Artículo **Revista de Aplicaciones de la Ingeniería**
Marzo, 2019 Vol.6 No.18 1-10

Robot seguidor de línea basado en visión artificial con ROS y OpenCV

Computer vision based line follower robot with ROS and OpenCV

VARGAS-TORRES, César †*, SANTIAGO-PAZ, Jayro

Centro de Investigación Avanzada (CIATEQ), Campus Guadalajara
Centro de Investigación y de Estudios Avanzados (CINVESTAV), Campus Guadalajara

ID 1er Autor: *César Augusto, Vargas-Torres* / **ORC ID:** 0000-0001-6102-8515, **CVU CONACYT ID:** 700271

ID 1er Coautor: *Jayro, Santiago-Paz* / **ORC ID:** 0000-0002-7036-0074, **CVU CONACYT ID:** 331279

DOI: 10.35429/JEA.2019.18.6.1.10 Recibido: 19 de Enero, 2019; Aceptado 02 de Marzo, 2019

Resumen

El objetivo de esta investigación es presentar el diseño y la implementación de un robot seguidor de línea basado en visión artificial con el sistema operativo para robots (ROS) y la biblioteca de visión artificial de código abierto (OpenCV). La razón para usar ROS y OpenCV es crear una plataforma robusta que pueda ser la base de robots más complejos como los vehículos de guiado automatizado (AGVs). Una cámara es el sensor que le permite al robot obtener imágenes de la ruta, que se procesan en un algoritmo para detectar la línea y obtener el error de retroalimentación del controlador Proporcional Integral y Derivativo (PID). La ventaja de utilizar una cámara en lugar de sensores infrarrojos es que la cámara devuelve un error con mayor resolución, además de que es posible detectar colores y obstáculos. El resultado de combinar una cámara, un controlador PID, ROS y OpenCV es un robot seguidor de línea estable y robusto, fácil de integrar con otros sistemas.

Robot seguidor de línea, Visión artificial, Sistema Operativo para Robots (ROS)

Abstract

The objective of this research is to present the design and implementation of a computer vision based line follower robot with Robot Operating System (ROS) and Open Source Computer Vision Library (OpenCV). The reason to use ROS and OpenCV is to create a robust platform that can be the base of more complex robots like the Automated Guided Vehicles (AGVs). A camera is the sensor that allows the robot to obtain images from the path, that will be processed in an algorithm to detect the line and gets the feedback error for Proportional Integral Derivative (PID) controller. The advantage of using a camera instead of infrared sensors is that the camera returns an error with higher resolution and is possible to get color identification and obstacle detection. The results of combining a camera, PID controller, ROS and OpenCV is a stable and robust line follower robot, easy to integrate with other systems.

Line follower robot, Computer vision, Robot Operating System (ROS)

Citación: VARGAS-TORRES, César, SANTIAGO-PAZ, Jayro. Robot seguidor de línea basado en visión artificial con ROS y OpenCV. Revista de Aplicaciones de la Ingeniería. 2019 6-18: 1-10

* Correspondencia del Autor (Correo electrónico: vargas.cesar88@gmail.com)
† Investigador contribuyendo como primer autor.

Introducción

Los robots seguidores de línea son máquinas móviles capaces de detectar y seguir una guía en el piso, por lo general la guía se representa como una línea negra sobre una superficie clara con alto contraste de color (Pandacan, Sanaatiyan, 2009). La aplicación industrial más común de los robots seguidores de línea son los AGV que se utilizan para transportar materiales en almacenes y líneas de producción. El funcionamiento del los robots seguidores de línea se basa en las siguientes tres operaciones básicas:

1: Medir la desviación en la trayectoria del robot. Para realizar esta tarea se suelen utilizar dos o más sensores infrarrojos, ya que a mayor a cantidad de sensores mejor será la medición Otra opción es utilizar una cámara que obtenga imágenes desde la parte frontal del robot y se procesen en un algoritmo de visión artificial, capaz de detectar la línea y medir la desviación en la trayectoria. A dicha medición se le conoce como error.

2: Generar una señal de control que corrija la trayectoria del robot. Para lograrlo se suele utilizar un controlador PID retroalimentado con el error obtenido en la medición de la desviación de la trayectoria del robot.

3: Regular de la velocidad angular y dirección de los motores a partir de la señal de control generada por el controlador PID, de esta manera el robot corregirá su curso (Sai-nan, 2008).

