DESDE 2013
https://repository.uaeh.edu.mx/revistas/index.php/icbi/issue/archive
Pädi Boletín Científico de Ciencias Básicas e Ingenierías del ICBI

ISSN: 2007-6363

Publicación Semestral Pädi Vol. 10, No. Especial 4 (2022) 100-109

Robot móvil autónomo con reconocimiento y navegación hacia botellas de plástico
Autonomous mobile robot with recognition and navigation towards plastic bottles

D. Vázquez-Lucero [a], E. Luna-Taylor [a,*], I. Santillán [a], C. Higuera [a]

[a] *Tecnológico Nacional de México Campus La Paz, 23080, La Paz, Baja California Sur, México.*

Resumen

En este trabajo, se propone el diseño y construcción de un prototipo de robot autónomo para el reconocimiento de botellas de plástico, aplicando una estrategia de segmentación semántica, a través de redes neuronales convolucionales. Así mismo, se propone el diseño de un sistema de navegación basado en lógica difusa, para lograr que el robot se aproxime a las botellas identificadas. Los experimentos se realizaron en un ambiente controlado, alcanzando un 96.6% de precisión en las pruebas de reconocimiento, mientras que en las pruebas de navegación, el robot alcanzó en todos los casos la posición de las botellas identificadas.

Palabras Clave: Medio ambiente, Robot móvil autónomo, Visión artificial, Red convolucional, Lógica difusa.

Abstract

In this work, we propose the design and construction of an autonomous robot prototype for the recognition of plastic bottles, applying a semantic segmentation strategy, through convolutional neural networks. Also, we propose the design of a navigation system based on fuzzy logic, to ensure that the robot approaches the identified bottles. The experiments were carried out in a controlled environment, reaching 96.6% accuracy in the recognition tests, while in the navigation tests, the robot reached the position of the identified bottles in all cases.

Keywords: Environment, Autonomous mobile robot, Artificial vision, Convolutional neural network, Fuzzy logic.

1. Introducción

La contaminación degrada y destruye hábitats de playa únicos que son utilizados por animales y plantas. Así mismo, limita la posibilidad de utilizar las playas con fines económicos, recreativos y estéticos. El plástico en el entorno acuático genera una preocupación cada vez mayor debido a su persistencia y las consecuencias que tiene en el medioambiente, la vida silvestre y la salud humana (APA, 2022).

Con el objetivo de aportar soluciones a esta problemática, en este trabajo se propone el diseño y construcción de un robot móvil autónomo, entrenado a través de redes neuronales convolucionales y lógica difusa, para identificar botellas de plástico y desplazarse hacia estas.

Para evaluar la estrategia de reconocimiento y navegación propuesta, se eligió trabajar en un ambiente controlado, dentro de un edificio académico, con diferentes botellas de plástico y la presencia de otros objetos.

Diferente de otras propuestas recientes con objetivos similares (Kulshreshtha, 2021), (De La Torre *et al*., 2020), (Ramírez *et al*., 2020), (Bai, 2018), donde el reconocimiento se realiza al nivel de áreas rectangulares que encierran a los objetos, en este proyecto se aplica *Segmentación Semántica* (Ronneberger *et al*., 2015), para reconocer los objetos al nivel de pixeles. Esto permite delimitar el contorno de las botellas, y estimar su área y centroide de forma más precisa. Con estos datos, se aplica posteriormente un algoritmo basado en lógica difusa para aproximar el robot a las botellas identificadas.

Como muestran los resultados de los experimentos realizados, con la estrategia propuesta se logra un alto porcentaje de reconocimiento de las botellas, y en todas las pruebas realizadas, el robot se desplaza hasta la posición de las botellas identificadas. Una característica a destacar de esta propuesta, es que el sistema resulta sencillo de implementar, ya que únicamente requiere de las imágenes capturadas por la cámara para guiar la navegación del robot. Esto es, no requiere de cálculos Odométricos, o información extra de otros **dispositivos como Encoders Rotativos (Segura et al., 2019),** Bluetooth

*Autor para la correspondencia: jorge.lt@lapaz.tecnm.mx
Correo electrónico: M21310005@lapaz.tecnm.mx (Diego Vázquez-Lucero), jorge.lt@lapaz.tecnm.mx (Jorge Enrique Luna-Taylor), israel.sm@lapaz.tecnm.mx (Israel Marcos Santillán-Méndez), cesar.hv@lapaz.tecnm.mx (Cesar Higuera-Verdugo).

Historial del manuscrito: recibido el 16/05/2022, última versión-revisada recibida el 11/08/2022, aceptado el 29/08/2022, publicado el 05/10/2022.
DOI: https://doi.org/10.29057/icbi.v10iEspecial4.9268

(AlenziI *et al.*, 2019), (Pawar *et al.*, 2019), o comunicación vía Wi-Fi (Mangayarkarsi *et al.*, 2020), (Jha *et al.*, 2019), para guiar el robot hasta su objetivo.

1.1. Antecedentes

Los robots móviles se utilizan en una gran variedad de aplicaciones, por ejemplo, en vigilancia, exploración, operaciones de búsqueda y rescate, limpieza, automatización industrial, entre otras. Para esto, deben contar con algoritmos especializados que les permitan navegar a través de su entorno. La navegación de un robot incluye todas las acciones que llevan a que este se desplace desde su posición actual hasta una posición de destino (López *et al*., 2020).

Los constantes avances en la tecnología de los microprocesadores, sensores, cámaras, entre otros componentes, y el desarrollo de nuevas técnicas de inteligencia artificial, como las redes neuronales artificiales y la lógica difusa, hacen de la robótica móvil uno de los campos con mayor crecimiento (Abeliuk and Gutiérrez, 2021). En (Guarnizo *et al*., 2021) presentan un perfil histórico de algoritmos aplicados a problemas de control, así como de las principales aplicaciones de la robótica móvil.

Actualmente, las redes neuronales artificiales son utilizadas en el control de sistemas con dinámicas complejas, especialmente en sistemas no lineales que varían en el tiempo y que resultan complejos de regular con métodos convencionales (Valverde and Gachet, 2007). En el campo de la robótica, una aplicación de las redes neuronales es en tareas de percepción a través de visión artificial. En este campo, se utilizan para la detección y clasificación de los objetos presentes en el entorno del robot (Andreu-Perez *et al.*, 2017).

