Campus Puebla

Materia

Integración de Robótica y Sistemas Inteligentes TE3003B

Challenge IV

Integrantes

José Jezarel Sánchez Mijares A01735226

Antonio Silva Martínez A01173663

Dana Marian Rivera Oropeza A00830027

Fecha: 03 de mayo de 2024

Contenidos

Resumen

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.

En este reporte vamos a profundizar en la aplicación de los elipses de confianza en el Puzzlebot para generar recorridos más certeros, mostrando la generación del modelo cinemático, su linealización, la generación de los jacobianos y finalmente la generación de la elipse de confianza para el modelaje.

Objetivos

- Mover el robot con un controlador de lazo abierto en línea recta y recopilar los datos de posición obtenidos
- Girar el robot de un ángulo inicial a un ángulo final y obtener los datos de posición final
- Generar la experimentación con el robot real, Gazebo y la simulación
- Mostrar la elipse de confianza en la simulación y generar un recorrido de punto a punto dentro de la simulación

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.

Jacobianos para relacionar la pose del robot (x, y, theta) - Relaciona los cambios en las variables de la posición del robot (x, y, theta) con los cambios en la velocidad, en este caso se puede hacer la relación de dos maneras, ya sea directa o inversa, en la primera relacionando los cambios con las velocidades lineales y angulares dadas tomando como ejemplo el caso del robot diferencial y en el segundo de los casos cuando se toma en cuenta un punto al que se quiere llegar y se busca obtener las velocidades necesarias para llegar a este punto ya sea en un tiempo dado o simplemente para llegar de un punto A a un punto B.

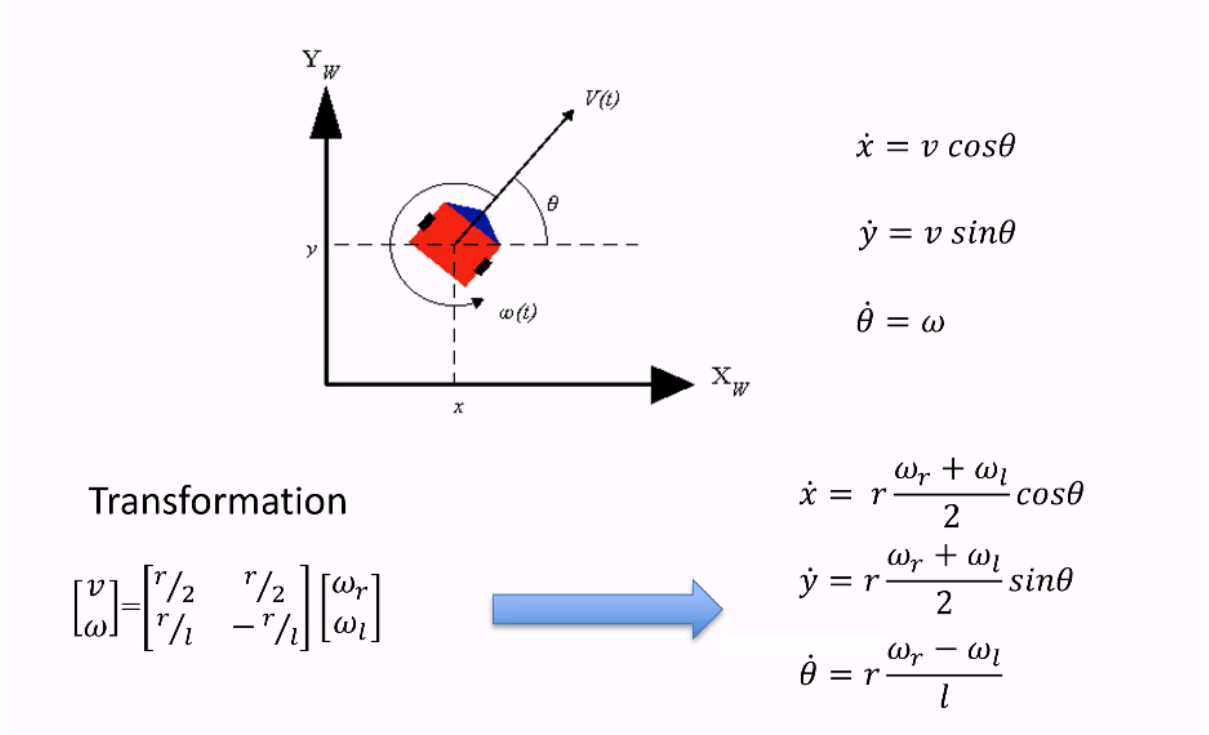
Mensaje tipo odometry - En el contexto de ROS los mensajes de odometría sirven para enviar información de la posición del robot, mostrándonos valores de ubicación y posición