El presente trabajo expone el diseño e implementación de un robot seguidor de línea basado en un sistema de visión artificial que aprovecha las ventajas de ROS y OpenCV. El trabajo está estructurado de la siguiente manera: La segunda sección presenta los fundamentos teóricos, la tercera sección presenta el diseño e implementación del robot, la cuarta sección expone los resultados y la quinta sección presenta las conclusiones.

Fundamentos teóricos

ROS

El sistema operativo para robots es un marco de trabajo de código abierto utilizado para escribir software para robots.

Se compone de una colección de herramientas y librerías compatibles con gran variedad de plataformas robóticas.

Aunque ROS no es un sistema operativo como tal, proporciona servicios como la abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidades de uso común, comunicación entre procesos y administración de paquetes, además cuenta con funcionalidades de alto nivel como llamadas síncronas y asíncronas, bases de datos centralizadas y un sistema de configuración para el robot.

ROS está basado en una arquitectura de grafos donde el procesamiento toma lugar en nodos que pueden recibir, enviar y multiplexar mensajes de sensores, controles, estados, planificaciones y actuadores, entre otros (Open Source Robotics Foundation, 2018). La filosofía de ROS se puede resumir en los siguientes cinco principios:

1. Punto-a-Punto: Una arquitectura punto-a-punto permite los nodos que componen el robot puedan dialogar directamente con cualquier otro de forma síncrona o asincrónica, según sea necesario.

2. Multilenguaje: ROS se puede programar en varios lenguajes de programación, lo importante es que las clases desarrolladas permitan la generación de mensajes basados en el Lenguaje de Definición de Interfaz (IDL).

3. Basado en herramientas: En lugar de un entorno de tiempo de ejecución monolítico, ROS adoptó un diseño de microkernel, que utiliza una gran cantidad de pequeñas herramientas para construir y ejecutar sus diversos nodos.

4. Ligero: Los algoritmos se empaquetan en ejecutables independientes, reutilizables y de tamaño reducido.

5. Gratuito y de código abierto: Es soportado por una gran comunidad y se ha convertido un estándar en la industria de la robótica (Mazzari, 2018).

Otro aspecto importante de ROS es su sistema de archivos, que está organizado de forma jerárquica en disco, donde destacan los siguientes conceptos:

ISSN 2410-3454

Paquete: Es un directorio que contiene nodos, bibliotecas externas, datos, archivos de configuración y un archivo de configuración llamado manifest.xml.

La pila: Es un directorio que contiene paquetes un archivo de configuración llamado stack.xml (Open Source Robotics Foundation, 2018).

Como se mencionó anteriormente, ROS permite ejecutar una gran cantidad de nodos en paralelo que requieren intercambiar datos de forma síncrona y asíncrona. El responsable de alcanzar ese objetivo es el grafo de cómputo de ROS, que involucra los conceptos que se menciona a continuación:

Maestro: Es un servicio de declaración y registro de nodos, que hace posible que los nodos se encuentren e intercambien datos.

Nodo: Es una instancia de un ejecutable. Un nodo puede ser un sensor, motor, algoritmo, etc (Open Source Robotics Foundation, 2018).

Temas: Un tema es un sistema de transmisión de datos asíncrono basado en un sistema de suscripción / publicación, donde uno o más nodos pueden publicar datos en un tema, y uno o más nodos pueden leer datos sobre ese tema (Open Source Robotics Foundation, 2018).

Servicios: Es un método de comunicación síncrona entre dos nodos (Open Source Robotics Foundation, 2018).

Mensaje: Es una estructura de datos compuesta por tipos de datos primitivos como cadenas de caracteres, booleanos, enteros, punto flotante, etc (Open Source Robotics Foundation, 2018).

OpenCV

Es una librería para el procesamiento de imágenes y visión artificial ampliamente utilizada en el ámbito académico y comercial. Cuenta con más de 2500 algoritmos de visión artificial y aprendizaje automático clásicos y de vanguardia. Estos algoritmos se pueden usar para detectar rostros, identificar objetos, clasificar gestos humanos, rastrear movimientos de la cámara, extraer modelos 3D de un objeto, etc.

OpenCV tiene una comunidad de más de 47 mil personas y más de 14 millones de descargas. Soporta los lenguajes C++, Python, Java y MATLAB y es compatible con Linux, Windows, Android y MAC OS (OpenCV Team, 2018).

Controlador PID

El controlador PID es un mecanismo de control por retroalimentación ampliamente utilizado en sistemas de control industriales. Este mantiene una relación entre la salida y la entrada de referencia, comparándolas y usando la diferencia como medio de control.

