Implementación de algoritmos de desplazamiento para un robot Raspberry Pi

Joel Hernández Pla, Universidad Autónoma de Barcelona

Abstracto — Este proyecto es una alternativa a los sofisticados vehículos autónomos o a los seguidores de líneas. Desarrollado en dos módulos principales, uno de visión y otro de movimiento, se pretende conseguir un robot de bajo coste, compuesto por un ordenador mini Raspberry Pi conectado a un módulo de movimiento en ruedas del modelo MD25, que consiga desplazarse de forma autónoma sin la necesidad de la interacción humana. Esto se consigue mediante los módulos antes comentados, aplicándolos a dispositivos de bajo presupuesto por tal de hacerlos más accesibles. Para ello, se sigue una ejecución secuencial comenzando por la captura del escenario y el procesado de la imagen para realizar el cálculo de la ruta y ejecutarla en el módulo de movimiento.

Abstract — This project is an alternative to the sophisticated autonomous vehicles or the followers of lines. Developed in two principal modules, one of vision and another for the motion, it pretend get a low cost robot, made-up by a mini computer Raspberry Pi connected to one motion module of model MD25, that it gets move of autonomous way without need the human interaction. It is possible through the modules before commented, implemented in low budget to do more accessible. For this, follow a sequential execution beginning to the scene capture and the image processing to perform the path-finding and execute it in the movement module.

Index terms — *fiducial marker*, computer, detection, *Raspberry Pi*, wireless comunication, *path-finding*.

Palabras claves— *fiducial marker*, ordenador, detección, *Raspberry Pi*, comunicación inalámbrica, *path-finding*.

1 Introducción

Este proyecto es una continuación de un proyecto anterior dentro del ámbito de la Robótica y la Visión por Computador.

Con este proyecto, junto con su predecesor, se pretende conseguir la autonomía de un robot construido a través de dispositivos de relativamente bajo coste. De esta manera, se podría prestar un servicio de asistencia a nivel empresarial, en el ámbito social o en el ámbito educativo, dado que su aplicación resulta bastante amplia debido a su bajo presupuesto. Por ejemplo, el transporte de artículos o materiales dentro de una fábrica de producción o un almacén, la asistencia a personas con minusvalías visuales o la reduccion de costes en un vehículo autónomo.

2 Predecesor

El proyecto anterior consistía un *path*-finding, o lo que es lo mismo, un cálculo de una ruta dentro de un escenario mediante una cámara web conectada a un dispositivo Raspberry Pi y otro Raspberry Pi conectado a un módulo de control de ruedas para simular un robot. De esta manera se conseguía calcular una ruta la primera vez, cuando se ponía en marcha todo el sistema, procesando la imagen que se captaba y, a partir de un algoritmo de distribución de puntos aleatorios sobre la imagen y un algoritmo de búsqueda de trayectoria que se sirve de ellos.

Algunos inconvenientes de este sistema y que se han querido solventar en el actual proyecto son:

- Imposibilidad de detectar el robot en cualquier otra posición que no sea la inicial.
- Imposibilidad de reanudar el cálculo de la ruta si esta falla.

Por lo tanto, a raíz de estos errores, surgió el proyecto que comentamos en este artículo.

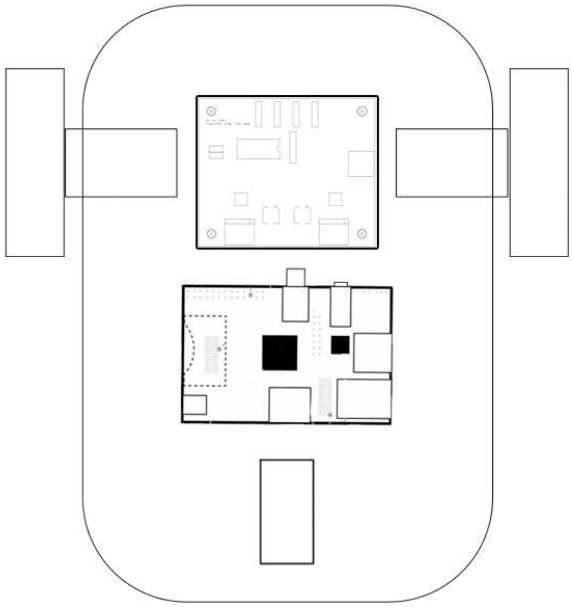


Figura 2. Diagrama del robot.

2.1 Arquitectura del sistema

La arquitectura de este sistema constaba de un robot compuesto por un módulo de movimiento llamado MD25 y una Raspberry Pi, encapsulado en una fiambrera con tres ruedas, dos de ellas unidas al módulo MD25. En este proyecto, se le decidió denominar Pluto.
Figura 2. Diagrama de Pluto.

En este módulo se cargaron aquellos ficheros que correspondían al módulo de movimiento del sistema.

Lo que correspondería al módulo de visión en este proyecto es el segundo dispositivo Raspberry Pi, que recibió el nombre de Zeus, que está conectado a una cámara web con la que se capturaba el escenario para así procesar la imagen, calcular la ruta y enviarla al otro Raspberry Pi para que fuera ejecutada.

