Proyecto 2: Control Cinemático de un Robot

Julio Ernesto Sanchez Díaz 148221, Diego Amaya Willhelm 149119, Gumer Israel Rodríŋguez Martínez 149109, Gabriel Reynoso Romero 150904

I. Introducción

EL objetivo de este proyecto fue implementar un modelo cinemático de control de un automóvil autónomo utilizando las herramientas Gazebo (como simulador) y ROS como interfaz y control. Para lograrlo se implementaron en código C++ algoritmos que direccionan el automóvil de su pose inicial a una pose deseada (final) . Como referencia se utilizó el modelo de control mencionado en el libro "Robotics, Vision and Control" de Peter Corke, complementándolo con el add-on de Matlab del mismo autor. El control se suscribe a la pose del robot (proporcionada por Gazebo) y a la pose deseada (proporcionada por el usuario), una vez que se obtienen las dos poses, usando el modelo de control se calcula el valor de la velocidad y el ángulo del volante y se envían al modelo de nuestro automóvil mediante la publicación en dos tópicos de ROS leídos por el simulador Gazebo. Es importante mencionar que este control se retroalimenta con la pose actual del robot, a una velocidad en Hz especificada por el usuario, hasta alcanzar la pose final (deseada).

II. Marco Teórico

La cinemática de un robot nos permite estudiar los movimientos de un robot con respecto a un sistema de referencia. En un análisis cinemático la posición, velocidad y aceleración de cada uno de los elementos del robot son calculados sin considerar las fuerzas que causan el movimiento con apoyo a las propiedades geométricas del robot y de su entorno basadas en el tiempo de movimiento.

La cinemática de un vehículo a estudiar requiere de un análisis profundo para determinar el algoritmo de control de trayectoria más adecuado y que sea capaz de implementarse en nuestro proyecto.

Como describe Pau Perez Padial [1] en su trabajo de titulación "Estudio de algoritmos de control de trayectoria de un vehículo virtual" , un coche puede ser descrito como un cuerpo rígido moviéndose a través de un plano. Escribiendo la pose de la siguiente manera:

$$q = (x, y, theta) \tag{1}$$

. La estructura del coche muestra el origen en el centro del eje trasero y los punto del eje x sobre el eje central del coche.

Llamando "s" a la velocidad con signo del coche y "phi" al ángulo de las ruedas. Tomamos a "L" como la distancia entre el eje frontal y el trasero, fijando un ángulo de giro el coche circulará en una trayectoria circular cuyo radio será "rho".

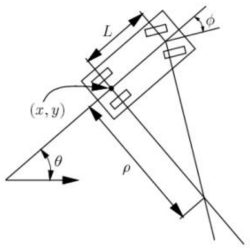

Figure 1. Esquema cinemático de un coche

Usando esta notación, el movimiento de un coche se puede representar con las siguientes ecuaciones.

$$\dot{x} = f_1(x, y, \theta, s, \phi)$$
$$\dot{y} = f_2(x, y, \theta, s, \phi)$$
$$\dot{\theta} = f_3(x, y, \theta, s, \phi)$$

Figure 2. Ecuaciones

El método cinemático o geométrico llamado "chasing the carrot" se basa en una idea muy sencilla, mover el vehículo hasta un punto destino. Para lograrlo se traza una línea desde el centro del sistema de coordenadas del vehículo perpendicular al camino a seguir. El "Carrot point" o punto de destino se define como el punto en el camino a una "look-ahead distance" alejada del punto de intersección de la línea perpendicular al camino a seguir con el propio camino a seguir. El parámetro más importante es el error de orientación, definido como el ángulo entre la orientación actual del vehículo y una línea que va desde el centro de coordenadas del vehículo hacia el "carrot point".

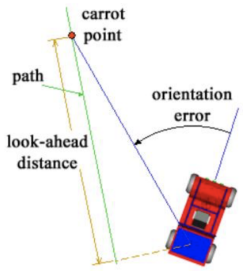

Figure 3. Algoritmo Follow the carrot

Otro método es el llamado "pure pursuit". El concepto de este método es calcular la curvatura necesaria para llevar el vehículo desde su posición actual a una posición

meta. Se define un círculo de tal manera que pasa a través del punto objetivo y de la posición actual del vehículo. A partir de este y por medio de un algoritmo de control se elige un ángulo de dirección en relación a este círculo. El propio vehículo cambia su curvatura repetidamente usando arcos de este estilo, siempre con la posición meta delante de él. Gracias esto se crea la analogía de que el vehículo "persigue" un punto en movimiento que siempre estará a cierta distancia delante de él.

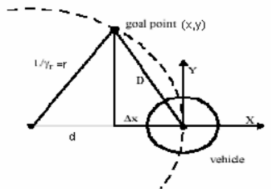

Figure 4. Algoritmo Pure Pursuit

La gráfica está mostrada en coordenadas del vehículo.