El algoritmo de control incluye tres modos básicos, el proporcional, el integral y el derivativo. El proporcional se obtiene multiplicando el error por una constante Kp, el integral es la suma de los errores anteriores multiplicados por una constante Ki y el derivativo predice errores futuros mediante la multiplicación de la diferencia entre el error actual y anterior por una constante Kd (Lorandi, Hermida, Ladrón de Guevara, Hernández, 2011) En la figura 1 se observa el diagrama bloques de un controlador PID.

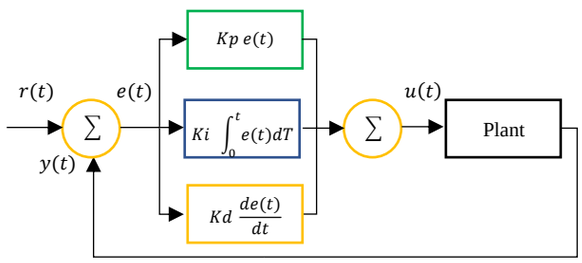

Figura 1 Diagrama a bloques de un controlador PID
Fuente: Elaboración Propia (2018)

Los controladores PID se usan en la mayoría de los sistemas de control, especialmente cuando el modelo matemático de la planta no se conoce y por lo tanto, no se pueden emplear métodos de diseño analíticos.

El proceso de ajuste de las constantes del controlador PID se conoce como sintonización y en casi todos los casos se hace en el sitio (Ogata, 2010). Por la ventaja de no requerir un modelo matemático del sistema a controlar y lo simple que es la implementación en un microcontrolador, el controlador PID es una gran elección para el control de la trayectoria de un robot seguidor de línea.

ISSN 2410-3454

A continuación, se muestra el pseudocódigo de la implementación de un controlador PID para un robot seguidor de línea.

```
Tiempo de muestreo <- 0.1
Kp                 <- 0.5
Ki                 <- 0.005
Kd                 <- 0.3
Error              <- 0
Error Anterior     <- 0
Proporcional       <- 0
Integral           <- 0
Derivativo         <- 0
Señal de control   <- 0
Procedimiento Cálculo PID
Repetir
Leer Error En la desviación del robot
   Proporcional    <- Error
   Integral        <- Integral + Error *
Tiempo de muestreo
   Derivativo      <- (Error - Error
Anterior) / Tiempo de muestreo
   Error Anterior  <- Error
   Señal de control <- Kp * Proporcional +
Ki * Integral + Kd * Derivativo
   Escribir Señal de control
   Hasta que Siempre
Fin Procedimiento
```

Diseño e implementación

Como etapa previa al diseño, se especificaron los requerimientos del robot seguidor de línea. Tales establecen que el robot debe ser un vehículo con una estructura mecánica de cuatro ruedas y tracción independiente capaz de alcanzar una velocidad de máxima de 1 m/s. El sensor para la detección del error debe ser una cámara, el mecanismo de control debe ser un controlador PID y la arquitectura del software debe estar basada en ROS y OpenCV. A partir de los requerimientos mencionados se seleccionó una estructura mecánica y se definió una arquitectura de hardware y de software.

Estructura mecánica

La estructura mecánica seleccionada es un chasis de aluminio de 4 ruedas con dimensiones de 24 x 23 x 8 cm y un peso de 1 kg. En la Figura 2 se aprecia el chasis de 4 ruedas.

Figura 2 Chasis de 4 ruedas
Fuente: Elaboración propia (2018)

Arquitectura de hardware

La arquitectura del hardware se basa en una Raspberry pi 3 B con un procesador de cuatro núcleos a 1.2 GHz que funge como controlador central y es responsable de recibir las imágenes capturadas por una cámara Logitech C920 de 1080p a 30 fps y de enviar la señal de control generada por el controlador PID a los cuatro motores de 12 V y 100 rpm. En la figura 3 se aprecia un diagrama a bloques de la arquitectura del hardware.