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.

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

Para comenzar a resolver este reto, partiremos del análisis del modelo cinemático del robot que está dado por:



Con el objetivo de poder programar su comportamiento es necesario hacer un proceso de discretización usando el método de euler:

$$\mathbf{h}(\mathbf{s}_{k-1}, \mathbf{u}_k) = \begin{bmatrix} s_{x,k-1} + \Delta t \cdot v_k . \cos(s_{\theta,k-1}) \\ s_{y,k-1} + \Delta t \cdot v_k . \sin(s_{\theta,k-1}) \\ s_{\theta,k-1} + \Delta t \cdot \omega_k \end{bmatrix}$$

El modelo $h(S_{k-1}, U_k)$ es un modelo ideal, por lo tanto en el mundo real no nos conviene usar este tipo de sistemas, por eso, es necesario tomar en cuenta las incertidumbres mediante un modelo de distribución Gaussiana. Lo representamos a continuación:

$$S_k \sim N(\mu_k, \Sigma_k)$$

Donde μ es la media y la Σ es la matriz de covarianza por lo tanto nuestro modelo con incertidumbres se representa de la siguiente manera:

$$S_k = h(S_{k-1}, u_k) + q_k$$

Donde q_k es el ruido, y tiene la siguiente distribución gaussiana:

$$q_k \sim N(0, Q_k)$$

Donde Q_k grande es la matriz de covarianza del ruido resolviendo esta ecuación nuestro S_k al ser dependiente de *h* nuestra distribución ya no es gaussiana por lo tanto tenemos que linealizar *h* en nuestro modelo cinemático con incertidumbres.

$$S_k = h(S_{k-1}, u_k) + q_k$$

Recordando las transformaciones afines que mencionan que si una variable gaussiana *X* con distribución normal

$$Y=AX+Bu$$

Es una transformación lineal, nuestra nueva variable aleatoria *Y* tendrá una distribución gaussiana normal con esto podemos calcular la media y la matriz de covarianza de *Y*

$$\mu_y = A\mu_X + b$$
$$\Sigma_y = A\Sigma_X A^T$$

Para linealizar el sistema discreto tenemos usamos la matriz jacobiana, donde derivamos parcialmente con respecto a los estados que tenemos, obtenemos lo siguiente

$$H_k = \nabla_{Sk} h(s_{k-1}, u_k)|_{s_{k-1}=\mu_{k-1}}$$

$$\mathbf{h}(\mathbf{s}_{k-1}, \mathbf{u}_k) = \begin{bmatrix} s_{x,k-1} + \Delta t \cdot v_k . \cos(s_{\theta,k-1}) \\ s_{y,k-1} + \Delta t \cdot v_k . \sin(s_{\theta,k-1}) \\ s_{\theta,k-1} + \Delta t \cdot \omega_k \end{bmatrix}$$