Para poder comenzar todo el flujo de datos y todo el proceso, se precisaba de un dispositivo Android que tenía como función principal iniciar todo el proceso.

2.2 Flujo

El flujo de los datos o el proceso del cálculo de la ruta era el siguiente:

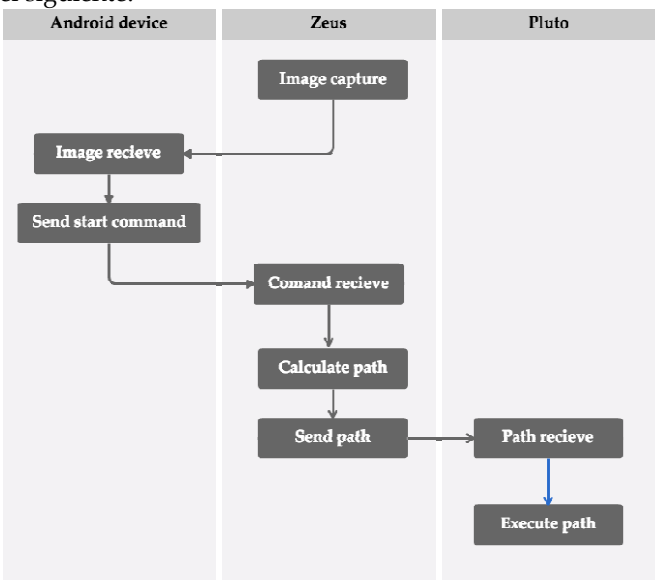


Figura 2. Diagrama del proceso.

Primero se capturaba una imagen del escenario para que se iniciara el proceso con el dispositivo Android que mandaba una señal a Zeus para tratar la imagen y calcular la ruta que se enviaba, posteriormente, a Pluto para que la ejecutara.

Para la detección de obstáculos, se usó un threshold con binarización para convertir todos los píxeles a blanco o negro dependiendo del color que tuvieran mientras que para el cálculo de la ruta se usó un algoritmo PRM junto con un algoritmo Lagart. Como se explica en el proyecto, el algoritmo PRM consiste en distribuir nodos aleatoriamente sobre el área usable de la imagen y, después, encontrar la ruta más corta de un nodo a otro, en nuestro caso desde el inicio hasta el objetivo. Para calcular dicha ruta, se usa el algoritmo Lagart, el cual es un algoritmo iterativo que calcula la distancia euclideana de cada vecino al nodo vecino y las ordena para escoger aquel nodo cuya distancia euclideana sea más pequeña.

Finalmente, esta ruta era enviada a Pluto, quien la ejecutaba.

3 METODOLOGÍA

3.1 Arquitectura del sistema a nivel de hardware

Como hemos visto, la anterior configuración del sistema constaba de dos dispositivos de bajo coste, dos Raspberry Pi, y un dispotivo móvil mientras que en este proyecto se ha optado por sustituir un Raspberry Pi y el dispositivo móvil por un ordenador. La intención de este cambio es reducir los componentes o dispositivos utilizados en el sistema, reducir la complejidad de la conexión de los mismos, y aumentar la capacidad de cómputo del mismo.

Las funciones de los distintos dispositivos serían las siguientes:

- **Raspberry Pi**: este dispositivo controla el movimiento del robot mediante un controlador de las ruedas llamado MD25.
- **Ordenador**: este dispositivo se encarga de la captura de la imagen del escenario y de su procesado para encontrar el robot y el punto objetivo.

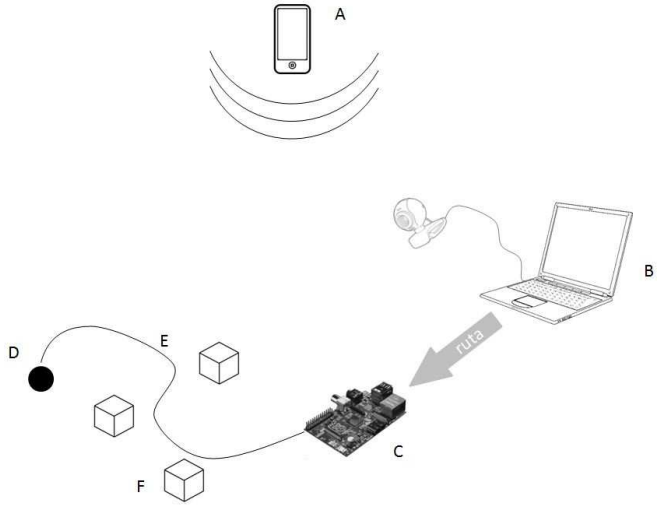


Figura 3. Esquema de componentes involucrados: A) Android, B) Ordenador con cámara web, C) Raspberry Pi y módulo MD25, D) punto objetivo, E) trayectoría calculada y F) obstáculos

En el sistema tenemos también un sistema Android para simular una red WiFi interna para evitar problemas con IP's.

