Experimental Verification of the Leader-Follower Formation Control of Two Wheeled Mobile Robots with Obstacle Avoidance

J. L. Mata-Machuca, L. F. Zarazua, and R. Aguilar-López

***Abstract*—In this work the experimental verification of the control of two wheeled mobile robots with differential configuration under a leader-follower formation is presented, where the leader performs the task of tracking a predefined trajectory, as well as, it is able to avoid obstacles and it sends information of the point-to-point trajectories reached to the follower mobile robot. The follower has the ability to estimate the trajectory of the leader robot only based on the information received from the leader, for arbitrary initial conditions. Experimental verifications with two wheeled mobile robots TurtleBot3 Burger illustrate the performance of the proposed technique, using the robot operating system (ROS) and Python.**

***Index Terms*—leader-follower formation, experimental verification, obstacle avoidance, wheeled mobile robots, differential configuration.**

I. Introducción

Un aspecto importante en la robótica cooperativa es lograr controlar sistemas de robots móviles de manera simultánea para llevar a cabo una tarea de forma coordinada [1], [2], [3]. La coordinación de robots representa una complejidad debido a las áreas que intervienen, tales como, procesamiento de información, comunicación, esquemas de control, seguimiento de trayectorias, evasión de obstáculos, escenario de pruebas, etc. Algunas aplicaciones se encuentran enfocadas en la coordinación de robots móviles para el control de tráfico evitando colisiones entre los vehículos [4], en el diseño de algoritmos de localización para operaciones de búsqueda y rescate [5], en el control para vehículos autónomos [6], en la impresión simultánea de piezas en 3D [7], en la manipulación y transporte de objetos [8], entre otras.

Aunque existen distintos métodos para el control de la formación de los robots móviles [9], [10], [11], [12], el esquema líder-seguidor ha sido mayormente utilizado debido a su simplicidad, flexibilidad, confiabilidad y adaptabilidad [13]. En esta estrategia, uno o varios robots del sistema se designan como líderes de la formación y el resto como seguidores, a los cuales se les especifica la postura (orientación y posición) deseada en relación con el líder o los líderes [14]. Por ejemplo, el trabajo [15] presenta un sistema de visión adaptativo para el control en formación líder-seguidor de robots móviles, el cual consiste en colocar una cámara, sin calibrar los parámetros, a bordo del robot seguidor con orientación y posición arbitrarias y mediante un observador adaptativo se estiman los parámetros desconocidos de la cámara así como los coeficientes del plano en movimiento con respecto al marco de referencia de la cámara, para calcular los puntos recorridos por el robot líder se emplea un controlador adaptativo basado en imágenes; en [16] se introduce un algoritmo independiente de las mediciones de posición y velocidad para el control de seguimiento en formación líder-seguidor; en [17] se describe el desarrollo de una estrategia de control mediante visión artificial que permite la coordinación de múltiples robots, el robot líder se mueve en una trayectoria incierta con una velocidad desconocida, el controlador esta diseñado para mantener a los robots seguidores a cierta distancia detrás del líder, usando información visual de la posición del líder; [18] presenta una técnica de control de la formación líder-seguidor para robots monociclo con restricciones de entrada, además se demuestran las condiciones para mantener la formación y la estabilización asintótica; [19] aborda el problema del control de seguimiento de múltiples robots móviles que avanzan en formación a lo largo de trayectorias en línea recta, se asume que solo un robot líder de de la red de robots móviles tiene la información de la trayectoria de referencia. Sin embargo, pocas aplicaciones prácticas verifican la viabilidad de la realización física de las técnicas propuestas. Siempre es un desafío llevar a cabo las pruebas experimentales, tomando en cuenta los errores en las mediciones de los sensores y la comunicación.

Para la evasión de obstáculos se tienen varios enfoques, tales como, campos potenciales, el histograma de campo vectorial (VFH), el algoritmo de contorno, entre otros [20], [21], [22], [23]. La técnica de campos potenciales considera al robot móvil como una partícula con un campo potencial de atracción, mientras que los obstáculos tienen potencial de repulsión, entonces se generan fuerzas repulsivas sobre el robot manteniéndolo alejado de los obstáculos [24], [25], [26]. El VFH es un algoritmo para llevar a cabo la detección y evasión de obstáculos, el método toma como entrada los datos adquiridos por los sensores y se forman tres niveles de representación de los datos, el primero es un mapa de dos dimensiones dividido por celdas donde se guardan las distancias, el segundo es un histograma polar, construido

Este trabajo ha sido financiado por la Secretaría de Investigación y Posgrado del Instituto Politécnico Nacional con el proyecto con número de registro SIP20201803.

J. L. Mata Machuca, Departamento de Tecnologías Avanzadas, UPIITA, Instituto Politécnico Nacional, IPN 2580, Ciudad de México 07340, México e-mail: jmatam@ipn.mx.

L. F. Zarazua, Ingeniería Mecatrónica, UPIITA, Instituto Politécnico Nacional, IPN 2580, Ciudad de México 07340, México e-mail: lzarazuaa1400@alumno.ipn.mx.

R. Aguilar-López, Departamento de Biotecnología y Bioingeniería, CINVESTAN-IPN, , IPN 2508, Ciudad de México 07360, México e-mail: raguilar@cinvestav.mx.

alrededor de la posición momentánea del robot y el tercero es la dirección que debe seguir el robot para evadir un obstáculo y alcanzar a su objetivo [27], [28], [29], [30]. Otros enfoques para evasión de coliciones son aplicados en sistemas con un gran número de elementos que interactúan en un escenario común, tales como, ORCA [22], es una técnica geométrica que tiene como objetivo garantizar la evasión de colisiones entre los robots en un sistema multiagente, y en [23] se propone una técnica que realiza la evasión de colisiones mediante la acción de fuerzas y energías potenciales, la aplicación que se presenta en [23] es con respecto a la interacción de un grupo de peatones en movimiento.

La contribución principal de este trabajo consiste en la verificación experimental del control en formación líder-seguidor de dos robots móviles terrestres con configuración diferencial mediante herramientas computacionales de código abierto (open source), los cuales tienen el objetivo de seguir trayectorias predefinidas, realizando la evasión de obstáculos. Se asume que los obstáculos son estáticos y estan presentes al momento de iniciar las pruebas. Para la verificación experimental, se emplean robots tipo TurtleBot3 Burger® que cuentan con un sensor láser de distancia de 360° tipo LIDAR el cual es usado para detectar los obstáculos. Por lo tanto, la tarea a cumplir por parte del robot líder es la de seguir alguna trayectoria proporcionada por el usuario y evadir los obstáculos, mientras que el robot seguidor se encarga de reconstruir la trayectoria del líder a partir de la información recibida. El algoritmo de evasión de obstáculos que se basa en el VFH [31], se ha implementado en ROS [32] y Python, utilizando únicamente un sensor LIDAR, lo que permite ser aplicado a otros robots o sistemas; el algoritmo líder-seguidor consiste en emplear el método de localización adaptativa de Monte Carlo (AMCL [33]) para estimar la posición y orientación de un robot a medida que comienza a moverse.