Por otro lado, la lógica difusa se utiliza en sistemas donde los datos y sus relaciones no pueden describirse en términos matemáticos precisos. En este tipo de sistemas, se aplica la lógica difusa pretendiendo emular la estrategia de control que seguiría un experto humano en el control manual de un proceso (Santos, 2011). Dentro de la robótica móvil, una aplicación de la lógica difusa es en tareas de navegación. Por ejemplo, en (Segura *et al*., 2019) proponen la implementación de un sistema de control difuso tipo Mamdani, para la navegación y evasión de obstáculos, sobre una plataforma con recursos computacionales limitados. En (López *et al*., 2020) presentan una arquitectura basada en control difuso para la selección de acciones de navegación, con el objetivo de que un robot móvil tome decisiones, con base en el nivel de carga de la batería.

La idea de diseñar y construir robots móviles para la recolección de basura no es nueva. Por ejemplo, en (De La Torre *et al*., 2020) diseñaron y construyeron un robot móvil recolector y compactador de botellas de plástico en playas con arena fina, aplicando una red neuronal tipo MobileNet. En (Ramírez *et al*., 2020) diseñaron y desarrollaron un robot recolector de basura utilizando Redes Neuronales Convolucionales con arquitectura SDD y Faster R-CNN.

En (Bai *et al*., 2018) presentan el diseño de un robot recolector de basura que opera sobre césped. El robot utiliza un sensor inercial, un sistema odométrico y un receptor GPS para producir información de ubicación. Para la identificación y evasión de obstáculos utiliza un sensor ultrasónico, así como una cámara web para detectar y reconocer la basura, a través de una Red Neuronal Convolucional.

En (Kulshreshtha *et al*., 2021) presentan el diseño de un robot recolector de basura. Los sensores que incluyen son una cámara, un sensor ultrasónico y un módulo GPS. Probaron tres modelos de red neuronal para la detección de objetos (Mask-RCNN, YOLOv4 y YOLOv4-tiny).

En el presente artículo, se propone una estrategia que combina Segmentación Semántica con Lógica Difusa, para el reconocimiento de botellas de plástico y aproximación de un robot hacia estas, utilizando únicamente información de una cámara de video. Esta estrategia no se encuentra reportada en algún otro trabajo de la literatura revisada.

2. Desarrollo

En este proyecto, se diseñó y construyó un prototipo de robot móvil, entrenado a través de redes neuronales convolucionales, para reconocer cuatro tipos de botellas de plástico, y desplazarse hacia a estas, aplicando un sistema de navegación basado en lógica difusa. Independientemente del tipo de botella, el sistema les asigna la misma clasificación. Es decir, no hace distinción en el proceso de entrenamiento de la red, ni durante el reconocimiento y navegación hacia estas. A continuación, se presentan las etapas de desarrollo de este trabajo.

2.1. Diseño del robot

Como primera etapa, se diseñó y construyó un prototipo de robot móvil (ver Figura 1), dotándolo de los componentes de hardware necesarios, para trabajar de forma autónoma. En la Tabla 1 se muestran los componentes de hardware del robot.

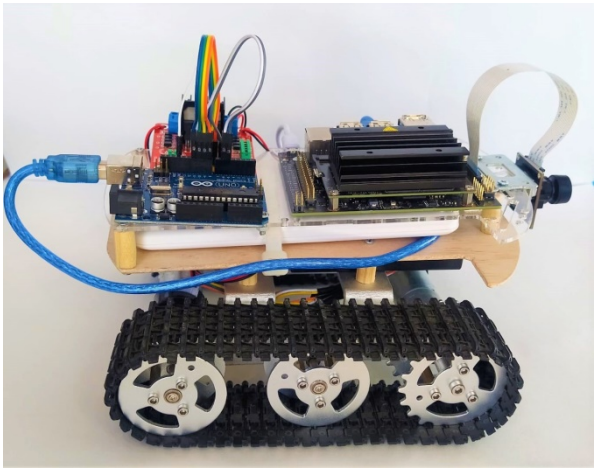

Figura 1: Prototipo de robot móvil.

Tabla 1. Componentes de hardware del robot.

Chasis con llantas sobre Orugas
Dos motorreductores Dc de 9-12 V
NVIDIA Jetson Nano B01
Módulo de cámara IMX219
Arduino Uno R3
Puente H L298
Banco de energía 12000 mah
Dos pilas de iones de litio de 3.7 V.

Entre sus principales componentes, el robot incluye una tarjeta Jetson Nano, cuya función es recibir las imágenes de la

cámara, sobre las cuales aplica un modelo entrenado de red neuronal convolucional, para reconocer las botellas presentes. Posteriormente, identifica los contornos de las botellas, calcula el área de cada una, y obtiene el centroide de la botella cuyo contorno tiene mayor área. Finalmente, la Jetson Nano transmite el área y el centroide a una tarjeta Arduino Uno, vía protocolo Serial.

La tarjeta Arduino recibe el área y centroide, y aplica un sistema de inferencia difusa, para calcular la potencia que debe imprimir a cada motor, con el objetivo de que el robot se aproxime a la botella reconocida con mayor área. En la Figura 2 se muestra un diagrama de los componentes de hardware del robot.

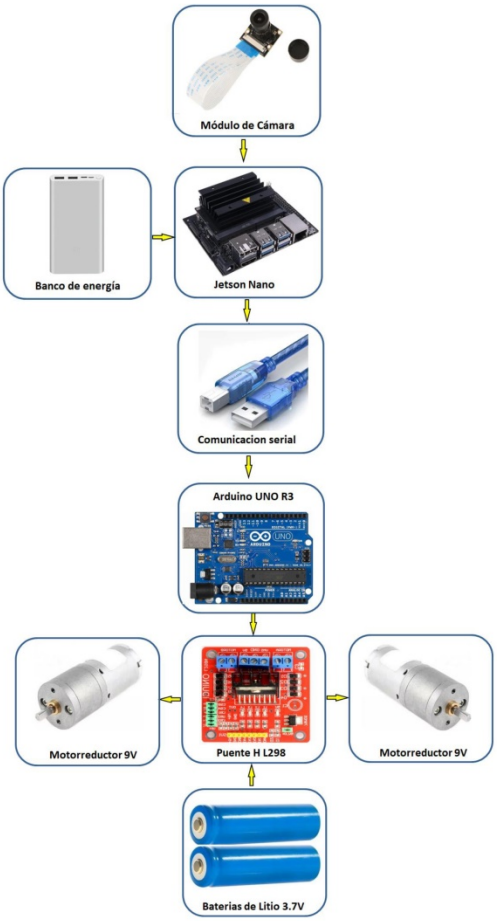

Figura 2: Diagrama de componentes de hardware.

2.2. Estrategia de Reconocimiento

El reconocimiento de las botellas se implementó a través de *Segmentación Semántica*, aplicando una red neuronal tipo U-Net (Ronneberger *et al*., 2015). Para esto, se tomó como base el modelo ResNet18 con los pesos pre-entrenados. A este modelo base, se le eliminaron las dos últimas capas (FC y Softmax), y se le agregaron cuatro capas Convolucionales Transpuestas, combinadas con cuatro nuevas capas Convolucionales. En la Figura 3 se muestra la arquitectura de la red utilizada. La codificación para definir la arquitectura y el entrenamiento de la red neuronal, se basó en el trabajo presentado en (Sensio. J., 2020).

