Campus Puebla

Materia

Integración de Robótica y Sistemas Inteligentes TE3003B

Challenge V

Integrantes

José Jezarel Sánchez Mijares A01735226

Antonio Silva Martínez A01173663

Dana Marian Rivera Oropeza A00830027

Fecha: 11 de mayo de 2024

Contenidos

Resumen

En este reporte hablaremos sobre el proceso para calcular los valores de kl y kr, las pruebas que generamos usando el Puzzlebot físico para calcular la incertidumbre en la posición del robot, haciendo referencia a la variación de la posición de las ruedas al momento de desplazarse de manera lineal y angular.

Objetivos

- Calcular los valores de kl y kr
- Mejorar la elipse de confianza

Introducción

Matriz de covarianza para localización - La matriz de covarianza, en este contexto, es una manera de representar las diferentes ecuaciones y variables de posición de una sola función, siendo esta una manera más sencilla de representar posiciones en x, y y la orientación del mismo, normalmente es empleada para relacionar estas ecuaciones de posición y el incremento o decremento que presentan como una unidad.

Odometry - En el contexto de ROS los mensajes de odometría sirven para enviar información de la posición del robot, mostrándonos valores de ubicación y posición estimados a partir de los diferentes sensores con los que cuenta el robot, son una normativa para la comunicación en robots estandarizando el envío de información relativa al movimiento del mismo, combinando las señales enviadas por los sensores para obtener un estimado de la posición y enviar la información a diferentes nodos como el controlador para generar el error y corregir el movimiento.

Pose Message - Dentro de ROS, los mensajes de pose nos permiten conocer la orientación y posición de un robot o del objeto que estemos analizando, además de los tres valores de posición y orientación nos regresa información adicional como

Elipse de confidencia - Los elipses de confianza son un concepto utilizado principalmente en estadística y probabilidad, se refiere al rango dentro del cual se puede considerar un valor como “bueno”, se genera un rango de valores a partir de la media obtenida y un rango de confianza donde los datos dentro de este rango son considerados como válidos, en este reporte vamos a considerar la elipse de confianza como una representación gráfica de la confianza que se tiene en los valores dados y buscamos que el movimiento de el Puzzlebot esté dentro de estos rangos de confianza tanto en la simulación como en el movimiento físico que se genere.

Solución del challenge

Anteriormente generamos el modelo cinemático del robot discretizando los valores, esto para poder adaptar el modelo ideal a un modelo que incluya incertidumbres, kl y kr, estos valores nos ayudan a calcular la posición de las llantas en diferentes momentos contemplando la incertidumbre Q_k , esta es dada por la ecuación:

$$Q_k = \bigtriangledown_{wk} \Sigma_{\Delta k} \bigtriangledown_{wk}^T$$

Donde la covarianza *k* se puede obtener a partir de las constantes *kr kl* relacionados con el error pasado presentado en los encoders de las llantas

$$\Sigma_{\Delta,k} = \begin{bmatrix} k_r|\omega_{r,k}| & 0 \\ 0 & k_l|\omega_{l,k}| \end{bmatrix}$$

Para el cálculo de los valores kr y kl, se llevaron a cabo diversas mediciones y experimentos utilizando tanto el robot físico como simulaciones en RVIZ. En los experimentos físicos, se generaron casos de experimentación marcando el avance lineal del robot al enviar la misma velocidad y medir las variaciones de posición, lo que permitió calcular el error lineal. Para este cálculo, se creó un plano de referencia físico y se registró el movimiento real y su variación con respecto al plano a lo largo de 15 experimentos.

Para el error angular, se siguió un principio similar midiendo las variaciones con respecto al plano de referencia utilizando un transportador, lo que generó un error más preciso en los experimentos.

Además, se crearon simulaciones en RVIZ para verificar que los valores de kr y kl caen dentro de la elipse de confianza calculada, ajustada con los nuevos valores obtenidos de las pruebas anteriores, confirmando así la validez del movimiento.

Para poder comprobar que las mediciones fueran correctas se realizó una simulación en gazebo en donde se realizaron las mismas pruebas que con el robot real, siendo que de igual forma que con el robot real se obtuvieron una regresión lineal para poder visualizar que tan preciso es nuestro robot.

Los códigos que se destacan en la elaboración de esta actividad fueron el controlador, siendo que en este script está diseñado para enviar comandos de velocidad a un robot móvil a través de ROS. Utiliza un enfoque de lazo abierto, lo que significa que no tiene en cuenta la retroalimentación del sistema para ajustar los comandos de velocidad en función de la posición o el estado actual del robot. En lugar de eso, simplemente envía comandos de velocidad constantes durante un tiempo específico.