Es importante mencionar que para la aplicación que se presenta en este trabajo es suficiente la implementación de los métodos VFH y AMCL, debido a que los obstáculos externos son estáticos sobre la trayectoria predefinida y el problema principal es la evasión de los obstáculos y el seguimiento de la trayectoria. En las técnicas propuestas en [22], [23] cada robot se puede ver como un obstáculo en movimiento.

II. Planteamiento del Problema

El modelo cinemático es esencial para el análisis de robots móviles y para los controladores. Para comenzar a entender el movimiento de los robots móviles primero es necesario representar la posición de un robot con configuración diferencial en un marco de referencia global $x_G - y_G$ y en un marco de referencia local $x_L - y_L$ [34], como se muestra en la Fig. 1, donde, r (m) es el radio exterior de las llantas, b (m) es un parámetro que se mide del punto P a la mitad del ancho de las llantas, b se considera como una distancia media de la llanta en contacto con la superficie, y las variables $\dot{\varphi}_{izq}$ (rad/s) y $\dot{\varphi}_{der}$ (rad/s) son las velocidades angulares de los motores de los lados izquierdo y derecho, respectivamente. La posición de P en el marco de referencia global se especifica mediante las coordenadas (x, y), y la diferencia angular entre los marcos de

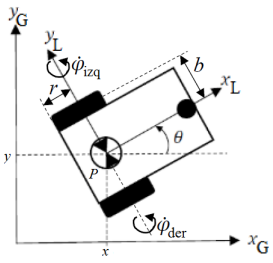


Fig. 1. Marco de referencia global y el marco de referencia local del robot.

referencia global y local viene dada por θ. Entonces, se define la posición del robot como un vector $\xi_G \in \mathbb{R}^3$ en función de estas variables, es decir, $\xi_G = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$.

Para describir el movimiento del robot en términos de las variables del movimiento global, se utiliza la matriz de rotación ortogonal,

$$R(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

La matriz (1) puede ser usada para determinar el movimiento del robot en ambos marcos de referencia, es decir permite cambiar del marco de referencia local de robot al global y viceversa, de la siguiente manera,

$$\dot{\xi_L} = R(\theta)\,\dot{\xi_G} \tag{2}$$

$$\dot{\xi_G} = R(\theta)^{-1}\,\dot{\xi_L} \tag{3}$$

donde, $\dot{\xi_G} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{\theta} \end{bmatrix}^T$.

La Fig. 2 representa la formación líder-seguidor, donde, d_1 es la distancia mínima entre líder y seguidor, y d_2 es la distancia que separa al líder del seguidor sobre la trayectoria recorrida por el líder con respecto al seguidor. Entonces, el robot líder envía al seguidor los puntos recorridos en la trayectoria. Con los datos recibidos el seguidor alcanzará los puntos indicados por el líder, y debido a que tienen la misma velocidad se mantendrán a una distancia constante d_2 sobre la trayectoria entre líder y seguidor.

En la Fig. 2 la curva de color rojo representa un tramo de la trayectoria a seguir, la cual esta formada por n pares

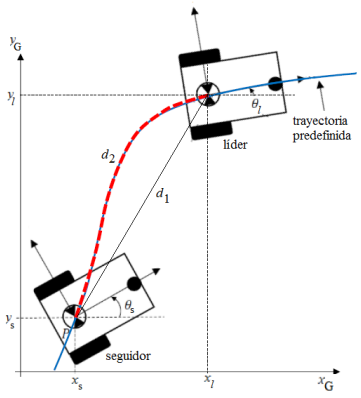


Fig. 2. Formación líder-seguidor.

de coordenadas, por lo tanto, la curva de color rojo se puede aproximar por $n-1$ segmentos que resultan de unir pares de coordenadas consecutivas, siendo $P_l = (x_l, y_l) = (x_n, y_n)$ y $P_s = (x_s, y_s) = (x_1, y_1)$ las coordenadas donde se localizan el líder y el seguidor, respectivamente, entonces se obtienen d_1 y d_2 como sigue,

$$d_1 = \sqrt{(x_l - x_s)^2 + (y_l - y_s)^2} \quad (4)$$

$$d_2 = \sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \quad (5)$$

III. Características y Especificaciones

De manera general, en la Fig. 3 se muestran los aspectos principales que han sido considerados para el control en formación líder-seguidor: (1) comunicación WiFi entre la computadora, el robot líder y el seguidor, (2) robot seguidor, (3) robot líder, (4) obstáculos sobre la trayectoria predefinida, (5) computadora con sistema operativo ROS para comunicarse con los robots, (6) marcas de distintas formas para una mejor identificación con el sensor LIDAR, (7) iluminación favorable para la operación del sensor LIDAR, (8) suelo plano y con coeficiente de fricción mayor que 0.4, (9) trayectoria predefinida, (10) punto final de la trayectoria, (11) distancia mínima de separación entre los objetos o móviles de 12 cm, (12) distancia mínima entre la computadora y el área de trabajo de 2 m, y (13) Dimensión máxima del área de trabajo de 6 m $\times$ 6 m.

Cada robot TurtleBot3 Burger® presenta las siguientes especificaciones,

- 1 Tarjeta Raspberry Pi 3 Model B, con sistema operativo Raspbian.
- 2 Actuadores Dynamixel XL430-W250-T: par máximo 1.4 N·m (a 11.1 V, 1.3 A), microcontrolador ST CORTEX-M3, encoder absoluto sin contacto (12 bit, 360°), resolución 4096 pulsos/rev, algoritmo de control PID, velocidad de transmisión 9600 bps $\sim$ 4.5 Mbps
- 1 Sensor láser de distancia 360° LIDAR LDS-01: rango de distancia de detección 120mm$\sim$3500mm, exactitud de distancia (120mm$\sim$499mm) $\pm$ 15mm, exactitud de distancia (500mm$\sim$3500mm) $\pm$ 5 %, rango de escaneo 300$\pm$10rpm, alcance angular 360°, resolución angular 1°, frecuencia de muestreo 1.8kHz.
- 1 Tarjeta OpenCR (32-bit ARM CORTEX-M7)
- Velocidad máxima de traslación de 0.22 m/s
- Velocidad máxima de rotación de 2.84 rad/s
- Tamaño de 138mm $\times$ 178mm $\times$ 192mm
- Giroscopio, acelerómetro y magnetómetro
- Bateria Li-Po 11.1V, 1800mAh

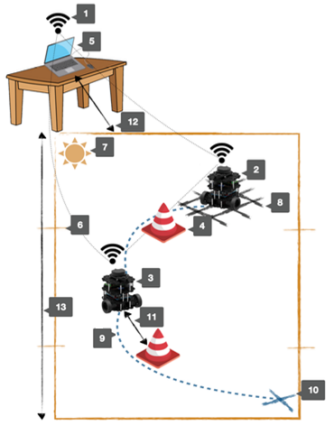


Fig. 3. Control en formación líder-seguidor con evasión obstáculos.

