz intrínseca  $M$ .

### 2.2. Distorsiones del lente

Dado que en este proyecto se extrae información métrica a partir de imágenes tridimensionales, es obligatorio remover las distorsiones que introduce la utilización de cámaras [4].

#### Distorsión Radial

Causa que líneas rectas en el espacio se rendericen como líneas curvas en el sensor CCD, es el tipo de distorsión que produce las deformaciones más severas cercanas a los bordes del sensor.

Se modela con los primeros términos de una serie de Taylor alrededor de  $r = 0$ , siendo  $r$  la distancia al centro óptico. Para cámaras baratas, se pueden caracterizar con los primeros dos términos que se denominan convencionalmente  $k_1$  y  $k_2$ . Para cámaras altamente distorsionadas se agrega un término adicional  $k_3$ . La ubicación radial de un punto sobre el sensor se corrige según las ecuaciones [5]

$$\begin{aligned} x_{\text{corregido}} &= x(1 + k_1r^2 + k_2r^4 + k_3r^6), \\ y_{\text{corregido}} &= y(1 + k_1r^2 + k_2r^4 + k_3r^6). \end{aligned} \quad (1)$$

#### Distorsión tangencial

Ocurre debido a defectos en el proceso de fabricación que resultan en un lente no alineado exactamente al plano de imagen. La distorsión tangencial se caracteriza por dos parámetros denominados coeficientes de descentrado del objetivo  $p_1$  y  $p_2$  según [5]

$$\begin{aligned}x_{\text{corregido}} &= x + [2p_1xy + p_2(r^2 + 2x^2)], \\y_{\text{corregido}} &= y + [p_1(r^2 + 2y^2) + 2p_2xy].\end{aligned}\tag{2}$$

### Modelo para la triangulación

Con las correspondencias obtenidas, se infiere profundidad a partir de la disparidad. Para lograr tal cometido el dispositivo debe responder al modelo paralelo-frontal, es decir [1]:

1. Ambos planos de imagen son coplanares, con sus ejes ópticos paralelos separados por una distancia conocida  $T$  e iguales longitudes focales.
2. Ambos puntos principales  $c_x^{izq}$  y  $c_x^{der}$  tienen las mismas coordenadas de píxel.
3. Las imágenes son de filas alineadas (cada fila de píxeles de una cámara se alinea exactamente con la correspondiente fila en la otra).

Tomando las coordenadas horizontales de los puntos  $x^l$  y  $x^r$  en cada cámara, se define disparidad a la resta:  $d = x^l - x^r$ . Con el modelo resultante se puede derivar la profundidad  $Z$  según

$$\frac{T - (x^l - x^r)}{Z - f} = \frac{T}{Z} \implies Z = \frac{fT}{x^l - x^r}.\tag{3}$$

donde la relación entre disparidad y profundidad es no lineal e inversamente proporcional [1].

### 2.3. Geometría Epipolar

En este trabajo se obtienen las correspondencias utilizando la geometría epipolar a fines de simplificar la búsqueda. Es la geometría proyectiva intrínseca de dos vistas solapadas (combina dos modelos pinhole). Permite dado un punto en una imagen, buscar su correspondencia a lo largo de la línea epipolar que le corresponde, es decir sobre la intersección del plano de imagen de la otra cámara con el plano epipolar (Figura 1) [2,7].

La matriz fundamental  $F$  constituye la principal herramienta en reconstrucción tridimensional, porque representa geoméricamente la restricción epipolar. Permite el mapeo de un punto en una imagen a su línea epipolar correspondiente en la otra.