$$\mathbf{H}_k = \begin{bmatrix} \frac{\partial h_1}{\partial s_{x,k}} & \frac{\partial h_1}{\partial s_{y,k}} & \frac{\partial h_1}{\partial s_{\theta,k}} \\ \frac{\partial h_2}{\partial s_{x,k}} & \frac{\partial h_2}{\partial s_{y,k}} & \frac{\partial h_2}{\partial s_{\theta,k}} \\ \frac{\partial h_3}{\partial s_{x,k}} & \frac{\partial h_3}{\partial s_{y,k}} & \frac{\partial h_3}{\partial s_{\theta,k}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\Delta t \cdot v_k \cdot \sin(\mu_{\theta,k-1}) \\ 0 & 1 & \Delta t \cdot v_k \cdot \cos(\mu_{\theta,k-1}) \\ 0 & 0 & 1 \end{bmatrix}$$

Por lo tanto podemos linealizar el sistema usando series de taylor de primer orden sobre μ, u_k como se muestra a continuación:

$$\mu_k = h(\mu_{k-1}, u_k)$$

Y la pose se puede obtener usando el sistema linealizado donde s_k es una variable gaussiana

$$s_k \approx \mu_k + H_k(s_{k-1} - \mu_{k-1})$$

Ahora estos conceptos los podemos poner en práctica para nuestro Robot, sabiendo que las incertidumbres son gaussianas es posible calcular la covarianza relacionada con la posición del robot

$$\Sigma_k = H_k \Sigma_{k-1} H_k^T + Q_k$$

H jacobiano linealizado
Covarianza en el instante anterior
H transpuesta
Más el ruido para cada instante de tiempo
covarianza = incertidumbre

Nosotros sabemos que en el momento inicial la pose de nuestro robot es 0, así podemos determinar que la covarianza es 0

$$\mu_0 = 0, \; and \; \Sigma_0 = 0$$

Las incertidumbres entre x,y es representada por una elipse alrededor del robot, llamada elipse de confianza. mientras el robot se mueva a lo largo del eje x las incertidumbres en y aumentaran más rápido que en y debido a un error de drift.

El objetivo del reto es generar la elipsoide de confianza mediante las covarianzas dadas por una relación entre las incertidumbres y el comportamiento del sistema que se explicó previamente.
Sabiendo que la covarianza está dada por la siguiente ecuación:

$$\Sigma_1 = H_1 \Sigma_0 H_1^T + Q_1$$

Podemos obtenerla mediante los siguientes pasos

Paso 1
Calcular la posición para cada instante dado por la siguiente ecuación:

$$\boldsymbol{\mu_1} = \mathbf{h}(\boldsymbol{\mu_0}, \mathbf{u_1})$$

$$\boldsymbol{\mu_1} = \begin{bmatrix} s_{x,1} \\ s_{y,1} \\ s_{\theta,1} \end{bmatrix} = \begin{bmatrix} s_{x,0} + \Delta t \cdot V_1 \cdot \cos(s_{\theta,0}) \\ s_{y,0} + \Delta t \cdot V_1 \cdot \sin(s_{\theta,0}) \\ s_{\theta,0} + \Delta t \cdot \omega_1 \end{bmatrix}$$

Paso 2:
Obtener la matriz del sistema lineal para cada instante de tiempo

$$\boldsymbol{H_1} = \begin{bmatrix} 1 & 0 & -\Delta t \cdot V_1 \cdot \sin(s_{\theta,0}) \\ 0 & 1 & \Delta t \cdot V_1 \cdot \cos(s_{\theta,0}) \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0.1 \\ 0 & 0 & 1 \end{bmatrix}$$

Paso 3:
Calcular la Q_k que representa las incertidumbres. Este cálculo está relacionado con los encoders ya que estos son los sensores con los que estamos trabajando que se encargan de darnos información, debido a que estos no son perfectos, obtenemos incertidumbres (Q_k) que calculamos de la siguiente manera:

$$Q_k = \nabla_{\omega_k} \cdot \Sigma_{\Delta,k} \cdot \nabla_{\omega_k}^T$$

La covarianza k podemos obtenerla de la siguiente manera donde *kr kl* son constantes relacionadas con el error presentado en los encoders de las llantas.