3.2 Arquitectura del sistema a nivel de software

A nivel de software, tenemos los mismos módulos que en

el proyecto predecesor, los cuales son:

- Módulo de detección
- Módulo de movimiento
- Comunicación

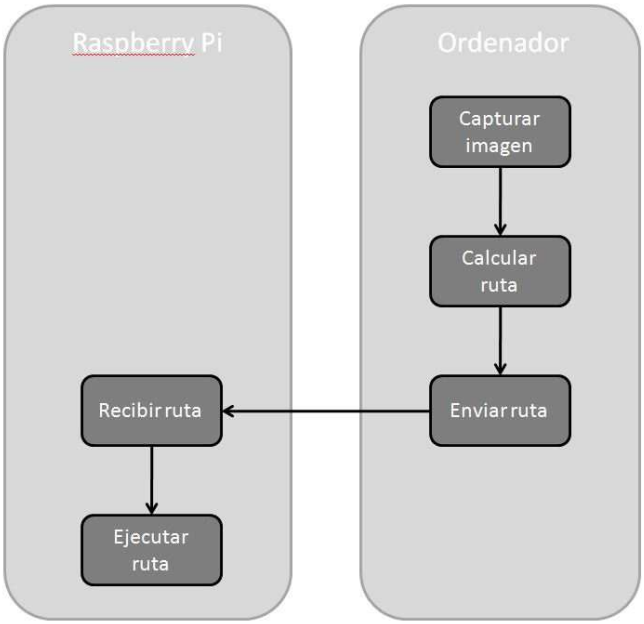


Figura 4. Diagrama del proceso.

En este caso, el esquema del proceso es más o menos similar al del proyecto anterior y se siguen los mismos pasos, pero varian los módulos que se ocupan de estas partes.

Sin embargo, hay una diferencia que destacar: en el proyecto anterior el proceso era secuencial y si en algún momento se quería volver a calcular una ruta, el robot tenía que colocarse de nuevo en la posición original y volver a calcular la ruta. En este proyecto, por el contrario, el Raspberry Pi siempre está escuchando el puerto de comunicación para saber si recibe una ruta nueva o continua ejecutando la última recibida, de esta manera no era necesario reiniciar el sistema al completo si se quería calcular una nueva ruta, si el punto final objetivo cambiaba de posición, si se añadía un obstáculo en medio del sistema, etc. Con esto, conseguimos que el sistema fuera más flexible y pudiera "reiniciarse" en cualquier momento. Además, conseguimos que el sistema también fuera más robusto, uno de los principales problemas del proyecto anterior. Y, también se añadió la opción de parar el sistema en cualquier momento.

3.3 Módulo de detección

Este módulo está implementado dentro del ordenador que hemos comentado anteriormente.

Uno de los principales problemas, que ya hemos comentado, con los que se habían encontrado en el proyecto anterior era la detección del robot, dado que solo funcionaba si se encontraba en la posición original y el punto objetivo siempre era fijo.

Con tal de mejorar esto y hacerlo más fluido y flexible y no tan restrictivo, se decidió introducir dos marcas diferentes, uno sobre el robot y otro con el que identificaríamos el punto final. Dichas marcas fueron *fiducial markers* unos dibujos geométricos característicos, usuados a menudo con este fin en varios proyectos de detección de objetos.

Figura 5. A la izquierda el *marker* para el robot y a la derecha el que usamos para el objetivo.

3.3.1 Detección del marker

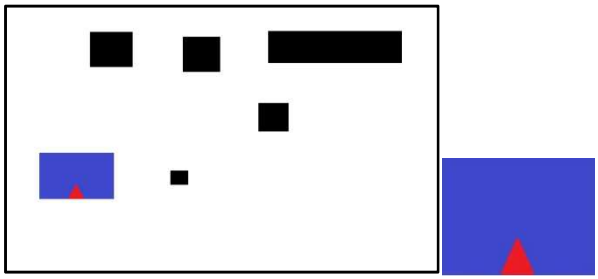
Ésta es una de las partes en las que más se ha centrado el proyecto, siendo el principal punto de trabajo del mismo. Mediante la webcam y el *marker*, comentado anteriormente, que tiene nuestro robot, se probaron diferentes opciones hasta encontrar la que decidió usarse como definitiva.

Para llegar al último algoritmo utilizado, pasamos primero por varias alternativas que fuimos descartando a medida que iban surgiendo inconvenientes.

Con ello se localiza el robot en cualquier lugar del escenario y pudiendo detectar, también, su orientación.

3.3.1.1 Estrategías planteadas

En un primer momento se pensó en usar una estrategia de *template matching* porque es uno de los algoritmos más sencillos que se podía aplicar con la librería OpenCV y se estuvo trabajando y desarrollando dicha idea hasta que vimos que tenía algunas desventajas dado que dependiendo de la iluminación, la orientación, la inclinación de la cámara, los colores de los objetos, etc, no se detectaba correctamente.



Figuras 6 y 7. Simulación de entorno con robot, y el robot mismo.