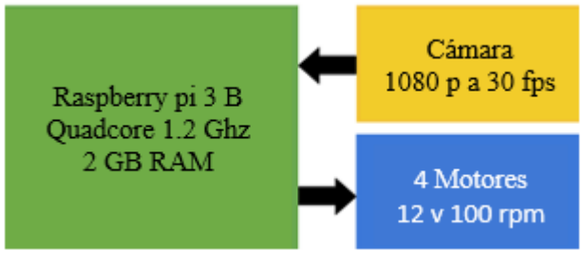

Figura 3 Diagrama a bloques de la arquitectura del hardware
Fuente: Elaboración propia (2018)

Arquitectura del software

La arquitectura de software está basada en el sistema operativo Ubuntu Mate 16.04 con ROS Kinetic y OpenCV 3.4.0 instalados en una capa superior. En la capa de aplicación se contará con cuatro nodos de ROS, de los cuales el primero funge como controlador principal, el segundo es un algoritmo para la medición del error, el tercero es la implementación del controlador PID y el último el controlador de los motores. En la figura 4 se aprecia el diagrama a bloques de la arquitectura de software.

ISSN 2410-3454

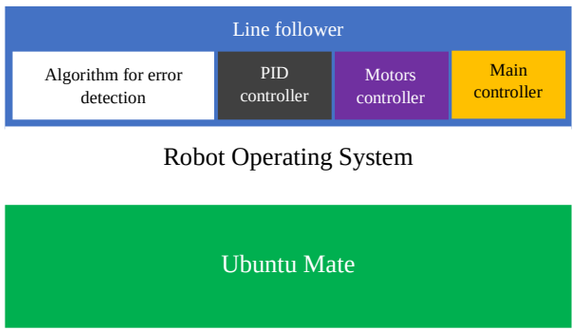

Figura 4 Diagrama a bloques de la arquitectura del software
Fuente: Elaboración propia (2018)

Diagrama de conexiones

En el diagrama de conexiones se aprecian los elementos necesarios para el control del robot. Para suministrar energía se utiliza una batería Lipo de 11.1 V a 5500 mAh que energiza a los dos puentes H dobles y a un regulador de voltaje con salida USB a 5V. La Raspberry pi 3 B se conecta al regulador de voltaje y la cámara web a la Raspberry vía USB. En la Figura 5 se muestra el diagrama principal.

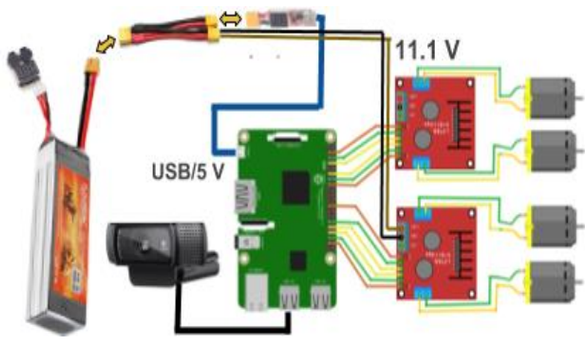

Figura 5 Diagrama de conexiones
Fuente: Elaboración propia (2018)

Diagrama de nodos de ROS

El diagrama de la Figura 6 muestra los nodos de ROS y la relación entre ellos.

El nodo de la izquierda se encarga de la detección del error, el central genera la señal de control, el de la derecha envía la señal de control a los motores y el nodo superior es el controlador central que marca la pauta del funcionamiento de los demás.

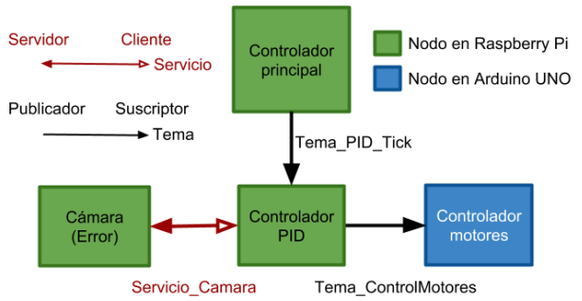

Figura 6 Diagrama de nodos de ROS
Fuente: Elaboración propia (2018)

Diagrama de clases

El diseño del software del robot se basó en el paradigma de programación orientada a objetos y se utilizó el lenguaje C++ para la implementación, por tal motivo se crearon clases que representan los elementos del robot. En la figura 7 se muestra el diagrama de clases.