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

Finalmente para completar el cálculo de la *Qk* tenemos que calcular el modelo linealizado con respecto a las llantas

$$\nabla_{\omega_k} = \begin{bmatrix} \frac{\partial h_1}{\partial \omega_{r,k}} & \frac{\partial h_1}{\partial \omega_{l,k}} \\ \frac{\partial h_2}{\partial \omega_{r,k}} & \frac{\partial h_2}{\partial \omega_{l,k}} \\ \frac{\partial h_3}{\partial \omega_{r,k}} & \frac{\partial h_3}{\partial \omega_{l,k}} \end{bmatrix} = \frac{1}{2} r \Delta t \begin{bmatrix} \cos(s_{\theta,k-1}) & \cos(s_{\theta,k-1}) \\ \sin(s_{\theta,k-1}) & \sin(s_{\theta,k-1}) \\ \frac{2}{l} & -\frac{2}{l} \end{bmatrix}$$

Así podemos sustituir de nuevo en la ecuación $\Sigma_1 = H_1 \Sigma_0 H_1^T + Q_1$ para obtener la covarianza en ese momento, ese cálculo se realizará en cada ejecución de nuestro código.

Calculamos la propagación de la incertidumbre usando el resultado de la transformación de afinidad

$$\boldsymbol{\Sigma}_1 = \mathbf{H}_1 \cdot \boldsymbol{\Sigma}_0 \cdot \mathbf{H}_1^T + \mathbf{Q}_1$$

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0.1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0.1 & 1 \end{bmatrix} + \begin{bmatrix} 0.5 & 0.01 & 0.01 \\ 0.01 & 0.5 & 0.01 \\ 0.01 & 0.01 & 0.2 \end{bmatrix}$$

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.5 & 0.01 & 0.01 \\ 0.01 & 0.5 & 0.01 \\ 0.01 & 0.01 & 0.2 \end{bmatrix}$$

$$\boldsymbol{\Sigma}_2 = \mathbf{H}_2 \cdot \boldsymbol{\Sigma}_1 \cdot \mathbf{H}_2^T + \mathbf{Q}_2$$

$$\boldsymbol{\Sigma}_2 = \begin{bmatrix} 1 & 0 & -0.01 \\ 0 & 1 & 0.0995 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.5 & 0.01 & 0.01 \\ 0.01 & 0.5 & 0.01 \\ 0.01 & 0.01 & 0.2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -0.01 & 0.0995 & 1 \end{bmatrix} + \begin{bmatrix} 0.5 & 0.01 & 0.01 \\ 0.01 & 0.5 & 0.01 \\ 0.01 & 0.01 & 0.2 \end{bmatrix}$$

$$\boldsymbol{\Sigma}_2 = \begin{bmatrix} 0.998 & 0.0207 & 0.0180 \\ 0.0207 & 1.0040 & 0.0399 \\ 0.0180 & 0.0399 & 0.4000 \end{bmatrix}$$

Para obtener Q_k tenemos que medir las incertidumbres generadas por los encoders, ya que son los sensores con los que estaremos trabajando y los que generan las incertidumbres.
Por lo tanto podemos asumir que tienen la siguiente distribución gaussiana en cada rueda

$$\begin{bmatrix} \omega_{r,k} \\ \omega_{l,k} \end{bmatrix} \sim N\left(0, \Sigma_{\Delta,k}\right) \qquad \Sigma_{\Delta,k} = \begin{bmatrix} k_r |\omega_{r,k}| & 0 \\ 0 & k_l |\omega_{l,k}| \end{bmatrix}$$

Donde la kr y kl son los errores asociados a cada velocidad de la rueda con respecto a las computarizadas. Para poder seguir tenemos que calcular el jacobiano basado en las ruedas.