Como ya hemos comentado, uno de los principales inconvenientes de este método es que se tenía aplicar con varias imágenes del robot, guardadas con diferentes orientaciones, para intentar que el algoritmo fuera más robusto. A pesar de ello, si la orientación no era ninguna de las que habíamos guardado nosotros préviamente, éste no funcionaba.

Por eso, se contempló la idea de hacer una búsqueda por contornos, como segunda opción, para detectar el contorno del robot, pero esta opción daba demasiado falsos positivos para poder seguir adelante con el proyecto de forma correcta.

Al descartar esta idea se calló en la cuenta de usar un *fiducial marker* dado que se trata de una forma concreta siendo difícil que un obstáculo tenga la misma forma o estructura. Además, estos *markers*, son más fáciles de detectar porque a los algoritmos de detección les resulta más sencillo encontrar los *keypoints* (puntos claves de una imagen que ayudan a describirla) y los distintos descriptores que emplean. Para ello se uso la imagen con la estructura que hemos visto anteriormente, siendo más sencillo reconocerlo.

Primeramente se comenzó a hacer este proceso de forma propia. Dando buenos resultados en imágenes planas como vemos en la siguiente figura.

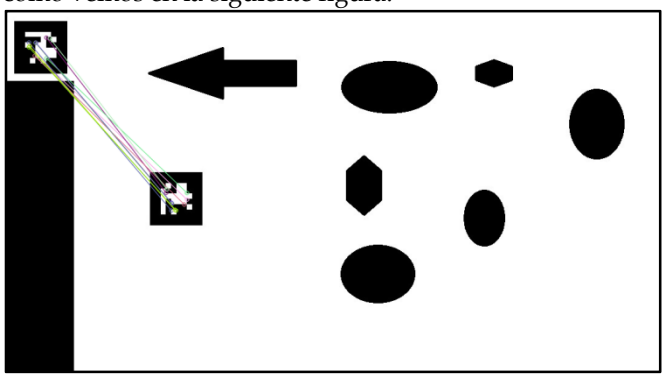


Figura 8. Resultado de simulación del algoritmo propio.

Para ello, utilizamos los descriptores de la imagen del escenario y del *fiducial marker* y los *keypoints* de ambas imágenes para reconocer el objeto dentro del escenario. Dado que parecía que funcionaba correctamente, terminamos de desarrollar el código para obtener la posición en la imagen y la orientación que tenía respecto de una que consideramos como la original. Pero cuando hicimos pruebas con imágenes más reales, nos dimos cuenta que la inclinación de la cámara y del objeto dificultaba que se reconociera correctamente la orientación en cualquier punto.

Con lo cual se buscó una librería externa que pudiera resolver nuestro problema, adaptando gran parte del código a nuestro caso y solventado estos errores con los que nos habíamos encontrado.

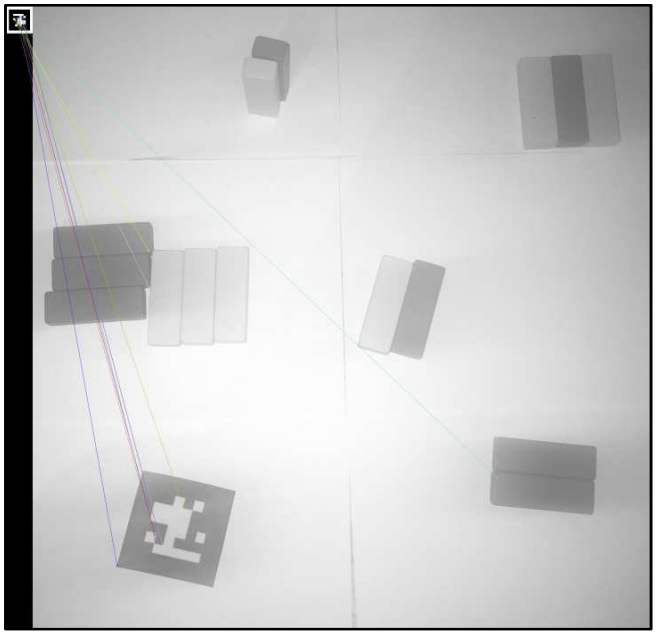


Figura 9. Primera prueba real.

Para ello, tuvo que hacerse además la calibración de la cámara web que usamos, con un código que también nos facilitó esta misma librería.

Dicho código, funcionaba a grandes rasgos como el que habíamos hecho nosotros mismos, pero añadiendo un elemento bastante importante y que daba robustez al programa. Este elemento es RANSAC que consistía en una selección aleatoria de los *keypoints* y que se repite varias veces para hasta que varios resultados son iguales, dando así algo más de robustez al código.

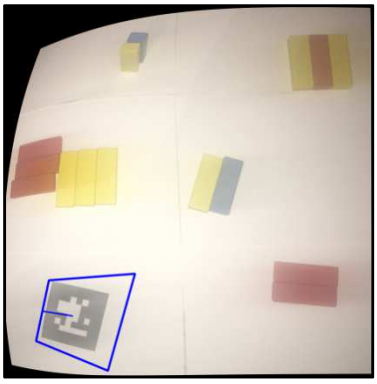


Figura 10. Resultados con experimientación.