## 3. Calibración de cámaras

Se realiza calibración fotogramétrica mediante objeto plano para conocer los parámetros intrínsecos y extrínsecos de cada vista. El objeto utilizado es un tablero de ajedrez del cual se conocen las coordenadas de los puntos a capturar en el sistema de coordenadas del objeto (intersecciones de los casilleros). Una

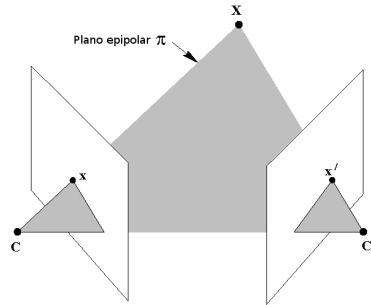


Figura 1: Geometría epipolar.

vez capturados dichos puntos y refinados a nivel sub-píxel, se obtienen las homografías (transformaciones proyectivas entre planos) que describen la relación entre los puntos del objeto y los capturados.

Cada homografía se desglosa en el producto de dos matrices (Ecuación 4), la matriz intrínseca  $M$  y la matriz extrínseca  $W$  que representa el cambio de coordenadas de un punto  $\tilde{Q}$  del objeto al punto  $\tilde{q}$  de la cámara (mediante la rotación  $R$  y la traslación  $T$ ).

$$\tilde{q} = sMW\tilde{Q}. \quad (4)$$

Gracias a que los dos primeros vectores columna  $r_1$  y  $r_2$  de  $R$  son ortonormales, surgen las ecuaciones 5 y 6 para cada una de las vistas de la forma

$$h_1^T(M^{-1})^T M^{-1}h_2 = 0, \quad (5)$$

y

$$h_1^T(M^{-1})^T M^{-1}h_1 = h_2^T(M^{-1})^T M^{-1}h_2. \quad (6)$$

El producto  $(M^{-1})^T M^{-1}$  es la matriz  $B$  simétrica que tiene entre sus componentes los parámetros intrínsecos de la cámara. Dicha simetría permite reescribir las ecuaciones 5 y 6 como productos unidimensionales por un vector de seis componentes según

$$\begin{bmatrix} v_{12}^T \\ (v_{11} - v_{22})^T \end{bmatrix} b = 0.$$

Se resuelve este sistema de ecuaciones para encontrar  $b$ , lo cual define la matriz  $B$  desde la que se extraen los parámetros intrínsecos. Dado que aún no se resolvieron las distorsiones, estos parámetros no son los correctos. Entonces, las coordenadas de píxel se obtienen utilizando las coordenadas del punto  $\tilde{Q} = [X^W, Y^W, Z^W]$  en el espacio según

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} f_x X^W / Z^W + c_x \\ f_y Y^W / Z^W + c_y \end{bmatrix},$$

para utilizar en

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \begin{bmatrix} x_d \\ y_d \end{bmatrix} + \begin{bmatrix} 2p_1 x_d y_d + p_2 (r^2 + 2x_d^2) \\ p_1 (r^2 + 2y_d^2) + 2p_1 x_d y_d \end{bmatrix}.$$

Con esta ecuación evaluada en múltiples puntos se forma un sistema de ecuaciones para calcular los parámetros de distorsión, luego de lo cual se obtienen los parámetros intrínsecos reales. Los extrínsecos se extraen de la Ecuación 4.

### 3.1. Calibración estéreo

La calibración estéreo consiste en calcular la relación geométrica entre cámaras, mediante la matriz de rotación  $R$  y el vector de traslación  $T$ . Utilizando las extrínsecas de cada par de visualizaciones del tablero se puede obtener dicha relación según las ecuaciones 7 y 8 [1,10].

$$R = R_r (R_l)^T \quad (7)$$

y

$$T = T_r - RT_l. \quad (8)$$

Se toma la mediana de  $R$  y  $T$  como aproximación inicial y luego se ejecuta un algoritmo iterativo de Levenberg-Marquardt para encontrar la solución final. Dicho algoritmo es un proceso iterativo de convergencia a través de la técnica de los mínimos cuadrados ponderados [6].