El controlador comienza iniciando un nodo ROS con el nombre 'send_velocity_node'. Luego, crea un publicador (vel_pub) que publica mensajes de tipo Twist. Estos mensajes de Twist contienen información sobre la velocidad lineal y angular que se enviarán al robot.

```python
def send_velocity():
    rospy.init_node('send_velocity_node', anonymous=True)
    vel_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
    rate = rospy.Rate(100)  # 10 Hz

    # Crear el mensaje Twist
    twist_msg = Twist()
    twist_msg.linear.x = 0.15 # Velocidad lineal deseada en m/s
    twist_msg.angular.z = 0.005

    start_time = time.time()

    while not rospy.is_shutdown() and time.time() - start_time < 23.0:
        vel_pub.publish(twist_msg)
        rate.sleep()

    # Detener el robot al finalizar los 4 segundos
    twist_msg.linear.x = 0.0
    twist_msg.angular.z = 0.0

    vel_pub.publish(twist_msg)

if __name__ == '__main__':
    try:
        send_velocity()
    except rospy.ROSInterruptException:
        pass
```

Código Covarianza

Para el cálculo de la odometría se implementó las siguientes mejoras en nuestro código:
Se agregó la siguiente función para el cálculo de las covarianzas del sistema

```python
def get_covariance(self, dt):
    gra_wk = 0.5 * self.radius * dt * np.array([[np.cos(self.theta), np.cos(self.theta)],[np.sin(self.theta), np.sin(self.theta)],[2.0/self.wheelbase, -2.0/self.wheelbase]])

    self.cov_delta_q = np.array([[self.kr * abs(self.wr_speed), 0.0], [0.0, self.kl * abs(self.wl_speed)]])

    self.Qk_matrix = np.matmul(np.matmul(gra_wk,self.cov_delta_q), np.transpose(gra_wk))

    self.H_matrix = np.array([[1.0,0.0, -dt * self.vel * np.sin(self.theta)],[0.0,1.0,dt * self.vel * np.cos(self.theta)],[0.0,0.0,1.0]])

    self.covariance_matrix = np.matmul(np.matmul(self.H_matrix, self.covariance_matrix),np.transpose(self.H_matrix)) + self.Qk_matrix

    return self.covariance_matrix

```

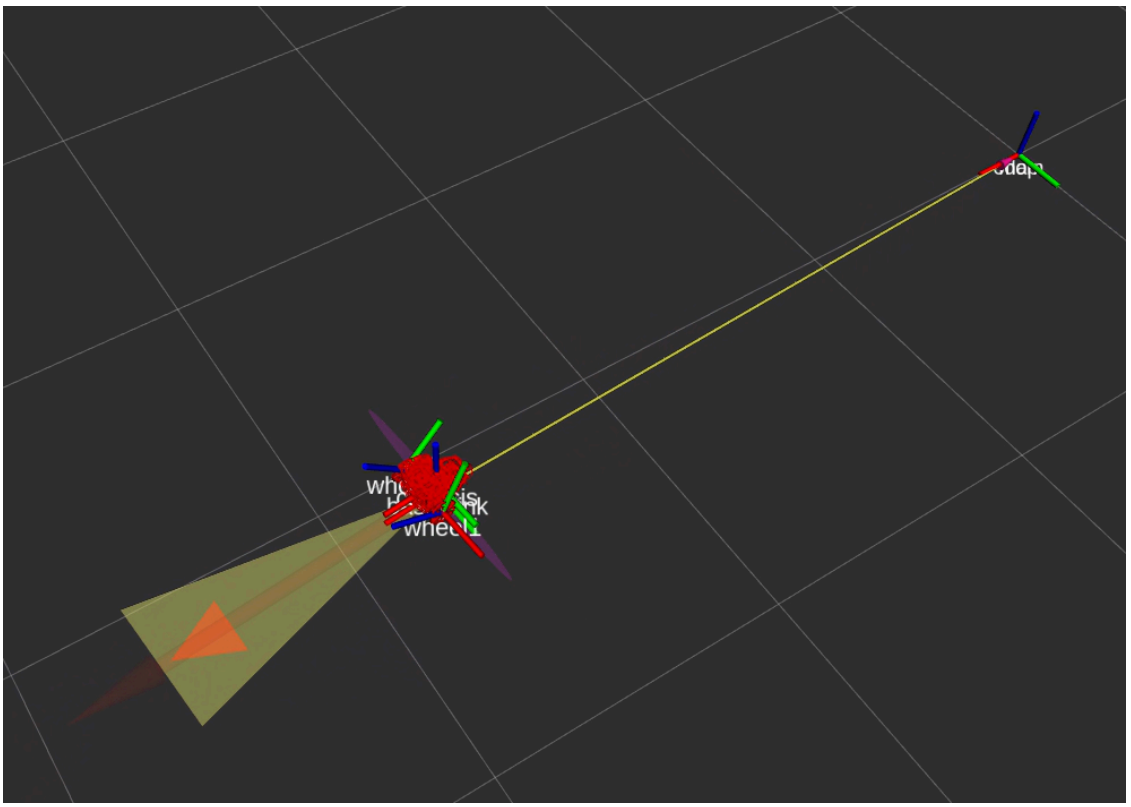
Ahora agregamos una matriz de covarianza (generada por la función get_covariance) a nuestra función “get_odometry”, lo que nos permitió publicar la covarianza y obtener la elipse de confianza

```python
def get_odometry(self,current_time,x,y,theta,covarianza):
        self.odom_msg.header.stamp = current_time
        self.odom_msg.header.frame_id = "odom" #or odom
        self.odom_msg.child_frame_id = "base_link"
        self.odom_msg.pose.pose.position.x = x
        self.odom_msg.pose.pose.position.y = y
        quaternion=Quaternion(*quaternion_from_euler(0,0,theta))
        self.odom_msg.pose.pose.orientation= quaternion

        #
        co = np.zeros((6,6),dtype=float)
        co[:2,:2] = covarianza[:2,:2]
        co[-1,:2] = covarianza[-1,:2]
        co[:2,-1] = covarianza[:2,-1]
        co[-1,-1] = covarianza[-1,-1]
        self.odom_msg.pose.covariance = co.reshape(36).tolist()
        self.odom_msg.twist.twist.linear.x=self.vel
        self.odom_msg.twist.twist.angular.z= self.w
        return self.odom_msg