Una vez comprobado que funcionaba correctamente, decidió añadirse el *fiducial marker* correspondiente al objetivo, dando resultados positivos.

A pesar de estos buenos resultados, comprobamos que también había errores porque no se detectaba correctamente el 100% de las veces. Por lo que probamos con diferentes tamaños de los *markers* y manteniendo la cámara, cómo vemos en la siguiente figura.

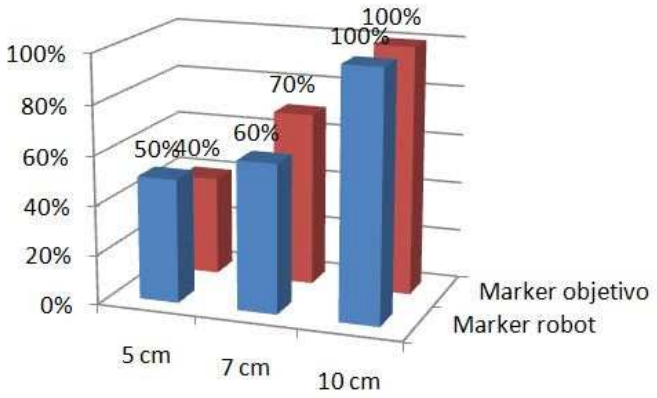


Figura 11. Gráfico de las veces que se encontraban los *markers*. Eje X, tamaño del *marker*, eje Y, porcentage de detección.

De esta manera, el resultado final que se decidió es que el tamaño de los *markers* debía ser de 10x10 cm.

3.3.2 Cálculo de ruta

Esta parte podemos incluirla en este módulo, dado que se ejecuta en el mismo dispositivo e interactua estrechamente con el módulo en cuestión. Además, éste código también forma parte de la detección de los obstáculos, por lo que queda todo unificado.

Como hemos comentado, en el predecesor de este proyecto, la detección de obstáculos se hacía con una binarización de la imagen a la que después se le aplicaba un threshold para detecctar los objetos del escenario. De esta manera, se conseguía lo que llamamos un *workspace*, o área disponible para el movimiento del robot. De esta manera, se distribuían los diferentes nodos en este *workspace* y se calculaba la ruta. Todo esto se hacía con los algoritmos PRM y Lagart, los cuáles hemos seguido utilizando para este proyecto.

Para darle más robustez al algoritmo y no tener que reiniciarlo, se decidió hacer que sí no se encontraba una ruta, se limpiara el *workspace* de nodos y se volvieran a distribuir, nuevamente al azar y se siguiera el proceso.

Además, se han hecho varias simplificaciones del código para el cálculo de la rotación que tiene que hacer el robot, sustituyendo la separación por cuadrantes que se había realizado en el anterior por una fórmula en la que intervienen la orientación actual y la orientación deseada después del movimiento, orientación que se puede obtener fácilmente mediante relaciones trigonométricas que se encuentran en la librería *math* de Python.

3.4 Módulo de movimiento

Este módulo está implementado en la Raspberry Pi que se coloca dentro del robot. Es un módulo más sencillo que el de detección dado que se encarga de ejecutar la ruta que recibe y lo hace a través de un controlador de las ruedas llamado MD25.

Cómo hemos dicho, el controlador puede hacer que el robot cambie de orientación y que se mueva en línea recta, con lo cual para hacer un movimiento primero se orienta los grados que se le indican y después se mueve la distancia que recibe.

3.5 Comunicación

Al tratarse de dos módulos computacionales, se pensó en una comunicación inalámbrica mediante un enlace Bluetooth, pero se descartó dado que en el anterior proyecto se usaba un tercer dispositivo, así que para poder permitir la incorporación de un tercer dispositivo, nuevamente, o ampliar el sistema a múltiples robots conectados a un solo sistema de cámaras, como podría ser una de sus aplicaciones, decidió dejarse la conexión inalámbrica vía WiFi. Además, se optó también por este tipo de conexión dado que el uso de la librería *socket()* de Python, para la conectividad entre dispositivos se ha convertido casi en un estándar y hay mucha información al respecto y es sencillo de programar.

La importancia de este módulo reside en la transferencia de datos entre dispositivos por tal de que los cálculos más complejos los ejecute el ordenador o una máquina de cómputo mayor, para así cargar a los dispositivos de bajo coste de menos trabajo computacional y que ejecuten la ruta de manera más fluida.

4 Experimentación

4.1 Comunicación

Por tal de facilitar la conexión entre los dispositivos, conectados a una red inalámbrica, y para evitar confrontaciones de IP con otros elementos ajenos al proyecto, se utilitzo un teléfono smartphone, un BQ Aquaris E5 concretamente para la generación de una red WiFi, aprovechando la herramienta que Android ofrece para compartir los datos como punto de anclaje WiFi. De esta manera, conseguíamos crear una red interna sin demasiada complejidad.