Implementación en código:

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

```
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)
```

Resultados

Iteraciones del puzzle bot

Comparaciones con el real y el ODOM		
Vel_x=1 t=4s		
Corridas	Odom	Puzzlebot_kin
1	3.900136948	3.899938822
2	3.799977064	3.800146818
3	3.899937153	3.910191059
4	3.800004005	3.800203085
5	4.000000954	4.000226021
6	3.400146008	3.399894953
7	3.900124073	3.900384903
8	3.899974108	3.899833918
9	3.900115967	3.900228977

10	3.79999113	3.800097942
11	3.89999795	3.902559042
12	3.820374012	3.810267925
13	3.80005908	3.800022125
14	3.80030179	3.79977107
15	3.799975157	3.800182104
16	3.900155306	3.90404582
17	3.899349213	3.899176836
18	3.900229693	3.904088497
19	3.788683653	3.788425922
20	3.910987854	3.895989656

Vel_z=0.5		
Corridas	Odom	Puzzlebot_kin
1	0.8278163373	0.8276870168
2	0.827710424	0.8276903617
3	0.8277265452	0.8271244823
4	0.8296466586	0.8296655993
5	0.829100859	0.8290826035
6	0.8315607656	0.8268459713
7	0.8289186945	0.8280355683
8	0.8277093535	0.827541082
9	0.8280120673	0.8296579127
10	0.8267839747	0.8272590573
11	0.8270559306	0.8276258327
12	0.8293839879	0.8293839879
13	0.8289147936	0.8293501204
14	0.8259703072	0.8279786852
15	0.8310718425	0.8287908504
16	0.8292218757	0.8290009447
17	0.8138619523	0.8133140728
18	0.8275209637	0.8293313371
19	0.8289974878	0.8273985057
20	0.8269092177	0.8262576543

Vel_z=0.1 - Vel_x=0.1 t=4 s						
Corridas	Odom			Puzzlebot_kin		
	x	y	z	x	y	z
1	0.3797653474	0.0746755597	0.193518774	0.3800706206	0.07480132923	0.1936805222
2	0.3818522561	0.0755530105 3	0.1946276256	0.3820464733	0.07562414383	0.1947297164
3	0.379292702	0.0744979045 4	0.1932699091	0.3801451635	0.07484710078	0.1937215605
4	0.3721915324	0.0716133916 6	0.1895084491	0.3714368159	0.07131553295	0.1891097664
5	0.3805878868	0.0706051999 5	0.1914177735	0.3810772716	0.07522897087	0.1942159932
6	0.3782640018	0.0740801578 3	0.1927249313	0.3781008769	0.07401179618	0.192638125
7	0.3814238656	0.0753700617 8	0.1943998178	0.3810808159	0.07522334319	0.194217034
8	0.3819609985	0.0755962350 3	0.1946849208	0.3801459159	0.07485132877	0.1937224727
9	0.3806665313	0.0750632722 9	0.190254759	0.3802118385	0.07487542978	0.1937570082
10	0.3798978481	0.0747462462 5	0.1935908182	0.3801719612	0.07485680928	0.193735887
11	0.3802446185	0.0748876143 5	0.1895518023	0.379142869	0.07443571215	0.1931904925
12	0.3802520552	0.0748956431	0.1937787951	0.3790324833	0.07439756272	0.1931325016
13	0.3792926764	0.0744991388 7	0.1932699559	0.379560725	0.0746052313	0.1934115837
14	0.380282639	0.0749052701 6	0.1937946184	0.3805803277	0.07502340961	0.1939523463
15	0.382481833	0.0758050716 3	0.1949610511	0.3812992643	0.07532062259	0.1943339369
16	0.3808832829	0.0751531836 1	0.1941134122	0.3801294804	0.07484084915	0.1937133388
17	0.3696659871	0.0689200316 3	0.1859187645	0.3700499176	0.07077223336	0.1883786266
18	0.380242208	0.0748936123 8	0.1937737546	0.3800855271	0.07482452416	0.1936902292
19	0.3810400063	0.0728036324 3	0.191084419	0.3805955717	0.07502585987	0.1939597957