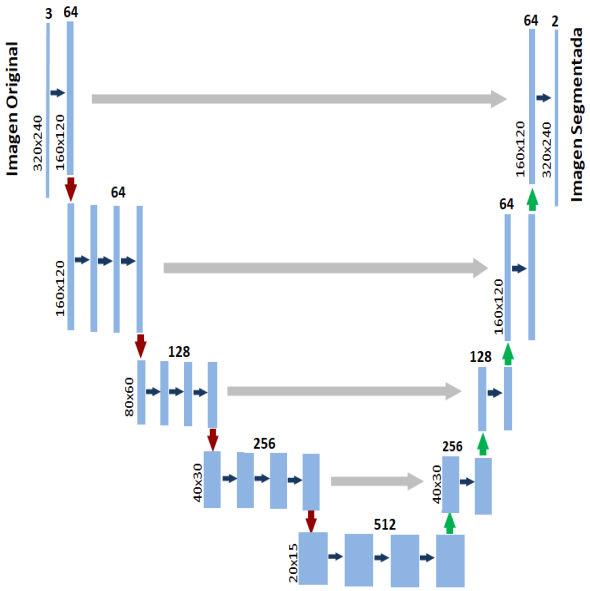

Figura 3: Arquitectura U-Net utilizada.

2.2.1. Entrenamiento de la red

Para el entrenamiento y pruebas de reconocimiento de la red, se capturaron 216 imágenes utilizando cuatro tipos diferentes de botellas de plástico, en un ambiente controlado. Las imágenes se almacenaron con un tamaño de 320x240 pixeles, en formato JPG. Para cada imagen, se realizó manualmente la segmentación de las botellas que aparecen en la misma, utilizando el software *VGG Image Annotator (VIA) versión 2.0.11*[1]. Posteriormente, se generó un archivo en formato JSON, con la definición de los polígonos que corresponden a los contornos de las botellas segmentadas.

A partir de estos polígonos, se aplicó una rutina escrita en Python, para generar archivos en formato PNG, con las imágenes segmentadas, nombradas como máscaras (ver Algoritmo 1). En estos archivos, el color de cada pixel corresponde a la clase de objeto al que pertenece. Se eligió el color rojo (128,0,0 en formato RGB), para las botellas, y color negro (0,0,0), para el resto de los objetos, considerados como fondo. En la Figura 4 se muestra un ejemplo de una imagen y su máscara correspondiente.

Algoritmo 1. Generación de archivos en formato PNG con las imágenes segmentadas.

ruta = './DataSet/Imagenes/'
arhivo_json = abre archivo *ruta*+'poligonos.json'
datos_json = carga *archivo_json*
imgs = toma información de las imágenes de *datos_json*
para cada *imgId* **en** *imgs*:
  nombre_img = lee nombre de la imagen *imgId*
  regiones = lee lista de las regiones en *imgId*
  imagen = abre y lee imagen *ruta*+*nombre_img*

[1] https://www.robots.ox.ac.uk/~vgg/software/via/

alto = número de filas de *imagen*
ancho = número de columnas de *imagen*
máscara = crea matriz de $(alto \times ancho \times 3)$
para *i* **en** el rango de 0 a número de *regiones*-1**:**
  region = lee la región *i* de *regiones*
  clase = lee la clase asociada a *region*
  si *clase* es 'botella'**:** *R,V,A*=128,0,0
  si no: *R,V,A*=0,0,0
  si el tipo de *region* es 'polygon'**:**
    total_puntos = lee cantidad de puntos de *region*
    contorno = crea matriz de *total_puntos* elementos
    para *i* **en** el rango de 0 a *total_puntos*-1**:**
      x = lee la coordenada *x* del punto *i*
      y = lee la coordenada *y* del punto *i*
      contorno[i] = crea punto (*x, y*)
    para *i* **en** el rango de 0 a *ancho*-1**:**
      para *j* **en** el rango de 0 a *alto*-1**:**
        si punto (*i, j*) se encuentra en el área de *contorno***:**
          máscara[*j*, *i*, 0] = *A*
          máscara[*j*, *i*, 1] = *V*
          máscara[*j*, *i*, 2] = *R*
  guarda *máscara* como archivo PNG en *ruta*

Figura 4: Imagen original y máscara correspondiente.

Posteriormente, las intensidades de color de las imágenes originales se normalizaron con valores entre 0 y 1. A las máscaras se les sustituyó los tres canales de color de cada pixel, por un valor único que representa la clase de objeto a la que pertenece. Finalmente, las imágenes, tanto las originales como las máscaras, se almacenaron en formato Numpy (ver Algoritmo 2).

Algoritmo 2. Asignación de clase de objeto por pixel

ruta = './DataSet/Imagenes/'
archivos = lee lista de archivos de *ruta*
define *mapa_color_etiqueta*()**:**
  mapa_color = [[0, 0, 0],[128, 0, 0]]
  // en *mapa_color* se definen los colores de todas las clases
  // de objetos que se requiera clasificar
  color_etiqueta = crea tensor de 256^3 elementos
  para *i*, *mapa* **en** enumeración *mapa_color***:**

color_etiqueta[(*mapa*[0]*256+*mapa*[1])*256+*mapa*[2]]=*i*
  retorna *color_etiqueta*
para *archivo* **en** *archivos***:**
  imagen = abre y carga imagen *ruta*+*archivo*+'.jpg'
  imagen = convierte formato BGR a RGB de *imagen*
  imagen = *imagen*/255.0
  guarda *imagen* en *ruta* con formato Numpy
color_etiqueta = llama a *mapa_color_etiqueta*()
para *archivo* in *archivos***:**
  máscara = abre y carga imagen *ruta*+*archivo*+'.png'
  máscara = convierte formato BGR a RGB de *máscara*
  idx = ((*máscara*[:, :, 0] * 256 + *máscara* [:, :, 1]) * 256 + *máscara* [:, :, 2])
  máscara = *color_etiqueta*[*idx*]
  // esta es una operación de numpy a nivel de matrices,
  // unidimensional (*color_etiqueta*) y bidimensional (*idx*),
  // resultado la matriz bidimensional con la clase por pixel
  guarda *máscara* en *ruta* con formato Numpy

Del total de imágenes, se utilizó el 75% para entrenamiento y el 25% para pruebas de reconocimiento. En la Tabla 2 se muestran los parámetros utilizados para el entrenamiento de la red.