### 3.2. Rectificación estéreo calibrada: Algoritmo de Bouguet

La rectificación el paso final para obtener el modelo paralelo-frontal. La rotación  $R$  se divide a la mitad entre ambas cámaras, resultando en las matrices  $r_l$  y  $r_r$ . Con tales rotaciones los planos se vuelven coplanares y sus ejes ópticos paralelos. Luego se calcula la matriz  $R_{rect}$ , la cual produce una rotación alrededor del centro óptico. Las columnas de dicha matriz se obtienen según

$$e_1 = \frac{T}{\|T\|}, \quad (9)$$

$$e_2 = \frac{[-T_y T_x 0]^T}{\sqrt{T_x^2 + T_y^2}}, \quad (10)$$

y

$$e_3 = e_1 \times e_2. \quad (11)$$

La alineación de filas de las cámaras se obtiene utilizando  $R_l = R_{rect} r_l$  y  $R_r = R_{rect} r_r$  para las cámaras izquierda y derecha respectivamente. Las imágenes originales se remapean al modelo paralelo frontal con las líneas epipolares

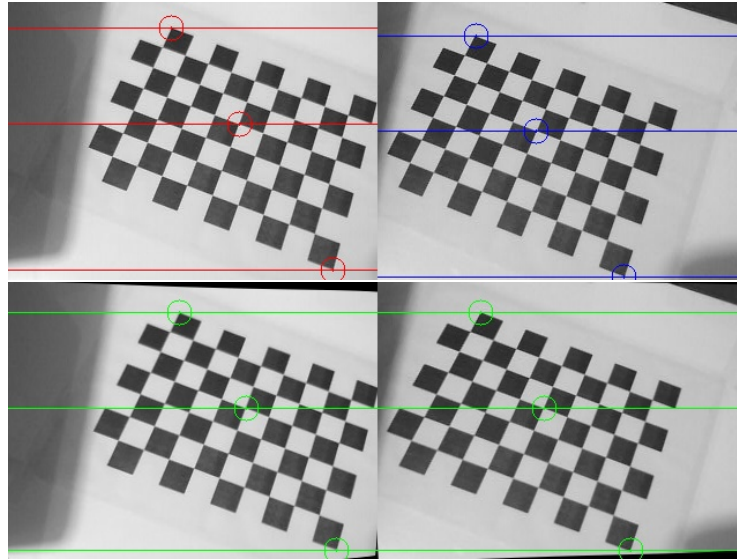


Figura 2: Par original (Arriba). Par rectificado (Abajo).

coincidentes a lo largo de filas. Luego de la rectificación los puntos aparecen en la misma fila (Figura 2).

Las imágenes que se utilizaron para la calibración también se pueden utilizar para verificarla. Al resolver la homografía que relacionaba el tablero de ajedrez con el plano de imagen, se obtuvieron por cada vista los valores de rotación  $R$ , traslación  $T$ , intrínsecas de cámara y parámetros de distorsión. Tales parámetros se pueden utilizar para re proyectar los puntos del objeto y obtener el error cuadrático medio entre los valores re proyectados y los puntos característicos refinados a nivel subpíxel. Se considera que dicho valor debe ser menor a 1 píxel para configuraciones de alta precisión [11]. En este proyecto el valor obtenido es 0.36, lo cual significa que los puntos re proyectados están en promedio a 0.36 unidad de píxel de su posición real.

## 4. Módulo de Navegación

### 4.1. Generación del mapa de disparidad

En este proyecto se utiliza el método basado en áreas con suma de diferencias absolutas como función de costo para generar los mapas de disparidad según

$$SAD(x, y) = \sum_{(i, j) \in W} |I_1(i, j) - I_2(x + i, y + j)| \quad (12)$$

para la vecindad  $W$  centrada en  $(x, y)$  de las imágenes  $I_1$  e  $I_2$ .