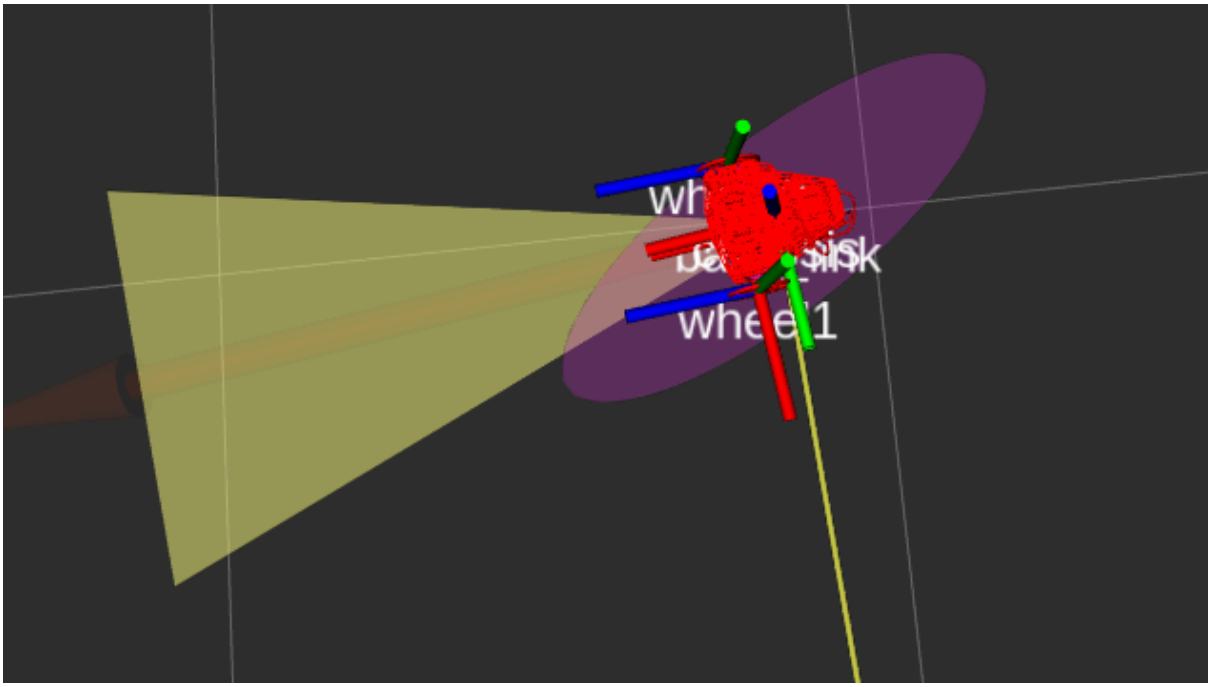
20	0.3813076322	0.07533300225	0.1943388826	0.3821242917	0.07566163678	0.1947714827

