

Fecha de publicación 05, 06, 2024, date of current version 05, 06, 2024.

Evidencia Final. Navegación Autónoma de un robot diferencial (Junio 2024)

A. González¹, D. Rodríguez², J. Martínez³, I. García.⁴, E.A. Carrizalez⁵.

¹ Instituto Tecnológico y de Estudios Superiores de Monterrey

² Escuela de Ingeniería y Ciencias

³ Departamento de Ingeniería Robótica y Sistemas Computacionales.

Contacto con el autor: A. González1 (A01351820@tec.mx).

“Este trabajo está apoyado en parte por el Departamento de Ingeniería Robótica y Sistemas Computacionales, además de nuestro socio formador Manchester Robotics”

RESUMEN Este informe presenta los resultados del desafío final asignado por Manchester Robotics en el curso de Integración de Robótica y Sistemas Inteligentes. El desafío tenía como objetivo revisar los conceptos introducidos durante todo el semestre, centrándose en mostrar diferentes algoritmos de navegación reactiva para robots móviles. El desafío se dividió en varias secciones, comenzando con la utilización de la estrategia de control de posición multipunto previamente desarrollada para el Puzzlebot. Usando el simulador Gazebo, desarrollado por MCR2, el equipo probó el comportamiento de diferentes algoritmos de navegación reactiva, basados en los algoritmos Bug 0 y Bug 2, para evitar obstáculos y alcanzar posiciones objetivo. Los principales logros y contribuciones del proyecto incluyen el desarrollo e implementación de algoritmos de navegación reactiva. Se crearon y probaron algoritmos eficientes basados en Bug 0 y Bug 2 para la planificación de rutas y evasión de obstáculos. También se usaron los algoritmos Go to Goal y Follow Walls, e incluimos algoritmos de detección de arucos para identificar puntos clave y corregir su posicionamiento, además se implementaron un filtro de Kalman para mejorar la estimación de la posición y orientación del robot, combinando múltiples mediciones y estimaciones previas para proporcionar una estimación más precisa del estado del robot.. La integración de sensores y análisis de covarianza también fue clave. Se integraron y calibraron sensores, como LIDAR y una cámara, y se analizó la creciente covarianza alrededor de la pose del robot para mejorar la precisión y robustez del sistema. Por último, se realizaron pruebas exhaustivas en diversos escenarios. Se diseñaron y probaron múltiples escenarios en Gazebo, definiendo correctamente los archivos de lanzamiento y el tiempo de muestreo adecuado para cada simulación.

ABSTRACT This report presents the results of the final challenge assigned by Manchester Robotics in the Robotics and Intelligent Systems Integration course. The challenge aimed to review the concepts introduced throughout the semester, focusing on demonstrating different reactive navigation algorithms for mobile robots. The challenge was divided into several sections, starting with the use of the multipoint position control strategy previously developed for the Puzzlebot. Using the Gazebo simulator, developed by MCR2, the team tested the behavior of different reactive navigation algorithms, based on the Bug 0 and Bug 2 algorithms, to avoid obstacles and reach target positions. The main achievements and contributions of the project include the development and implementation of reactive navigation algorithms. Multiple scenarios were designed and tested in Gazebo, correctly defining the launch files and the appropriate sampling time for each simulation . Efficient algorithms based on Bug 0 and Bug 2 were created and tested for route planning and obstacle avoidance. The Go to Goal and Follow Walls algorithms were also used, in addition to including artifact detection algorithms to identify key points and correct their positioning, and a Kalman filter was implemented to improve the estimation of the robot's position and orientation, combining multiple measurements and previous estimates to provide a more accurate estimation of the robot's state. Sensor integration and covariance analysis was also key. Sensors, such as LIDAR and a camera, were integrated and calibrated, and the increasing covariance around the robot pose was analyzed to improve the accuracy and robustness of the system. Finally, extensive testing was performed in a variety of scenarios. Multiple scenarios were designed and tested in Gazebo, correctly defining the launch files and the appropriate sampling time for each simulation.

INDEX TERMS autonomous navigation, Bug 0 algorithms, Bug 2 algorithms, covariance, Gazebo, Gazebo simulator, intelligent robotics, multipoint position control, obstacle avoidance, path planning, reactive navigation, robot simulation, sensor integration, Puzzlebot, arucos, go to goal, follow wall

I. INTRODUCTION

En este proyecto, se desarrollarán y evaluarán algoritmos de navegación reactiva para robots móviles, utilizando el simulador Gazebo desarrollado por MCR2. La navegación reactiva es un aspecto fundamental de la robótica autónoma, permitiendo a los robots evitar obstáculos en tiempo real utilizando sensores exteroceptivos como LiDAR, cámaras y sonares. Este proyecto se enfocará en la implementación y prueba de los algoritmos Bug 0 y Bug 2, los cuales se utilizan para la evitación de obstáculos y la navegación hacia una posición objetivo. Además, se incorporarán los algoritmos Go to Goal y Follow Walls, que son esenciales para la navegación hacia metas específicas y el seguimiento de paredes respectivamente, y se incluirán algoritmos de detección de arucos para identificar puntos clave y corregir su posicionamiento.

El desafío se estructurará en varias etapas clave. Primero, se desarrollarán los algoritmos de navegación reactiva, asegurando su integración con los diversos sensores del robot. Estos sensores, que incluyen LiDAR, cámaras y encoders, permitirán al robot percibir su entorno de manera precisa y reaccionar adecuadamente ante obstáculos imprevistos. El proceso de desarrollo implica no solo la programación de los algoritmos, sino también su calibración y ajuste fino para garantizar un rendimiento óptimo, así como la implementación y ajuste de filtros de Kalman para mejorar la estimación de la posición y orientación del robot.

Posteriormente, se realizaron pruebas exhaustivas de estos algoritmos en una serie de escenarios simulados dentro de Gazebo. Estos escenarios estarán diseñados para replicar diversas condiciones y desafíos que un robot móvil podría enfrentar en el mundo real, incluyendo diferentes configuraciones de obstáculos y rutas de navegación. Las pruebas permitirán evaluar el desempeño y la efectividad de cada algoritmo en términos de precisión, rapidez y robustez.

La integración de sensores y el análisis de covarianza serán aspectos cruciales para mejorar la precisión de la navegación. A través de un cuidadoso análisis de los datos recopilados por los sensores, se podrá ajustar la covarianza y mejorar la fiabilidad de las estimaciones de posición del robot. Este proceso ayudará a minimizar errores y a aumentar la consistencia del comportamiento del robot en distintos entornos.

Finalmente, se analizarán los resultados obtenidos de las pruebas, y se discutirán las posibles mejoras y aplicaciones de los algoritmos desarrollados. Este análisis incluirá una

evaluación detallada de los logros alcanzados, las limitaciones encontradas y las oportunidades para futuras mejoras. Se prestará especial atención a cómo los diferentes algoritmos pueden ser optimizados y adaptados para una variedad de aplicaciones en el campo de la robótica inteligente.

El contenido del informe se organiza de la siguiente manera: Objetivos, Estado del Arte, Materiales y Métodos, Resultados, Discusión de Resultados, Conclusiones, Referencias y Anexos. Este informe pretende proporcionar una documentación exhaustiva y detallada del proceso seguido y los resultados obtenidos, subrayando la importancia de los algoritmos de navegación reactiva y su aplicación en la robótica móvil. A través de este proyecto, se demuestra la capacidad de los estudiantes para aplicar conceptos teóricos en un entorno práctico, contribuyendo al avance de la robótica inteligente. Este esfuerzo no solo destaca la relevancia de la navegación reactiva en la robótica moderna, sino que también subraya el potencial de estos algoritmos para mejorar la autonomía y la eficiencia de los robots en una amplia gama de aplicaciones industriales y de investigación.

II. OBJECTIVES

Nuestro proyecto tiene varios objetivos generales y específicos que buscamos alcanzar utilizando el formato SMART (específicos, medibles, alcanzables, relevantes y con un tiempo definido). Los objetivos generales son completar la pista con el menor número de colisiones posible, llegar al punto objetivo de manera precisa, lograr que el robot se reubique automáticamente si se pierde, y que se quede inmóvil una vez alcanzado el punto final. Para lograr estos objetivos generales, hemos definido algunos objetivos específicos clave.

Primero, necesitamos asegurar que los scripts de los algoritmos Bug 0 y Bug 2 funcionen correctamente en los laberintos propuestos. Bug 0 y Bug 2 son algoritmos de navegación reactiva que permiten al robot evitar obstáculos y seguir un camino hacia el objetivo. Bug 0 sigue el contorno de un obstáculo hasta que encuentra una ruta despejada hacia la meta mediante la comparación de distancias, mientras que Bug 2 también sigue el contorno del obstáculo pero utiliza una línea directa hacia el objetivo para evaluar el regreso a su ruta. Nuestro objetivo es implementar y validar estos algoritmos, logrando que completen los laberintos con una tasa de éxito del 95% y reduciendo las colisiones a menos de 3 por prueba, en un período de 1 semana.

También es esencial que integremos las referencias de odometría usando el nodo que desarrollamos en entregas anteriores. La odometría es crucial para rastrear la posición y orientación del robot a lo largo del tiempo, permitiendo ajustes precisos durante la navegación. Nuestro objetivo es lograr una precisión de posicionamiento con un error máximo del 5% en pruebas controladas, implementando y calibrando este sistema en un período de 3 días.

Queremos garantizar que el controlador del nodo "Go to Goal" lleve al robot al punto indicado de manera precisa, ajustando correctamente las ganancias. El algoritmo "Go to Goal" se centra en dirigir el robot directamente hacia una meta específica utilizando una estrategia de control proporcional que ajusta la dirección del robot en función de la distancia y el ángulo respecto al objetivo. Nuestro objetivo es que el robot alcance el objetivo con un margen de error menor a 10 cm, completando la calibración en 3 días.