Además, se emplea una Computadora con procesador Intel Core i7-9700 CPU @ 3GHz, memoria RAM 16 GB, con Ubuntu 16.04. Las ganancias de los controladores de velocidad PID de los motores se han obtenido a partir de la opción de auto-sintonización: $K_p = 1.1$, $K_d = 0.03$, $K_I = 0.11$. La resolución del mallado de las trayectorias es igual a 1cm, se ha ajustado de manera experimental.

IV. Propuesta de Solución

Se procede a realizar los algoritmos que se requieren para lograr el control en formación líder-seguidor.

Para el seguimiento de trayectorias se utiliza la matriz de fuerzas (mapa de vectores que cubre toda el área de trabajo) como método de seguimiento por donde el líder se desplaza. La trayectoria de referencia se obtiene discretizando la trayectoria propuesta y guardando estos puntos en un archivo con extensión .txt. Posteriormente, esta serie de puntos son procesados por el algoritmo que genera la matriz de fuerzas y son guardados en un archivo con extensión .npy el cual puede ser interpretado por Python. En la Tabla I se observa el algoritmo usado en el programa principal para generar la matriz de fuerza y en la Tabla II se muestra como se calcula para cada punto de la matriz el elemento destino o meta.

TABLA I
ALGORITMO GENERADOR DE LA MATRIZ DE FUERZA.

Genera_matriz_de_fuerza.py

```
num_archivo = '3' # Ingresar del 1 al 3 o el número de trayectorias a
escoger y que hayan sido previamente generadas.
puntos = 'Trayectoria'+num_archivo+'.txt' # Nombre del archivo a leer.
nombre = 'MatrizDeFuerza'+num_archivo+'.npy' # Nombre del archivo
de salida que contiene la matriz de fuerza.
nombrep = 'PuntosAjustados'+num_archivo+'.npy' # Nombre del
archivo de salida que contiene los puntos ajustados al mapa.
nombreGrafica1 = 'GraficoPorSeguir'+num_archivo+'.jpg' # Nombre
de la gráfica a seguir.
nombreGrafica2 = 'GraficoInterpolado'+num_archivo+'.jpg' # Nombre
de la gráfica con interpolación.
nombreGrafica3 = 'GraficoDistancias'+num_archivo+'.jpg' # Nombre de
la gráfica con interpolación.
map_size_x = 250.0 # Medida del mapa en "x" en cm.
map_size_y = 250.0 # Medida del mapa en "y" en cm.
resolution = 1.0 # Medida del mallado en cm.
matrix = np.zeros((map_size_x/resolution,map_size_y/resolution,2),
dtype='f') # Generar matriz de tamaño x*y*2
matrix_dist = np.zeros( (map_size_x/resolution,map_size_y/resolution),
dtype='f' ) # Generar matriz de tamaño x*y*2 para guardar distancias.
ind_lookahead = np.zeros( (map_size_x/resolution,map_size_y
/resolution),dtype='d' ) # matriz con número de índices a avanzar.
margen = 50 # Longitud en cm desde el borde hasta el punto
mas lejano de la trayectoria.
min_dist = 0.05 # Distancia mínima de separación entre 2 puntos de
trayectoria en m.
```

TABLA II
RUTINA PARA OBTENER EL PUNTO META.

near

```
def near(initial_position,xind,yind,tree,xy):
dist, index = tree.query(initial_position) # Obtener la distancia al punto
mas cercano y el indice que lo identifica.
global matrix_dist
matrix_dist[xind,yind]=dist # Guardar la distancia.
# Encontrar el punto mas cercano
if dist>4*min_dist: # Si la distancia es mayor a 4 veces la distancia
mínima de separación entre puntos
        lookahead_offset = np.int(0) # avanza al punto mas cercano.
else:
         lookahead_offset = np.int(2) # avanza con dirección al tercer
punto mas cercano.
ind_lookahead[xind,yind] = lookahead_offset # Guarda en la matriz de
indices a avanzar.
lookahead_target = xy[(index + lookahead_offset) % len(xy)] # Elige
punto objetivo.
x1, y1 = initial_position # Posición inicial.
x3, y3 = lookahead_target # Posición a alcanzar.
matrix[xind,yind,0] = x3-x1 # Guardar la distancia en x.
matrix[xind,yind,1] = y3-y1 # Guardar la distancia en y.
```

TABLA III
PROGRAMA PRINCIPAL DEL NODO DEL LÍDER.

MovimientoLíder

```
class MovimientoLider: # Clase del movimiento del robot líder.
  def __init__(self):
      self.inicializacion() # Inicia la función con los primeros
          parámetros.
      while self.poseflag==False: # Se mantiene parado mientras no
          haya recibido la primera posición del robot.
           self.setStop() # Coloca la velocidad del robot en 0.
   def seguimientoSinGUI(self):
      for i in range(0,numero_de_iteraciones):
           self.follow()# Llama a la función de seguir trayectoria.
  def seguimientoConGUI(self):
      self.turn_on=False # Bandera que indica si avanza o se para.
      self.lane_subscriber = rospy.Subscriber("/lane",Int32, # subscriptor
           self.laneCallback,queue_size=1)# de la trayectoria escogida.
      self.turn_on_subscriber = rospy.Subscriber("/turn_on",# subscriptor
           Bool,self.turn_onCallback,queue_size=1)# de avanzar o parar.
     while (not rospy.is_shutdown()):
           if self.turn_on==True and self.lane>0 and
                self.lane<=self.numTrayectorias:
                self.follow() # Llama a la función de seguir trayectoria.
           else:
                self.setStop() # Hace que el robot se pare.
   def main():
      rospy.init_node('MovimientoLider', anonymous=True)#Inicia el
          nodo MovimientoLider.
      gui = len(sys.argv)>1# Decide si usar GUI o no.
      Lider = MovimientoLider()
      print 'Nodo inicializado'# indica que el nodo se inicializó.
      if gui:
           Lider.seguimientoSinGUI()
      else:
           Lider.seguimientoConGUI()
      Lider.setStop()# Detiene al robot.
```

Una vez obtenida la matriz de fuerza se propone un código para que el líder pueda seguir la trayectoria. El seguimiento de trayectoria se realiza en el nodo "MovimientoLíder". En la Tabla III se observa el programa principal del nodo encargado de asignarle el movimiento al líder, este programa al construir el objeto configura todas las condiciones y funciones a ejecutar. La Tabla IV presenta el diagrama de flujo del método *"Follow"*, el cual tiene como propósito principal decidir si ejecuta el seguimiento de trayectoria o esquiva algún obstáculo

TABLA IV
MÉTODO PARA SEGUIR LA TRAYECTORIA.