Tabla 2: Parámetros de entrenamiento de la red

Función de optimización	Adam
Función de pérdida	BCEWithLogitsLoos
Factor de aprendizaje (lr)	0.0001
Factor de decaimiento (wd)	0.001
Tamaño de lote	10
Número de épocas	100

La implementación de la red y su entrenamiento, se realizó en lenguaje Python versión 3.9.7, utilizando las librerías Pytorch versión 1.9.0, Torchvisión versión 0.10.0 y OpenCV 4.5.5, sobre el sistema operativo Ubuntu 20.04. En la Tabla 3 se muestra las características del equipo donde se ejecutó el entrenamiento.

Tabla 3: Características del equipo de cómputo para entrenamiento

Tarjeta madre	Gigabyte H81M-H
Procesador	Intel Core i7 3.6Ghz x8
Memoria RAM	16 GB
Disco duro	SSD 240 GB
GPU	NVIDIA GeForce GTX Titán X

El tiempo de ejecución del entrenamiento fue de seis segundos por época, dando un total de 10 minutos por las 100 épocas. En la Figura 5 se muestra el comportamiento de la función de pérdida durante el entrenamiento, aplicada sobre las muestras de entrenamiento (línea azul), y sobre las muestras de prueba (línea verde). Al final de las 100 épocas, la función de pérdida para las muestras de entrenamiento fue de 0.026 y para las muestras de pruebas de 0.050.

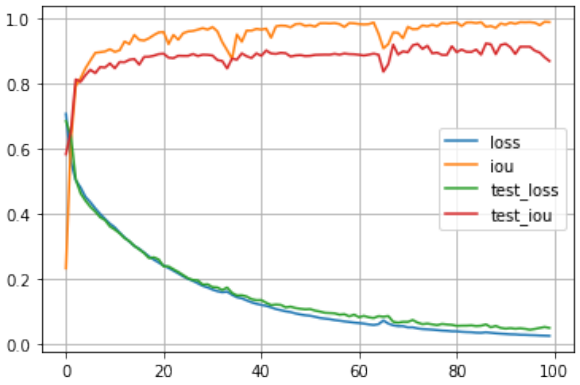

Figura 5. Comportamiento de la función de pérdida durante el entrenamiento.

2.2.2. *Reconocimiento de las botellas*

Una vez generado el modelo de la red entrenada, se almacenó en un archivo con formato Pytorch (.pt), y se cargó sobre la Jetson Nano. El proceso de reconocimiento de las botellas sobre la Jetson, inicia con la captura de una imagen (*frame*) a través de la cámara, para pasarla al sistema de inferencia de la red. A partir de la salida de la red, se obtiene la máscara inferida, correspondiente a la imagen de entrada. Posteriormente, a partir de la máscara inferida, se generan los contornos de las botellas reconocidas, a través de la librería OpenCV. Después se obtiene el área de cada contorno, y se calcula el centroide del contorno con área mayor (ver Algoritmo 3).

Esto último, considera que la botella reconocida con área mayor, es la que, posiblemente, se encuentra más próxima al robot, y por lo tanto se convierte en el objetivo de alcance del mismo. El tiempo requerido para completar este proceso por la Jetson, es de aproximadamente 0.35 segundos, por lo que el robot es capaz de procesar tres imágenes por segundo, lo cual resultó suficiente para lograr un desplazamiento fluido hacia las botellas.

Algoritmo 3. Sistema de reconocimiento sobre la Jetson Nano

```
define clase HiloCámara:
    define constructor(cámara):
        atributo cámara = parámetro cámara
        atributo última_imagen = Nulo
        atributo activo = Verdadero
    define ejecución:
        mientras activo sea Verdadero:
            última_imagen = lee frame del buffer de la cámara
            pausa un milisegundo
puerto_serial = crea y configura objeto de puerto serial
cámara = crea y configura objeto cámara
hiloCámara = crea objeto HiloCámara(cámara)
inicia ejecución de hiloCámara
modelo = carga el modelo entrenado de la red
pasa el modelo a la GPU
activa el modo de inferencia del modelo
repite:
  si hiloCámara.última_imagen no es Nulo:
    imagen = hiloCámara. última_imagen
    imagen = convierte formato BGR a RGB de imagen
    imagen_torch = convierte imagen a formato Torch
    imagen_torch = permuta la tercera dimensión de
                   imagen_torch como primera
dimensión
    // el modelo requiere que los canales de color sean la
    // primera dimensión.
    imagen_torch = imagen_torch.float()/255  //se
normaliza
    imagen_torch = agrega una dimensión a imagen_torch
    // se agrega una dimensión porque el modelo procesa
por
    // lote de imágenes (el lote lleva una sola imagen).
    imagen_cuda = se transfiere imagen_torch a la GPU
    salida = evaluación de imagen_cuda con el modelo, y
se
             se extrae del lote la única imagen procesada.
    máscara = genera matriz con la clase inferida por pixel
              con la función torch.argmax(máscara,
axis=0)
    máscara = pasa máscara a la CPU en formato Numpy
    máscara = se filtra la clase de interés (botellas)
    contornos = genera contorno de las botellas en
máscara
                con findContours de OpenCV
    áreaMayor = 0
    contornoMayor = Nulo
    para contorno en contornos:
      área = calcula el área de contorno con contourArea
             de OpenCV
      si área > áreaMayor:
        áreaMayor = área
        contornoMayor  =  contorno
    si áreaMayor > 0:
      momentos = calcula los momentos de contornoMayor
                 con moments de OpenCV
      si se generaron momentos:  // momentos0,0 > 0
        centroideX = calcula centroide de contornoMayor
                     // momentos1,0 / momentos0,0
        cadena ="<area>"+str(áreaMayor)+
                "<cx>"+str(centroideX)+"<fin>\n"
        envía cadena a la placa Arduino por puerto_serial
mientras no se pulsa tecla <ESC>
hiloCámara.activo = Falso
sincroniza finalización con hiloCámara
libera recursos de cámara y puerto_serial
```

En la Figura 6 se muestra un ejemplo de imagen de entrada (imagen superior), y la imagen segmentada por la red, con la generación del contorno y centroide de la botella con mayor área (imagen inferior).

2.3. *Sistema de navegación*

Una vez que se obtiene el área y el centroide de la botella con mayor área, la Jetson Nano transmite estos datos a la tarjeta Arduino Uno, vía protocolo serial. Tomando como datos el área y el centroide, la tarjeta Arduino ejecuta un algoritmo basado en lógica difusa, implementado en C++, para calcular la potencia que debe de imprimirse a cada motor, con el objetivo de que el robot se desplace hacia la botella correspondiente.