Vel_z=0.3 - Vel_x=0.3 t=4s						
Corridas	Odom			Puzzlebot_kin		
	x	y	z	x	y	z
1	0.9076996728	0.5768944177	0.5768944177	0.9080706369	0.5776916189	0.5381195953
2	0.9208768149	0.6063262239	0.5512779533	0.9202901379	0.6050020025	0.5506728224
3	0.9247129751	0.6153323653	0.5553690153	0.9210203762	0.6065755388	0.5514058933
4	0.9228803936	0.6110053333	0.5534103768	0.9223208765	0.6096446479	0.5528025439
5	0.9109735343	0.626394478	0.5604034442	0.9227181062	0.6105664753	0.5532136488
6	0.9229897168	0.6112763918	0.5535283879	0.9231247778	0.6114706456	0.5536387655
7	0.9239015743	0.6134642263	0.5545137424	0.9229996687	0.6113054961	0.5535439071
8	0.9229577078	0.6111953474	0.5534945523	0.9237577244	0.6130571602	0.5543383507
9	0.9206903403	0.6058428142	0.5510731309	0.9204540615	0.6051940635	0.5507898665
10	0.9243750809	0.614569067	0.5550104719	0.9235512649	0.6126049843	0.5541314069
11	0.9213491307	0.6074819703	0.551794666	0.9235512649	0.6126049843	0.5541314069
12	0.9205515184	0.6056272057	0.5509552774	0.9218143608	0.6084718889	0.5522661502
13	0.9155428644	0.61857889	0.5568414685	0.9214444119	0.6075673305	0.5518573896
14	0.9216834678	0.6081843668	0.5521289649	0.9217676413	0.6083296663	0.5521991855
15	0.9205410154	0.605507385	0.5509169256	0.9202255391	0.6047978063	0.5505871743
16	0.921071552	0.6067469047	0.5514779395	0.9230076133	0.6113279584	0.5535476897
17	0.9219900934	0.608901426	0.5524528015	0.9240616452	0.6137024809	0.5546485237

18	0.9232329074	0.6118536595	0.5537858768	0.9228121574	0.6108888296	0.5533469005
19	0.9204541606	0.6054327218	0.5508592315	0.921812073	0.6084725917	0.5522713975
20	0.9231615385	0.611623011	0.5536964198	0.9228377847	0.6107817812	0.5533314401

Estas mediciones se realizaron en rviz para poder calibrar la elipsoide de confianza, sin embargo los resultados obtenidos están dispuestos a cambios ya que al estar simulados no cuenta con variaciones y por ende es necesario calibrar con gazebo.

Movimiento del Puzzlebot con la elipsoide de confianza



Plot de la odometría


```
header:
  seq: 1563
  stamp:
    secs: 1714795460
    nsecs: 999063968
  frame_id: "odom"
child_frame_id: "base_link"
pose:
  pose:
    position:
      x: 2.1369824886
      y: 0.0
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
  covariance: [0.0005351619318459864, 6.266011255253688e-21, 0.0, 0.0, 0.0, 7.288233893274345e-21, 6.2660
11255253688e-21, 0.09014859768451752, 0.0, 0.0, 0.0, 0.06330802753098985, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 7.288233893274345e-21, 0.06330802753098985, 0.
0, 0.0, 0.0, 0.05929772098016479]
twist:
  twist:
    linear:
      x: 0.1
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: 0.0
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
---
```


Video:

https://youtu.be/AAeNyMUEsNw?feature=shared

Conclusiones

Para la elaboración del reporte se ha tenido que explorar el concepto de elipses de confianza como una representación gráfica crucial para evaluar la certeza en los valores obtenidos. Este concepto se aplica de manera significativa en el contexto del Puzzlebot, donde buscamos generar recorridos más precisos y seguros.

Nuestro enfoque incluyó el desarrollo del modelo cinemático del robot, su posterior linealización y la generación de los jacobianos necesarios para el modelado. Todo esto, junto con la implementación de la elipse de confianza, nos permitió visualizar de manera efectiva la certeza en los movimientos planificados del Puzzlebot.

Para lograr nuestros objetivos, fue fundamental considerar la matriz de covarianza para la localización, así como comprender en profundidad los mensajes tipo odometry en ROS. Estos elementos técnicos fueron fundamentales para garantizar la precisión y confiabilidad de nuestros resultados.

La implementación práctica de estos conceptos en el código nos permitió no solo calcular la odometría de manera efectiva, sino también visualizar la elipse de confianza en la simulación,

lo que facilita la evaluación y ajuste continuo de nuestros algoritmos de control y planificación de movimiento.

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).