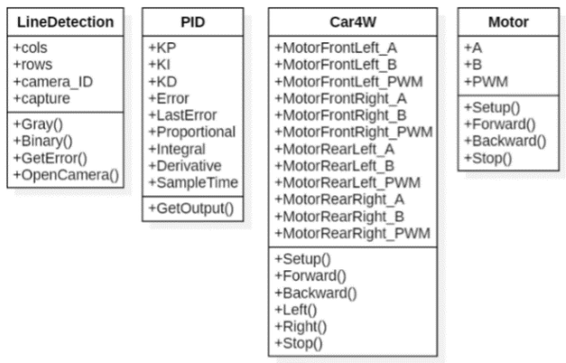

Figura 7 Diagrama de clases
Fuente: Elaboración propia (2018)

3.7 Algoritmo para medición del error

El algoritmo comienza con la captura de una imagen de la ruta desde parte frontal del robot (Ver Figura 8), dicha imagen se procesa, comenzando con un suavizado que reduce el ruido ambiental (Ver Figura 9). La imagen resultante se convierte a escala de grises (Ver Figura 10) y posteriormente se binariza (Ver Figura 11).

Finalmente, de la imagen binaria, se analizan todos los pixeles del eje horizontal posicionados en el centro del eje vertical, con el fin de contar los pixeles negros y sumar su posición respecto al eje horizontal. Mediante la división entre la suma de las posiciones horizontales de los píxeles negros entre la cantidad de los mismos, se obtiene la posición del píxel que representa el centro de la línea.

ISSN 2410-3454

Para obtener el error en la trayectoria, se suma la posición del píxel que representa el centro de la línea menos la mitad de la cantidad de píxeles horizontales de la imagen (Ver Figura 12). A continuación se muestra el pseudocódigo de la implementación del algoritmo para la medición del error en la trayectoria del robot.

```
Suma          <- 0
Cantidad      <- 0
Error         <- 0
Procedimiento Medición del error en la trayectoria
   Repetir
      Capturar Imagen
      Suavizar Imagen
      Convertir a gris Imagen
      Binarizar Imagen
      Para y <- 0 Hasta Imagen.cols  Con Paso 1 Hacer
         Si Imagen[rows*0.5][y] == 0  entonces
            Cantidad    <- Cantidad + 1
            Suma        <- Suma + y
      Fin para
         Si Cantidad > 0 entonces
         Error          <- Suma / Cantidad
      Escribir Error - Imagen.cols* 0.5
   Hasta que Siempre
Fin Procedimiento
```

Figura 8 Captura de imagen
Fuente: Elaboración propia (2018)

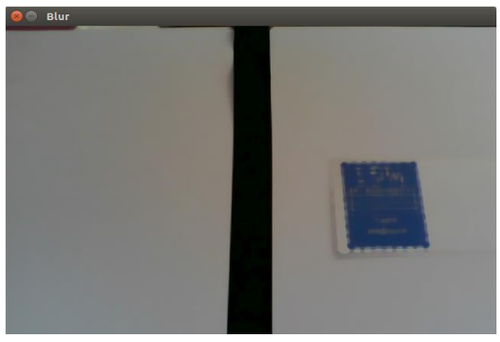

Figura 9 Imagen suavizada
Fuente: Elaboración propia (2018)

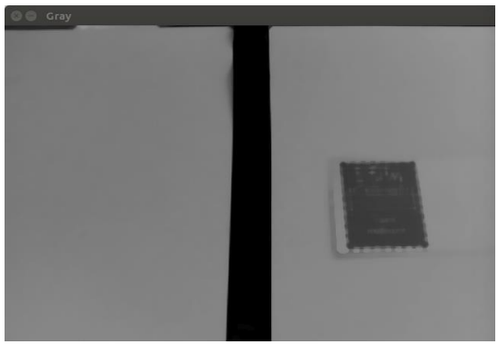

Figura 10 Imagen en escala de grises
Fuente: Elaboración propia (2018)

Figura 11 Imagen binarizada
Fuente: Elaboración propia (2018)

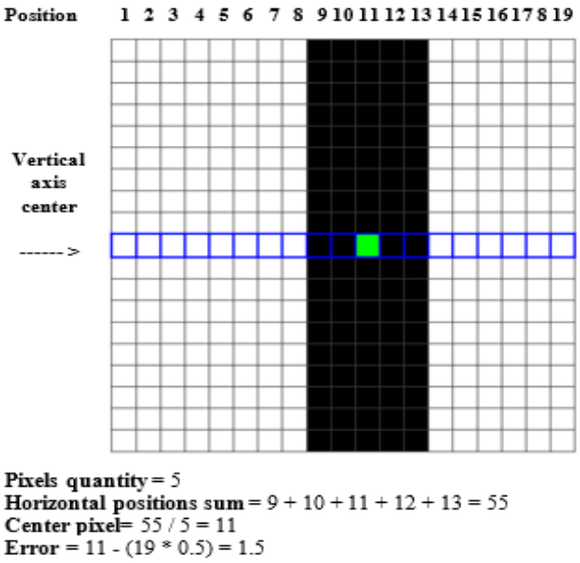

Figura 12 Ejemplificación de medición de error.
Fuente: Elaboración propia (2018)

Resultados

Una vez concluida la implementación se realizaron pruebas unitarias para garantizar que todos los subsistemas funcionan correctamente. Las pruebas se enfocaron en el algoritmo para medición del error, el controlador PID y una prueba funcional que demuestre la correcta integración de todos los componentes de hardware y software.

