Navegación reactiva de un robot aplicando una estrategia bifásica en una Raspberry Pi 3

M. A. Navarrete-Sánchez[1], **O. Vite-Chávez**[1], **Ro. Olivera-Reyna**[1], **Re. Olivera-Reyna**[1], **S. Mercado-Pérez**[1], **J. U. Muñoz-Minjares**[1] y **A. Marín-Hernández**[2]

[1] *Universidad Autónoma de Zacatecas, Unidad Académica de Ingeniería Eléctrica, Campus Jalpa*
Lib. Jalpa Km. 156+380, Fracc. Solidaridad, Carretera Guadalajara-Saltillo, Jalpa, Zacatecas, México, CP 99601.
mnavarrete@uaz.edu.mx
[2] *Centro de Investigación en Inteligencia Artificial - Universidad Veracruzana*
Sebastián Camacho 5, Zona Centro, Xalapa-Enríquez, Veracruz, C.P. 91000

Resumen

Este trabajo presenta una estrategia bifásica, para la navegación reactiva en tiempo real, de un robot móvil que se desplaza en un ambiente estructurado. En la primera fase, se hace la adquisición de datos a través de un sensor Kinect, para ser procesados por la tarjeta Raspberry Pi 3 y detectar los elementos encontrados en el campo de visión del sensor. La segunda fase trabaja con la información obtenida para tomar las decisiones necesarias y evitar la colisión con los elementos detectados en el entorno. Finalmente, se muestran los resultados de las pruebas experimentales en entornos diferentes.

Palabras clave— Navegación Reactiva, Robot móvil, Estrategia bifásica.

I. Introducción

La robótica es un área de conocimiento que ha avanzado significativamente en los últimos años alcanzado grandes éxitos en la industria manufacturera. Tal es el caso de los brazos robóticos que realizan tareas repetitivas como soldar o manipular componentes eléctricos con alta precisión y a gran velocidad. Una de las principales desventajas de estos robots es la falta de movilidad. La incapacidad de desplazarse de un lugar a otro los limita a estar en una área de trabajo reducida y definida por su ubicación de instalación [1]. En contraste, un robot móvil es capaz de moverse libremente dentro de la planta, lo que le permite tener una mayor flexibilidad y eficacia al realizar una tarea [2].

En la literatura se pueden encontrar diversos trabajos diseñados para la navegación robótica, que utilizan técnicas de procesamiento digital de imágenes en ambientes parcialmente controlados, utilizando un sensor Kinect que incluye una cámara RGB, un sensor de profundidad 3D, un arreglo de micrófonos y motor de inclinación. Y que, en ciertas circunstancias se ha combinado con otros tipos de sensores. Por ejemplo, Chang, Y., $et\ al.$ [3], utilizaron un sensor Kinect para capturar imágenes y el algoritmo Faster R-CNN para un aprendizaje profundo, incorporando el software middleware entre el Sistema Operativo Robótico (ROS, por sus siglas en inglés) y las aplicaciones que se ejecutan internamente, para identificar objetos con un robot móvil utilizando una Raspberry Pi de bajo costo.

Thapa V., $et\ al.$ [4], por su parte evaluaron la efectividad de ROS, utilizando un Kinect como sensor de visión y un sistema Raspberry Pi. Los resultados generados indicaron que dentro del rango típico del sensor Kinect funciona con alta precisión, pero con un retraso mínimo del robot para evitar obstáculos.

Otro trabajo que hace uso del sensor Kinect es presentado por Martínez, C., $et\ al.$ [5], donde desarrollaron un sistema de navegación reactiva difusa que emplea los datos de profundidad del sensor Kinect, algoritmos de visión por computadora y lógica difusa. Con esto generaron ángulos de giro suave para la navegación de un robot móvil realizando pruebas con la plataforma móvil ERA-MOBI, obtuvieron resultados de giros suaves con un porcentaje de evasión de obstáculos del 85.7 %.

Por su parte Ruiz, J., $et\ al.$ [6], describen la integración del sensor Kinect en un robot móvil para mejorar su navegación reactiva, gracias a su capacidad de detección de obstáculos que le permite desplazarse de forma más segura en entornos con obstáculos a diferentes alturas.

