Campus Puebla

Materia

Integración de Robótica y Sistemas Inteligentes TE3003B

Challenge VI

Integrantes

José Jezarel Sánchez Mijares A01735226

Antonio Silva Martínez A01173663

Dana Marian Rivera Oropeza A00830027

Fecha: 17 de mayo de 2024

Contenidos

Resumen

En este reporte vamos a abordar la implementación de algoritmos de evasión de obstáculos en una simulación en Gazebo, específicamente el BUG0 y BUG2, para esto usaremos los diferentes sensores del Puzzlebot, para el envío de información que alimentará a los algoritmos y a su vez al controlador.

Objetivos

- Generar los algoritmos BUG0 y BUG2 para evasión de obstáculos
- Implementación de algoritmos en una simulación en Gazebo donde el Puzzlebot simulado atraviese los mapas dados por Manchester Robotics

Introducción

En el ámbito de la robótica, la navegación reactiva es un método en el que los robots se desplazan sin la necesidad de un mapa predefinido del entorno. En lugar de depender de un conocimiento previo, los robots utilizan información en tiempo real obtenida de sus sensores para detectar y evadir obstáculos. Esta forma de navegación es especialmente útil en entornos desconocidos y dinámicos, donde los robots deben adaptarse continuamente a cambios imprevistos en su entorno.

Dentro de este contexto, los algoritmos BUG se destacan como una solución eficaz para la evasión de obstáculos. Propuestos en la década de los ochenta, estos algoritmos se basan en la detección de obstáculos a través de sensores y la generación de rutas alternativas para alcanzar un destino específico. Entre estos algoritmos, el BUG0 y el BUG2 son fundamentales.

Algoritmos BUG - Los algoritmos BUG son algoritmos de navegación con un enfoque en evasión de obstáculos, trazando diversas rutas según sea el BUG del que estemos hablando, propuestos en los ochentas estos algoritmos recuperan información de los sensores para generar rutas de evasión, cada uno siendo una mejora de la versión anterior, sin embargo

BUG0 - este fue el primero de los algoritmos de tipo BUG, consiste en dirigirse hacia la meta y en caso de encontrar un obstáculo seguirlo hasta ser posible para después seguir su ruta hacia el punto de llegada.

BUG1 - este algoritmo es una mejora de BUG0, busca dirigirse al punto de llegada, sin embargo en caso de encontrar obstáculos los rodea por completo y al llegar al punto donde inició a rodear el obstáculo busca la ruta más óptima para acercarse al punto de llegada y continúa hasta llegar al mismo.

BUG2 - esta es la versión optimizada de los dos BUGs anteriores, este consiste en trazar una ruta directa, comúnmente una línea recta, que va siguiendo hasta encontrarse con un obstáculo, lo rodea hasta volver a encontrar la ruta que trazó en un inicio y continúa hasta llegar al punto de llegada.

El uso de sensores es fundamental para poder cumplir con las tareas que se nos solicitan, siendo uno de estos el sensor lidar (rplidar A1). Los sensores LIDAR (Light Detection and Ranging) se usan para medir distancias por medio del uso de lásers en un entorno, regresando señales en tiempo real, por medio de la luz reflejada busca aproximar las distancias a objetos de manera precisa, normalmente se utilizan para generar “nubes de puntos” que nos dan una idea de lo que rodea al sensor, suelen estar compuestos por un láser que emite pulsaciones veloces de luces infrarrojas, un sensor interno que detecta los lásers que se reflejan y rebotan con los objetos del ambiente y un procesador que calcula el tiempo de rebote para aproximar la distancia de los objetos, además este se encarga de generar la nube de puntos de todas las lecturas que genera.

El Puzzlebot cuenta con un RPLIDAR A1, que es un LIDAR de bajo costo y eficiente funcionamiento que gira en 360° permitiéndonos generar una nube de puntos 2D que se aplica en mapeo, localización y modelación del ambiente, consiste en un sensor LIDAR montado en un sistema de motores que permite al sensor girar y hacer una lectura más completa de su entorno.

En este reporte, abordaremos la implementación de los algoritmos BUG0 y BUG2 en una simulación en Gazebo utilizando el Puzzlebot. Este robot, equipado con un sensor LIDAR RPLIDAR A1, captura datos del entorno para alimentar los algoritmos y el controlador de navegación. El RPLIDAR A1, un sensor eficiente y de bajo costo, gira 360° para generar una nube de puntos 2D, proporcionando una visión detallada del entorno para aplicaciones de mapeo, localización y modelación.