Un tercer método es el "vector pursuit" que determina el movimiento del vehículo por medio de "theory of screws". La "screw theory" se utiliza para representar el movimiento de cualquier solido rígido en relación a un sistema de coordenadas dado, haciendo útil en aplicaciones de control de trayectoria, de esta manera cualquiera movimiento puede ser descrito como una rotación frente a una línea en el espacio con una determinada pendiente. Es importante definir "screw", éste consiste en una línea definida en un sistema de coordenadas y un cierto ángulo. El movimiento del sólido rígido en cada instante puede ser representado como si fuera pegado a un tornillo y rotando en este tornillo a cierta velocidad angular. En la siguiente figura se muestra el movimiento de un sólido rígido que rota con velocidad angular a lo largo de un tornillo, que tiene una línea central definida y una pendiente. La velocidad en cualquier punto del sólito rígido es igual a la velocidad debida a la rotación más velocidad de translación debido a la pendiente del tornillo.

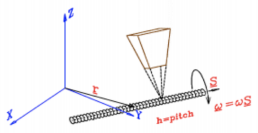

Figure 5. Algoritmo Vector Pursuit

El "Screw control" se desarrolló como un intento para que el vehículo no solo llegue a un cierto destino, sino que llegue con una orientación y curvatura correcta.

El modelo descrito en el libro de Peter Corke [2] es el siguiente:

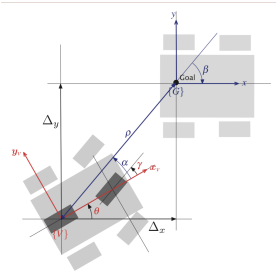

Figure 6. Moviendo a una pose

Y las ecuaciones del modelo cinemático son las siguientes:

$$\begin{bmatrix} \rho \\ \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \sqrt{(\Delta_x^2 + \Delta_y^2)} \\ atan(\frac{\Delta_y}{\Delta_x}) - \theta \\ -\theta - \alpha \end{bmatrix}$$

Figure 7. Modelo cinemático

$$\rho = \sqrt{\Delta_x^2 + \Delta_y^2}$$
$$\alpha = \tan^{-1}\frac{\Delta_y}{\Delta_x} - \theta$$
$$\beta = -\theta - \alpha$$

Figure 8. Formulas 1

$$v = k_\rho \rho$$
$$\gamma = k_\alpha \alpha + k_\beta \beta$$

Figure 9. Formulas 2

El control de un robot es una implementación que permite converger al sistema a un valor deseado. Se logra retroalimentando valores al sistema por parte de sensores u otras herramientas, esto con el objetivo de que si se presentaran errores respecto al valor deseado, se comandas acciones a los actuadores con la intención de modificar los valores de las siguientes entradas. Lo anterior se consigue a partir de modelos matemáticos que se intentan converger al valor deseado en cada ciclo.

Por último, la simulación es una herramienta de programación esencial para cualquier usuario que desee probar su código. Gazebo es un simulador que permite probar algoritmos, diseños, comportamiento y hasta entrenamiento de algoritmos de inteligencia artificial usando diferentes escenarios casi realistas. Esto gracias a su motor gráfico que permite observar y sobre todo analizar resultados.

III. Esbozo de la solucion

Para resolver el problema, comenzamos por implementar la conectividad con ROS a través de un nodo, el cual tiene dos suscriptores, uno para la pose deseada y otro para la pose actual del robot; y dos publicadores, uno para publicar la velocidad y otro para el ángulo de giro. El programa le dirá al robot que no se mueva hasta que se reciba una trayectoria deseada. Cuando se detecta una trayectoria, lo primero que se hace es determinar si el punto final se encuentra frente al robot o atrás de él, para ello nos ayudamos del producto vectorial: -Si el punto final se encuentra frente al robot, es decir, en un ángulo entre pi medios y menos pi medios y el producto punto entre los vectores de orientación y aquel que va desde el robot a la pose final será positivo, será una trayectoria normal donde el automóvil avanzará de frente. -Si el punto final se encuentra detrás del robot, el producto punto entre los vectores de orientación y aquel que va desde el robot a la pose final será menor a cero y los posteriores valores de ángulo y velocidad deberán ser cambiados de signo para generar una trayectoria en reversa. Posteriormente se verifica que la pose actual esté fuera de la distancia o error tolerable a la pose final y se calculan la velocidad (v) y el ángulo (gamma) con la pose final y la pose actual aplicando el modelo de Ackerman.