Mane, S., $et\ al.$ [7], propusieron un algoritmo de detección

y evasión de obstáculos en tiempo real utilizando una cámara Kinect estereoscópica pasiva. La idea básica detrás del método de detección de obstáculos fue encontrar un mapa de profundidad de la imagen capturada por la cámara Kinect y mapear las coordenadas del mundo real. Fue probado en entornos interiores con una Raspberry Pi 2, el sistema fue simple, robusto y eficiente.

Este trabajo propone una estrategia de dos fases, la primera consiste en sensado y procesamiento; y la segunda fase es de navegación. Para la primera fase del censado se utilizará un sensor único de tipo Kinect y la información se procesará por medio de una Raspberry Pi 3 de bajo costo. En cuanto a la fase de navegación, se implementará la técnica de navegación reactiva.

El artículo está organizado en: Sección II análisis de requerimientos. Sección III estrategia bifásica. Sección IV implementación. En la Sección V pruebas. Los resultados se presentan en la Sección VI y finalmente se presentan las conclusiones en la Sección VII.

II. Análisis de requerimientos

En la actualidad, una de las líneas de la investigación dentro de la robótica móvil que ha tenido mucho auge, es la robótica de servicio [8]. A nivel internacional se han obtenido grandes logros en aspectos como son: robots guías en museos, asistentes de asilos de ancianos, recolectores de datos en ambientes de producción y en actividades que ponen en peligro a los seres humanos. Independientemente del tipo de robot que se utilice, estos realizan actividades ya sean peligrosas o repetitivas para las cuáles su ayuda es muy valiosa. Pero la mayoría de estos robots aún siguen siendo teleoperados [9, 10] o programados para realizar tareas en específico, debido a la complejidad del ambiente al que son expuestos. La navegación de un robot en entornos no controlados es una tarea que tiene una complejidad elevada. Para que el robot se desplace de un punto a otro y parte de sus prioridades es no colisionar con los objetos que pudiera encontrarse en su camino, conlleva a que el robot tenga la capacidad de tomar sus propias decisiones sin tener que consultar a un ser humano.

Diferentes autores describen de manera distinta los esquemas de navegación, pero todos poseen un objetivo común, llevar el vehículo a su destino de forma segura. La capacidad de reacción ante situaciones inesperadas debe ser también considerada, ya que es parte fundamental para el cumplimiento de la tarea en cualquier entorno.

Las etapas involucradas para que un robot móvil realice una correcta navegación son:

- **La percepción del mundo.** Recopilación de información sobre el entorno a través de sus sensores.
- **La planificación de la trayectoria.** Utilización de metodologías para diseñar trayectorias libres de obstáculos para alcanzar el objetivo.
- **El seguimiento del camino.** Control del movimiento para seguir la trayectoria construida por la segunda etapa de forma continua.

La navegación en los robots móviles suele utilizar un mapa, que les permite realizar tareas de forma apropiada. Una de las desventajas de este método es que supone que los objetos en el mapa permanecen estáticos. En la vida real es muy difícil de cumplir, debido a que en los ambientes los objetos no son totalmente estáticos y el robot móvil necesita realizar un proceso de mapeo continuo, lo que genera una pérdida de tiempo para que éste pueda realizar la tarea que le fue asignada. Además de las grandes cantidades de datos que son generados y que se tienen que almacenar para su posterior uso.

II.1. Requerimientos

El reconocimiento por visión tiene tareas como la extracción de información a partir de videos o imágenes, basada en la posición de la cámara, la detección y reconocimiento de objetos. En la propuesta que se plantea en este trabajo, consiste en una estrategia de navegación reactiva basada en la detección de objetos mediante un sensor Kinect sobre una plataforma Raspberry Pi 3. Las pruebas se realizaron dentro de un entorno estructurado y en uno no estructurado, ambos controlados. Se considera entorno estructurado y controlado aquel en el cual los objetos son seleccionados previamente para su detección en un área limitada de m^2 con un borde de 0.01 m de altura. Basándose en la tarea específica y con el conocimiento previo en el área, se establecieron las características de hardware y software:

Hardware

- Raspberry Pi 3: 1GB en RAM, procesador ARMv7 de cuatro núcleos 900 MHz , USB $\times$ 4, video HDMI, Ethernet, alimentación $5\,V$ y $1.5\,A$, slot Micro SD y GPIO 40 PINS, monitor VGA o HDMI, adaptador HDMI$-$VGA (en caso de que el monitor sea VGA), fuente de alimentación de $5\,V$ y $1\,A$ (mínimo).
- Kinect: cámara RGB, captura de imagen a color con una resolución de 640 X 480 pixeles, sensor de profundida con patrones de puntos infrarrojos, 30 Hz, alcance de medición de 0.7 a 4 m.
- Plataforma de robot móvil: cuatro motorreductores de CD, controlador para motores, tarjeta embebida Raspberry Pi 3, batería y sensor de visión Kinect.
  En la Fig. 1 se muestra la plataforma robótica estructurada.

Figura 1: *Robot móvil*

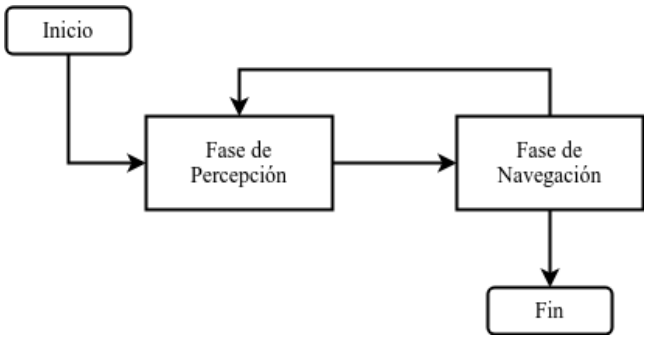


Figura 2: *Diagrama de flujo general de navegación*

Software

- Sistema operativo Raspbian GNU/Linux, librerías OpenCV, para la codificación se utilizó el conjunto de librerías y controladores Freenect (OpenKinect), por la flexibilidad que brinda el sensor Kinect en diversos sistemas operativos como Linux, Windows y OSX. Estas librerías están desarrolladas en lenguaje C, y permite trabajar utilizando otros lenguajes de programación como Python, Java, C# y C++ [5, 6].

III. Estrategia bifásica

Mediante el empleo de la estrategia bifásica se presenta una solución a la problemática de la navegación reactiva, (ver Fig. 2). Con base en un esquema de dos fases, en la primera fase, se hace la adquisición de datos a través del sensor Kinect, para ser procesados por la tarjeta Raspberry Pi 3. La segunda fase trabaja con la información obtenida para tomar las decisiones necesarias y evitar la colisión con los elementos detectados en el entorno.

III.1. Fase de percepción

Esta fase se encarga de percibir el entorno, esto es, obtener las imágenes y procesar la información para detectar los objetos que se encuentran en el entorno de acuerdo al diagrama de flujo que se muestra en la Fig. 3.

- ***Datos de profundidad del Kinect.*** Encargada de recibir los datos de profundidad que envía el sensor. El primer paso es obtener la información que es enviada por el sensor (cámara). Esta información es digitalizada para después continuar con la obtención de las transformaciones morfológicas, finalmente por medio de la función *erode* de OpenCV se obtienen los mapas de profundidad. En esta fase se lleva a cabo el proceso para obtener los mapas de profundidad que permite estimar la distancia de los objetos y poder aplicar las maniobras evasivas ante los objetos. Esta tarea se realiza con el siguiente proceso: primero se trazan los contornos dentro de los mapas de profundidad que se obtuvieron de la fase anterior, una vez trazados los contornos mediante diversas funciones se detectan los objetos y se almacena la información de los elementos detectados. Después de obtener los mapas de profundidad el siguiente paso es detectar las cosas que se encuentran en el entorno.
- ***Información de objetos detectados.*** Una vez detectados los elementos dentro del mapa de profundidad se almacena la información de las cosas detectadas, como la dimensión, distancia, etc.