Como el emparejamiento se basa en intensidades, puede ser inexacto debido a ruido en la imagen, por ello en vez del píxel independiente se chequea el píxel junto con su vecindad. Dicha vecindad centrada en el píxel a buscar se desplaza a lo largo de la línea epipolar en la otra imagen, en cada una de las coordenadas se calcula la Ecuación 12 y donde se obtenga el menor valor estará la correspondencia para dicho punto. Generar el mapa de disparidad desde las imágenes rectificadas consta de tres etapas [9,1]:

1. Prefiltrado: las imágenes se normalizan para reducir diferencias en brillo y realzar texturas.
2. Búsqueda de correspondencias: se buscan características con textura suficiente a lo largo de líneas epipolares.
3. Posfiltrado: se verifica la diferencia entre sumas de diferencias absolutas entre la mejor coincidencia y la siguiente en orden ascendente. Si la diferencia no supera el valor esperado no se considera la disparidad calculada para ese punto.

#### 4.2. Interpretación del mapa de disparidad

Para detectar y esquivar posibles obstáculos al frente del robot, los mapas de disparidad obtenidos a partir de las imágenes rectificadas se dividen en una ventana central y dos laterales como se observa en la Figura 3.

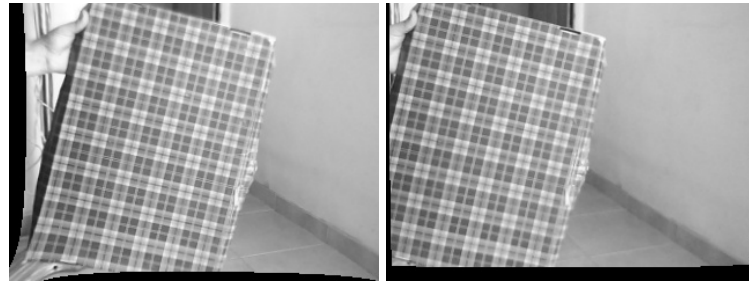
Por cada mapa se procede según:

- De las ventanas, se calculan los píxeles  $p$  cuyo valor de disparidad  $D(p)$  es superior a un umbral definido  $T$ .
- Se examina tal cantidad. Si el porcentaje que representa con respecto al total de píxeles en dicha ventana no supera un porcentaje predefinido  $r$ , no existen obstáculos en esa dirección. Si se supera, se rota hacia la derecha o la izquierda de acuerdo al menor porcentaje obtenido de las ventanas laterales [8].
- Si no se detecta obstáculo en la ventana central, se examinan las ventanas laterales. La detección de obstáculo en las mismas provoca una rotación en la dirección opuesta.

### 5. Configuración del robot

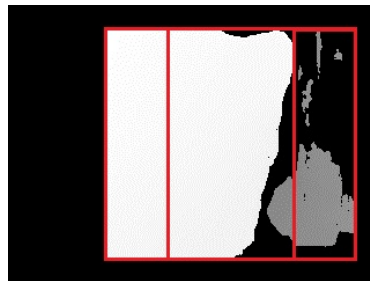
Este proyecto se realiza con el Lego<sup>®</sup> Mindstorms<sup>®</sup> NXT ensamblado en el formato de robot diferencial, esto es con dos ruedas a cada lado del cuerpo del robot controladas por sendos motores, más una tercer rueda que brinda estabilidad (Figura 4). Con esta configuración se puede utilizar odometría, es decir estimar su posición en función de las rotaciones de las ruedas, su diámetro y la distancia del eje.

Como dispositivo de sensado se utiliza la webcam estéreo marca Minoru<sup>®</sup> ubicada al frente del robot. El software que procesa las imágenes desde el sensado



(a) Izquierda rectificada

(b) Derecha rectificada



(c) Disparidad normalizada

Figura 3: Interpretación del mapa de disparidad.