Follow

```
def follow(self): #Calcula el siguiente punto a llegar y hace que el
robot se oriente en dirección a esa posición.
x1,y1=[self.lider_pose.x,self.lider_pose.y]# Lee la posición en x, y
# Obtener el índice en x para la matriz de fuerza.
x_index=np.int((x1+self.map_size_x/200)*(100/self.resolution))
# Obtener el índice en y para la matriz de fuerza.
y_index=np.int((y1+self.map_size_y/200)*(100/self.resolution))
# Definir límites del índice en "x" y en "y"
x_index = self.limitar(x_index,0,self.map_size_x/self.resolution-1)
y_index = self.limitar(y_index,0,self.map_size_y/self.resolution-1)
if self.lane>0:# Si se seleccionó una trayectoria entonces se lee la
trayectoria deseada.
     dist_x, dist_y = self.Trayectorias[self.lane-1][x_index,y_index,:]
# Obtener la distancia a la que se quiere llegar.
else: # Detiene al robot si no se seleccionó trayectoria.
          self.setStop()
          return
self.an = int(self.convert2pi(np.arctan2(dist_y,dist_x))*180/np.pi)
# Ángulo objetivo.
if (self.dist_min>self.obs_min_dist) or (self.histogram[self.an]==0):
# No Aplica VFH si no esta cerca o es un espacio libre.
self.move2angle(dist_y,dist_x)
# Establece la velocidad correcta para llegar al objetivo.
else: # Aplica VFH.
self.VectorFieldHistogram() # Detiene al robot.
```

cercano en caso de ser necesario. Adicionalmente incluye condiciones sobre la trayectoria seleccionada.

En la Tabla V se muestra el código del método *"Vector Field Histogram"*, para su ejecución en el líder. Este método tiene como función encontrar el ángulo más cercano a la trayectoria por donde pueda pasar el móvil realizando la evasión de obstáculos. En el caso específico del histograma, recibe un vector indicando si hay obstáculo o no, con el cual el método de "Follow" indica la instrucción de seguir la trayectoria o evadir el obstáculo. Este nodo es el encargado de suscribirse al algoritmo AMCL y reorientarse para mantener la trayectoria con base en la matriz de fuerza.

La Tabla VI presenta el diagrama de flujo del método "Move2angle", el cual a partir de la dirección a seguir orienta al robot líder al ángulo meta por medio del control de su velocidad angular. Cabe mencionar que para realizar un mejor seguimiento cuando el móvil está desorientado por más de 45°, la velocidad lineal es cero para garantizar la rotación en su propio eje, lo cual reduce el error al tener una mejor maniobrabilidad.

Para el robot seguidor, se propone el algoritmo que cumpla con la formación líder-seguidor, denominado *"Nodo Seguidor"*. Este nodo se suscribe a las posiciones publicadas por el líder y las posiciones del seguidor en la construcción del objeto *"FollowerRobot"*. Cada vez que se recibe una nueva posición del líder es almacenada en una lista de Python llamada *WPL*. El método *FollowerCallback* mostrado en la Tabla VII, es utilizado por el robot seguidor para cubrir los mismos puntos alcanzados por parte del robot líder, reorientando al robot seguidor bajo el mismo lazo de control implementado en el líder, en función de la posición y el conjunto de puntos de referencia almacenados en la lista WPL.

Para el control de los motores se emplea la cinemática inversa de los robots móviles en configuración diferencial. El

TABLA V
ALGORITMO VFH.

```
VectorFieldHistogram
class LaserVFH: # Clase para realizar el algoritmo VFH.
  def __init__(self):
    self.poseflag=False # Bandera del dato de posición obtenido.
    self.lidar_err=0.004 # error para evitar ver obstáculos falsos.
    self.lider_pose=Pose() # Inicializa el objeto de posición.
    self.pose_sub = rospy.Subscriber("/tb3_0/amcl_pose",
    PoseWithCovarianceStamped, self.poseCallback, queue_size=1)
    # Se subscribe a la posición del líder obtenida por AMCL.
    self.laser_sub = rospy.Subscriber("/tb3_0/scan", LaserScan,
    self.LaserCallback, queue_size=1) # Ejecuta la conversión al
    # histograma de vfh cuando se recibe una muestra del láser.
    self.histogram_publisher = rospy.Publisher('/tb3_0/histogram',
    Histograma, queue_size=1)# publicar el histograma
    self.histogram = Histograma() # guardar el histograma.
    self.an = range(0,360,1) # Vector que representa los 360 grados.
    self.H = np.zeros(360,dtype='f') # Histograma de unos y
    ceros que indica la presencia de obstáculos.
    self.Ho = np.zeros(360,dtype='f') # Histograma de unos y
    ceros que indica la presencia de obstáculos.
    self.hp = np.zeros(360,dtype='f') # Derivada del histograma.
    self.plot=False # Bandera para graficar los histogramas.
  def main():
    rospy.init_node('Laser_VFH', anonymous=True)
    L=LaserVFH() # Inicializa el nodo VFH.
    rospy.spin()
```

TABLA VI
MÉTODO PARA ORIENTAR AL ROBOT.

```
move2angle
def move2angle(self,dist_y,dist_x): # Orienta la robot en el ángulo que
indican las distancias de entrada al mismo tiempo que avanza.
self.rate = rospy.Rate(10) # Maximo de veces se repite por segundo la
función.
ka=1.0 # Constante para incrementar la velocidad de giro.
avel=self.convert2pi(np.arctan2(dist_y,dist_x))-self.lider_pose.theta
# Calcula la diferencia angular entre las 2 posciones.
avel=(avel+np.pi) %(2*np.pi)-np.pi # Orienta la velocidad en el sentido
de giro mas corto.
if abs(avel)>45*np.pi/180: # Si el giro es mayor a 45 grados hace que
la velocidad lineal sea 0, para evitar desviaciones largas provocadas por
        lvel=0.0 # la velocidad lineal.
else:
        lvel=self.lineal# Conserva la velocidad lineal constante.
avel=ka*avel# Aumenta o disminuye la velocidad de giro.
avel=self.limitar(avel,-self.lim_angular,self.lim_angular)# Límita la
velocidad angular dentro de un margen para evitar giros bruscos.
self.vel_msg.linear.x=lvel# Guarda la velocidad lineal.
self.vel_msg.angular.z =avel# Guarda la velocidad angular.
self.velocity_publisher.publish(self.vel_msg)# Publica la velocidad para
que el robot vaya a la velocidad calculada.
self.rate.sleep()# Ejecuta la pausa sin interrumpir otros procesos.
```