Otro objetivo a desarrollar es la implementación del algoritmo "Follow Walls". Este algoritmo permite al robot seguir el contorno de las paredes de manera eficiente, lo cual es útil en entornos complejos donde es necesario mantener una trayectoria cercana a los límites del área de navegación. Nuestro objetivo es que el robot mantenga una distancia constante de 10 cm a las paredes con un margen de error de ± 3 cm, implementando y probando en 4 días.

Además, integraremos algoritmos de detección de arucos para identificar puntos clave y corregir su posicionamiento. Los arucos son marcadores visuales que el robot puede detectar usando su cámara, proporcionando referencias precisas para la localización y corrección de la trayectoria del robot. Esta capacidad es fundamental para asegurar que el robot pueda reconocer y ajustar su posición en relación con el entorno. Nuestro objetivo es detectar arucos con una tasa de precisión del 90% y corregir la posición del robot con un error máximo del 5%, completando la integración y pruebas en 4 días.

Asimismo, utilizaremos filtros de Kalman para mejorar la estimación de la posición y orientación del robot. Los filtros de Kalman son algoritmos de filtrado óptimo que combinan múltiples mediciones y estimaciones previas para proporcionar una estimación más precisa del estado del robot. Nuestro objetivo es implementar y ajustar los filtros de Kalman para que mejoren la precisión de la odometría y la detección de arucos, logrando una reducción del error de estimación a menos del 5%. La implementación y ajuste de los filtros de Kalman se realizará en un período de 3 días.

Nuestro proyecto también se enfoca en desarrollar un sistema modular que pueda ser implementado junto con la lectura de arucos y el filtro de Kalman en un período no mayor a dos semanas, que es la entrega final de nuestro reto. La integración de estos componentes permitirá

entregar un proyecto final completamente integrado, siguiendo una documentación adecuada y con nodos publicadores para futuras entregas. Para alcanzar este objetivo, nos enfocaremos en la modularidad del sistema. Cada componente será diseñado de manera independiente, permitiendo una fácil integración y flexibilidad en el desarrollo. Además, nos comprometemos a seguir una documentación exhaustiva que detalle el funcionamiento de cada módulo, facilitando su comprensión y mantenimiento en el futuro. El sistema resultante será escalable y adaptable a futuras mejoras, asegurando que pueda ser extendido y mejorado conforme se desarrolle nuevas tecnologías y algoritmos en el campo de la robótica.

III. STATE OF THE ART

La navegación autónoma de robots móviles ha sido objeto de una investigación continua y un desarrollo creciente en el campo de la robótica inteligente. Con el avance tecnológico, se han explorado diversas estrategias y algoritmos para permitir que los robots se desplacen de manera autónoma en entornos complejos. Desde la implementación de enfoques clásicos como la descomposición celular y los métodos de campo potencial hasta la adopción de técnicas heurísticas y algoritmos de optimización, se han desarrollado una variedad de soluciones para abordar el problema de la planificación de rutas y la evitación de obstáculos.

Algoritmos como Bug 0 y Bug 2 han demostrado su eficacia en la navegación reactiva, permitiendo que los robots eludan obstáculos de manera eficiente mientras se mueven hacia una posición objetivo. Estos algoritmos se basan en estrategias simples pero efectivas, que utilizan información sensorial en tiempo real para tomar decisiones de navegación. La integración de sensores exteroceptivos, como LIDAR, cámaras y sonares, ha mejorado significativamente la percepción del entorno por parte de los robots móviles, permitiéndoles detectar y evitar obstáculos de manera más precisa.

El desarrollo de simuladores como Gazebo ha sido fundamental para el diseño y la evaluación de algoritmos de navegación reactiva. Estas herramientas proporcionan entornos virtuales realistas donde los algoritmos pueden ser probados y validados antes de su implementación en robots reales. La capacidad de diseñar y probar escenarios en un entorno simulado ha acelerado el proceso de desarrollo, y ha facilitado la comparación de diferentes enfoques y estrategias de navegación.

Investigaciones recientes, como el estudio realizado por Krishnavamshi Perumbhudur sobre el "Path Planning Algorithm for a Two Wheel Differential Mobile Robot in Robot Operating System", y el proyecto de la Universidad de Michigan sobre la "Autonomous Navigation", han

destacado la aplicación práctica de algoritmos de navegación reactiva en entornos dinámicos. Estos proyectos han demostrado la efectividad de los algoritmos Bug 0 y Bug 2 en la planificación de rutas y la evitación de obstáculos, así como la importancia de la integración de sensores y la simulación para mejorar la precisión y robustez del sistema de navegación, hablaremos de ellas a continuación para ver un poco a profundidad lo que planea cada una:

A. PATH PLANNING ALGORITHM FOR A TWO WHEEL DIFFERENTIAL MOBILE ROBOT IN ROBOT OPERATING SYSTEM

El estudio realizado por Krishnavamshi Perumbhudur, titulado "Path Planning Algorithm for a Two Wheel Differential Mobile Robot in Robot Operating System", presenta un enfoque novedoso en la planificación de rutas para robots móviles en entornos dinámicos. Este trabajo se destaca por su implementación de algoritmos de navegación, incluyendo el enfoque de campo potencial y el diagrama de Voronoi, con el objetivo de encontrar trayectorias óptimas mientras se evitan obstáculos. Utilizando el entorno de desarrollo Robot Operating System (ROS), Perumbhudur desarrolló un nuevo algoritmo de planificación de rutas en Python, integrándose con ROS para permitir la navegación autónoma del robot. Además, el estudio incluye la definición de un modelo de robot personalizado en el formato URDF, lo que facilita su simulación en el entorno de simulación Gazebo. Mediante el uso de técnicas de mapeo y localización simultáneas (SLAM), con el plugin gmapping de ROS, se logró obtener un mapa de ocupación del entorno para mejorar la precisión de la navegación.

El trabajo de Perumbhudur ofrece una comparación exhaustiva de su algoritmo propuesto con otros métodos de navegación establecidos, evaluando su desempeño en términos de longitud de ruta y tiempo de ejecución. Estos resultados destacan la relevancia y el potencial de su enfoque para aplicaciones prácticas en robótica móvil autónoma.

B. AUTONOMOUS NAVIGATION DERECK WONNACOTT

En el ámbito de la navegación autónoma con ROS, el trabajo realizado por la Universidad tecnológica de Michigan presenta otro ejemplo relevante. En este caso, los investigadores desarrollaron un sistema de navegación autónoma para un robot móvil utilizando ROS, enfocándose en la integración de varios sensores y la implementación de algoritmos avanzados de navegación.

El proyecto tenía como objetivo principal diseñar y desarrollar un sistema de navegación autónoma, que permitiera al robot moverse eficientemente en entornos interiores complejos. Para ello, se utilizó una combinación

de sensores LiDAR, cámaras y sensores de ultrasonido, proporcionando al robot la capacidad de percibir su entorno de manera precisa. La información obtenida de estos sensores se procesaba mediante algoritmos de SLAM (Simultaneous Localization and Mapping) para crear mapas del entorno y localizar el robot dentro de ellos (ar5iv) (MDPI).

Un aspecto fundamental del trabajo fue la implementación de un planificador de rutas, que permitía al robot moverse de un punto a otro evitando obstáculos y optimizando su trayectoria. Este planificador se basaba en técnicas de búsqueda heurística y algoritmos como A* y Dijkstra, que son comunes en la literatura de robótica para la planificación de rutas en tiempo real. Además, el proyecto incorporó módulos de control de movimiento y estrategias de navegación que aseguraban un desplazamiento suave y preciso del robot en entornos dinámicos (ar5iv) (MDPI).

La implementación de un sistema de navegación autónoma con ROS por parte de la Universidad de Michigan demuestra la eficacia de este marco para integrar sensores avanzados y algoritmos de navegación, destacando su capacidad para mejorar la percepción y el movimiento autónomo en robots móviles. Este trabajo contribuye al campo de la robótica móvil proporcionando ejemplos prácticos y soluciones técnicas que pueden ser adaptadas y extendidas en investigaciones futuras y aplicaciones industriales.

METODOLOGÍAS Y HERRAMIENTAS DE PLANEACIÓN DE PROYECTOS

La implementación de un sistema de navegación autónoma con ROS por parte de la Universidad de Michigan demuestra la eficacia de este marco para integrar sensores avanzados y algoritmos de navegación, destacando su capacidad para mejorar la percepción y el movimiento autónomo en robots móviles. Este trabajo contribuye al campo de la robótica móvil proporcionando ejemplos prácticos y soluciones técnicas que pueden ser adaptadas y extendidas en investigaciones futuras y aplicaciones industriales.

IV. MATERIALS AND METHODS

1) COMPONENTES DEL PUZZLEBOT:

Materiales:

- NVIDIA Jetson Nano
- Manchester Robotics Puzzlebot Jetson Edition
- Manchester Robotics Hacker Board
- Encoders reductores de velocidad
- Batería de litio recargable del modelo “INIU”
- Cámara Raspberry pi “Arduino IMX29”

- Piezas de como llantas, tornillos, entre otros que son propiamente del Puzzlebot brindado por Manchester Robotics.

- Laptop con tarjeta gráfica de NVIDIA.

- Para el entorno virtual del robot, se le instaló ubuntu 18.04 a la jetson nano, se instaló ROS-Melodic y todas las dependencias de ROS a continuación señaladas:

- ROS Serial

- Python 3

- Rospy

- El Puzzlebot de Manchester Robotics es el robot utilizado en este proyecto. Este robot está equipado con motores específicos para controlar sus llantas, lo que le permite moverse y navegar.

- Además, cuenta con sensores de cámara y LiDAR que le permiten percibir su entorno. Estos sensores son fundamentales para detectar obstáculos y crear mapas del entorno por donde el robot se desplaza.