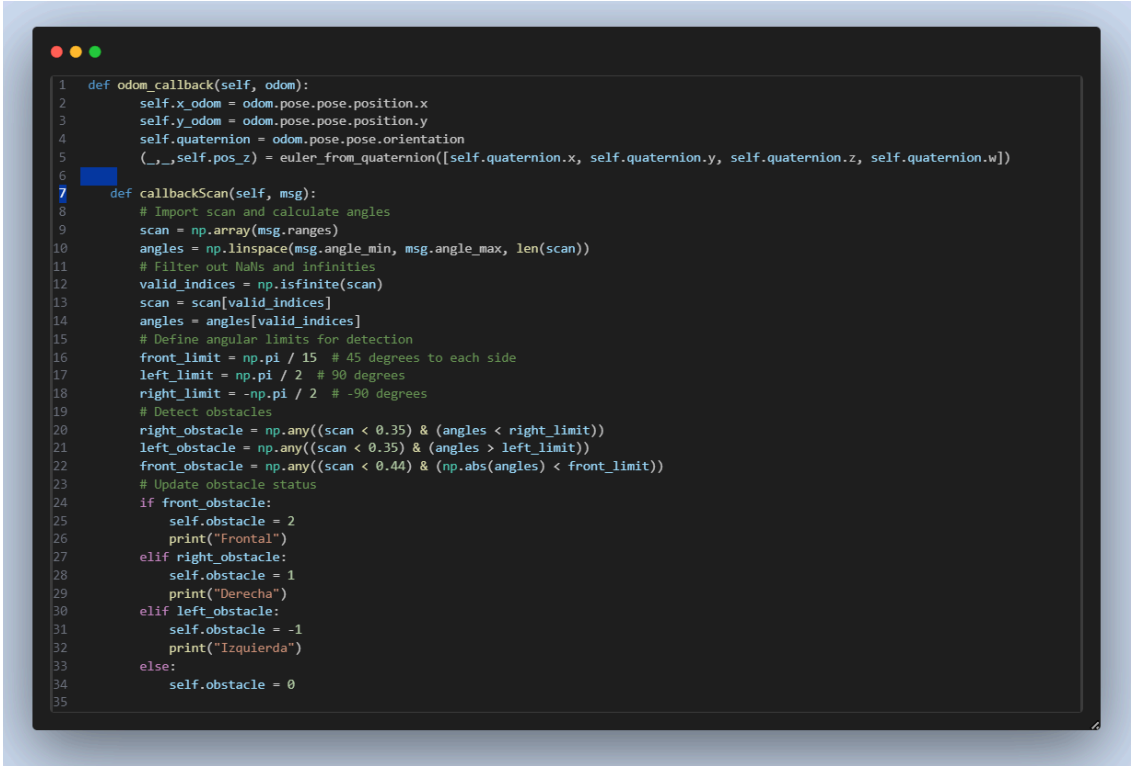
Solución del challenge

La estructura del código para de los algoritmo bug es similar ya que la estructura que se sigue cuenta con dos clases de las cuales comparten la del PID.

El funcionamiento del controlador PID es el siguiente

Se crea la clase Control PID implementando un controlador para ajustar la velocidad del robot según el error en distancia y orientación, esta se inicializa con } las constantes del PID (kp, ki, kd), el tiempo de muestreo (dt), y otras variables necesarias para el cálculo del error. Posteriormente el método points_callback actualiza las coordenadas del objetivo hacia el cual el robot debe dirigirse, mientras que el método get_pid devuelve la predicción ajustada para corregir la trayectoria del robot.

El funcionamiento del los algoritmos es el siguiente:
La clase Bugs implementa el algoritmo Bug0 para la navegación del robot. Esta clase maneja la suscripción a los tópicos relevantes de ROS, publica comandos de velocidad y ejecuta la lógica del algoritmo Bug0. El constructor de la clase Bugs inicializa el nodo ROS, configura los suscriptores y publicadores, y define las variables necesarias para la navegación.
Los métodos odom_callback y callbackScan actualizan la posición del robot y detectan obstáculos mediante la odometría y el sensor LIDAR, respectivamente.