TABLA VII
MÉTODO PARA EJECUTAR CADA VEZ QUE SE RECIBE UNA NUEVA POSICIÓN EN EL SEGUIDOR.

```
FollowerCallback
def FollowerCallback(self, data):
  if self.turn_on==True: # Ejecuta rutina si se le indicó que avance.
    wplen=len(self.wp) # Guarda cuantas posiciones hay por alcanzar.
    if wplen> 60: # Si es mayor a 60 posiciones ejecuta el algoritmo
      de seguimento de trayectoria, para no chocar.
      # Datos del líder
      xL = self.wp[-1].pose.pose.position.x #posición mas antigua en x.
      yL = self.wp[-1].pose.pose.position.y #posición mas antigua en y.
      orientation_q = self.wp[-1].pose.pose.orientation #orientacion
      mas antigua.
      orientation_list = [orientation_q.x, orientation_q.y, orientation_q.z,
      orientation_q.w] # Guarda los valores en una lista.
      (rollL, pitchL, yawL) = euler_from_quaternion (orientation_list)
      # Datos del seguidor
      xF = data.pose.pose.position.x # Obtiene posición del seguidor
      yF = data.pose.pose.position.y # Obtiene posición del seguidor
      orientation_q = data.pose.pose.orientation # Obtener la orientación
      del seguidor
      orientation_list = [orientation_q.x, orientation_q.y, orientation_q.z,
      orientation_q.w] # Guardar la orientación del seguidor en una lista.
      (rollF, pitchF, yawF) = euler_from_quaternion (orientation_list)
      #print 'xF=',xF,'yF=',yF # Imprime la posición del seguidor.
      ka = 1.0 # Constante para incrementar la velocidad de giro.
      a = self.convert2pi(np.arctan2(yL-yF,xL-xF))-self.convert2pi(yawF)
      # Calcula la diferencia angular entre las 2 posiciones.
      avel = (a+np.pi) %(2*np.pi)-np.pi # Orienta la velocidad en el
      sentido de giro mas corto.
      if abs(avel)>45*np.pi/180: # Si el ángulo a girar es mayor a
                  45 grados.
          lvel = 0.0 # Gira sin velocidad lineal.
      else:
          lvel = self.follower_lin_vel # Gira con
                  velocidad lineal constante.
      #print 'velocidad angular=',avel # Imprime el ángulo a rotar.
      avel = self.limitar(ka*avel,-self.lim_angular,self.lim_angular)
      # Obtiene la velocidad de giro contemplando su límite.
      self.vel_msg.linear.x = lvel # Guarda la velocidad lineal.
      self.vel_msg.angular.z = avel # Guarda la velocidad angular.
      self.pub.publish(self.vel_msg) # Publica la velocidad para que
      el robot seguidor para que vaya a la velocidad calculada.
      self.wp.pop() # Elimina la posición alcanzada.
    else:
              self.vel_msg.linear.x,self.vel_msg.angular.z = [0.0,0.05]
      Guarda la velocidad para girar sobre su propio eje.
      self.pub.publish(self.vel_msg) # Publica la velocidad para que el
      robot seguidor gire sobre su propio eje.
```

objetivo es calcular la velocidad que se requiere en cada una de las llantas para obtener el cambio en la dirección y mantener una velocidad lineal constante. Esto se explica a continuación.

Sea V_k (m/s) la velocidad lineal requerida para cada robot móvil, entonces tomando la ecuación de la cinemática inversa se tiene que las velocidades de los motores estan dadas por,

$$\dot{\varphi}_{der} = \frac{V_k + \dot{\theta} b}{r} \tag{6}$$

$$\dot{\varphi}_{izq} = \frac{V_k - \dot{\theta} b}{r} \tag{7}$$

Debido a la restricción de la velocidad máxima de los motores ($\dot{\varphi}_{max}$), la velocidad angular máxima ($\dot{\theta}_{max}$) que se puede obtener para los robots móviles es,

$$\dot{\theta}_{max} = \frac{\dot{\varphi}_{max} r - V_k}{b} \tag{8}$$

Las ecuaciones (6)-(8) han sido implementadas en ROS para controlar los motores del líder y del seguidor.

Con respecto a la parte de procesamiento, se han desarrollado diferentes nodos en ROS. La manera en que los diversos nodos se comunican es mediante el uso de tópicos (variables compartidas), los cuales también son enlistados en la sección de transmisión de mensajes. Se proponen, para el líder y seguidor la interconexión de los nodos mostrados en las Figs. 4(a)-(b), donde se presenta el esquema de control en formación líder-seguidor con seguimiento de trayectorias y evasión de obstáculos. La Fig. 4(c) representa el sistema físico que será empleado para realizar los esquemas de control propuestos.

La manera de formar esta interconexión es directa, esto se debe a que dentro de cada nodo hay objetos que se suscriben

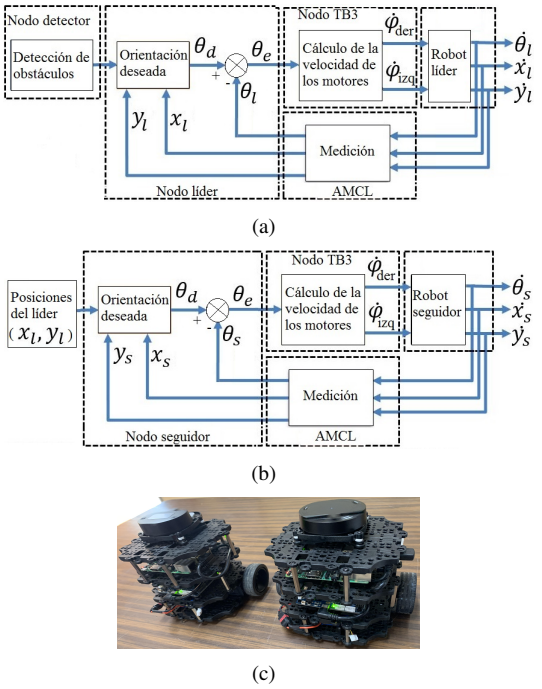


Fig. 4. Esquema de control en formación líder-seguidor. (a) Sistema de control para el líder, (b) sistema de control para el seguidor, (c) robots líder y seguidor.