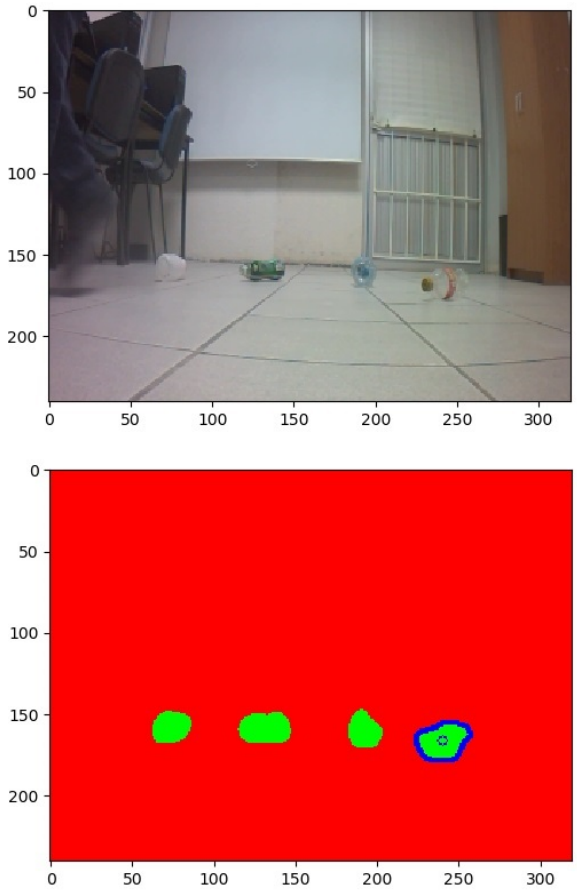

Figura 6: Reconocimiento y cálculo de centroide

Para la implementación de este sistema, se definieron tres conjuntos difusos. Un primer conjunto define la función de pertenencia para la variable lingüística de entrada *área* (ver Figura 7), un segundo conjunto para la variable de entrada *pos_x*, que corresponde a la componente sobre el eje x del centroide (ver Figura 8), y un tercer conjunto difuso para la variable lingüística de salida *potencia* (ver Figura 9).

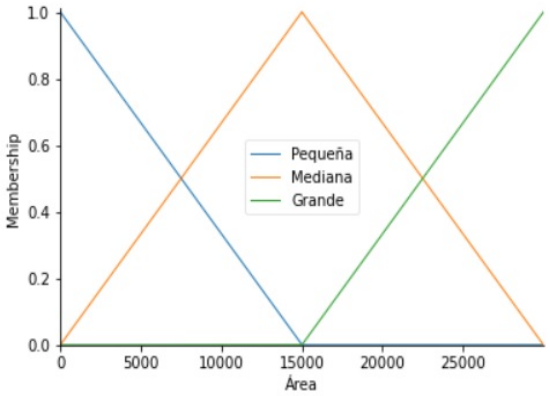

Figura 7: Conjunto difuso para la variable de entrada *área*.

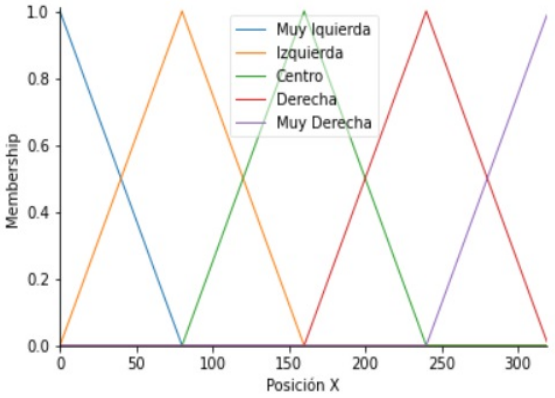

Figura 8: Conjunto difuso para la variable de entrada *pos_x*.

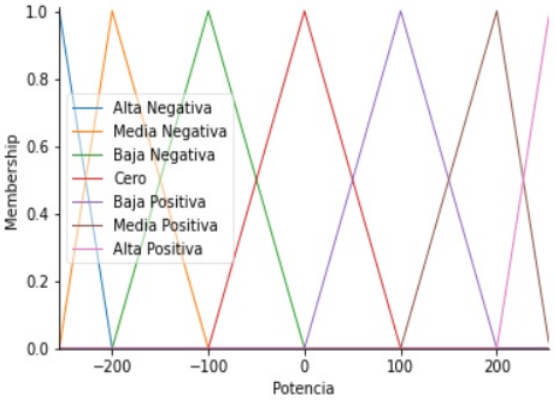

Figura 9: Conjunto difuso para la variable de salida *potencia*.

En la Tabla 4 se muestra la definición de las reglas para la inferencia difusa de la potencia del motor derecho, y en la Tabla 5 se muestra lo correspondiente para el motor izquierdo. El sistema de inferencia utilizado es del tipo Mamdani, y para la defuzzificación se aplica el método Singleton (ver Algoritmos 4, 5 y 6).

Tabla 4: Reglas de inferencia para la potencia del motor derecho.

Pos.x / Área	Muy Izq.	Izq.	Centro	Der.	Muy Der.
Pequeña	Media Pos.	Media Pos.	Alta Pos.	Baja Pos.	Baja Pos.

Mediana	Baja Pos.	Baja Pos.	Media Pos.	Cero	Cero
Grande	Baja Pos.	Baja Pos.	Cero	Cero	Baja Neg.

Tabla 5: Reglas de inferencia para la potencia del motor izquierdo.

Pos.x / Área	Muy Izq.	Izq.	Centro	Der.	Muy Der.
Pequeña	Baja Pos.	Baja Pos.	Alta Pos.	Media Pos.	Media Pos.
Mediana	Cero	Cero	Media Pos.	Baja Pos.	Baja Pos.
Grande	Baja Neg.	Cero	Cero	Baja Pos.	Baja Pos.