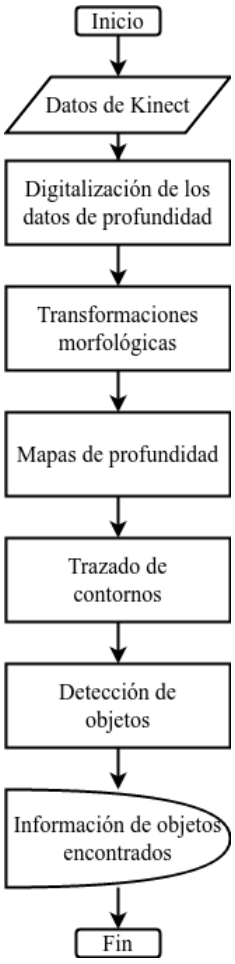


Figura 3: *Diagrama de flujo de la fase de percepción*

III.2. Fase de navegación

Esta fase trabaja con las coordenadas y las distancias de los objetos obtenidas de la fase de percepción; mediante una serie de condiciones, la plataforma robótica tendrá la capacidad de tomar decisiones para realizar movimientos de forma que esquive los objetos evitando colisiones, como se muestra en la Fig. 4.

El proceso para tomar las decisiones comienza a partir de identificar el objeto más cercano considerando la información recibida de la fase de percepción. Después se obtiene la distancia a la cual se encuentra el objeto y ésta información es evaluada bajo las siguientes condiciones:

- Si la distancia es menor a $0.55\,m$, entonces se compara la coordenada x del objeto con el punto medio del eje x de la imagen. Si la coordenada x es mayor al punto

medio, entonces la plataforma girará a la derecha; de lo contrario, girará a la izquierda. El valor $0.55\,m$ de la distancia a comparar está determinada por el límite del rango de distancias del sensor Kinect ($0.4\,m$ - $8\,m$), eligiendo $0.55\,m$ para dar un mayor margen y así evitar la colisión.

- Si la distancia es igual a $10\,m$, con una tolerancia de ($\pm 0.05e$) para los errores de medición y ruido que se pueda presentar en la lectura, entonces la plataforma retrocede y gira hacia la derecha. En este caso la distancia es igual a $10\,m$ debido a que este dato se usa como referencia inicial en la función que encuentra el objeto más cercano.
- Si no se cumple ninguna de las condiciones anteriores entonces la plataforma se mueve hacia adelante.

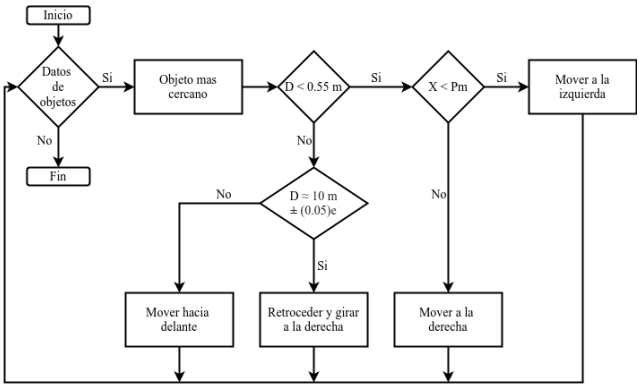


Figura 4: *Diagrama de flujo de la fase de navegación*

IV. Implementación en Raspberry Pi 3

En esta sección se describen los pasos que se siguieron para la implementación de cada una de las fases descritas anteriormente sobre la tarjeta Raspberry Pi 3, con base en la programación de cada uno de los algoritmos aquí descritos.

IV.1. Fase de percepción

Para las pruebas experimentales realizadas se utilizó una cabina y haciendo los ajustes necesarios se aprecia con facilidad el funcionamiento de la fase de percepción. La cabina de prueba se conformó por tres láminas de poliestireno expandido (unicel) de $1\ m^2$ cada una, y dentro de esta se colocó el sensor Kinect con los objetos para realizar las pruebas de percepción tal como se muestra en la Fig. 5.