- La computadora a bordo del Puzzlebot incluye una Jetson y una Hackerboard. Estas son unidades de procesamiento, que proporcionan la potencia necesaria para analizar los datos de los sensores y ejecutar los algoritmos de navegación.

2) HERRAMIENTAS DE PROGRAMACIÓN Y SIMULACIÓN:

- El lenguaje de programación utilizado en este proyecto es Python. Python es un lenguaje popular en robótica debido a su facilidad de uso y la gran cantidad de bibliotecas disponibles para trabajar con robots.

- Se utiliza el simulador Gazebo para crear un entorno virtual donde el Puzzlebot puede moverse y realizar tareas. Gazebo permite simular el comportamiento del robot en diferentes escenarios antes de implementarlo en el mundo real.

- RViz es una herramienta de visualización, que se utiliza para representar gráficamente el entorno del robot y los datos recopilados por sus sensores. Esto ayuda a comprender mejor cómo el robot interpreta su entorno.

- ROS (Robot Operating System) es el marco de desarrollo utilizado para programar y controlar el Puzzlebot. ROS proporciona una serie de paquetes y herramientas que simplifican el desarrollo de aplicaciones robóticas.

3) TÉCNICAS DE PROGRAMACIÓN Y SIMULACIÓN:

- Se implementan algoritmos de navegación reactiva basados en Bug 0 y Bug 2. Estos algoritmos permiten al robot moverse y evitar obstáculos en tiempo real.

- Los sensores de cámara y LiDAR se integran al Puzzlebot y se calibran para garantizar una percepción precisa del entorno.

- Se realiza un análisis de covarianza alrededor de la pose del robot para mejorar la precisión y la robustez del sistema de navegación.

4) PROCESO METODOLÓGICO:

- El desarrollo y las pruebas de los algoritmos se realizan en el simulador Gazebo. Se definen escenarios de prueba y se ajustan los parámetros para evaluar el rendimiento de los algoritmos en diferentes situaciones.

- Se llevan a cabo pruebas exhaustivas en una variedad de escenarios simulados para evaluar el rendimiento de los algoritmos en términos de longitud de ruta, tiempo de ejecución y capacidad para evitar obstáculos.

5) TEORÍA

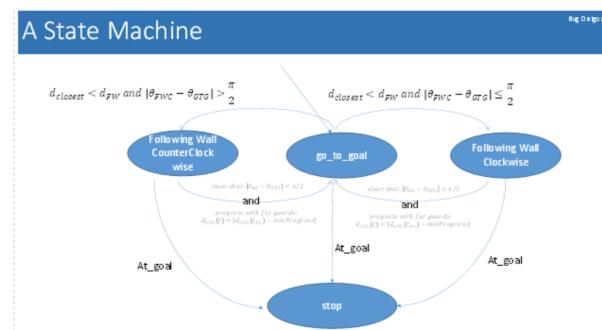


Fig 1. Máquina de estados por la cual nos estamos guiando, además de las ecuaciones y la jerarquía que lleva.

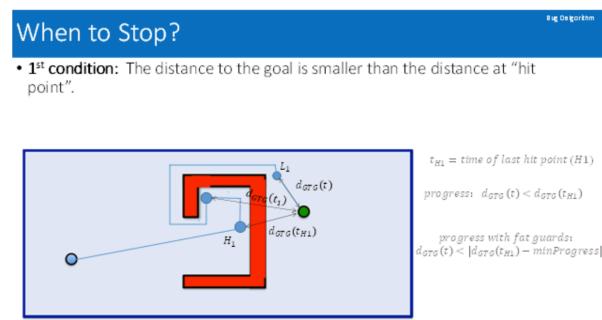


Fig 2. Teoría del bug 0 así como las ecuaciones que modelan la primera condición del mismo

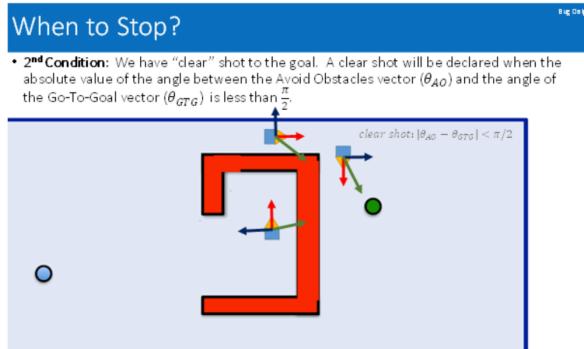


Fig 3. Teoría del bug 0 así como las ecuaciones que modelan la segunda condición del mismo

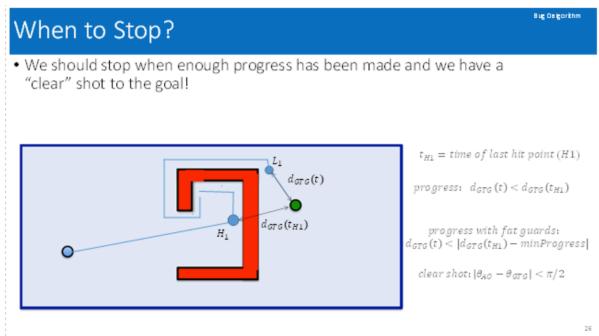


Fig 4. Teoría del bug 0 así como las ecuaciones que modelan la condición del cuándo detenerse y en qué caso es aplicable.

Kalman filter localisation

- Motion model for the mobile robot:

$$\mathbf{s}_k = \mathbf{h}(\mathbf{s}_{k-1}, \mathbf{u}_k) + \mathbf{q}_k \quad \mathbf{q}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_k)$$

- Robot observation model:

$$\mathbf{z}_{i,k} = \mathbf{g}(\mathbf{m}_i, \mathbf{s}_k) + \mathbf{r}_{i,k} \quad \mathbf{r}_{i,k} \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k)$$

- If \mathbf{h} and \mathbf{g} are linear functions with respect to their variables, the Kalman filter can be used directly and optimality is guaranteed. Otherwise, Extended Kalman filter is used where both functions \mathbf{h} and \mathbf{g} are linearised around current robot pose estimate using first-order Taylor expansion.

Fig 5. Teoría de filtros de kalman así como las ecuaciones que la modelan.

Kalman filter localisation

- Based on Extended Kalman filter theory, the aim of the linearised model is to propagate covariance matrices based on Gaussian distributions.

- Thus, provided that $\mathbf{s}_{k-1} \sim \mathcal{N}(\mu_{k-1}, \Sigma_{k-1})$ is available, the prediction step of the extended Kalman filter is done in a similar manner to motion-based localisation.

$$\hat{\mathbf{s}}_k = \mathbf{h}(\mu_{k-1}, \mathbf{u}_k)$$

$$\hat{\Sigma}_k = \mathbf{H}_k \Sigma_{k-1} \mathbf{H}_k^T + \mathbf{Q}_k$$

Fig 6. Teoría de filtros de kalman así como las ecuaciones que la modelan.

V. RESULTS

En la sección de resultados, se presentan los datos recopilados durante la implementación y pruebas del proyecto. Estos datos incluyen mediciones de rendimiento, tiempos de ejecución, y evaluaciones de la navegación autónoma del Puzzlebot en distintos escenarios simulados

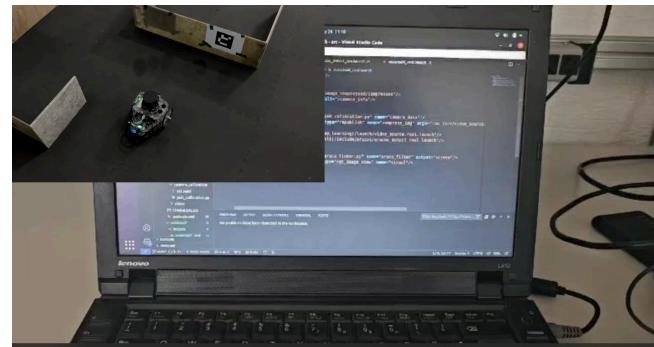


Fig 7. Debug bug0.

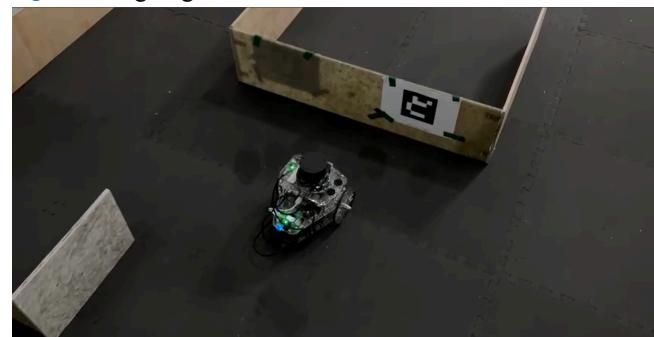


Fig 8. Prueba física bug 0 y bug 2.

Simulation statistics

	Bug0	Bug2
Número de pruebas realizadas	5	5
Porcentaje de recorridos exitosos	100%	80%
Colisiones promedio por recorrido	0.8	1.2

Fig 9. Teoría de filtros de kalman así como las ecuaciones que la modelan.

Simulation statistics

Prueba Bug0	Error de posición final (cm)	Objetivo alcanzado	Colisiones
1	2	Sí	0
2	5	Sí	0
3	8	Sí	0
4	1	Sí	0
5	3	No	1

Fig 10. Estadísticas de las pruebas realizadas Bug 0.

Simulation statistics

Prueba Bug2	Error de posición final (cm)	Objetivo alcanzado	Colisiones
1	0	Sí	0
2	3	Sí	0
3	5	Sí	0
4	1	Sí	0
5	4	Sí	1

Fig 11. Estadísticas de las pruebas realizadas Bug 2.



Fig 12. Lectura de Aruco.