ISSN 2410-3454

Prueba de algoritmo para medición de error

La prueba consistió en validar visualmente que el algoritmo es capaz de detectar el centro de la línea en ambientes con distintas condiciones de iluminación. El resultado demostró que en entornos con escasa iluminación el algoritmo no era capaz de detectar el centro de la línea, por lo que fue necesario integrar un sistema de iluminación en la parte frontal del robot que garantice que el campo de visión de la cámara mantenga una iluminación constante en cualquier ambiente.

El resultado de la mejora fue positivo. En la figura 13, 14 y 15 observamos los resultados de la medición del error, donde se aprecian dos líneas de color azul, la línea horizontal marca los pixeles que se analizaron en el algoritmo y la línea vertical representa el centro de la línea negra. Además del lado derecho de las imágenes se muestra el error medido.

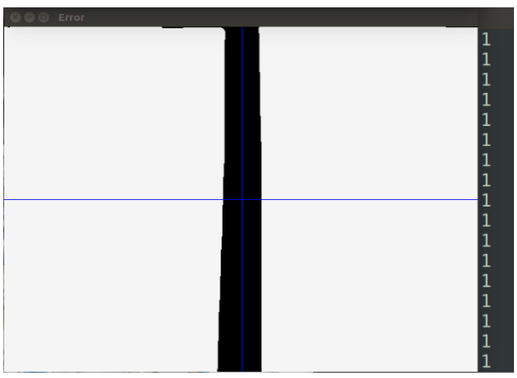

Figura 13 Medición del error al centro de la imagen
Fuente: Elaboración propia (2018)

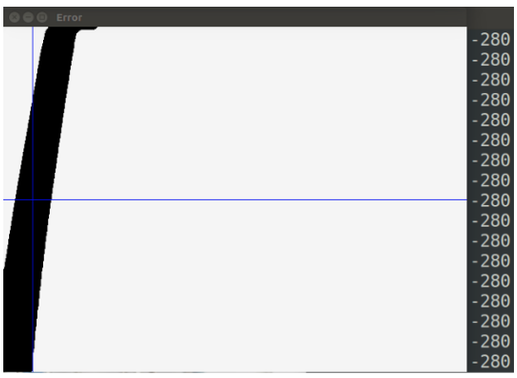

Figura 14 Medición del error a la izquierda de la imagen
Fuente: Elaboración propia (2018)

Figura 15 Medición del error a la derecha de la imagen
Fuente: Elaboración propia (2018)

Sintonización del controlador PID

La sintonización del controlador PID es un proceso delicado, ya que si no se realiza de forma adecuada el robot presentará un comportamiento inestable que podría ocasionar oscilaciones o la pérdida de la ruta.

Para garantizar que la sintonización es la adecuada, se validó el comportamiento del robot durante cinco horas en diferentes rutas, además de generar perturbaciones externas con la finalidad de desestabilizar al sistema para observar su respuesta.

Después de repetir el proceso de sintonización en 20 ocasiones se definió el valor de las constantes Kp, Ki y Kd que presentaron mejor comportamiento en el controlador PID. En la Tabla 1 se muestran los distintos valores de las constantes con las cuales se hicieron pruebas.