En la fase de percepción, se adquieren los datos de profundidad del entorno mediante el sensor Kinect. Estos datos son procesados para crear el mapa de profundidad a través del Algoritmo 1. Los datos de profundidad se discretizan usando el método *binning* para después aplicar la función *erode* generando el resultado que se muestra en la Fig. 6.

El mapa de profundidad es procesado con la función que se muestra en el Algoritmo 2. La primera acción detecta los bordes y después los contornos de los objetos dentro del mapa. Posteriormente se evalúa cada contorno y se descartan aquellos

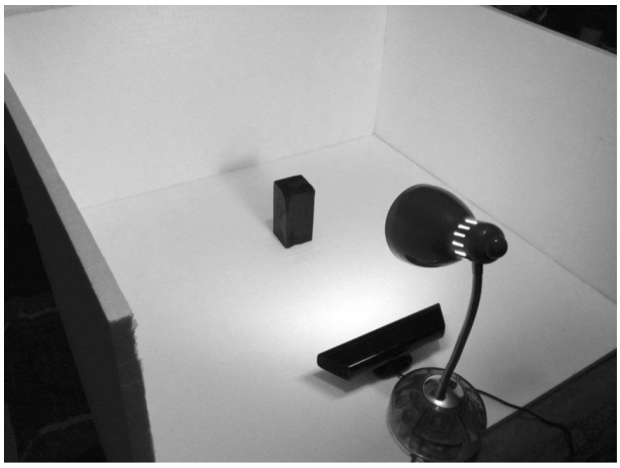


Figura 5: *Cabina de pruebas para la fase de percepción*

Algoritmo 1. Subfase de mapa de profundidad

Entrada: Datos de profundidad desde el sensor.

Salida: Mapa de profundidad inicio.

```
1 inicio
2   # Obtención de datos del sensor
3   datos = Kinect.profundidad
4   # Se discretizan los datos de profundidad mediante el método binning
5    mapa = binning(datos)
6   # Transformaciones morfológicas
7   mapaf = erode(mapa)
8 fin
9 return mapaf
```

Figura 6: *Mapa de profundidad a partir del sensor Kinect*

que no cumplan con las condiciones de evaluación. Los contornos que cumplen con dichas condiciones, serán los objetos que se encuentran dentro del entorno, de los cuales se guardarán los datos tales como las coordenadas (x, y) de su ubicación dentro de la imagen, su dimensión y la distancia con respecto al sensor Kinect.

Algoritmo 2. Subfase de detección de objetos

Entrada: Mapa de profundidad.

Salida: Información de objetos detectados.

```
inicio
    edges = canny(mapa)
    contornos = findContours(edges)
    i = 0
    for obj in contornos do
        x, y, radio = enclosingCircle(obj)
        distancia = mapa[x, y]
        lista[i] = (x, y, radio, distancia)
        i++
        ,
    end
fin
return lista
```

IV.2. Fase de navegación

La implementación se realizó mediante la creación de dos funciones principales. La primera función se encarga de encontrar el objeto más cercano dentro del arreglo que se recibe en la fase de percepción. La función se basa en dos variables donde se guardarán los valores del objeto más cercano. La variable x_c guarda la coordenada x del objeto dentro de la imagen mientras que la segunda variable, d_c, almacena el valor de la distancia del objeto con respecto a la plataforma y se encuentra inicializada con una tolerancia de $\pm 0.05e$ (error aproximado). Si el objeto mas cercano detectado se localiza demasiado lejos de la plataforma, su distancia seguirá siendo menor a 10, debido a que es un valor que esta fuera del rango de distancias del sensor Kinect.

El arreglo de objetos se va recorriendo y comparando con el valor de la distancia de cada objeto y la variable d_c, al ser menor la distancia del objeto se guardan los datos de distancia y la coordenada x. El sensor dejará de detectar objetos cuando esté demasiado cerca a una pared o un objeto, en este caso la función regresará los valores iniciales de las variables xc y d_c, tal como se muestra en el Algoritmo 3.

Algoritmo 3. Función *objetoCercano()*

Entrada: Información de objetos detectados.