CARPETA CON VIDEO DEMOSTRATIVOS DE LAS PRUEBAS

https://tecmx-my.sharepoint.com/:f/g/personal/a01706320_tec_mx/Eik7YEiv6CdDieMdjTO4AgUBtDYVJgO4YubeoX5nCyhYQg?e=FG7tRN

VI. DISCUSSION OF RESULTS

Hemos logrado avances significativos en nuestra comprensión y aplicación de diversas tecnologías y metodologías, esenciales para la navegación autónoma de robots. A lo largo del semestre, nos enfrentamos a una serie de desafíos técnicos que no solo pusieron a prueba nuestros conocimientos, sino que también fortalecieron nuestra capacidad de trabajar en equipo y resolver problemas de manera eficiente.

PROBLEMAS IDENTIFICADOS:

INTEGRACIÓN DE SISTEMAS Y ALGORITMOS

Uno de los aspectos más destacados del proyecto fue la integración de diversos sistemas y algoritmos para la navegación autónoma del Puzzlebot. La combinación de la odometría basada en los encoders y la detección de marcadores Aruco mediante el filtro de Kalman resultó en una mejora sustancial de la precisión en la localización del robot. Este enfoque permitió fusionar datos de distintas fuentes, lo cual es crucial para mantener la precisión en entornos desconocidos y dinámicos.

AJUSTES Y ADAPTACIONES

El cambio en la configuración de la pista de pruebas, aunque no drástico, nos obligó a realizar ajustes y mejoras en nuestros algoritmos. Este proceso fue una oportunidad invaluable para aplicar los conceptos aprendidos durante el curso en un contexto práctico, demostrando nuestra capacidad de adaptación y resolución de problemas en tiempo real. La necesidad de modificar significativamente el código existente para adaptarse a nuevas condiciones subraya la importancia de la flexibilidad y la robustez en el desarrollo de sistemas robóticos.

VALIDACIÓN Y ROBUSTEZ

La validación del software y hardware fue un componente crucial del proyecto. A través de simulaciones en Gazebo y pruebas en entornos reales, pudimos evaluar y garantizar la precisión y robustez de nuestros sistemas. Las pruebas unitarias e integradas aseguraron que cada módulo del sistema funcionara correctamente y que la comunicación entre ellos fuera eficiente. Este enfoque exhaustivo de validación no solo mejoró el rendimiento del sistema, sino que también identificó áreas de mejora, promoviendo una innovación continua.

EFICIENCIA EN LA NAVEGACIÓN

La implementación de algoritmos de navegación como bug0 y bug2, junto con comportamientos de follow walls y go to goal, permite una navegación autónoma eficiente del Puzzlebot. La capacidad del robot para reconocer y utilizar los marcadores Aruco mejoró su habilidad para ubicarse y evitar colisiones, demostrando un alto nivel de integración y funcionalidad. Este éxito no sólo refleja nuestra comprensión técnica y habilidades en programación, sino también nuestra capacidad para trabajar de manera colaborativa y eficiente bajo presión.

FUTURAS DIRECCIONES DE INVESTIGACIÓN

A pesar de los logros alcanzados, el proyecto también reveló áreas potenciales para futuras investigaciones. La aplicación del filtro de Kalman y otros algoritmos en contextos del mundo real, bajo condiciones más desafiantes, es una dirección prometedora. Asimismo, la integración de sensores adicionales y técnicas de

percepción avanzada, como el reconocimiento de objetos y la detección semántica, podría mejorar aún más la navegación autónoma. La exploración de estrategias de planificación de rutas más adaptativas y robustas también representa una oportunidad para enfrentar entornos cambiantes y desconocidos con mayor eficacia.

VI. CONCLUSIONES

Las conclusiones generales resaltan la importancia de la investigación continua en el campo de la navegación autónoma de robots móviles. La revisión del estado del arte revela un panorama dinámico de enfoques y tecnologías en constante evolución, desde algoritmos clásicos de planificación de rutas hasta técnicas más avanzadas de percepción y aprendizaje profundo. Este análisis proporciona una base sólida para el desarrollo y la implementación de algoritmos de navegación reactiva, como los evaluados en el reto final de Manchester Robotics.

Uno de los retos más significativos fue la integración eficiente de los mini-retos en el proyecto final. Logramos combinar múltiples componentes y sistemas, incluyendo la odometría con las velocidades de WR y WL, y la implementación de algoritmos como bug0 y bug2, así como los comportamientos de follow walls y go to goal. La implementación de filtros de Kalman y el uso de marcadores Aruco mejoraron la precisión de la localización del robot, lo que resultó en una navegación autónoma y eficiente del Puzzlebot a través de la pista del proyecto.

El curso no solo nos preparó para enfrentar desafíos técnicos complejos, sino que también resaltó la importancia de la validación de software y hardware en el desarrollo de sistemas robóticos. Mediante simulaciones en Gazebo y pruebas en entornos reales, aseguramos que nuestros sistemas fueran robustos y capaces de operar de manera óptima incluso en condiciones adversas. La validación exhaustiva permitió identificar áreas de mejora y optimizar el rendimiento del sistema.

Este curso ha sido una experiencia transformadora que ha combinado teoría y práctica, preparando a los estudiantes para enfrentar desafíos complejos en el campo de la robótica. La capacidad para resolver problemas, integrar sistemas y trabajar en equipo son habilidades que hemos desarrollado y demostrado a lo largo de este proyecto, subrayando nuestra preparación para contribuir eficazmente en el ámbito de la robótica.

Estas conclusiones no solo tienen implicaciones para la investigación académica en robótica móvil, sino también para aplicaciones prácticas en una variedad de industrias, desde la logística y la manufactura hasta la exploración espacial y la atención médica. El avance en la navegación autónoma de robots móviles no solo aumentará la eficiencia y la seguridad de las operaciones robóticas, sino que

también abrirá nuevas oportunidades para la innovación y el progreso tecnológico en el futuro.

1) GONZÁLEZ: Este proyecto demuestra la eficacia del filtro de Kalman en la mejora de la precisión de la odometría y la localización del Puzzlebot en entornos simulados. Como horizonte de trabajo adicional, se podría explorar la aplicación del filtro de Kalman en contextos del mundo real para evaluar su rendimiento bajo condiciones más desafiantes y variables. Además, sería interesante investigar la integración de sensores adicionales, como cámaras de profundidad, para mejorar aún más la percepción y navegación autónoma del robot.

2) RODRÍGUEZ: A lo largo de este proyecto se resalta la importancia de la integración de múltiples fuentes de datos, como los encoders y los marcadores Aruco, para así poder mejorar la precisión de la navegación autónoma. Un área de trabajo futura sugerida sería la optimización de los algoritmos de predicción y actualización del filtro de Kalman para reducir el tiempo de procesamiento y mejorar la eficiencia del sistema. Asimismo, se podría investigar la aplicación de técnicas de aprendizaje automático para adaptar dinámicamente los parámetros del filtro según el entorno operativo.

3) MARTÍNEZ: En conclusión, este proyecto evidencia el potencial de los algoritmos de navegación reactiva y el uso de filtros de Kalman para sortear obstáculos y alcanzar objetivos en entornos simulados. Una vía de investigación futura podría ser la implementación de estrategias de control más avanzadas, como la optimización de la función de costo en el filtro de Kalman, para mejorar el desempeño del robot en entornos dinámicos y no estructurados. Además, se podría explorar la mejora de los algoritmos de estimación para adaptarse mejor a distintos tipos de entornos y situaciones.

4) GARCÍA: Este proyecto proporciona una visión clara de los desafíos y oportunidades en la navegación autónoma de robots móviles mediante el uso de filtros de Kalman y marcadores ArUco. Como áreas de investigación futura, se podría explorar la integración de técnicas de percepción avanzada, como la detección de objetos en 3D y la semántica ambiental, para mejorar la capacidad del robot de interactuar de manera más inteligente con su entorno. Además, sería interesante investigar la implementación de estrategias de planificación de rutas más adaptativas y robustas para enfrentar entornos cambiantes y desconocidos.

5) CARRIZALEZ: El proyecto demuestra que el filtro de Kalman es una herramienta efectiva para mejorar la precisión de la navegación autónoma en entornos simulados, utilizando datos de encoders y marcadores Aruco. Como futuras direcciones de investigación, se podría explorar el uso de técnicas de navegación basadas en

mapas, como la SLAM (Simultaneous Localization and Mapping), para permitir que los robots construyan y actualicen mapas de su entorno en tiempo real. Además, sería interesante investigar la implementación de estrategias de navegación cooperativa para permitir que múltiples robots colaboren en tareas de exploración y mapeo. Referente a los filtros de Kalman sería interesante cómo podrían acoplarse más sensores para reducir la incertidumbre a la odometría incluso en entornos no conocidos.

VII. PROJECT MANAGEMENT

<https://drive.google.com/drive/folders/1BBA0bnLqppkWnMNvfbKoO9VImWiZtc7g?usp=sharing>

VIII. AGRADECIMIENTOS

Nos gustaría expresar nuestro más sincero agradecimiento a los siguientes profesores, cuyo apoyo y orientación han sido fundamentales para el desarrollo de este paper:

Primero, queremos agradecer al Profesor Jesús Arturo Escobedo Cabello por su valiosa asesoría durante todo el proceso de formación dentro del bloque. Sus conocimientos y experiencia han sido de gran ayuda para la conceptualización y el diseño de nuestro proyecto.

También deseamos agradecer al Profesor Juan Manuel Ledesma Rangel por su tiempo y dedicación al revisar y comentar nuestros trabajos del submódulo así como el apoyo dado al consultar por asesoramiento. Sus sugerencias y observaciones críticas han mejorado significativamente la calidad de nuestro trabajo.

Asimismo, queremos expresar nuestro agradecimiento al Profesor Benigno Muñoz Barrón por sus valiosos aportes en el desarrollo e implementación de control y de nuevas tecnologías asociadas a los cobot. Su experiencia en este campo ha sido fundamental para obtener resultados precisos y confiables.