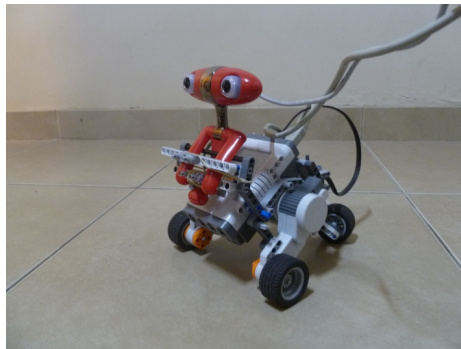


Figura 4: Robot móvil controlado por visión

hasta el cálculo del mapa de disparidad y que controla los movimientos del robot se ejecuta en la mini computadora Raspberry Pi.

Este proyecto utiliza el LeJOS NXJ como reemplazo del firmware original del Lego, posibilitando interactuar con el Lego utilizando Java. Se controla el robot mediante las clases Java (`classes.jar`) que implementan el API (Application Programming Interface) de leJOS NXJ, particularmente `DifferentialPilot` para controlar los movimientos del robot y `OdometryPoseProvider` para registrar la posición actual del mismo. La clase `DifferentialPilot` necesita saber a que puertos se conectan los motores, el diámetro de las ruedas y el ancho del eje. La clase `OdometryPoseProvider` estima la ubicación actual y la dirección a la cual apunta el robot (`heading`). Utiliza coordenadas cartesianas, con ángulos en grados para representar posición y orientación del Lego.

## 6. Arquitectura de control

La arquitectura de control es la que brinda un marco para la programación de sistemas robóticos inteligentes. Define paradigmas que establecen la manera en que se relacionan las principales acciones de un robot (SENSAR, PLANEAR y ACTUAR). Estos paradigmas son tres: el reactivo, el deliberativo y el híbrido (reactivo/deliberativo). En este proyecto se utiliza el paradigma reactivo por ser el apropiado para aplicaciones en tiempo real, dado que propone una conexión directa entre el sensado y las acciones sin requerir etapas de modelado. Las acciones posibles del robot se descomponen en comportamientos, que actúan en paralelo. Estos comportamientos se traducen en comandos para los motores.

En particular se utiliza la Arquitectura de Subsunción, la cual es un ejemplo del paradigma reactivo. En la misma los comportamientos tienen jerarquía y los superiores predominan sobre (subsumen) los niveles inferiores. Requiere la existencia de un árbitro para definir cuál es el comportamiento activo en un momento dado.

El desarrollo se realiza utilizando la implementación existente en el API de leJOS NXJ para el modelo de subsunción. La misma consta de una clase `Arbitrator` para definir el comportamiento activo y una interface `Behavior`, la cual modela los comportamientos y consta de los métodos:

- `takeControl`: Para solicitar el control del robot en el caso que corresponda.
- `action`: Código que se ejecuta una vez que este comportamiento toma el control.
- `suppress`: Código para cancelar inmediatamente al método `action`.

Se implementa una clase adicional denominada `Minoru`. Con ésta se genera la información relativa al sensado disponible para los comportamientos definidos en este proyecto: Esquivar, Flanquear y Avanzar (ordenados de mayor a menor según nivel de jerarquía). En primera instancia, el robot se orienta al punto destino que es parámetro del sistema, posteriormente los comportamientos entran en competencia por el control.

Descripción de comportamientos:

- Esquivar: Toma el control si existe colisión. Devuelve el control una vez concretada la rotación determinada por la interpretación del mapa de disparidad.
- Flanquear: Si hubo colisión hace que el robot transite la distancia necesaria para sortear el obstáculo. Una vez finalizado dicho recorrido, el robot se reorienta al destino.
- Avanzar: Toma el control si el camino está despejado. Controla la distancia al punto destino, si la misma se encuentra por debajo de un umbral determinado, detiene el robot.

## 7. Pruebas de odometría

Para estimar la precisión de la odometría se hizo transitar al robot seis recorridos de 1,80 m de distancia, tres de los cuales fueron con el camino libre y los tres restantes con un obstáculo a la mitad del recorrido. Como obstáculo se utilizó una caja de 29 cm de ancho, 24 cm de alto y 14 cm de profundidad. Se midió el error en cm obtenido en cada uno de ellos. Tal error en cm es la diferencia entre el centro de eje del robot y el punto definido como destino (Tablas 1 y 2).