Salida: Coordenada x y distancia del objeto cercano inicio.

```
inicio
    xc = 0
    dc = 10
    for obj in contornos do
        if ob.distancia <= dc then
            dc = ob.distancia
            xc = ob.x
        end
    end
fin
return xc, dc
```

La segunda función muestra la integración de la fase de navegación descrito en el Algoritmo 4. Esta función tiene como tarea la toma de decisiones para que la plataforma no colisione con los objetos detectados. Primero se encuentran los datos del objeto más cercano y posteriormente el dato de distancia se evalúa con las condiciones definidas dentro de la función.

Algoritmo 4. Fase de navegación reactiva

```
inicio
    xc, dc = objetoCercano()
    if dc <= 0.55 then
        if dc <= pm then
            girar a la izquierda
        end
        else
            girar a la derecha
        end
    end
    else if dc == 10 then
        retroceder
        girar a la derecha
    end
    else
        avanzar
    end
fin
```

V. Pruebas

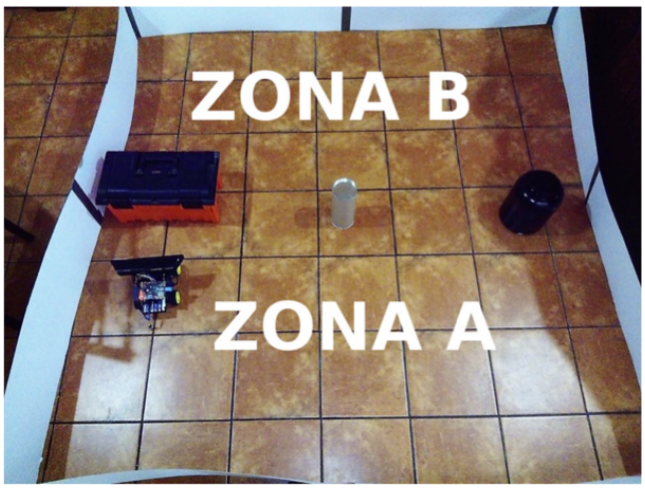
Una vez concluidas las etapas de desarrollo y programación de la estrategia en la plataforma, se procedió a realizar las pruebas experimentales dentro de dos entornos diferentes. En el primer entorno se considera un ambiente estructurado, este consta de un cuadrado de $2\,m^2$ con una altura de $0.5\,m$, tal como se muestra en Fig. 7a. Se consideraron tres objetos de diferentes tamaños como obstáculos, lo cual facilitó que se identificaran rápidamente los objetos dentro del rango de visión del sensor. En el segundo entorno, Fig. 7b, se considera un escenario no estructurado (no representa restricción de espacio ni de objetos) colocados de forma aleatoria.

Identificación de:

Zonas: Los objetos dentro del entorno se distribuyeron de manera que se identifican dos zonas, tal como se observa en la Fig. 7a.

Objetos: Se realizó con una distribución de objetos como se muestra en la Fig. 7a. Durante esta prueba se evaluaron 170 fotogramas. La distribución de los objetos dentro del entorno permitió que la plataforma pudiera moverse con mayor libertad, en relación a la distribución de la primera prueba. Al igual que en el caso anterior, no se presentaron colisiones y el rango de los objetos que se identificaron fue el mismo, de 0 a 7 unidades.

Entorno no estructurado: La tercera prueba se realizó en un entorno no estructurado, se utilizó la distribución de objetos que se muestra en la Fig. 7b. Una posición inicial



(a) *Entorno estructurado con objetos e identificación de zonas*

(b) *Entorno no estructurado con objetos aleatorios*

Figura 7: *Diseño del ambiente para pruebas experimentales*

de la plataforma junto de un objeto. Para esta prueba se evaluaron 200 fotogramas.

VI. Resultados

Se presentan los resultados arrojados en relación al número de objetos y el tiempo de reacción. A partir de las tres pruebas realizadas, las primeras dos se llevaron a cabo dentro del entorno estructurado y la tercera en ambiente no estructurado.