```

Finalmente corremos las funciones en conjunto para mandarlo a odometría

```python
def calculate_odometry(self):
    current_time = rospy.Time.now()  # Get current time

    if self.first:
        self.previous_time = current_time
        self.first = False
    else:
        dt = (current_time - self.previous_time).to_sec()  # get dt
        self.previous_time = current_time

        #Get the covariance
        covariance_matrix=self.get_covariance(dt)

        #get the pose of the robot
        self.get_positon(self.wr_speed,self.wl_speed,dt)

        # Create Odometry message
        odom_msg = self.get_odometry(current_time,self.x,self.y,self.theta,covariance_matrix)

        # Publish Odometry message
        self.odom_pub.publish(odom_msg)

        # Publish transform
        self.transform(odom_msg)
```

Movimiento del Puzzlebot con la elipsoide de confianza tuneado con las medidas realizadas



Video

https://youtu.be/yAKRGK0y_0U

Conclusiones

La matriz de covarianza es una herramienta crucial para representar la incertidumbre en las posiciones y orientaciones de un robot. Nos permite relacionar las ecuaciones y variables de posición en una sola función, simplificando la representación de la información espacial. En este contexto, la matriz de covarianza nos ayuda a entender cómo varían las posiciones en x, y, y la orientación, y cómo estas variaciones están relacionadas entre sí.

Las elipses de confianza son una representación gráfica de la confianza que tenemos en los valores de posición y orientación de un robot. Nos permiten visualizar el rango de valores que consideramos "buenos" o válidos, basados en la media y un intervalo de confianza. En nuestro caso, utilizamos la elipse de confianza para verificar que el movimiento del robot esté dentro de estos rangos durante las simulaciones y pruebas físicas.

En cuanto a la solución del desafío de la odometría, implementamos mejoras en nuestro código para calcular las covarianzas del sistema y publicarlas junto con la odometría. Esto nos permitió obtener una elipse de confianza que representa la precisión de la estimación de la posición del robot, ayudándonos a verificar la validez de nuestras mediciones y ajustar los valores de kr y kl para mejorar la precisión del robot.

Referencias

Martínez, M. M. (2024). MCR2_RandomVariables_Linearisation. Manchester Robotics. https://github.com/ManchesterRoboticsLtd/TE3003B_Integration_of_Robotics_and_Intelligent_Systems/blob/main/Week%203/Presentations/MCR2_RandomVariables_Linearisation.pdf

Nikolaus Correll, Bradley Hayes, Christoffer Heckman and Alessandro Roncone. Introduction to Autonomous Robots: Mechanisms, Sensors, Actuators, and Algorithms, MIT Press, 2022 (forthcoming).