Algoritmo 4. Función para giro de los motores en Arduino

```
IN1 = número de pin asociado a IN1 del puente H
IN2 = número de pin asociado a IN2 del puente H
ENA = número de pin asociado a INA del puente H
IN3 = número de pin asociado a IN3 del puente H
IN4 = número de pin asociado a IN4 del puente H
ENB = número de pin asociado a INB del puente H
define giroMotores(potDer, senDer, potIzq, senIzq)
  envía señal LOW a IN1, IN2, IN3 e IN4
  envía valor potIzq a ENA
  envía valor potDer a ENB
  si senIzq es 1:
    envía señal HIGH a IN1
  si no:
    envía señal HIGH a IN2
  si senDer es 1:
    envía señal HIGH a IN4
  si no:
    envía señal HIGH a IN3
```

Algoritmo 5. Función del sistema difuso para cálculo de las potencias de los motores en Arduino

```
// define matriz para el conjunto difuso Área
conDifArea = [[0,0,15000],          // Pequeña
              [0,15000,30000],      // Mediana
              [15000,30000,30000]]  // Grande
// define matriz para el conjunto difuso Centroide
conDifCx = [[0,0,80],               // Muy Izquierda
            [0,80,160],             // Izquierda
            [80,160,240],           // Centro
            [160,240,320],          // Derecha
            [240,320,320]]          // Muy Derecha
// define matriz para el conjunto difuso Potencia
conDifPot = [[-255,-255,-200],      // Alta Negativa
             [-255,-200,-100],      // Media Negativa
             [-200,-100,0],         // Baja Negativa
             [-100,0,100],          // Cero
             [0,100,200],           // Baja Positiva
             [100,200,255],         // Media Positiva
             [200,255,255]]         // Alta Positiva
// matriz para inferencia de la potencia del motor derecho,
// las filas corresponden al índice del conjunto del área, las
// columnas al centroide y el valor de la celda al de potencia
reglasPotDer = [[5,5,6,4,4],
                [4,4,5,3,3],
                [4,4,3,3,2]]
// matriz para inferencia de la potencia del motor izquierdo
reglasPotIzq = [[4,4,6,5,5],
                [3,3,5,4,4],
                [2,3,3,4,4]]
define cálculoPotencias(área, centr_x):
  pertW = crea matriz de cuatro elementos flotantes
  potMedDerW = crea matriz de cuatro elementos flotantes
  potMedIzqW = crea matriz de cuatro elementos flotantes
  contPertW = crea matriz de cuatro elementos enteros
  contPertW = 0
  para i de 0 a 2:
    // si el área pertenece al conjunto difuso i
    si área > conDifArea[i][0] y área < conDifArea[i][2]:
      para j de 0 a 4:
        // si el centr_ x pertenece al conjunto difuso j
        si centr_x>conDifCx[j][0] y centr_x<conDifCx[j][2]:
          // aplica las reglas de inferencia para las potencias
          potMedDerW[contPertW] = conDifPot[reglasPotDer[i][j]][1]
          potMedIzqW[contPertW] = conDifPot[reglasPotIzq[i][j]][1]
          si área < conDifArea[i][1]:
            auxWA = (área-conDifArea[i][0]) / (conDifArea[i][1]-conDifArea[i][0])
          si no:
            auxWA = (conDifArea[i][2]-área) / (conDifArea[i][2]-conDifArea[i][1])
          si centr_x < conDifCx[j][1]) {
            auxWB = (centr_x-conDifCx[j][0])/ (conDifCx[j][1]-conDifCx[j][0])
          si no:
            auxWB = (conDifCx[j][2]-centr_x) / (conDifCx[j][2]-conDifCx[j][1])
          pertW[contPertW] = mínimo(auxWA, auxWB)
          incrementa contPertW
  // aplica defuzzificación
  sumaNumeDer = 0
  sumaNumeIzq = 0
  sumaDeno = 0
  para i de 0 a contPertW-1:
    sumaNumeDer += potMedDerW[i] × pertW[i]
    sumaNumeIzq += potMedIzqW[i] × pertW[i]
    sumaDeno += pertW[i]
  potDer = sumaNumeDer / sumaDeno
  potIzq = sumaNumeIzq / sumaDeno
  si potDer > 0:
    senDer = 1
```

si no:
senDer = 0
si *potIzq* > 0**:**
senIzq = 1
si **no:**
senIzq = 0
potDer = *absoluto*(*potDer*)
potIzq = *absoluto*(*potIzq*)
retorna *potDer*, *senDer*, *potIzq*, *senIzq*

Algoritmo 6. Módulo principal del sistema de navegación sobre el Arduino

define *configuración( )***:**
velocidad de transferencia del puerto serial =115200
tiempo de espera de datos en el puerto serial = 10 ms
define pines de salida *IN1,IN2,IN3,IN4,ENA,ENB*
define *ciclo( )***:**
si hay datos disponibles en el puerto serial**:**
cadena = lee cadena del puerto serial, hasta salto de línea
si *cadena* contiene a la subcadena "<area>"**:**
posEtiArea = posición de la subcadena "<area>"
posEtiCx = posición de la subcadena "<cx>"
posEtiFin = posición de la subcadena "<fin>"
área = subcadena entre las posiciones *posEtiArea*+6 y *posEtiCx*, en formato entero
centr_x = subcadena entre las posiciones *posEtiCx*+4 y *posEtiFin*, en formato entero
si *área* > 0 y *centr_x* > 0**:**
potDer,*senDer*,*potIzq*,*senIzq* = *cálculoPotencias*(*área*,*centr_x*)
giroMotores(*potDer*,*senDer*,*potIzq*,*senIzq*)
retardo de un milisegundo

3. Experimentos y Resultados

Los experimentos del método propuesto se dividieron en dos tipos. Por un lado, se realizaron pruebas para evaluar la capacidad de reconocimiento, y posteriormente se realizaron pruebas para evaluar el sistema de navegación.

3.1. Pruebas de reconocimiento

Para validar el método de reconocimiento, se realizaron 50 pruebas, en las cuales se colocó el robot y las botellas en posiciones y orientaciones diferentes. De esta forma, se tomaron 50 imágenes, donde cada una puede contener desde cero hasta las cuatro botellas. Las imágenes pueden contener otros objetos del entorno, como reguladores, botes de basura, cables, mesas, sillas, entre otros. De las 50 imágenes de prueba, 32 contienen botellas y 18 no contienen. En total, se distribuyeron 42 botellas dentro del grupo de 32 imágenes.

En las Figuras 10 y 11 se muestran ejemplos de pruebas de reconocimiento. Para tomar la imagen de la Figura 10, no se colocaron otros objetos entre las botellas (sí en el entorno). Para la imagen de la Figura 11, se colocó un regulador, un bote de basura y una silla, entre las botellas. Se puede observar en ambas figuras, que la red reconoció correctamente las botellas, ignorando el resto de los objetos.