Identificación de:

Zonas: La gráfica de la Fig. 8a, muestra los objetos identificados (línea azul) y el tiempo de reacción (línea roja) en el transcurso de las pruebas, donde el dispositivo móvil llevó a cabo la captura de 150 imágenes. El rango de los objetos que este identificó van desde 0 a 7 unidades y el tiempo que tardó el proceso en enviar la instrucción de control fue de $0.108\,s$ a los $0.794\,s$. Como se puede observar en las gráficas, el tiempo se incrementa en relación al número de objetos detectados; así también se puede observar que los picos más altos se dan cuando el número de objetos detectados igual a cero. Esto es debido al comportamiento mismo de la función *findContours()* que deberá de evaluar cada contorno para descartar aquellos que no cumplen con las condiciones de evaluación, existan o no obstáculos, aumentando tiempo de procesamiento.

Objetos: Los resultados son en base al tiempo de reacción de $0.109\,s$ a los $0.293\,s$, tal como se muestra en la gráfica de la Fig. 8b.

Entorno no estructurado: Los resultados se pueden observar a pesar de encontrarse cerca de un objeto al inicio, realiza las maniobras para evitar el contacto con el objeto, pero no evita sufrir colisiones y específicamente en las capturas 1 y 2, posteriormente en las capturas 125 a la 135, debido a la posición de la plataforma con respecto al objeto donde el sensor no pudo percibir al objeto. El rango de objetos que se identificaron en este caso fue de 0 a 8 unidades y el tiempo de reacción estuvo en el rango de los $0.109\,s$ a $0.794\,s$, (ver Fig. 8c).

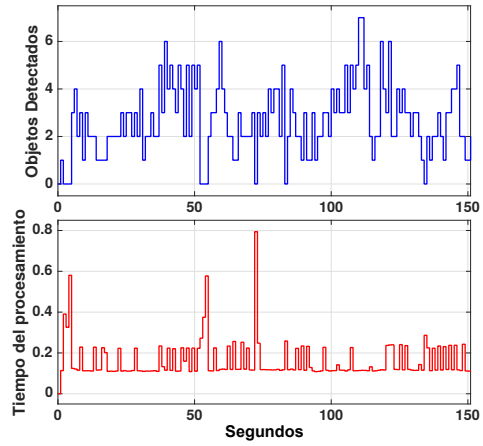
VII. Conclusiones

La integración entre un sensor de visión como lo es el Kinect y una tarjeta la Raspberry Pi 3 ambos de bajo costo, es posible. Se menciona en los trabajos relacionados, el uso del sensor Kinect, diferentes plataformas para el procesado como el sistema operativo robótico ROS, o plataformas robóticas complejas y algoritmos robustos para la navegación reactiva, con la necesidad de almacenar previamente mapas del entorno. A pesar de las ventajas que presentan estos trabajos, destacamos en el nuestro la no creación de un mapa para la navegación. La navegación reactiva que utilizamos es sencilla y cumple con el objetivo de este trabajo, permite evitar posibles colisiones con obstáculos. Se utiliza como único sensor de visión el dispositivo Kinect y el procesado de la información es llevado a cabo por la tarjeta Raspberry Pi 3.

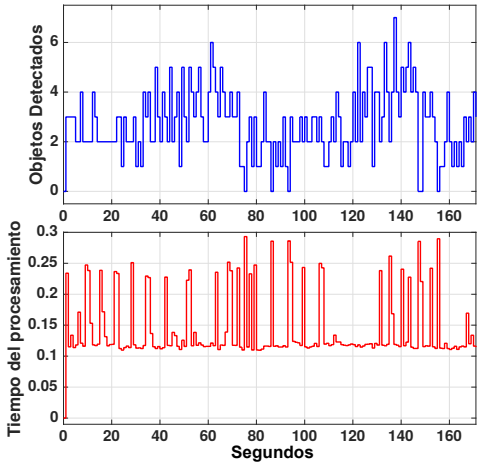
A pesar de la simplicidad de la estrategia de dos fases, proporciona resultados satisfactorios. Se obtuvo un 0 % de colisiones dentro de un entorno controlado y un 5.5 % en un entorno no controlado. Estas fueron dadas por las limitantes del sensor debido al rango de visibilidad y las distancias con las que trabaja el Kinect las cuales son $0.5m$ de valor mínimo y $3m$ de valor máximo, impidiendo que detecte objetos fuera de este rango de distancias. Cabe destacar que debido a la forma en que trabaja el Kinect para obtener el mapa de profundidad, el sensor puede detectar objetos en ambientes con oscuridad total por lo cual se descartó el uso de una iluminación ideal.