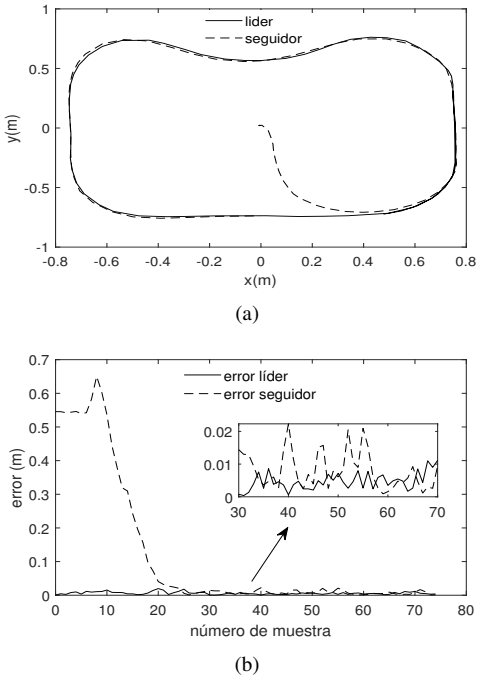


Fig. 5. Control en formación líder-seguidor. (a) Trayectoria 1, (b) error del líder y error del seguidor.

a tópicos, y otros que publican la información procesada, por lo que de manera natural pueden trabajar en conjunto. Para ejecutar todos los nodos se hace uso de otro tipo de archivo de ROS, archivos con extensión *launch*, los cuales permiten mandar a llamar los nodos de manera automática. Para lograr la interconexión que se propone en los diagramas de las Figs. 4(a)-(b), primero se ejecuta el nodo que inicializa los motores y el sensor LIDAR, posteriormente se activa el mapa para la localización, una vez inicializado el mapa se ejecutan los nodos líder y seguidor, finalmente se llaman por medio de otro archivo *launch* el localizador del líder y el localizador del seguidor, una vez inicializados todos los nodos antes mencionados se lleva a cabo el control en formación líder-seguidor.

V. Resultados

Para la verificación experimental del esquema líder-seguidor se inicia haciendo que el líder siga la trayectoria 1 que aparece en la Fig. 5(a) y después se ejecuta el nodo del seguidor, en este caso se considera que no hay obstáculos.

El desempeño del esquema de control se evalúa para cada muestra de la siguiente manera,

$$\text{error líder} = \sqrt{(x_d - x_l)^2 + (y_d - y_l)^2} \quad (9)$$

$$\text{error seguidor} = \sqrt{(x_l - x_s)^2 + (y_l - y_s)^2} \quad (10)$$

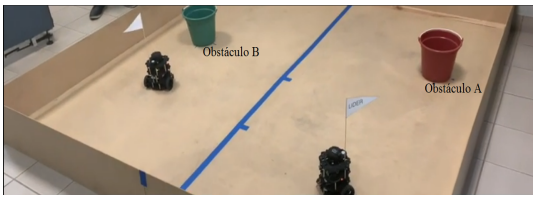
donde, $P_d = (x_d, y_d)$ es la posición deseada, $P_l = (x_l, y_l)$ es la posición del líder y $P_s = (x_s, y_s)$ es la posición del seguidor.

Para realizar las pruebas experimentales se ejecuta el programa *"roslaunch esquema_lider_seguidor lider.launch"* por medio de la computadora iniciando la recolección de datos de las posiciones con el comando *"rosrecord tb3_0/amcl_pose tb3_1/amcl_pose"*. Los datos recabados son usados para calcular el error que existe entre los puntos de la trayectoria predefinida y los puntos recorridos por el robot líder, mostrado en la Fig. 5(b), se tiene que el robot líder alcanza la trayectoria deseada y presenta un error máximo de 1 cm durante el resto del recorrido. Asimismo, en la Fig. 5(b) se muestra el error entre los puntos de la trayectoria alcanzados por el robot líder y los puntos estimados por el robot seguidor, se nota que el robot seguidor reconstruye la trayectoria en 30 muestras, además, como se puede ver en el recuadro, se tiene un error máximo de 2 cm.

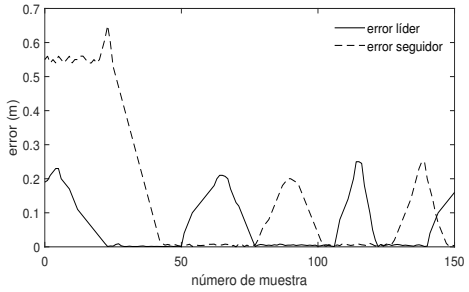
Para validar el desempeño del algoritmo de evasión de obstáculos la prueba consiste en agregar los obstáculos A y B en el área de trabajo (Fig. 6(a)) y ejecutar los algoritmos de seguimiento de trayectoria y evasión de obstáculos. Inicialmente se colocaron ambos robots a 50 cm de distancia uno del otro. Como se puede observar en la Fig. 6(b), el error disminuye cuando se mantiene dentro de la trayectoria y aumenta cuando se aleja de la trayectoria e inicia la evasión del obstáculo, lo cual valida el desempeño adecuado del esquema de control con evasión de obstáculos, debido a que el líder toma como prioridad no colisionar con el objeto y después regresa a la trayectoria deseada. En la Fig. 6(b) se muestra que el robot seguidor también realiza evasión de obstáculos tomando en cuenta las coordenadas del líder. Es importante mencionar que durante el seguimiento de la trayectoria tanto el robot líder como el robot seguidor fueron capaces de evadir los obstáculos e incorporarse a la trayectoria nuevamente.

Para verificar que el esquema de control en formación líder-seguidor es robusto para cambios bruscos del ángulo de dirección se proponen las trayectorias de las Figs. 7(a) y 8(a). Como se observa en las Figs. 7(b) y 8(b) el robot

líder mantiene un error variable de 0 a 2cm, esto se debe a que en los cambios de dirección el robot tiene que orientarse y esto provoca que exista una diferencia con respecto a la trayectoria de referencia. Además, la Fig. 7(b) muestra que el robot seguidor alcanza la trayectoria del líder y también se mantiene oscilando alrededor de ésta con un error máximo de 3cm. Como se puede observar, una limitación del esquema de control líder-seguidor es que en todos los experimentos los errores se mantienen oscilando, es decir, aunque los errores estan acotados no presentan convergencia asintótica a cero.