4.2 Escenario

El escenario que utilizamos para hacer las pruebas con el robot esta formado por un fondo blanco, para evitar que haya errores para detectar los *markers* y los obstáculos.

Además, se debe poner la cámara a cierta altura para captar bien la escena para que permita un buen cálculo de la ruta e intentado que los cables de corriente y conexión de los diferentes dispositivos interfieran lo menos posible.

En este momento, se llegó a tener algunos problemas, dado que se había trabajado con un ordenador portátil con Windows 8.1 como sistema operativo nativo, y los protocolos que usa dicho sistema dificultaban las tareas o la codificación de algunos módulos, con lo cuál se optó por la implantación de una máquina virtual en el ordena-

dor que hemos mencionado, dado que las versiones más recientes de Windows que van instaladas como nativas, no permiten un arranque dual, lo que dificulta la instalación de un segundo sistema operativo. Esto ocasionó un retraso en el desarrollo de los distintos módulos, lo cual dificultó que se pudieran desarrollar de la forma prevista.

5 RESULTADO

Como resultados finales de este proyecto, conseguimos que se hiciera un buen cálculo de la ruta y se detectara correctamente los dos *fiducial markers*, tal y como podemos observar:

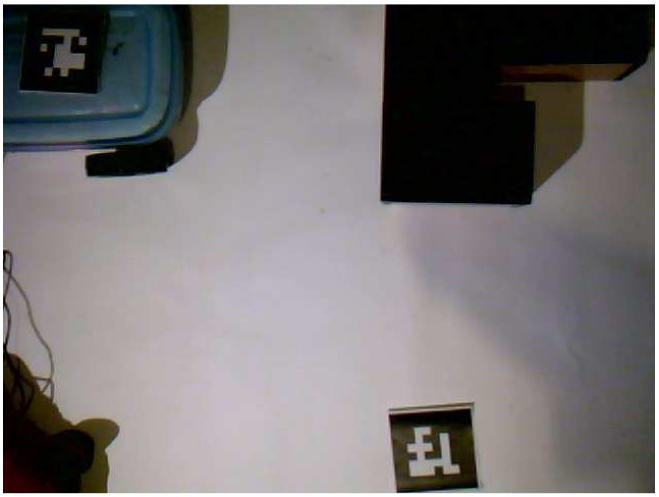


Figura 12. Imagen captada del escenario.

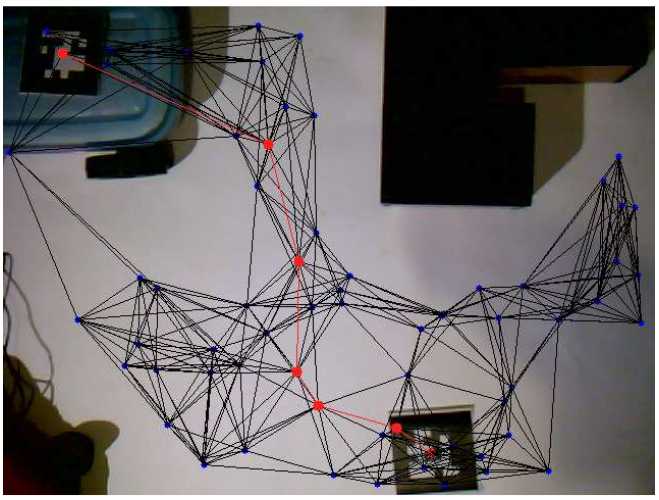


Figura 13. Trayectoria encontrada por el algoritmo.

Los resultados del módulo de visión son un éxito. Encuentra siempre una ruta que no interfiere con los obstáculos. Detecta correctamente los *markers*, los obstáculos, y encuentra la ruta.

El proceso que se sigue para ello, tal y como se ha explicado anteriormente, consiste en la aplicación de distintos filtros, de los cuáles podemos encontrar las distintas imágenes resultantes en el apéndice, que permiten anular los obstáculos como zonas útiles. También se considera un éxito este módulo dado que se ha mejorado en gran medida respecto del proyecto anterior, detectando el robot en cualquier orientación y posición del escenario, ampliando el mismo sin necesidad de usar delimitadores con lo cual se amplia el área de trabajo.

Por lo que corresponde al módulo de movimiento, nos encontramos con algunas dificultades, debido a parámetros internos del controlador de las ruedas, que hemos estado intentando compensar, llegando a conseguir que el robot llegue a su punto objetivo en un porcentaje de un 30%.

Éste es uno de los últimos inconvenientes con que nos hemos encontrado y con el que hemos estado liando a lo largo de la experimentación realizada en el último periodo del trabajo.

Así mismo, podemos concluir que el proyecto ha conseguido mayor robustez en condiciones de iluminación, haciendo pruebas en espacios más cerrados y menos iluminados, como en espacios totalmente iluminados; y mayor flexibilidad, por lo que corresponde a la localización del robot en cualquier punto del escenario.