Se presentó un retardo de tiempo entre el proceso de detección de objetos y el proceso de localización que solo perjudica al tiempo de reacción y acción de movimiento del robot, ya que requiere de tiempo para determinar la dirección a seguir. Debido a que no se consideró el tiempo de respuesta del proceso de detección de objetos, que era mayor que el del proceso de localización, esto provocó que el sistema tenga que esperar que la tarea de búsqueda de objetos finalice.

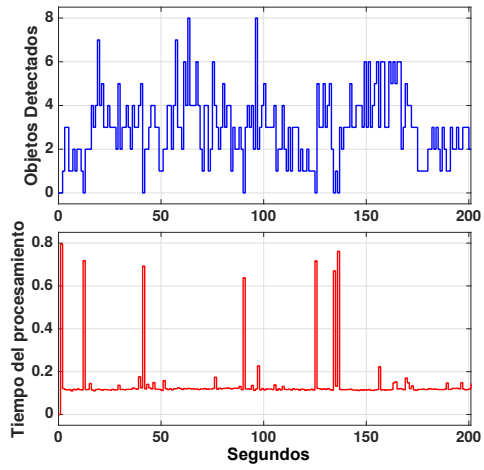
Estos primeros resultados dan la pauta para continuar con nuestra investigación. Como trabajos futuros se busca una na-



(a) *En la identificación de zonas*



(b) *Por la identificación de objetos*



(c) *En el entorno no estructurado*

Figura 8: *Tiempo de procesamiento por objetos detectados*

vegación libre de colisiones de cualquier robot móvil. Además, se pretende reducir los tiempos de procesamiento y el almacenamiento de información de manera significativa.

Referencias

[1] Roland Siegwart, Illah Reza Nourbakhsh y Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[2] Aníbal Ollero Baturone. *Robótica: manipuladores y robots móviles*. Marcombo, 2005.

[3] Yeong-Hwa Chang, Ping-Lun Chung y Hung-Wei Lin. «Deep learning for object identification in ROS-based mobile robots». En: *2018 IEEE International Conference on Applied System Invention (ICASI)*. IEEE. 2018, págs. 66-69.

[4] Vikas Thapa y col. «Obstacle avoidance for mobile robot using rgb-d camera». En: *2017 International Conference on Intelligent Sustainable Systems (ICISS)*. IEEE. 2017, págs. 1082-1087.

[5] Claudia Cruz Martínez y col. «Sistema de Navegación Reactiva Difusa para Giros Suaves de Plataformas Móviles Empleando el Kinect». En: *ReCIBE. Revista electrónica de Computación, Informática, Biomédica y Electrónica* 1.1 (2016).

[6] JR Ruiz-Sarmiento y col. «Navegación reactiva de un robot móvil usando kinect». En: *Actas ROBOT 2011* (2011).

[7] Sunil B Mane y Sharan Vhanale. «Real time obstacle detection for mobile robot navigation using stereo vision». En: *2016 International Conference on Computing, Analytics and Security Trends (CAST)*. IEEE. 2016, págs. 637-642.

[8] Ifr. *Service Robots*. URL: `https://ifr.org/service-robots/`.

[9] Riyad A El-laithy, Jidong Huang y Michael Yeh. «Study on the use of Microsoft Kinect for robotics applications». En: *Proceedings of the 2012 IEEE/ION Position, Location and Navigation Symposium*. IEEE. 2012, págs. 1280-1288.

[10] Miguel Angel García-Sánchez. «Diseño de estructuras para vehículos aéreos no tripulados». En: *Difu100ci@ Revista en Ingeniería y Tecnología, UAZ* 10.1 (2016).