Tabla 1: Error sin obstáculo.

Recorrido	Error en cm
1	3,1
2	3,2
3	4,5
Error promedio	3,6
Porcentaje de error	2,0 %

Tabla 2: Error con obstáculo.

Recorrido	Error en cm
1	5,1
2	5,2
3	6,3
Error promedio	5,53
Porcentaje de error	3,07 %

## 8. Pruebas de trayectoria

Según las mediciones, el tiempo promedio desde que se capturan las imágenes hasta que se genera el mapa de disparidad con su posterior interpretación es de 3,1 segundos en el Raspberry Pi. Según la calibración, la distancia focal promedio es de 455,7 píxeles. Tal medida se utiliza para calcular la profundidad  $Z$  utilizando la Ecuación 3. Se sabe que la distancia de línea base del dispositivo estereoscópico es  $T = 6$  cm, y que el algoritmo de correspondencia fija la coordenada de comienzo de búsqueda en 0 y el número de disparidades a buscar en 64. Estos parámetros proporcionan un mapa de disparidad que va de los 42,7 cm de profundidad al “infinito”.

Tales resultados sirven para especificar la velocidad máxima del robot y el umbral para el mapa de disparidad. Definiendo el umbral en 47 píxeles, se estarán



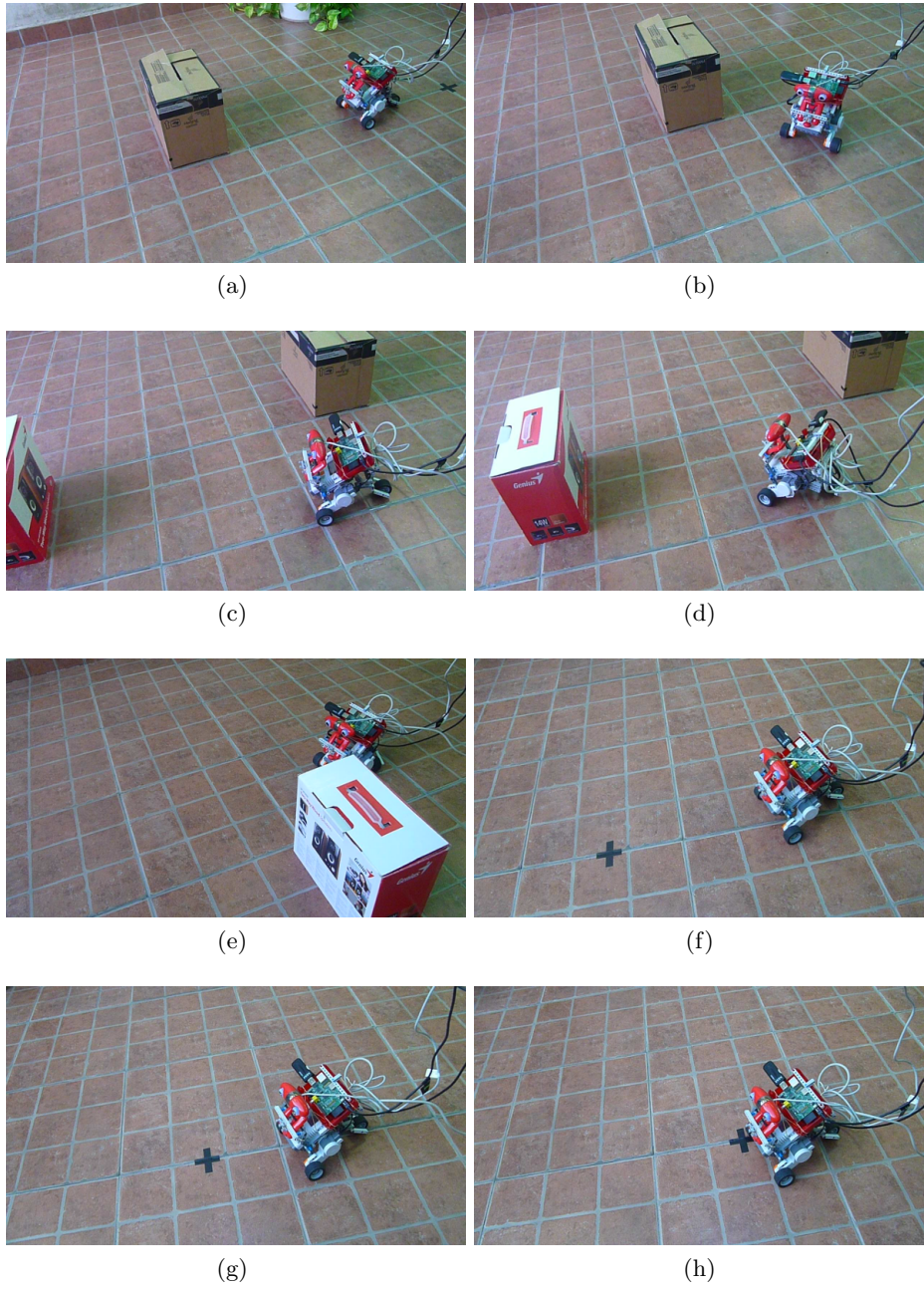


Figura 6: Capturas de la trayectoria ejemplo.

Dado que el software se ejecuta en una mini computadora, se tienen las restricciones dadas por su capacidad de procesamiento limitada. Por ello es necesario que el avance del Lego<sup>®</sup> Mindstorms<sup>®</sup> NXT sea a baja velocidad. Tal requerimiento permite que el sistema sea capaz de realizar el procesamiento total de las imágenes, desde la captura hasta la interpretación del mapa de disparidad, y con ello definir una orden para el robot a intervalos de tiempos apropiados para que el mismo transite sin colisionar al destino.