No podemos dejar de mencionar al Profesor Josué González García, cuya guía en la creación y entendimiento de los filtros de kalman ha sido inestimable. Sus comentarios detallados y su habilidad para comunicar ideas complejas nos han ayudado a mejorar la calidad de nuestro trabajo.

Por último, pero no menos importante, deseamos agradecer A el Profesor Pablo Estéfano Arroyo Garrido por su apoyo en la planeación y gestión de este proyecto, al mostrarnos las herramientas y competencias necesarias para guiar un proyecto de esta magnitud.

En conjunto, el aporte de estos cinco profesores ha sido vital en nuestro proceso dentro del proyecto. Apreciamos

enormemente su dedicación, conocimientos y apoyo constante a lo largo de este proyecto. Sus contribuciones han enriquecido nuestro aprendizaje y han sido fundamentales para el éxito de este papel.

Extendemos nuestro agradecimiento a todos los profesores mencionados y a aquellos que, aunque no se mencionan específicamente, también han brindado su apoyo y orientación. Su compromiso con la educación y su pasión por la investigación han dejado una huella duradera en nuestro desarrollo académico.

IX. REFERENCES

- [1] K. Perumbhudur, "Path Planning Algorithm for a Two Wheel Differential Mobile Robot in Robot Operating System." Order No. 28866528, Texas A&M University - Kingsville, United States -- Texas, 2021.
- [2] D. Wonnacott, M. Karhumaa, J. Walker, and N. Onder, "Autonomous Navigation Planning with ROS", pp. 2-12, 2012
- [3] Cañas, José M., Eduardo Perdices, Lía García-Pérez, and Jesús Fernández-Conde. 2020. "A ROS-Based Open Tool for Intelligent Robotics Education" *Applied Sciences* 10, no. 21: 7419. <https://doi.org/10.3390/app10217419>
- [4] Samuel Parra, Argentina Ortega, Sven Schneider, Nico Hochgeschwender, "A Thousand Worlds: Scenery Specification and Generation for Simulation-Based Testing of Mobile Robot Navigation Stacks", 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp.5537-5544, 2023.
- [5] Ali Karimoddini, Mubbashar Altaf Khan, Solomon Gebreyohannes, Mike Heiges, Ethan Trewitt, Abdollah Homaifar, "Automatic Test and Evaluation of Autonomous 2011.
- [6] Adler, B., Xiao, J., Zhang, J.: Autonomous exploration of urban environments using unmanned aerial vehicles. *Journal of Field Robotics*, vol. 31, no. 6, pp. 912–939 (2014)
- [7]MDPI and ACS Style Juarez-Lora, A.; Rodriguez-Angeles, A. Bio-Inspired Autonomous Navigation and Formation Controller for Differential Mobile Robots. *Entropy* 2023, 25, 582. <https://doi.org/10.3390/e25040582>

X. APÉNDICE

Anexos con código fuente utilizado en el proyecto así como el link del repositorio de Git Hub

Link: https://github.com/Andyglar/final_challenge

Link: <https://github.com/A01024901/ADJC-bot>

Bug 2

```

#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist, PoseStamped
from std_msgs.msg import Float32
from nav_msgs.msg import Odometry
from tf.transformations import euler_from_quaternion
from sensor_msgs.msg import LaserScan
import numpy as np
import conditions as cnd
import gtg as gtg
import wf as wf

#This class will make the puzzlebot move to a given goal
class AutonomousNav():
    def __init__(self):
        rospy.on_shutdown(self.cleanup)

    ##### ROBOT CONSTANTS #####
    self.r=0.05 #wheel radius [m]
    self.L = 0.19 #wheel separation [m]

    ##### Variables #####
    self.x = 0.0 #x position of the robot [m]
    self.y = 0.0 #y position of the robot [m]
    self.theta = 0.0 #angle of the robot [rad]

    ##### Variables #####
    self.x_target = 1.5
    self.y_target = 1.0
    self.goal_received = 1 #flag to indicate if the goal has been received
    self.lidar_received = 0 #flag to indicate if the laser scan has been received
    self.target_position_tolerance=0.10 #acceptable distance to the goal to declare the robot has arrived to it [m]
    wf_distance = 0.2
    self.integral = 0.0
    self.prev_error = 0.0
    stop_distance = 0.08 # distance from closest obstacle to stop the robot [m]
    v_msg=Twist() #Robot's desired speed
    self.wr=0 #right wheel speed [rad/s]
    self.wl=0 #left wheel speed [rad/s]
    self.current_state = 'GoToGoal' #Robot's current state

    ##### INIT PUBLISHERS #####
    self.pub_cmd_vel = rospy.Publisher('/cmd_vel', Twist,
queue_size=1)

    ##### SUBSCRIBERS #####
    rospy.Subscriber("/wl", Float32, self.wl_cb)
    rospy.Subscriber("/wr", Float32, self.wr_cb)
    rospy.Subscriber("/scan", LaserScan, self.laser_cb)
    rospy.Subscriber("puzzlebot_goal", PoseStamped, self.goal_cb)
    rospy.Subscriber("odom", Odometry, self.odom_cb)

    ##### INIT NODE #####
    freq = 20
    rate = rospy.Rate(freq) #freq Hz
    dt = 1.0/float(freq) #Dt is the time between one calculation and the next one
    fwf = True
    d_t1 = 0.0

```

```

d_t = 0.0
theta_ao = 0.0
theta_gtg = 0.0

##### MAIN LOOP #####
while not rospy.is_shutdown():

    print("\nX_robot: ", self.x)
    print("Y_robot: ", self.y)
    print("X_target: ", self.x_target)
    print("Y_target: ", self.y_target, "\n")

    if self.lidar_received:
        closest_range, closest_angle =
        self.get_closest_object(self.lidar_msg) #get the closest object range and angle

    if self.current_state == 'Stop':
        if self.goal_received and closest_range > wf_distance:
            print("Change to Go to goal from stop")
            self.current_state = "GoToGoal"
            self.goal_received = 0

    elif self.goal_received and closest_range < wf_distance:
        print("Change to wall following from Stop")
        self.current_state= "WallFollower"

    else:
        v_msg.linear.x = 0.0
        v_msg.angular.z =0.0
    elif self.current_state == 'GoToGoal':
        if self.at_goal() or closest_range < stop_distance:
            print("Change to Stop from Go to goal")
            print(self.at_goal())

    print(np.sqrt((self.x_target-self.x)**2+(self.y_target-self.y)**2))

    print(self.x_target)
    print(self.y_target)
    self.current_state = "Stop"

    elif closest_range < wf_distance:
        print("Change to wall following from Go to goal")
        self.current_state= "WallFollower"

    else:
        v_gtg, w_gtg = self.compute_gtg_control(self.x_target,
self.y_target, self.x, self.y, self.theta)
        v_msg.linear.x = v_gtg
        v_msg.angular.z = w_gtg

    elif self.current_state == 'WallFollower':
        theta_gtg, theta_ao = cnd.compute_angles(self.x_target,
self.y_target, self.x, self.y, self.theta, closest_angle)
        d_t =
np.sqrt((self.x_target-self.x)**2+(self.y_target-self.y)**2)
        print("Quit?: ",cnd.quit_wf_bug_two(theta_gtg, theta_ao,
self.x_target, self.y_target, self.x, self.y))

        if self.at_goal() or closest_range < stop_distance:
            print("Change to Stop")
            self.current_state = "Stop"
            fwf = True

        elif cnd.quit_wf_bug_two(theta_gtg, theta_ao,
self.x_target, self.y_target, self.x, self.y):
            print("Change to Go to goal from wall follower")
            self.current_state = "GoToGoal"
            fwf = True

        else:

```

```

        d_t1 =
np.sqrt((self.x_target-self.x)**2+(self.y_target-self.y)**2) if fwf else
d_t1
        clk_cnt = cnd.clockwise_counter(self.x_target,
self.y_target, self.x, self.y, self.theta, closest_angle) if fwf else clk_cnt
        fwf = False
        v_wf, w_wf =
wf.compute_wf_controller(closest_angle,closest_range, 0.18, clk_cnt,
dt)
        v_msg.linear.x = v_wf
        v_msg.angular.z = w_wf

        self.pub_cmd_vel.publish(v_msg)

        rate.sleep()

def at_goal(self):
    return
np.sqrt((self.x_target-self.x)**2+(self.y_target-self.y)**2)<self.target_
position_tolerance

def get_closest_object(self, lidar_msg):
    min_idx = np.argmin(lidar_msg.ranges)
    closest_range = lidar_msg.ranges[min_idx]
    closest_angle = lidar_msg.angle_min + min_idx *
lidar_msg.angle_increment
    closest_angle = np.arctan2(np.sin(closest_angle),
np.cos(closest_angle))

    return closest_range, closest_angle

def compute_gtg_control(self, x_target, y_target, x_robot, y_robot,
theta_robot):
    kvmax = 0.2 #linear speed maximum gain
    kwmax = 1.0 #angular angular speed maximum gain

    av = 1.0 #Constant to adjust the exponential's growth rate
    aw = 2.0 #Constant to adjust the exponential's growth rate

    ed = np.sqrt((x_target-x_robot)**2+(y_target-y_robot)**2)

    #Compute angle to the target position

    theta_target = np.arctan2(y_target-y_robot,x_target-x_robot)
    e_theta = theta_target-theta_robot

    #limit e_theta from -pi to pi
    e_theta = np.arctan2(np.sin(e_theta), np.cos(e_theta))
    #Compute the robot's angular speed
    kw = kwmax*(1-np.exp(-aw*e_theta**2))/abs(e_theta) #Constant
    to change the speed
    w = kw*e_theta

    if abs(e_theta) > np.pi/8:
        v = 0 #linear speed

    else:
        kv = kvmax*(1-np.exp(-av*ed**2))/abs(ed) #Constant to
        change the speed
        v = kv*ed #linear speed

    return v, w

def laser_cb(self, msg):
    ## This function receives a message of type LaserScan
    self.lidar_msg = msg
    self.lidar_received = 1

def wl_cb(self, wl):