En la Tabla 6 se muestran los resultados de las pruebas de reconocimiento realizadas con las 50 imágenes. Para la evaluación se consideraron tres métricas: *precision* (1), *recall* (2) y *accuracy* (3). Estás métricas se basan en los valores: *true positive* (TP) , *false positive* (FP), *true negative* (TN) y *false negative* (FN). Se consideraron como TP, los casos donde la red reconoce botellas, que realmente sí se encuentran en las imágenes. Como FP, los casos donde la red reconoce botellas, que no se encuentran en las imágenes. Como TN, los casos donde la red no reconoce botellas, en imágenes que no contienen alguna botella. Y como FN, los casos donde la red no reconoce botellas, que sí se encuentran en las imágenes.

$$PR = \frac{TP}{TP + FP} \quad (1)$$

$$RC = \frac{TP}{TP + FN} \quad (2)$$

$$AC = \frac{TP + TN}{TP + FP + TN + FN} \quad (3)$$

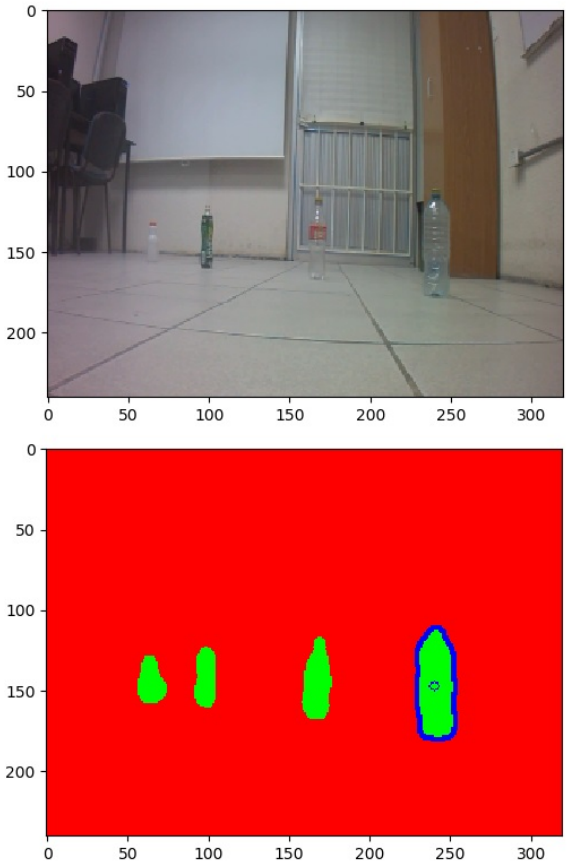

Figura 10: Ejemplo de prueba de reconocimiento.

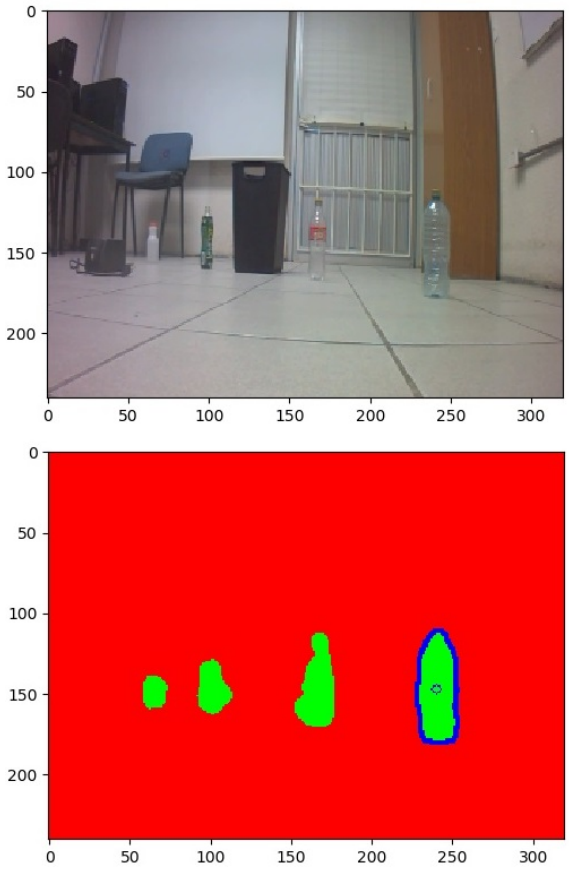

Figura 11: Prueba de reconocimiento con otros objetos cercanos.

Tabla 6: Resultados de las pruebas de reconocimiento.

TP	41
FP	1
TN	18
FN	1
Precision	97.6%
Recall	97.6%
Accuracy	96.7%

Los resultados de las pruebas muestran un 96.7% de precisión en el reconocimiento de las botellas. Este porcentaje de precisión en el reconocimiento, permitió la navegación fluida y directa del robot hacia las botellas. Además, es competitivo con respecto a los resultados reportados en otros trabajos recientes con objetivos similares.

Por ejemplo, en la Tabla 7 se muestran los resultados de las pruebas de reconocimiento en (Bai *et. al*., 2018), quienes aplican su estrategia sobre cinco categorías de objetos (variedad de botes, latas, cartón, bolsas de plástico y desperdicio de papel). Dependiendo de la categoría de los objetos, logran una precisión entre el 77.7% y el 91.97%.

Tabla 7: Pruebas de reconocimiento en (Bai *et al*., 2018)

Categoría	Precisión (%)
Botes	91.87
Latas	90.11
Cartón	90.94
Bolsas de plástico	85.68
Residuo de papel	77.7

En la Tabla 8 se presentan los resultados de los experimentos en (Kulshreshtha *et al*., 2021), quienes apllican tres modelos de redes neuronales para el reconocimiento de basura no biodegradable, en diferentes entornos como follaje verde, playas, caminos, regiones rocosas e interiores. Los resultados van del 83% al 97.1% de precisión, dependiendo del modelo de red utilizado.

Tabla 8: Pruebas de reconocimiento en (Kulshreshtha *et al*., 2021)

Modelo de Red	Precisión (%)
Mask-RCNN	83
YOLOv4-tiny	95.2
YOLOv4	97.1

3.2. Pruebas de navegación

Las pruebas para evaluar el sistema de navegación, consistieron en analizar las trayectorias que siguió el robot, para aproximarse a las botellas colocadas en 20 posiciones distintas, dentro del campo de visión del robot. Todas las posiciones se definieron dentro de un espacio de dos por tres metros. En la Figura 12 se muestran las trayectorias seguidas por el robot.

En todas las pruebas, el robot alcanzó las posiciones de las botellas, siguiendo trayectorias casi directas. Es decir, una vez que una botella es identificada a través de la red neuronal, en el 100% de las pruebas, el robot se desplazó correctamente hasta la posición de esta. Además, se realizaron pruebas con las botellas en movimientos, mientras el robot se desplazaba hacia estas (ver Video). En todos los casos, mientras las botellas no se salen del campo de visión, el robot ajusta su trayectoria para dirigirse a las nuevas posiciones.