6 PROPUESTAS DE MEJORA

6.1 Corriente

Respecto a este punto, es uno de los inconvenientes que sigue teniendo el sistema. Los cables que conectan los dispositivos a la corriente, dificultan el montaje del escenario o la libertad de movimiento del robot. Sobre todo las conexiones de éste, son las que más obstáculos causan, dado que no permiten que se mueva del todo libre, porque siempre hay que tener en cuenta la longitud de los cables.

Debido a esto, se ha pensado que una de las mejoras que se podrían tratar de implementar en un futuro sería proveer al dispositivo Raspberry Pi y módulo MD25 de una batería o un transformador pequeño con una capacidad suficiente para hacer las pruebas, o la conexión de pilas, como en los coches teledirigidos o como en muchos de los dispositivos de uso cuotidiano.

6.2 Compactación

Como opciones de mejora tendríamos la compactación de los módulos, aunque esta opción dependería de la aplicación que se le quisiera dar.

Por un lado, tendríamos la opción de unir los distintos módulos en la Raspberry Pi, utilizando una de las últimas versiones, que tienen más capacidad de cómputo, y conectando el dispositivo a una cámara web inalámbrica para liberar un poco el sistema de cables. De esta manera, la captura del escenario, el procesado de la imagen, la detección de los *fiducial markers*, el cálculo de la ruta y la ejecución de la misma, se harían por parte del Raspberry Pi. Esta sería una buena opción si el dispositivo cuenta

con la capacidad de cómputo suficiente, o no importa tanto el retraso que pueda suponer la lentitud del cálculo, es decir, para uso personal o para mostrar a los alumnos de una asignatura como funciona y que puedan probarlo.

De esta manera, además, eliminaríamos complejidad del sistema, al eliminar elementos en el sistema y se simplificarían algunas partes de los módulos o se podrían llegar a suprimir, como la comunicación.

Por otro lado tendríamos la opción de exportar la mayoría de cálculos a un servidor, conectado a un sistema de cámaras, por tal de abarcar más zona en la que el robot pueda moverse como podría ser en un almacén, y que el Raspberry Pi colocado en el robot solo debería ejecutar la ruta. De esta manera, también se podrían conectar más robots al sistema para una mayor diversidad de aplicaciones. O, usar un dispositivo UAV, pequeños dispositivos aéreos autónomos, con una cámara incorporada para facilitar la captación de la imagen, ya que dicho dispositivo podría seguir a nuestro robot.

7 Posibles aplicaciones

Como hemos comentado en el apartado anterior, este proyecto puede tener varias aplicaciones posibles como podría ser la implantación del sistema en un almacén, estudio de un sistema de *path-finding* de bajo coste, o la implantación dentro de un vehículo autónomo como ayuda para su movimiento.

Al ser dispositivos de relativamente bajo coste, aquellos que necesitaríamos para la construcción del robot, podría ser una buena opción para el estudio de una asignatura de robótica o incluso una en la que se trate la detección de obstáculos o figuras en una imagen. De esta manera, los estudiantes podrían ver una forma relativamente sencilla de unir los módulos, cómo se hace la detección de los *markers*, como se optimiza la imagen para agilizar la detección, cómo se calcula la ruta y como se ejecuta.

Por otro lado, mirando a un sector más comercial, podría implantarse en almacenes o plantas de producción de materiales, para facilitar la recogida y transporte de materiales dentro de las zonas. Esto se podría hacer contando con una "flota" de robots con los módulos que hemos comentado a lo largo del artículo, Raspberry Pi y módulo MD25, encapsulados, que transportarían los materiales una vez calculada la ruta en el servidor que estaría conectado a un sistema de cámaras por tal de tener una imagen de toda la planta o el almacén y poder calcularla a cualquier punto que se requiera.

8 Conclusión

Finalmente, al terminar el trabajo, hemos podido comprobar que se trata de un proyecto versátil, que permite una amplia variedad de aplicaciones, ya que se puede incluir como herramienta de apoyo en otros proyectos de desarrollo, implantacióna nivel empresarial, herramienta de apoyo a nivel educativo, y aquellas aplicaciones que se nos puedan ocurrir.

Dentro del proyecto se han tratado varios módulos y no solo ha sido beneficioso a nivel de autoaprendizaje el entender como funcionan de manera interna, si no la importancia de la buena conexión entre ellos para el proceso del *path-finding*. Dichos módulos pueden ser mejorados y pueden ser más complejos, pero explicados y codificados de la manera en que se ha hecho, se pueden llegar a entender como una introducción al mundo de la robótica y los calculadores de rutas.

Además, este proyecto ha proporcionado también información intrínseca de los diferentes sistemas que cooperan mejor con las distintas librerías que hemos tratado a lo largo del proyecto, la facilidad o complejidad entre unos y otros, y la velocidad en que se ejecutan los cálculos.

Agradecimientos

A Fernando Vilariño del Centro de Visión por Computador de la Universidad Autónoma de Barcelona por su guía en este proyecto y a Onur Ferhat del mismo centro, por su inestimable ayuda a lo largo de todo el proyecto, implicándose en la resolución de los errores surgidos a lo largo del mismo, en el desarrollo de este y por su cantidad de tiempo volcada en proporcionarme cualquier ayuda que necesitara o resolverne cualquier duda que tuviera.