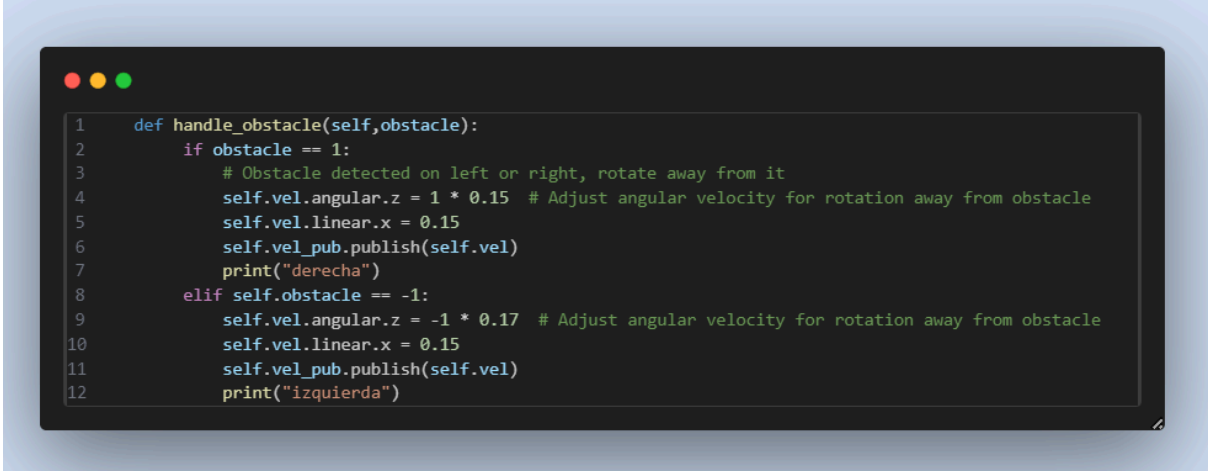


```python
def odom_callback(self, odom):
        self.x_odom = odom.pose.pose.position.x
        self.y_odom = odom.pose.pose.position.y
        self.quaternion = odom.pose.pose.orientation
        (_,_,self.pos_z) = euler_from_quaternion([self.quaternion.x, self.quaternion.y, self.quaternion.z, self.quaternion.w])

    def callbackScan(self, msg):
        # Import scan and calculate angles
        scan = np.array(msg.ranges)
        angles = np.linspace(msg.angle_min, msg.angle_max, len(scan))
        # Filter out NaNs and infinities
        valid_indices = np.isfinite(scan)
        scan = scan[valid_indices]
        angles = angles[valid_indices]
        # Define angular limits for detection
        front_limit = np.pi / 15  # 45 degrees to each side
        left_limit = np.pi / 2  # 90 degrees
        right_limit = -np.pi / 2  # -90 degrees
        # Detect obstacles
        right_obstacle = np.any((scan < 0.35) & (angles < right_limit))
        left_obstacle = np.any((scan < 0.35) & (angles > left_limit))
        front_obstacle = np.any((scan < 0.44) & (np.abs(angles) < front_limit))
        # Update obstacle status
        if front_obstacle:
            self.obstacle = 2
            print("Frontal")
        elif right_obstacle:
            self.obstacle = 1
            print("Derecha")
        elif left_obstacle:
            self.obstacle = -1
            print("Izquierda")
        else:
            self.obstacle = 0

```

El método handle_obstacle ajusta la velocidad y dirección del robot para evitar obstáculos detectados por el sensor LIDAR.