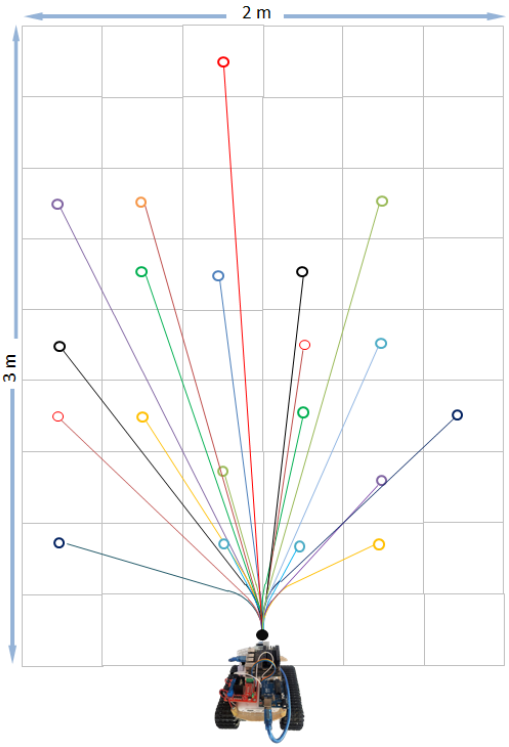

Figura 12: Trayectorias de las pruebas de navegación.

4. Conclusiones

Como primer acercamiento en la búsqueda de soluciones al problema de contaminación de nuestras playas, en este trabajo, se propone el diseño y construcción de un robot autónomo, para reconocimiento y navegación hacía botellas de plástico.

Para el sistema de reconocimiento, se entrenó una red neuronal tipo U-Net basada en la ResNet18, implementada en Python, con las librerías Pytorch, Torchvisión y OpenCV, sobre la tarjeta NVIDIA Jetson Nano B01. El sistema de navegación se diseño a través de lógica difusa, y se implementó en C++ sobre una tarjeta Arduino Uno R3.

Los experimentos se realizaron en un ambiente controlado, utilizando cuatro tipo diferentes de botellas. En las pruebas de reconocimiento se obtuvo un 96.7% de precisión. Mientras que en las pruebas de navegación, el robot alcanzó en todos los casos los objetivos establecidos, siguiendo trayectorias casi directas.

Una vez probado el método en un ambiente controlado, como trabajo futuro, se propone reentrenar la red neuronal y reconfigurar el sistema difuso, para que el robot sea capaz de reconocer y navegar en un ambiente de playa. El objetivo final de este proyecto, es incorporar al robot un sistema de evasión de obstáculos y un sistema de recolección de las botellas.

Agradecimientos

Agradecemos a la Dirección de Posgrado, Investigación e Innovación del Tecnológico Nacional de México, por el financiamiento de este trabajo, a través del proyecto 15319.22-P.

Referencias

Agencia de Protección Ambiental de Estados Unidos. (28 de Enero de 2022). La importancia de la Protección de las playas. Obtenido de https://espanol.epa.gov/espanol/la-importancia-de-la-proteccion-de-las-playas.

Abeliuk, A., Gutiérrez, C., (2021). Historia y evolución de la inteligencia artificial. Revista Bits de Ciencia 21, 14–21.

AleniziI, N., Alajmi, O., Alsharhan, S. Khudada, S., (2019). Autonomous Beach Cleaner. URC Conference Dubai. Disponible en línea: https://dspace.auk.edu.kw/handle/11675/8024 (accesado 11 julio 2022).

Andreu-Perez, J., Deligianni, F., Ravì, D., Yang, G., (2017). Artificial Intelligence and Robotics, UK-RAS Network.

Guarnizo, M. J. G., Bautista, D. D., Sierra, T. J. S., (2021). Una revisión sobre la evolución de la robótica móvil. Repositorio - Universidad Santo Tomás.

De La Torre, M. K., Cristóbal, Y. J. G., Sotelo, V. F., (2020). Diseño de un robot móvil recolector y compactador de botellas de plástico utilizando redes neuronales en playas con arena fina, tesis, Universidad Ricardo Palma, Facultad de Ingeniería, Escuela Profesional de Ingeniería Mecatrónica, Lima, Perú.

Bai, J., Lian, S., Liu, Z., Wang, K., Liu, D., (2018). Deep Learning Based Robot for Automatically Picking Up Garbage on the Grass, in IEEE Transactions on Consumer Electronics 64(3), 382-389. DOI: 10.1109/TCE.2018.2859629

Jha, A., Singh, A., Kerketta, R., Prasad, D., Neelam, K., Nath, V., (2019). Development of Autonomous Garbage Collector Robot. In: Nath, V., Mandal, J. (eds) Proceedings of the Third International Conference on Microelectronics, Computing and Communication Systems. Lecture Notes in Electrical Engineering, 556. Springer, Singapore. DOI: 10.1007/978-981-13-7091-5_46

Kulshreshtha, M., Chandra, S., Randhawa, P., Tsaramirsis, G., Khadidos, A., Khadidos, A., (2021). OATCR: Outdoor Autonomous Trash-Collecting Robot Design Using YOLOv4-Tiny. Electronics 10, 2292. DOI: 10.3390/electronics10182292.

López, L. E., Rubio, E. E., Sossa, A. J. H., Ponce, P. V. H., (2020). Selección de acciones para la navegación de un robot móvil basada en fuzzy q-learning. Research in Computing Science 149, 79–89.

Mangayarkarsi, P., Sarath, G., Sudheer, G., (2020). Arduino based trashbot. Tierärztliche Praxis 40, 1429–1436.

Pawar, S., Shinde, S., Fatangare, J., (2019). Remote Operated Floating River Cleaning Machine. Int. Res. J. Eng. Technol. 6, 3344–3347.

Ramírez, Z. A. R., Martín, O. M. I., Carballido, C. J. L., (2020). Robot inteligente recolector de basura asistido por redes neuronales artificiales, Tesis, Facultad de Ciencias de la Computación, Benemérita Universidad Autónoma de Puebla, Puebla, Puebla.

Ronneberger, O., Fischer, P., Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. Medical Image Computing and Computer-Assisted Intervention 9351.

Santos, M., (2011). Un Enfoque Aplicado del Control Inteligente, Revista Iberoamericana de Automática e Informática Industrial RIAI, 8 (4), 283-296. DOI: 10.1016/j.riai.2011.09.016

Segura, C. D. J., Molina, L. H., Rubio, E. E., (2019). Sistema de navegación y evasión de obstáculos aplicando un sistema de control difuso en una placa Arduino uno. Research in Computing Science 148, 291–303.

Sensio, J. (3 octubre 2020). Visión Artificial - Segmentación Semántica. Sensio-Inteligencia Artificial. https://juansensio.com/blog/050_cv_segmentacion.

Valverde, R., Gachet, D., (2007). Identificación de Sistemas Dinámicos Utilizando Redes Neuronales RBF. Revista iberoamericana de automática e informática industrial (RIAI), 4 (2), 32-42. DOI: 10.1016/S1697-7912(07)70207-8