```

```

## This function receives a the left wheel speed [rad/s]
self.wl = wl.data

def wr_cb(self, wr):
    ## This function receives a the right wheel speed.
    self.wr = wr.data

def odom_cb(self, msg):
    self.x = msg.pose.pose.position.x
    self.y = msg.pose.pose.position.y

    orientation_q = msg.pose.pose.orientation
    orientation_list = [orientation_q.x, orientation_q.y,
orientation_q.z, orientation_q.w]
    (r, p, yaw) = euler_from_quaternion(orientation_list)

    self.theta = np.arctan2(np.sin(self.theta), np.cos(self.theta))

def goal_cb(self, goal):
#    self.x_target = goal.pose.position.x
#    self.y_target = goal.pose.position.y

    self.goal_received=1

def cleanup(self):
    vel_msg = Twist()
    self.pub_cmd_vel.publish(vel_msg)

#####
##### MAIN PROGRAM #####
#####

if __name__ == "__main__":
    rospy.init_node("go_to_goal_with_obstacles1", anonymous=True)
    AutonomousNav()
```

Bug 0

```

#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist, PoseStamped
from std_msgs.msg import Float32
from nav_msgs.msg import Odometry
from sensor_msgs.msg import LaserScan
from tf.transformations import euler_from_quaternion
import numpy as np
import conditions as cnd
import gtg as gtg
import wf as wf

#This class will make the puzzlebot move to a given goal
class AutonomousNav():
    def __init__(self):
        rospy.on_shutdown(self.cleanup)

#####
##### ROBOT CONSTANTS #####
#####

self.r=0.05 #wheel radius [m]
self.L = 0.19 #wheel separation [m]

#####
##### Variables #####
#####

self.x = 3.0 #x position of the robot [m]
self.y = 3.0 #y position of the robot [m]
self.theta = np.pi/2 #angle of the robot [rad]

#####
##### Variables #####
#####


```

```

self.x_target = 0
self.y_target = 0
self.goal_received = 1 #flag to indicate if the goal has been
received
    self.lidar_received = 0 #flag to indicate if the laser scan has been
received
        self.target_position_tolerance=0.10 #acceptable distance to the
goal to declare the robot has arrived to it [m]
            wf_distance = 0.3
            self.integral = 0.0
            self.prev_error = 0.0
            stop_distance = 0.09 # distance from closest obstacle to stop the
robot [m]
                v_msg=Twist() #Robot's desired speed
                self.wr=0 #right wheel speed [rad/s]
                self.wl=0 #left wheel speed [rad/s]
                self.current_state = 'GoToGoal' #Robot's current state

#####
##### INIT PUBLISHERS #####
#####

self.pub_cmd_vel =
rospy.Publisher('/puzzlebot_1/base_controller/cmd_vel', Twist,
queue_size=1)

#####
##### SUBSCRIBERS #####
#####

rospy.Subscriber("puzzlebot_1/wl", Float32, self.wl_cb)
rospy.Subscriber("puzzlebot_1/wr", Float32, self.wr_cb)
rospy.Subscriber("puzzlebot_1/scan", LaserScan, self.laser_cb)
rospy.Subscriber("puzzlebot_goal", PoseStamped, self.goal_cb)
rospy.Subscriber("puzzlebot_1/base_controller/odom", Odometry,
self.odom_cb)

#####
##### INIT NODE #####
#####

freq = 20
rate = rospy.Rate(freq) #freq Hz
dt = 1.0/float(freq) #Dt is the time between one calculation and
the next one
    fwf = True
    d_t1 = 0.0
    d_t = 0.0
    theta_ao = 0.0
    theta_gtg = 0.0

#####
##### MAIN LOOP #####
#####

while not rospy.is_shutdown():
    if self.lidar_received:
        closest_range, closest_angle =
self.get_closest_object(self.lidar_msg) #get the closest object range and
angle

    if self.current_state == 'Stop':
        if self.goal_received:
            print("Change to Go to goal from stop")
            self.current_state = "GoToGoal"
        else:
            v_msg.linear.x = 0.0
            v_msg.angular.z = 0.0

    elif self.current_state == 'GoToGoal':
        if self.at_goal() or closest_range < stop_distance:
            print("Change to Stop from Go to goal")
            print(self.x_target)
            print(self.y_target)
            self.current_state = "Stop"
        elif closest_range < wf_distance:
            print("Change to wall following from Go to goal")
            self.current_state = "WallFollower"
        else:
            v_gtg, w_gtg = gtg.compute_gtg_control(self.x_target,
self.y_target, self.x, self.y, self.theta)
            v_msg.linear.x = v_gtg
            v_msg.angular.z = w_gtg

    elif self.current_state == 'WallFollower':
        theta_gtg, theta_ao = cnd.compute_angles(self.x_target,
self.y_target, self.x, self.y, self.theta, closest_angle)
        d_t =
np.sqrt((self.x_target-self.x)**2+(self.y_target-self.y)**2)
        if self.at_goal() or closest_range < stop_distance:
            print("Change to Stop")
            self.current_state = "Stop"
            ffw = True
        elif cnd.quit_wf_bug_zero(theta_gtg, theta_ao, d_t, d_t1):
            print("Change to Go to goal from wall follower")
            self.current_state = "GoToGoal"
            ffw = True
        else:
            d_t1 =
np.sqrt((self.x_target-self.x)**2+(self.y_target-self.y)**2) if ffw else
d_t1
            clk_cnt = cnd.clockwise_counter(self.x_target,
self.y_target, self.x, self.y, self.theta, closest_angle) if ffw else clk_cnt
            ffw = False
            v_wf, w_wf = wf.compute_wf_controller(closest_angle,
closest_range, 0.4, clk_cnt)
            v_msg.linear.x = v_wf
            v_msg.angular.z = w_wf

            self.pub_cmd_vel.publish(v_msg)
            rate.sleep()

def at_goal(self):
    return
np.sqrt((self.x_target-self.x)**2+(self.y_target-self.y)**2)<self.target_
position_tolerance

def get_closest_object(self, lidar_msg):
    min_idx = np.argmin(lidar_msg.ranges)
    closest_range = lidar_msg.ranges[min_idx]
    closest_angle = lidar_msg.angle_min + min_idx *
lidar_msg.angle_increment
    closest_angle = np.arctan2(np.sin(closest_angle),
np.cos(closest_angle))

    return closest_range, closest_angle

def laser_cb(self, msg):
    ## This function receives a message of type LaserScan
    self.lidar_msg = msg
    self.lidar_received = 1

def wl_cb(self, wl):
    ## This function receives a the left wheel speed [rad/s]
    self.wl = wl.data

def wr_cb(self, wr):
    ## This function receives a the right wheel speed.
    self.wr = wr.data

def odom_cb(self, msg):
    self.x = msg.pose.pose.position.x
    self.y = msg.pose.pose.position.y

    orientation_q = msg.pose.pose.orientation
    orientation_list = [orientation_q.x, orientation_q.y,
orientation_q.z, orientation_q.w]

```

```

print("Change to wall following from Go to goal")
self.current_state = "WallFollower"
else:
    v_gtg, w_gtg = gtg.compute_gtg_control(self.x_target,
self.y_target, self.x, self.y, self.theta)
    v_msg.linear.x = v_gtg
    v_msg.angular.z = w_gtg

elif self.current_state == 'WallFollower':
    theta_gtg, theta_ao = cnd.compute_angles(self.x_target,
self.y_target, self.x, self.y, self.theta, closest_angle)
    d_t =
np.sqrt((self.x_target-self.x)**2+(self.y_target-self.y)**2)
    if self.at_goal() or closest_range < stop_distance:
        print("Change to Stop")
        self.current_state = "Stop"
        ffw = True
    elif cnd.quit_wf_bug_zero(theta_gtg, theta_ao, d_t, d_t1):
        print("Change to Go to goal from wall follower")
        self.current_state = "GoToGoal"
        ffw = True
    else:
        d_t1 =
np.sqrt((self.x_target-self.x)**2+(self.y_target-self.y)**2) if ffw else
d_t1
        clk_cnt = cnd.clockwise_counter(self.x_target,
self.y_target, self.x, self.y, self.theta, closest_angle) if ffw else clk_cnt
        ffw = False
        v_wf, w_wf = wf.compute_wf_controller(closest_angle,
closest_range, 0.4, clk_cnt)
        v_msg.linear.x = v_wf
        v_msg.angular.z = w_wf

        self.pub_cmd_vel.publish(v_msg)
        rate.sleep()

def at_goal(self):
    return
np.sqrt((self.x_target-self.x)**2+(self.y_target-self.y)**2)<self.target_
position_tolerance

def get_closest_object(self, lidar_msg):
    min_idx = np.argmin(lidar_msg.ranges)
    closest_range = lidar_msg.ranges[min_idx]
    closest_angle = lidar_msg.angle_min + min_idx *
lidar_msg.angle_increment
    closest_angle = np.arctan2(np.sin(closest_angle),
np.cos(closest_angle))

    return closest_range, closest_angle

def laser_cb(self, msg):
    ## This function receives a message of type LaserScan
    self.lidar_msg = msg
    self.lidar_received = 1

def wl_cb(self, wl):
    ## This function receives a the left wheel speed [rad/s]
    self.wl = wl.data

def wr_cb(self, wr):
    ## This function receives a the right wheel speed.
    self.wr = wr.data

def odom_cb(self, msg):
    self.x = msg.pose.pose.position.x
    self.y = msg.pose.pose.position.y

    orientation_q = msg.pose.pose.orientation
    orientation_list = [orientation_q.x, orientation_q.y,
orientation_q.z, orientation_q.w]