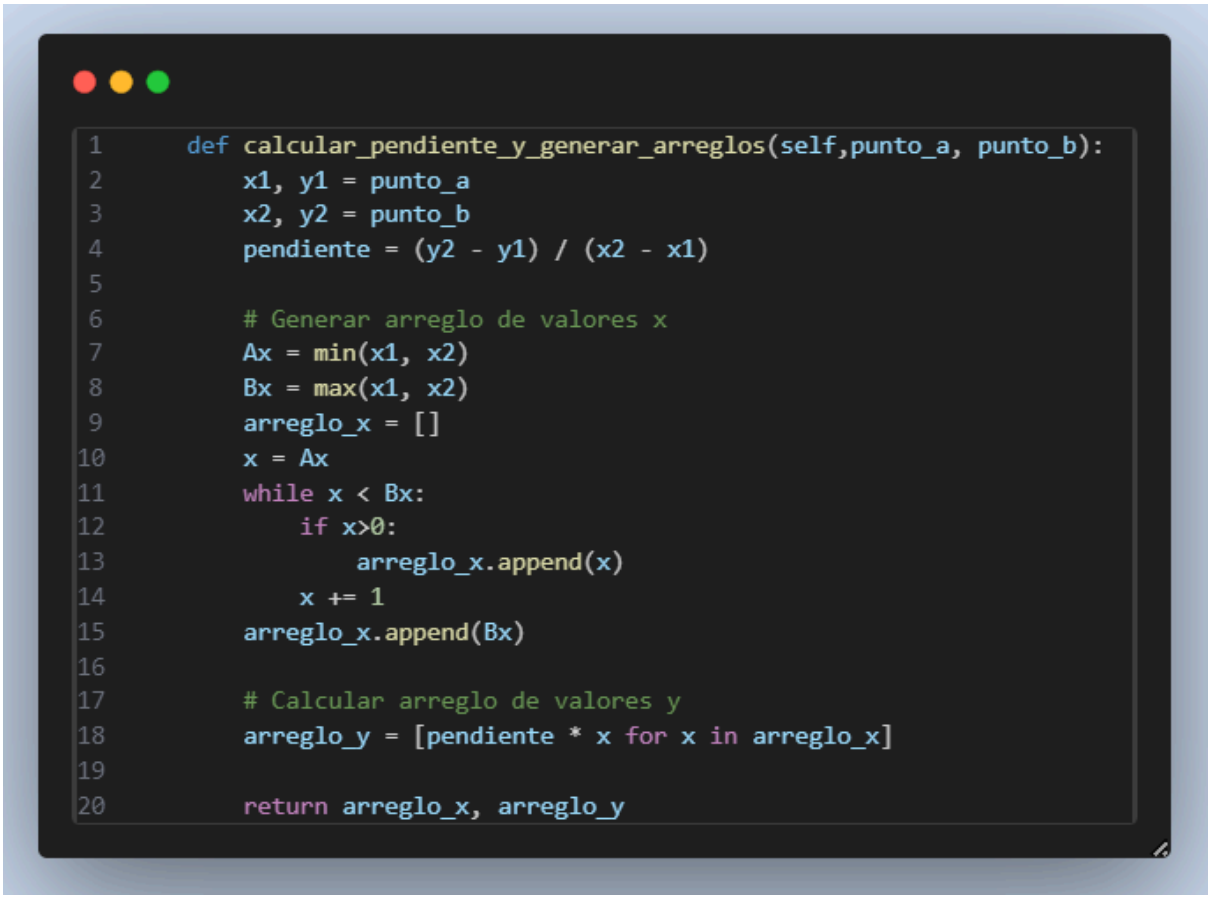


```python
    def handle_obstacle(self,obstacle):
        if obstacle == 1:
            # Obstacle detected on left or right, rotate away from it
            self.vel.angular.z = 1 * 0.15  # Adjust angular velocity for rotation away from obstacle
            self.vel.linear.x = 0.15
            self.vel_pub.publish(self.vel)
            print("derecha")
        elif self.obstacle == -1:
            self.vel.angular.z = -1 * 0.17  # Adjust angular velocity for rotation away from obstacle
            self.vel.linear.x = 0.15
            self.vel_pub.publish(self.vel)
            print("izquierda")
```

El método run() coordina la navegación del robot, asegurando que avance hacia sus objetivos mientras evita obstáculos, todo ello mediante la combinación de controladores PID y la lógica del algoritmo Bug0.