Puebla	Kp	Ki	Kd
1	0.050	0.000	0.000
2	0.100	0.000	0.000
3	0.150	0.000	0.000
4	0.200	0.000	0.000
5	0.250	0.000	0.000
6	0.300	0.000	0.000
7	0.350	0.000	0.000
8	0.400	0.000	0.000
9	0.450	0.000	0.000
10	0.500	0.000	0.000
11	0.550	0.000	0.000
12	0.600	0.000	0.000
13	0.550	0.001	0.020
14	0.500	0.002	0.040
15	0.450	0.003	0.060
16	0.400	0.004	0.080
17	0.350	0.005	0.100
18	0.300	0.006	0.120
19	0.300	0.007	0.140
20	0.300	0.008	0.160
21	0.300	0.009	0.180

ISSN 2410-3454

22	0.300	0.010	0.200
23	0.295	0.010	0.210
24	0.290	0.010	0.220
25	0.285	0.010	0.230
26	0.280	0.010	0.240
27	0.275	0.010	0.250
28	0.270	0.010	0.260
29	0.265	0.010	0.270
30	0.260	0.010	0.280

Tabla 1 Pruebas de sintonización de controlador PID
Fuente: Elaboración propia (2018)

Prueba funcional

La prueba funcional consistió en operar el robot durante seis horas en doce lapsos de 30 minutos continuos. Dicha prueba permitió validar que todos los elementos mecánicos, electrónicos y de software funcionan correctamente y satisfacen los requerimientos especificados antes del diseño. En la Figura 16 se visualiza el robot seguidor de línea operando en un almacén.

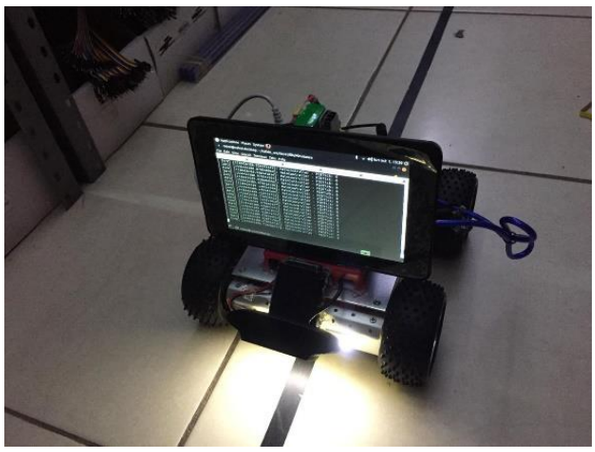

Figura 16 Robot seguidor de línea operando en un almacén
Fuente: Elaboración propia (2018)

Caso de uso

Como se demostró anteriormente, el robot seguidor de línea presentado en este trabajo tiene un comportamiento estable en distintos ambientes, además de que la integración de ROS y OpenCV lo hace lo suficientemente flexible y robusto como para ser la base de robots más complejos.

Un caso de uso donde se utilizó el robot presentado en este trabajo, es en el diseño y construcción de un AGV basado en RFID para el manejo de órdenes en un almacén.

Este robot se desplegó en la empresa Talos Electronics, dedicada a comercializar productos electrónicos, donde asiste al usuario en la recolección de productos en el almacén.

La tarea del AGV consiste en cargar los contenedores donde se depositan los productos, recibir los números orden que se van a recolectar, navegar de forma autónoma hasta la ubicación de los productos, indicarle al usuario la cantidad y ubicación de los mismos, recibir la confirmación de la recolección y repetir el proceso hasta terminar la recolección de todos los productos (Ver Figura 17). El despliegue del AGV permitió reducir el tiempo invertido en el manejo de órdenes y la disminución de errores en la selección de productos.

Figura 17 AGV basado en RFID para manejo de órdenes
Fuente: Elaboración propia (2018)

El funcionamiento del AGV se basa en la instalación de tags RFID a intervalos definidos sobre la guía que sigue el robot (Ver Figura 18). Posteriormente se guarda la distribución del almacén de Talos en un nodo de ROS como un mapa en forma de grafo, donde los tag RFID representan los vértices y la guía las aristas.

Otro nodo de ROS contiene un algoritmo de búsqueda en grafos que permite determinar la más corta para realizar el manejo de las órdenes, en la Figura 6 se muestra la representación del almacén de Talos en forma de grafo. En la Figura 19 se aprecia el mapa del almacén de Talos representado en forma de grafo.