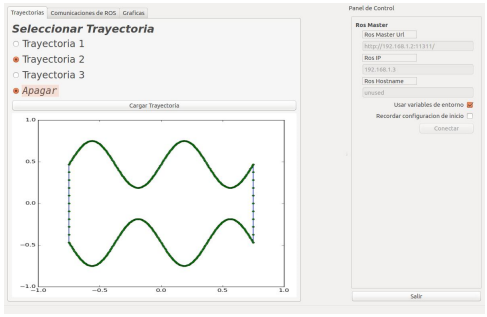


(a)

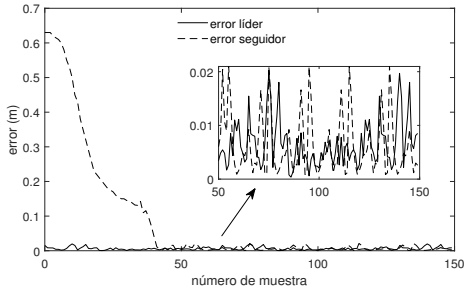


(b)

Fig. 6. Control en formación líder-seguidor con evasión de obstáculos. (a) Escenario con obstáculos A y B, (b) error del líder y error del seguidor.

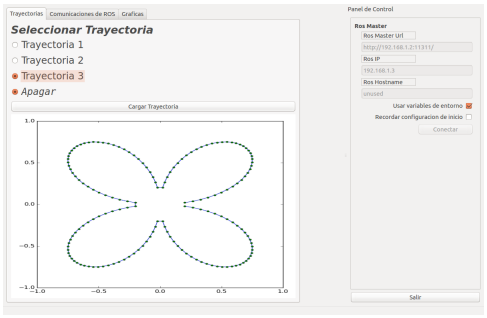


(a)

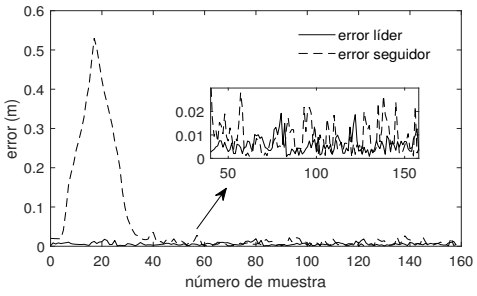


(b)

Fig. 7. Control en formación líder-seguidor. (a) Trayectoria 2, (b) error del líder y error del seguidor.



(a)



(b)

Fig. 8. Control en formación líder-seguidor. (a) Trayectoria 3, (b) error del líder y error del seguidor.

Finalmente, en la Fig. 9 se presenta un experimento para evaluar el comportameniento del esquema de control cuando los robots son colocados inicialmente en extremos opuestos del escenario, es importante notar que el robot seguidor es capaz de reconstruir la trayectoria del líder, sin embargo, la ruta que sigue no es óptima.

VI. Conclusiones

Se realizó la verificación experimental para el control de dos robots móviles TurtleBot3 Burger® con configuración diferencial que desarrollan la tarea del seguimiento de trayectorias y evasión de obstáculos bajo un esquema líder-seguidor. El robot líder lleva a cabo el seguimiento de una trayectoria de referencia y de manera simultánea envía la información de los puntos recorridos al seguidor. Por otra parte, el robot seguidor calcula la trayectoria con base en los datos recibidos. Los resultados presentados muestran el desempeño del esquema de control para tres trayectorias diferentes.

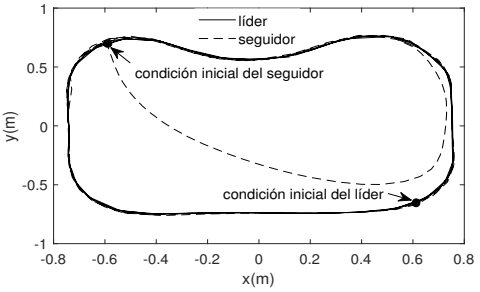


Fig. 9. Comportamiento con respecto a condiciones iniciales.

REFERENCIAS

[1] A. Khan, B. Rinner, A. Cavallaro, "Cooperative robots to observe moving targets," *IEEE Transactions on cybernetics*, vol. 48, no. 1, pp. 187-198, 2016.

[2] J. W. Wang, Y. Guo, M. Fahad, B. Bingham, "Dynamic plume tracking by cooperative robots," *IEEE/ASME Transactions on Mechatronics*, vol. 24, no. 2, pp. 609-620, 2019.

[3] E. Roszkowska, P. Dulewicz, L. Janiec, "Hierarchical hybrid control for multiple mobile robot systems," *IFAC-PapersOnLine*, vol. 52, no. 8, pp. 452-457, 2019.

[4] V. Digani, M. A. Hsieh, L. Sabattini, C. Secchi, "Coordination of multiple AGVs: a quadratic optimization method," *Autonomous Robots*, vol. 43, no. 3, pp. 539-555, 2019.

[5] C. E. Lee, T. K. Sung, "TWR based cooperative localization of multiple mobile robots for search and rescue application," *Journal of Korea Robotics Society*, vol. 11, no. 3, pp. 127-132, 2016.

[6] S. D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. H. Eng, D. Rus, M. H. Ang, "Perception, planning, control, and coordination for autonomous vehicles," *Machines*, vol. 5, no. 1, pp. 1-54, 2017.

[7] X. Zhang, M. Li, J. H. Lim, Y. Weng, Y. Tay, H. Pham, Q. C. Pham, "Large-scale 3D printing by a team of mobile robots," *Automation in Construction*, vol. 95, pp. 98-106, 2018.

[8] A. G. Barrientos, J. L. Lopez, E. S. Espinoza, J. Hoyo, G. Valencia, "Object transportation using a cooperative mobile multi-robot system," *IEEE Latin America Transactions*, vol. 14 no. 3, pp. 1184-1191, 2016.

[9] Z. Peng, S. Yang, G. Wen, A. Rahmani, Y. Yu, "Adaptive distributed formation control for multiple nonholonomic wheeled mobile robots," *Neurocomputing*, vol. 173, pp. 1485-1494, 2016.

[10] R. S. Ortigoza, M. M. Aranda, G. S.Ortigoza, V. M. H. Guzmán, M. A. M. Vilchis, G. S. González, J. C. H. Lozada, M. O. Carbajal, "Wheeled mobile robots: a review," *IEEE Latin America Transactions*, vol. 10, no. 6, pp. 2209-2217, 2012.

[11] M. A. Kamel, X. Yu, Y. Zhang, "Fault-tolerant cooperative control design of multiple wheeled mobile robots," *IEEE Transactions on control systems technology*, vol. 26, no. 2, pp. 756-764, 2018.

[12] L. Dong, Y. Chen, X. Qu, "Formation control strategy for nonholonomic intelligent vehicles based on virtual structure and consensus approach," *Procedia engineering*, vol. 137, pp. 415-424, 2016.