Bug 2

La clase Bugs se encarga de gestionar la comunicación con ROS, suscribiéndose a los tópicos relevantes y publicando comandos de velocidad. Los métodos odom_callback y callbackScan actualizan la posición del robot y detectan obstáculos mediante la odometría y el escaneo láser, respectivamente.



```python
def calcular_pendiente_y_generar_arreglos(self,punto_a, punto_b):
    x1, y1 = punto_a
    x2, y2 = punto_b
    pendiente = (y2 - y1) / (x2 - x1)

    # Generar arreglo de valores x
    Ax = min(x1, x2)
    Bx = max(x1, x2)
    arreglo_x = []
    x = Ax
    while x < Bx:
        if x>0:
            arreglo_x.append(x)
        x += 1
    arreglo_x.append(Bx)

    # Calcular arreglo de valores y
    arreglo_y = [pendiente * x for x in arreglo_x]

    return arreglo_x, arreglo_y
```

El método calcular_pendiente_y_generar_arreglos calcula la pendiente entre dos puntos dados, lo que permite determinar la dirección en la que se debe mover el robot para seguir una trayectoria recta entre esos puntos. Luego, genera una serie de puntos a lo largo de esta línea recta, lo que proporciona una ruta clara para que el robot siga.

Diferencias entre ambos algoritmos

El algoritmo Bug0 es una estrategia básica de navegación para robots móviles que se utiliza para guiar un robot desde su posición actual hasta un objetivo en un entorno con obstáculos. Su enfoque es relativamente simple: el robot se desplaza hacia el objetivo en línea recta y, si encuentra un obstáculo en su camino, lo rodea siguiendo el contorno del obstáculo hasta que puede volver a dirigirse hacia el objetivo en línea recta. Este enfoque puede ser efectivo en entornos simples, pero puede resultar en un comportamiento subóptimo en entornos más complejos, especialmente cuando hay obstáculos que podrían hacer que el robot se desvíe significativamente de su ruta hacia el objetivo.

Por otro lado, el algoritmo Bug2 es una mejora del Bug0 que aborda una limitación importante del Bug0: la posibilidad de que el robot quede atrapado en ciertos tipos de obstáculos. En Bug2, si el robot encuentra un obstáculo que lo obliga a alejarse del objetivo, se recordará la posición en la que se encontró el obstáculo y se continuará moviendo hacia el objetivo. Si encuentra otro obstáculo y se encuentra más cerca del objetivo que en el primer obstáculo, el robot abandonará el contorno del segundo obstáculo y se moverá hacia el objetivo. Si el segundo obstáculo lo aleja aún más del objetivo, el robot continuará

siguiendo el contorno del segundo obstáculo. Este proceso se repite hasta que el robot alcanza el objetivo o se da cuenta de que no puede llegar al objetivo debido a obstáculos.

Videos

Bug0: https://youtu.be/pKD4ys-JOcI
Bug2: https://youtu.be/m7CxLtQLxZU

Conclusiones

La navegación reactiva es un amplio campo de investigación dentro de la robótica, nos presenta diversos retos como la generación de algoritmos que sean capaces de identificar obstáculos y reaccionar en tiempos óptimos para evitar colisiones, sin embargo es una de las ramas de la navegación que ha adquirido mayor importancia en los últimos años con el incremento del interés en los vehículos autónomos y la búsqueda constante de su implementación en entornos abiertos donde no se puede predecir de manera segura los retos que se van a enfrentar, es por esto que su estudio y desarrollo es de gran importancia en la actualidad. En este caso logramos cumplir con los objetivos que planteamos al inicio de este reporte generando algoritmos que funcionaran de manera correcta en la simulación de Gazebo haciendo uso de los sensores que conforman al Puzzlebot, sin embargo aún queda lugar a la mejora específicamente en los algoritmos, ya que por si solos se pueden encontrar con situaciones en las que no sean capaces de completar los recorridos propuestos o que lo hagan con un alto uso de recursos.

Referencias

Navarrete-Sánchez, M. A., Vite-Chávez, O., Olivera-Reyna, Ro., Olivera-Reyna, Re., Mercado-Pérez, S., Muñoz-Minjares, J. U., & Marín-Hernández, A. (2019). Navegación Reactiva de un Robot aplicando una estrategia bifásica en una Raspberry Pi 3. Recuperado de https://www.researchgate.net/publication/344455019_Navegacion_Reactiva_de_un_Robot_aplicando_una_estrategia_bifasica_en

IBM. (2022). LIDAR. Recuperado de https://www.ibm.com/mx-es/topics/lidar

RPLIDAR A1M8 360 Degree Laser Scanner Development Kit Datasheet. (2016). Recuperado de https://www.generationrobots.com/media/rplidar-a1m8-360-degree-laser-scanner-development-kit-datasheet-1.pdf