```

```

(_,_,> self.theta) = euler_from_quaternion(orientation_list)
self.theta = np.arctan2(np.sin(self.theta), np.cos(self.theta))

def goal_cb(self,goal):
    self.x_target = goal.pose.position.x
    self.y_target = goal.pose.position.y

    self.goal_received=1

def cleanup(self):
    vel_msg = Twist()
    self.pub_cmd_vel.publish(vel_msg)

#####
##### MAIN PROGRAM #####
#####

if __name__ == "__main__":
    rospy.init_node("go_to_goal_with_obstacles1", anonymous=True)
    AutonomousNav()

```

Código Go to Goal

```

#!/usr/bin/env python
import rospy
import numpy as np
from dead_reckoning_class import dead_reckoning
from std_msgs.msg import Float32
from std_msgs.msg import Bool
from nav_msgs.msg import Odometry
from tf.transformations import quaternion_from_euler
from std_msgs.msg import Float32MultiArray
import tf2_ros #ROS package to work with transformations
from geometry_msgs.msg import TransformStamped

class localisation:
    def __init__(self, mode):
        ##### Inicio del Nodo #####
        rospy.init_node('localisation')
        rospy.on_shutdown(self.cleanup)

        if mode == "sim":
            wr_sub = "/puzzlebot_1/wr"
            wl_sub = "/puzzlebot_1/wl"
            self.ref = "puzzlebot_1/base_footprint"

        elif mode == "real":
            wr_sub = "/wr"
            wl_sub = "/wl"
            self.ref = "/base_link"

        ##### Subscriptores #####
        rospy.Subscriber(wr_sub, Float32, self.wr_cb)
        rospy.Subscriber(wl_sub, Float32, self.wl_cb)
        rospy.Subscriber("ar_array", Float32MultiArray, self.ary_cb)
        rospy.Subscriber("arucos_flag", Bool, self.flag_cb)

        ##### Publishers #####
        self.odom_pub = rospy.Publisher("odom", Odometry,
                                       queue_size=1)

        ##### Robot Constants #####
        self.r = 0.05
        self.l = 0.19
        self.dt = 0.02

        ##### Variables #####
        self.v = 0.0

```

```

self.w = 0.0
self.wr = 0.0
self.wl = 0.0
self.ar_arr = []
self.flag = False

self.odom = Odometry()
self.covariance = dead_reckoning(self.dt, 0.33, 1.72, 0)
#self.covariance = dead_reckoning(self.dt, 0.0, 0.0, 0)
rate = rospy.Rate(int(1.0/self.dt))
self.tf2_ros = tf2_ros.TransformBroadcaster() # Create a
TransformBroadcaster object
self.t = TransformStamped()

#while not rospy.get_time() == 0: print ("Simulacion no
iniciada")#Descomentar en simulacion

while not rospy.is_shutdown():
    self.get_robot_velocities()
    cov_mat, u = self.covariance.calculate(self.v, self.w, self.wr,
                                           self.wl, self.flag, self.ar_arr)
    self.get_transform(u)
    self.get_odom(cov_mat, u)
    print(u)

##### Publish #####
self.odom_pub.publish(self.odom)
rate.sleep()

def get_robot_velocities (self):
    self.v = (self.r * (self.wr + self.wl))/2
    self.w = self.r * (((2*self.v/self.r) - self.wl)-self.wl)/self.l

def get_odom (self , cov_mat , u):
    self.odom.header.frame_id = "odom"
    self.odom.child_frame_id = self.ref
    self.odom.pose.pose.position.x = u[0] #+ 0.1
    self.odom.pose.pose.position.y = u[1]

    quat = quaternion_from_euler(0 , 0 , u[2])
    self.odom.pose.pose.orientation.x = quat[0]
    self.odom.pose.pose.orientation.y = quat[1]
    self.odom.pose.pose.orientation.z = quat[2]
    self.odom.pose.pose.orientation.w = quat[3]

    self.odom.pose.covariance = [0.0] * 36

    self.odom.pose.covariance[0] = cov_mat[0][0] * 17 #Covariance
    in x
    self.odom.pose.covariance[1] = cov_mat[0][1] #Covariance in xy
    self.odom.pose.covariance[5] = cov_mat[0][2] #Covariance in x
    theta
    self.odom.pose.covariance[6] = cov_mat[1][0] #Covariance in y x
    self.odom.pose.covariance[7] = cov_mat[1][1] #Covariance in y
    self.odom.pose.covariance[11] = cov_mat[1][2] #Covariance in y
    theta
    self.odom.pose.covariance[30] = cov_mat[2][0] #Covariance in
    theta x
    self.odom.pose.covariance[31] = cov_mat[2][1] #Covariance in
    theta y
    self.odom.pose.covariance[35] = cov_mat[2][2] * 5 #Covariance
    in theta

    def get_transform(self,u):
        # Fill the transformation information
        self.t.header.stamp = rospy.Time.now()
        self.t.header.frame_id = "odom"
        self.t.child_frame_id = self.ref
        self.t.transform.translation.x = u[0]
        self.t.transform.translation.y = u[1]

```

```

self.t.transform.translation.z = 0.0

q = quaternion_from_euler(0, 0, u[2])
self.t.transform.rotation.x = q[0]
self.t.transform.rotation.y = q[1]
self.t.transform.rotation.z = q[2]
self.t.transform.rotation.w = q[3]
# A transformation is broadcasted instead of published
self.tf2_ros.sendTransform(self.t) #broadcast the
transformation

def wr_cb (self , msg):
    self.wr = msg.data

def wl_cb (self , msg):
    self.wl = msg.data

def ary_cb (self , msg):
    self.ar_arr = msg.data

def flag_cb (self , msg):
    self.flag = msg.data

def cleanup (self):
    print ("Apagando Localsation")
    self.odom_pub.publish(Odometry())

if __name__ == "__main__":
    localisation("real")

```

Dead reckoning

```

import numpy as np
np.set_printoptions(suppress = True)
np.set_printoptions(formatter = {'float': '{: 0.4f}'.format})

class dead_reckoning:
    def __init__(self , dt , x , y , theta):
        ##### Matrix init---#####
        self.u_calc = np.array([x , y , theta])
        self.u_real = np.array([0 , 0 , 0])
        self.u_prev = np.array([0 , 0 , 0])
        self.e = np.array([[0 , 0 , 0] , [0 , 0 , 0] , [0 , 0 , 0]])
        self.e_prev = np.array([[0 , 0 , 0] , [0 , 0 , 0] , [0 , 0 , 0]])
        self.q = np.array([[0 , 0 , 0] , [0 , 0 , 0] , [0 , 0 , 0]])

        ##### Constant Set ---#####
        self.dt = dt
        self.r = 0.05
        self.l = 0.19
        self.lw = 0.25 #KL
        self.rw = 0.25 #KR

    ##### Calc best estimated position ---#####
    def estimated_pos (self , v , w):
        th = np.copy(self.u_calc[2])
        u_o = np.array([self.u_calc[0] + self.dt * v * np.cos(th) ,
                       self.u_calc[1] + self.dt * v* np.sin(th) ,
                       self.u_calc[2] + self.dt * w])
        self.u_calc = np.copy(u_o)

    ##### Linearice model ---#####
    def linearization (self , v):
        th = np.copy(self.u_prev[2])

```

```

h = np.array([[1 , 0 , -self.dt * v * np.sin(th)] ,
             [0 , 1 , self.dt * v * np.cos(th)] ,
             [0 , 0 , 1]])

self.h = np.copy(h)

##### Calculate uncertainty ---#####
def uncertainty (self):
    e_o = self.h.dot(self.e_prev).dot(self.h.T) + self.q
    self.e = np.copy(e_o)

##### Order of execution ---#####
def calculate (self , v , w , wr , wl , flag ,arr):
    self.calcQ(wr , wl)
    self.estimated_pos(v , w)
    self.linearization(v)
    self.uncertainty()
    u = self.u_calc
    e = self.e
    if flag:
        self.correction(arr)
        u = self.u_real
        e = self.e_real

    self.u_prev = np.copy(u)
    self.e_prev = np.copy(e)

    print(u)
    print(e)

    return e , u

def calcQ (self , wr , wl):
    th = np.copy(self.u_prev[2])
    m = np.array([[self.rw * np.abs(wr) , 0] ,
                 [0 , self.lw * np.abs(wl)]])
    sigma = np.array([[np.cos(th) , np.cos(th)] ,
                     [np.sin(th) , np.sin(th)] ,
                     [2/self.l , -2/self.l]])

    calc = 0.5 * self.r * self.dt
    sigma = sigma * calc

    Q = sigma.dot(m).dot(sigma.T)
    self.q = np.copy(Q)

    def correction(self , arr):
        self.xa = arr[0]
        self.ya = arr[1]
        self.z_c = np.array([arr[2] , arr[3]])
        self.R = np.array([[0.03 , 0] , [0 , 0.004]])
        self.obs_model()
        self.uncertainty_pro()
        self.calc_u_e()

    def obs_model(self):
        d_x = self.u_calc[0] - self.xa
        d_y = self.u_calc[1] - self.ya
        p = d_x**2 + d_y**2
        #Observation Model
        self.z_ob = np.array([np.sqrt(p) ,
                             np.arctan2(d_y , d_x) - self.u_calc[2]])
        #Linearization
        self.G = np.array([[-(d_x/np.sqrt(p)) , -(d_y/np.sqrt(p)) , 0] ,
                          [d_y/p , - d_x/p , -1]])

    def uncertainty_pro(self):
        #Uncertainty propagation
        self.Z = self.G.dot(self.e).dot(self.G.T) + self.R

```

```
#Kalman Gain
self.K = self.e.dot(self.G.T).dot(np.linalg.inv(self.Z))

def calc_u_e(self):
    self.u_real = self.u_calc + self.K.dot(self.z_c - self.z_ob)
    self.e_real = (np.eye(3) - self.K.dot(self.G)).dot(self.e)
```