[13] J. Chen, D. Sun, J. Yang, H. Chen, "Leader-follower formation control of multiple non-holonomic mobile robots incorporating a receding-horizon scheme," *The International Journal of Robotics Research*, vol. 29, pp. 727-746, 2010.

[14] C. E. Bugarin, A. Y. Aguilar, "Control visual para la formación de robots móviles tipo uniciclo bajo el esquema líder-seguidor," Ingeniería investigación y tecnología, vol. 15, pp. 593-602, 2013.

[15] H. Wang, D. Guo, X. Liang, W. Chen, G. Hu, K. K. Leang, "Adaptive vision-based leader-follower formation control of mobile robots," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 4, pp. 2893-2902, 2016.

[16] X. Liang, H. Wang, Y. H. Liu, W. Chen, T. Liu, "Formation control of nonholonomic mobile robots without position and velocity measurements," IEEE Transactions on Robotics, 34(2), 434-446.

[17] C. M. Soria, R. Carelli, R. Kelly, J. M. Ibarra-Zannatha, "Coordinated control of mobile robots based on artificial vision," *International Journal of Computers Communications & Control*, vol. 1, no. 2, pp. 85-94, 2006.

[18] L. Consolini, F. Morbidi, D. Prattichizzo, M. Tosques, "Leader–follower formation control of nonholonomic mobile robots with input constraints," Automatica, vol. 44, pp. 1343-1349, 2008.

[19] A. Loria, J. Dasdemir, N. Alvarez Jarquin, "Leader–Follower Formation and Tracking Control of Mobile Robots Along Straight Paths," *IEEE Transactions on Control Systems Technology*, vol. 24, pp. 727-732, 2016.

[20] F. Rubio, F. Valero, C. Llopis-Albert, "A review of mobile robots: Concepts, methods, theoretical framework, and applications," *International Journal of Advanced Robotic Systems*, vol. 16, no. 2, 2019, Art. no. 1729881419839596.

[21] H. Omrane, M. S. Masmoudi, M. Masmoudi, "Fuzzy logic based control for autonomous mobile robot navigation," *Computational intelligence and neuroscience*, vol. 2016, 2016, Art. no. 9548482.

[22] J. Van den Berg, S. J. Guy, M. Lin, D. Manocha, "Reciprocal n-Body Collision Avoidance," in C. Pradalier, R. Siegwart, G. Hirzinger (eds), *Robotics Research*, Springer Tracts in Advanced Robotics, vol. 70, Berlin, Heidelberg: Springer, 2011, pp. 3-19.

[23] I. Karamouzas, B. Skinner, S. Guy, "Universal power law governing pedestrian interactions," *Physical review letters*, vol. 113, no. 23, 2014, Art. no. 238701.

[24] Y. J. Lee, Z. Bien, "Path planning for a quadruped robot: an artificial field approach," *Advanced Robotics*, vol. 16, no. 7, pp. 609-627, 2002.

[25] F. Bayat, S. Najafinia, M. Aliyari, "Mobile robots path planning: electrostatic potential field approach," *Expert Systems with Applications*, vol. 100, pp. 68-78, 2018.

[26] S. M. Rostami, A. K. Sangaiah, J. Wang, X. Liu, "Obstacle avoidance of mobile robots using modified artificial potential field algorithm," *EURASIP Journal on Wireless Communications and Networking*, vol. 1, no. 70, pp. 1-19, 2019, https://doi.org/10.1186/s13638-019-1396-2

[27] M. Pasha, R.A. Riaz, N. Javaid, M. Ilahi, R.D. Khan, "Control strategies for mobile robot with obstacle avoidance," *Journal of Basic and Applied Scientific Research* vol. 3, no. 4, pp. 1027-1036, 2013.

[28] J. Oroko, G. N. Nyakoe, "Obstacle avoidance and path planning schemes for autonomous navigation of a mobile robot: a review," in *Proceedings of the 2012 Mechanical Engineering Conference on Sustainable Research and Innovation*, 2014, pp. 314-318.

[29] G. Şahin, M. Balcilar, E. Uslu, S. Yavuz, M. F. Amasyali, "Obstacle avoidance with Vector Field Histogram algorithm for search and rescue robots," in *22nd Signal Processing and Communications Applications Conference (SIU)*, Trabzon, 2014, pp. 766-769, doi: 10.1109/SIU.2014.6830342.

[30] A. Babinec, M. Dekan, F. Duchoň, A. Vitko, "Modifications of VFH navigation methods for mobile robots," *Procedia Engineering*, vol. 48, pp. 10-14, 2012.

[31] I. Ulrich and J. Borenstein, "VFH/sup */: local obstacle avoidance with look-ahead verification," *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, San Francisco, CA, USA, 2000, pp. 2505-2511 vol.3, doi: 10.1109/ROBOT.2000.846405.

[32] A. Koubâa, *Robot Operating System (ROS): The complete reference (volume 4)*. Studies in Computational Intelligence 831, Springer, 2019.

[33] D. Fox, S. Thrun, W. Burgard, F. Dellaert, "Particle filters for mobile robot localization," in *Sequential Monte Carlo methods in practice*, New York, NY, USA: Springer, 2001, pp. 401-428.

[34] R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.

Juan L. Mata Machuca recibió el grado de Doctor en Ciencias en la Especialidad de Control Automático por el CINVESTAV-IPN, México, en 2013. Actualmente, es profesor investigador de tiempo completo en la Academia de Mecatrónica UPIITA-IPN, miembro Nivel 1 del Sistema Nacional de Investigadores y miembro de la Red de Expertos en Robótica y Mecatrónica del IPN. Es autor y coautor de publicaciones de investigación en revistas, conferencias, capítulos de libros y libros.

Luis F. Zarazua Aguilar recibió el título de Ingeniero en Mecatrónica de la UPIITA-IPN, en 2020. Le fue otorgada la Beca de Estímulo Institucional de Formación de Investigadores (BEIFI) y participó en proyectos de investigación institucionales. Realizó actvidades de investigación en el Laboratorio Institucional de la Red de Expertos en Robótica y Mecatrónica del IPN. Sus áreas de interés son la robótica, el control de sistemas mecatrónicos y la automatización industrial.

Ricardo Aguilar-López recibió el grado de Doctor en Ciencias en Ingeniería Química por la Universidad Autónoma Metropolitana en 1998. Realizó un posdoctorado en el Instituto Mexicano del Petróleo (2000) y además un doctorado en ciencias en la especialidad de control automático en el CINVESTAV-IPN (2003). Es autor y coautor de mas de 90 artículos publicados en revistas internacionales. Actualmente, es investigador en el Departamento de Biotecnología y Bioingeniería del CINVESTAV-IPN y miembro del Sistema Nacional de Investigadores desde 1998, (actualmente, nivel III).