En caso de que la pose actual se encuentre dentro del error tolerable, se publican velocidad y ángulo igual a cero para que el robot deje de moverse. Los valores de las constantes kalpha, kbeta y krho se determinaron experimentalmente, escogiendo los valores que nos arrojaron mejores resultados al correr la simulación, respetando la regla de convergencia: krho>0, kbeta<0 y kalpha-krho>0. A la pose final se le dejó siempre fija un ángulo theta igual a cero, el modelo completo no se pudo implementar debido a que tuvimos problemas para el desarrollo de dicho modelo.

IV. Experimentos

Se realizaron diferentes pruebas a las distintas partes del código. Lo primero fue probar ROS en el programa, es decir, la conexión con los nodos del robot, en los cuales recibe las poses y publica el control, estos se comparaban entre el "echo" hecho directamente en los tópicos y lo que imprimía el programa, una vez que se corroboró la conexión, se procedió a utilizar Gazebo como ambiente de simulación. En Gazebo se importó el modelo "AutoNOMOSmini" que sería el que interactuaría con el control de manera retroalimentada. Primero se probaron poses sencillas, como avanzar en el eje x 5 unidades. El resultado fue poco satisfactorio al principio porque a pesar de llegar a la pose indicada, el robot seguía moviéndose, por lo que se agregó un parámetro de error permitido, lo que implica que cuando la distancia entre el robot y las coordenadas finales sea menor a lo permitido, el carro obtiene velocidad y ángulo iguales a cero. Esto solucionó el problema. También nos percatamos en esta etapa que la velocidad del móvil depende directamente de la distancia al punto, por lo que se vuelve muy lento en cuanto más se acerca, lo cual hacía tediosa la simulación. Se continuó probando poses más complejas, con ángulos de giro involucrados. El resultado fue exitoso y comprobamos que el control convergía bien al valor del punto, incluso se ve cómo el carro gira durante la simulación para compensar y alcanzar la trayectoria. Tuvimos problemas para hacer girar el auto porque el modelo del robot en Gazebo utiliza diferentes unidades, arroja el ángulo de su pose en radianes, pero el ángulo de giro del volante lo debe recibir en grados. Arreglarlo fue sencillo porque es una simple conversión, pero nos llevo tiempo darnos cuenta de ello. También se presentaron problemas cuando el objetivo estaba detrás del robot, porque no reconocía que tenía que ir en reversa, este problema se solucionó utilizando vectores y álgebra lineal, verificando la posición deseada únicamente al ser recibida. Las pruebas finales completas fueron llevar el robot desde el origen a ciertos destinos: -(5,5) -(-5,-5) -(5,0) -(0,5) Así como agregar nuevas poses deseadas una vez alcanzada una anterior, o cuando todavia se estaba en camino a alcanzar un objetivo. -(0,0) a (3,3) a (-1,2)

Las poses deseadas mencionadas anteriormente son los ejemplos documentados en el video, sin embargo, además de éstos, se hicieron varios más parecidos. Dicho video se puede descargar en el siguiente link de Google Drive:
https://drive.google.com/open?id=1FYZ_Bcuzai2EVxlVqrDa7vW9cO6Y1gl8

V. Conclusiones

Una lección que nos llevamos es lo poderoso y útil que puede resultar GitHub para un proyecto colaborativo, ya que la mayoría de nosotros nunca habíamos trabajado con la herramienta. Ésta permite un mejor manejo del código y facilita la participación de todos en el proyecto en general. Una causa de problemas fue la instalación de Gazebo, esto debido a que los errores que manda al ejecutar las instrucciones de GitHub no contienen mucha información y la poca que contiene es muy técnica, lo arreglamos moviendo bastantes cosas e incluso teniendo que instalar el simulador desde cero. La parte más problemática del proyecto fue la poca documentación sobre el simulador utilizado para el desarrollo del proyecto, ya que la pose del vehículo que publica Gazebo se encuentra medida en radianes, mientras que el valor del ángulo que recibe el volante se encuentra en grados. Esto nos ocasionó bastantes problemas a la hora de probar nuestro código, no fue hasta que se decidió probar con números diferentes que nos dimos cuenta de esta característica. En general no tuvimos problemas para probar nuestro control, como se mencionó anteriormente el principal problema fue la conversión de radianes a grados para el valor de gamma, pero fuera de eso no existieron mayor complicaciones. También se utilizaron distintas condiciones para asegurar en todo momento el correcto funcionamiento del control. Por último, el simulador nos permite una experiencia mucho más real de una implementación en el modelo

físico, nos permite probar nuestro control he incluso poder ajustarlo para diferentes valores de las constantes K?s.

VI. Bibliografía

[1] Perez Padial Pau, Estudio de algoritmos de control de trayectorias de un vehículo virtual, Universitat Politécnica de Catalunya, Octubre, 2016

[2] Peter Corke. Robotics, Vision and Control. Springer, 2011