ISSN 2410-3454

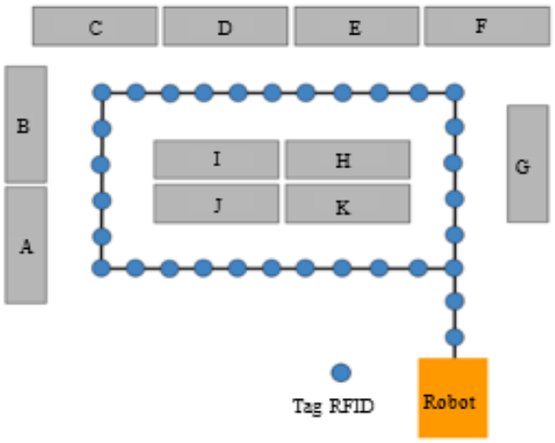

Figura 18 Tags RFID sobre la guía del AGV
Fuente: Elaboración propia (2018)

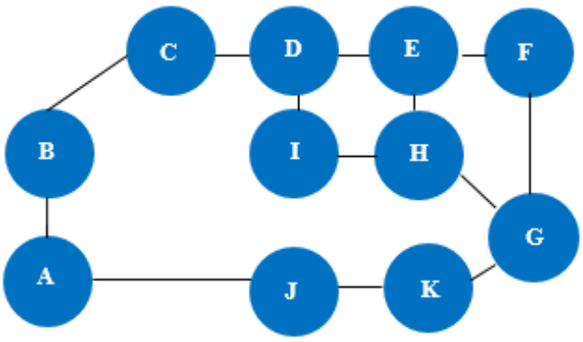

Figura 19 Almacén de Talos representado en forma de grafo
Fuente: Elaboración propia (2018)

Conclusiones

En este trabajo se presenta el diseño y construcción de un robot seguidor de línea basado en visión artificial con ROS y OpenCV, donde la medición del error en la trayectoria del robot se obtiene mediante la captura de imágenes de la ruta y su procesamiento, lo que permite obtener un error bastante preciso, que es la clave del comportamiento estable del robot.

Dicho robot pretende ser una plataforma robusta y flexible, que sea compatible con tecnologías estándar en la industria de la robótica con la finalidad de ser la base de robots móviles más complejos, tal el caso de los AGV, que pueden aprovechar la funcionalidad del seguidor de línea y las herramientas que ofrecen ROS y OpenCV para integrar más nodos con funcionalidades para el control de tráfico, detección de obstáculos, integración de interfaz hombre máquina, etc.

El despliegue del AGV para el manejo de órdenes en la empresa Talos Electronics, permitio reducir el tiempo requerido para en la realización de esta tarea un 47%, que se traduce en 10 horas hombre por semana, además garantiza que no haya errores en la recolección lo que ayuda a brindar un mejor servicio al cliente.

Agradecimientos

Se agradece al CONACYT y al CIATEQ unidad Guadalajara, al Dr. Jayro Santiago Paz por su valiosa retroalimentación y apoyo otorgado durante el desarrollo de la investigación.

Referencias

Lorandi, A. P., Hermida. G., Ladrón de Guevara, E., Hernández, J. (2011). Controladores PID y Controladores Difusos. Revista de la ingeniería industrial, Volumen 5, No.1.

Mazzari, V. (2016). ROS – Robot Operating System. (2018, 20 Octubre), The blog generación de robots. Disponible: https://www.generationrobots.com/blog/en/ros-robot-operating-system-2/

Ogata, K. (2010) Ingeniería de control moderna (3a. ed.), Mdrid: Prentice Hall Hispanoamerica.

Open Source Robotics Foundation (2018). About ROS. (2018, 22 Octubre). ROS. Disponible: http://www.ros.org/about-ros/

Open Source Robotics Foundation (2018). Navigating the ROS Filesystem. (2018, 22 Octubre). ROS. Disponible: http://wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem

Open Source Robotics Foundation (2018). Understanding ROS Nodes. (2018, 23 Octubre). ROS. Disponible: http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

Open Source Robotics Foundation (2018). Understanding ROS Topics. (2018, 23 Octubre). ROS. Disponible: http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

Open Source Robotics Foundation (2018). Understanding ROS Services and Parameters. (2018, 24 Octubre). ROS. Disponible: http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams

Open Source Robotics Foundation (2018). Creating a ROS msg and srv. (2018, 25 Octubre). ROS. Disponible: http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv

Open Source Robotics Foundation (2018). Creating a ROS msg and srv. (2018, 25 Octubre). ROS. Disponible: http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv

OpenCV Team (2018). About OpenCV. (2018, 19 Octubre). OpenCV. Disponible: https://opencv.org/about.html

Pandacan, M., Sanaatiyan, M. M.. (2009). Design and Implementation of Line Follower Robot. Second International Conference on Computer and Electrical Engineering. pp. 585-590.

Sai-nan, Liu. (2008). Optimization problem for AGV in automated warehouse system. 2008 IEEE International Conference on Service Operations and Logistics. pp. 1640-1642.

ISSN 2410-3454