Aruco finder

```
#!/usr/bin/env python3

import rospy
import numpy as np
from arucos import aruco
from fiducial_msgs.msg import FiducialTransformArray
from std_msgs.msg import Bool
from std_msgs.msg import Float32MultiArray

class ArucoFinder:
    def __init__(self, mode):
        ##### Inicio del Nodo #####
        rospy.init_node('aruco_finder')
        rospy.on_shutdown(self.cleanup)

        ##### Subscriptores #####
        rospy.Subscriber("fiducial_transforms", FiducialTransformArray,
                         self.ft_cb)

        ##### Publishers #####
        self.array_pub = rospy.Publisher("/ar_array", Float32MultiArray,
                                         queue_size=1)
        self.flag_pub = rospy.Publisher("/aruco_flag", Bool,
                                         queue_size=1)

        ##### Constants #####
        self.dt = 0.02
        self.robot2origin = np.array([
            [1, 0, 0, 0, 0.1],
            [0, 1, 0, 0, 1.7],
            [0, 0, 1, 0, 0],
            [0, 0, 0, 0, 1.0]])

        ##### Objetos #####
        arucos = [aruco(702, 0.0, 0.80), aruco(701, 0.0, 1.60),
                  aruco(703, 1.73, 0.80),
                  aruco(704, 2.63, 0.39), aruco(705, 2.85, 0.0), aruco(706,
                  2.87, 1.22),
                  aruco(707, 1.74, 1.22)]

        arucos_sim = [aruco(0, 0.02, 0.80), aruco(1, 0.02, 1.60),
                      aruco(2, 1.71, 0.80),
                      aruco(3, 2.58, 0.40), aruco(4, 2.85, 0.01), aruco(5,
                      2.85, 1.98)]

        if mode == "sim": self.arucos = arucos_sim
        elif mode == "real": self.arucos = arucos

        self.flag_msg = Bool()
        self.array_msg = Float32MultiArray()
        self.flag = False

        self.fiducial_transform = FiducialTransformArray()
        rate = rospy.Rate(int(1.0 / self.dt))

        while rospy.get_time() == 0 or self.flag: pass
```

```
while not rospy.is_shutdown():
    flag, ar = self.process_transforms()
    self.pub_msgs(flag, ar)
    rate.sleep()

def process_transforms(self):
    angle_r = 0
    angle = 0
    d = 0
    x = 0
    y = 0
    pub_msg = np.array([
        [1, 0.0, 0, 5],
        [0, 1, 0.0, 4],
        [0.0, 0, 1, 2],
        [0.0, 0.0, 0.0, 1.0]])
    if self.fiducial_transform.transforms:
        flag_msg = True
        for arucos in self.fiducial_transform.transforms:
            for posiciones in self.arucos:
                if arucos.fiducial_id == posiciones.ID:
                    x = arucos.transform.translation.x
                    y = arucos.transform.translation.y
                    z = arucos.transform.translation.z
                    origin2robot = posiciones.transform_robot2aruco(x, y,
                    z)
                    distance, angle_r = self.get_values(origin2robot)
                    comp, _ = self.get_values(pub_msg)
                    if comp > distance:
                        pub_msg = origin2robot
                        d = distance
                        angle = angle_r
                        x = posiciones.x
                        y = posiciones.y
                    else:
                        flag_msg = False
                        d = 0
                        angle = 0
                    array = [x, y, d, angle]
                    print(array)
                    return flag_msg, array

    def pub_msgs(self, flag, t):
        self.array_msg.data = t
        self.flag_msg.data = flag

        self.array_pub.publish(self.array_msg)
        self.flag_pub.publish(self.flag_msg)

    def get_values(self, m):
        d = m[3, 3]
        distance = np.linalg.norm(d)
        angle_rad = np.arctan2(d[1], d[0])
        return distance, angle_rad

    def ft_cb(self, msg):
        self.fiducial_transform = msg
        self.flag = True

    def cleanup(self):
        print("Apagando Localización")

if __name__ == "__main__":
    ArucoFinder("real")
```

Arucos

```
#!/usr/bin/env python3

import rospy
import numpy as np
from fiducial_msgs.msg import FiducialTransformArray
from std_msgs.msg import Bool
from std_msgs.msg import Float32MultiArray

class aruco:
    def __init__(self, ID, x, y):
        self.ID = ID
        self.x = x
        self.y = y

        self.robot2camera = np.array([
            [0.0, 0.0, 1.0, 0.1],
            [1.0, 0.0, 0.0, 0.0],
            [0.0, -1.0, 0.0, 0.065],
            [0.0, 0.0, 0.0, 1.0]])

        self.origin2aruco = np.array([
            [1, 0.0, 0, x],
            [0, 1.0, 0, y],
            [0.0, 0, 1, 0.1],
            [0.0, 0.0, 0.0, 1.0]])

    def transform_origin2robot(self, x, y, z):
        camera2aruco = np.array([
            [1, 0.0, 0, x],
            [0, 1.0, 0, y],
            [0.0, 0, 1, z],
            [0.0, 0.0, 0.0, 1.0]])

        aruco2camera = self.get_inv(camera2aruco)
        camera2robot = self.get_inv(self.robot2camera)

        self.origin2robot =
        self.origin2aruco.dot(aruco2camera).dot(camera2robot)

        return self.origin2robot

    def transform_robot2aruco(self, x, y, z):
        camera2aruco = np.array([
            [1, 0.0, 0, x],
            [0, 1.0, 0, y],
            [0.0, 0, 1, z],
            [0.0, 0.0, 0.0, 1.0]])

        robot2aruco = self.robot2camera.dot(camera2aruco)

        return robot2aruco

    def get_inv(self, T):
        R = T[:3, :3]
        t = T[:3, 3]

        R_inv = R.T

        t_inv = -R_inv @ t

        T_inv = np.eye(4)
        T_inv[:3, :3] = R_inv
        T_inv[:3, 3] = t_inv

        return T_inv
```

Camera calibration

```
#!/usr/bin/env python3

import rospy
import yaml
import os
from sensor_msgs.msg import CameraInfo

def yaml_to_CameraInfo(yaml_fname):
    # Load data from file
    with open(yaml_fname, "r") as file_handle:
        calib_data = yaml.load(file_handle)

    # Parse
    camera_info_msg = CameraInfo()
    camera_info_msg.width = calib_data["image_width"]
    camera_info_msg.height = calib_data["image_height"]
    camera_info_msg.K = calib_data["camera_matrix"]["data"]
    camera_info_msg.D = calib_data["distortion_coefficients"]["data"]
    camera_info_msg.R = calib_data["rectification_matrix"]["data"]
    camera_info_msg.P = calib_data["projection_matrix"]["data"]
    camera_info_msg.distortion_model =
    calib_data["distortion_model"]

    return camera_info_msg

if __name__ == "__main__":
    # Get fname from command line (cmd line input required)
    #import argparse
    #arg_parser = argparse.ArgumentParser()
    #arg_parser.add_argument("filename", help="Path to yaml file
containing "+\
    "#           "camera calibration data")
    #args = arg_parser.parse_args()
    ruta = os.path.dirname(os.path.abspath(__file__))
    filename = ruta + '/os.yaml'

    # Parse yaml file
    camera_info_msg = yaml_to_CameraInfo(filename)

    # Initialize publisher node
    rospy.init_node("camera_info_publisher", anonymous=True)
    publisher = rospy.Publisher("camera_info", CameraInfo,
queue_size=10)
    rate = rospy.Rate(10)

    # Run publisher
    while not rospy.is_shutdown():
        publisher.publish(camera_info_msg)
        rate.sleep()
```

XI. Integrantes



A. González. Nació el 18 de diciembre de 2002, en León, Guanajuato. Cursó estudios de Bachillerato en La Salle, de 2017 a 2020. Actualmente, se encuentra cursando la carrera de Ingeniero en Robótica y Sistemas Digitales en el Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Querétaro, México desde 2020 con planes de graduación para el presente año



D. Rodríguez. Nació el 1 de marzo de 2002 en San Luis Potosí, SLP. Cursó sus estudios de bachillerato en el colegio de bachilleres No. 28 desde el 2017 hasta el 2020. Actualmente, se encuentra cursando la carrera de Ingeniería en Robótica y Sistemas Digitales en el Instituto Tecnológico de Estudios Superiores de Monterrey, Campus Querétaro, México desde el 2020 con próxima graduación en 2024, durante su formación académica profesional incluye un semestre de intercambio en Madrid, España en el año 2023.



J. Martínez Nació el 5 de julio de 2002 en Atlacomulco, Estado de México. Obtuvo el título de Técnico en Mecatrónica otorgado por la Secretaría de Educación Pública (SEP) y cursó sus estudios en el Centro de Bachillerato Tecnológico No. 1 "Lic. Adolfo López Mateos" en Temascalcingo, Estado de México, desde 2016 hasta 2019. Actualmente, se encuentra cursando la carrera de Ingeniero en Robótica y Sistemas Digitales en el Instituto Tecnológico de Estudios Superiores de Monterrey, Campus Querétaro, México desde el año 2020 hasta la actualidad.



I. García Nacido en Michoacán el 23 de marzo del 2001, obtuvo el título de Técnico en Mecatrónica otorgado por la Secretaría de Educación Pública (SEP) y cursó sus estudios en el Centro de Estudios Científicos y Superiores del Estado de Michoacán en el periodo del 2016 al 2019. Actualmente estudiando la carrera de Ingeniería en Robótica y Sistemas Digitales por el Tecnológico de Monterrey campus Querétaro desde 2019 hasta la actualidad.



E. Carrizalez Nació el 19 de Enero de 2002 en Ciudad de México. Estudió la Preparatoria en el ITESM Campus Santa Fe. Actualmente, se encuentra cursando la carrera de Ingeniero en Robótica y Sistemas Digitales en el Instituto Tecnológico de Estudios Superiores de Monterrey, Campus Querétaro, México desde 2020 hasta la actualidad.