Bibliografia

[1] "Robótica y visión por computador con Raspberry Pi" - Daniel Jimenez Mendoza, 2014

[2] Libreía para fiducial markers, url - https://github.com/mattvenn/fiducial, consultada por última vez en Enero 2016

[3] "Finding optimal rotation and translation between corresponding 3D points", http://nghiaho.com/?page_id=671, consultada por última vez en Enero 2016

[4] "OpenCV-Python Tutorials", https://opencv-python-tutroals.readthedocs.org/en/latest/, consultada por última vez en Enero 2016

[5] "OpenCV documentation index", http://docs.opencv.org/, consultada por última vez en Enero 2016

[6] Siciliano B., Sciavicco L., Villani L. Oriolo G. *Robotics, Modelling, Planning and Control*. Springer

[7] Choset H., Lynch K., Hutchinson S., Kantor G., Burgard W., Kavraki L., Thrun S. *Principles of Robot Motion, Theory, Algorithms, and Implementation*.

[8] Forsyth D., Ponce J. *Computer Vision, a modern approach* (segunda edición). Pearson

APÉNDICE

A1. SECCIÓN DEL APÉNDICE

Imágenes resultantes de los filtros aplicados durante el procesado de la imagen:

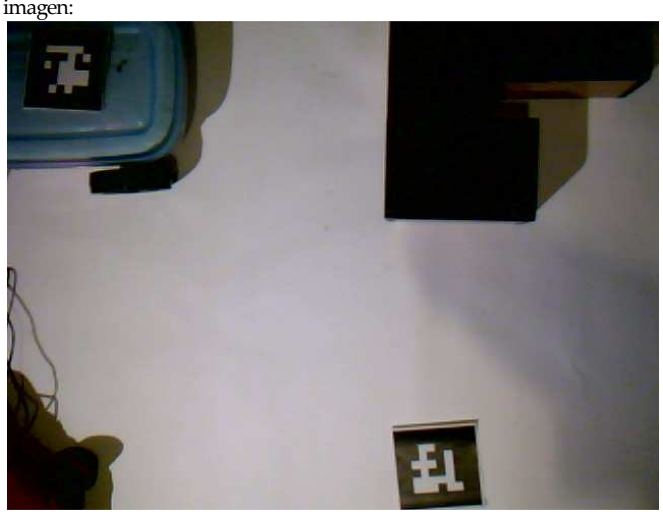


Imagen inicial

Imagen después de una dilatación de los obstáculos para tener en cuenta las dimensiones del robot.

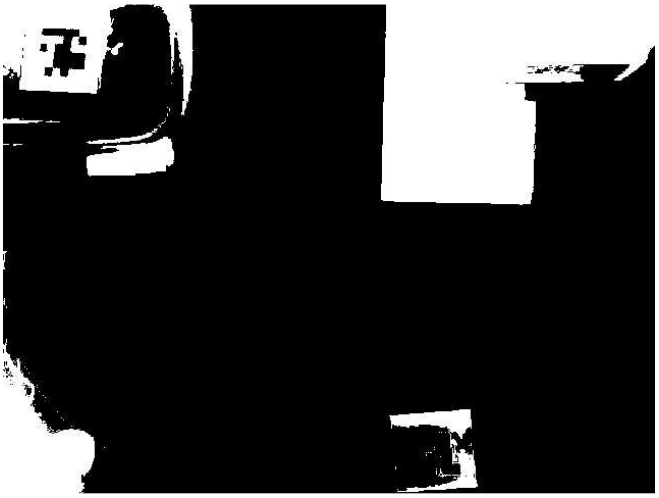


Imagen tras binariación inversa.

Workspace sobre el que se distribuirán los nodos.

Imagen tras filtrado *opening* y eliminación de los *markers* como obstáctulos.

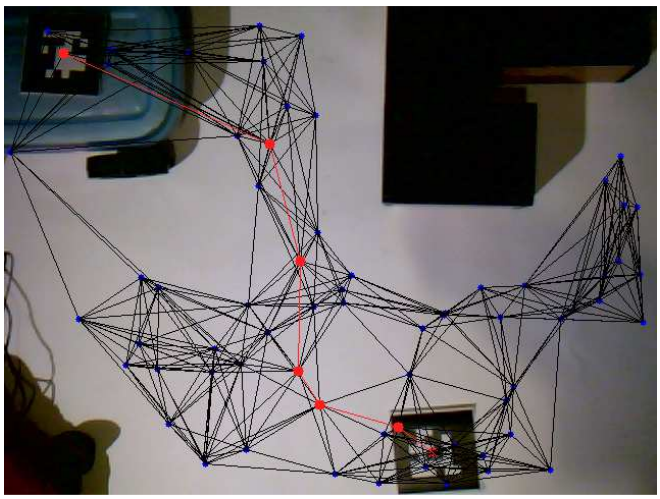


Imagen final del escenario con los nodos y la trayectoria dibujados.