La utilización de la mini computadora Raspberry Pi en conjunto con la plataforma de software elegida representa un condicionante para el desempeño del robot. Esto se puede resolver de tres maneras: mediante hardware más potente, una solución empaquetada o mediante el uso de rutinas de software hechas a medida en el Raspberry Pi.

Utilizar hardware más potente (como la notebook mencionada previamente), permitiría que los cálculos necesarios para la toma de decisiones se realicen en el orden de los milisegundos, posibilitando que el robot se mueva a velocidades superiores. Si se desea realizar una solución comercial empaquetada (como el robot limpiador Roomba o similar) utilizando los algoritmos desarrollados en este proyecto, se deben tener en cuenta los costos de producción y facilidad de ensamblaje. Es en este caso donde la mini computadora Raspberry Pi lleva la ventaja, dado que su costo, tamaño y peso son significativamente menores [12].

La mejor opción a futuro sería el desarrollo de software a medida. Para ello sería necesario migrar la mayoría de código Java a C o C++, utilizando rutinas propias de conexión para la cámara web y el robot, y desarrollar algoritmos óptimos de cálculo para poder comparar el desempeño. Se podría por ejemplo, intentar sacar provecho de la GPU buscando la manera de realizar algunos de los cálculos en dicho procesador.

Comparando este proyecto con trabajos actuales de objetivos similares realizados en el país, se concluye que se ha logrado dar un paso adelante hacia la autonomía en robótica. Esto se debe a que se logró desarrollar un sistema capaz de ejecutarse en tiempo real en una computadora de capacidad reducida, cuyo tamaño y peso permiten acercarse significativamente a la implementación de una solución empaquetada. Además, se ha podido brindar un comportamiento de mayor complejidad que sólo el de esquivar obstáculos, tal como es el caso de lo logrado en uno de los trabajos observados [13,14].

La precisión con la cual el robot llega al punto destino es dependiente de los errores de odometría, y está fuertemente condicionada por la correcta orientación del mismo al inicio del recorrido. Minimizando el umbral de distancia al destino es posible obtener mejores resultados. Se fijó dicho umbral en 2 cm con lo cual se alcanzó un error de 1,5% en el recorrido de 3 m.

## 10. Conclusiones y trabajos futuros

En este proyecto se ha desarrollado e implementado un sistema de navegación guiado por visión computacional para Lego<sup>®</sup> Mindstorms<sup>®</sup> NXT para ejecutarse en el ordenador de placa reducida Raspberry Pi.

El sistema desarrollado consta de las etapas necesarias de calibración del dispositivo estereoscópico para la remoción de distorsiones, la generación e interpretación de los mapas de disparidad para el esquivar de obstáculos y el control del robot. En su funcionamiento, se logra una precisión aceptable en la llegada al punto destino.

En cuanto a trabajos futuros, dos puntos son identificados como pasos próximos a seguir. Por un lado, los errores de posicionamiento en la llegada podrían disminuirse con la fusión de información proveniente de los otros sensores del robot (particularmente, acelerómetro y giróscopo), donde un filtro de Kalman sería capaz de obtener los parámetros óptimos para estimar el posicionamiento. Por otro lado, el sistema actualmente no tiene en cuenta la ubicación del punto final al momento de esquivar obstáculos, restricción que podría minimizar el recorrido realizado.

## Referencias

1. Bradski, G. and Kaehler, A.: Learning OpenCV: Computer vision with the OpenCV library. O'Reilly (2008)
2. R. Hartley and A. Zisserman: Multiple view geometry in computer vision. Press Syndicate of the University of Cambridge (2004)
3. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
4. Beauchemin, Seven S and Bajcsy, Ruzena: Modelling and removing radial and tangential distortions in spherical lenses: Multi-Image Analysis 1–21. Springer (2001)
5. Fryer, J. G. and Brown, D. C.: Photogrammetric Engineering and Remote Sensing: IAPR TC 7 Workshop on Analytical Methods in Remote Sensing for Geographic Information Systems: 52 (1986), 51–58
6. M. I. A. Lourakis: A Brief Description of the Levenberg-Marquardt Algorithm Implemented. Institute of Computer Science, Foundation for Research and Technology (2005)
7. G. L. Foresti and C. Micheloni and T. Ellis: Active video-based surveillance system: the low-level image and video processing techniques needed for implementation: Signal Processing Magazine, IEEE: 22 (2) (2005) 25–37
8. Nalpantidis, L. and Kostavelis, I. and Gasteratos, A.: Stereovision-based algorithm for obstacle avoidance: Intelligent Robotics and Applications 195–204 Springer (2009)
9. J. Braux-Zin: Toward Real-Time Dense 3D Reconstruction using Stereo Vision: School of Computer Science and Communication (2011)
10. A. Bódis-Szomoru and T. Dabóczy and Z. Fazekas: Calibration and sensitivity analysis of a stereo vision-based driver assistance system: Chapter in Stereo Vision, In-Tech, ISBN 978-953-7619-22 (2008)
11. Habib, A. and Jarvis, A. and Detchev, G. and Stensaas, D. and Moe, D. and Christopherson, J.: Standards and specifications for the calibration and stability of amateur digital cameras for close-range mapping applications: International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences: 37 (2008)
12. Business Logic layer, <http://www.irobot.com/us/learn/home/roomba.aspx>, Fecha de acceso: 09-04-2014.

13. Pire T.:Evasión de obstáculos en tiempo real usando visión estéreo:Departamento de Computación, Facultad de Ciencias Exactas y Naturales, UBA: 37 (2012)
14. Ortega, C. D. and Moyano, F. E.:Técnicas de Implementación de Visión Estereoscópica en Robótica:EST - Jornadas Argentinas de Informática e Investigación Operativa: 439-451 (2013)