Borrador TFG

Maksym Saldat

Marzo 2024

1. Introducción

2. Objetivos

2.1. Descripción general

2.2. Requisitos

- OP1
- OP2
- OP3

2.3. Metodología

2.4. Plan de trabajo

3. Desarollo del software

3.1. Dependencias

3.2. Estructura

3.2.1. Tablas de Control

En el desarrollo del presente Trabajo de Fin de Grado, se ha empleado la estructura de datos std::map proporcionada por el lenguaje de programación C++. Esta estructura se ha revelado como una herramienta fundamental para la gestión eficiente y organizada de datos clave en el sistema implementado.

'std::map' es una estructura de datos asociativa que permite almacenar elementos en pares clave-valor, donde cada clave está asociada a un único valor correspondiente. En el contexto de este proyecto, std::map ha sido utilizada para vincular los nombres de parámetros específicos a direcciones de registros correspondientes en la memoria interna de los motores Dynamixel.

Una de las ventajas fundamentales de std::map radica en su capacidad para proporcionar un acceso rápido y eficiente a los valores almacenados. Al utilizar

las claves asociadas, se pueden recuperar los valores correspondientes de manera instantánea, lo que resulta especialmente beneficioso en un entorno donde se necesita un acceso rápido a la información de los motores Dynamixel.

Otra característica destacada de std::map es su capacidad para manejar de forma eficiente la inserción, eliminación y búsqueda de elementos. Esto se traduce en un código más limpio, organizado y fácil de mantener, ya que std::map se encarga de las operaciones subyacentes de manera transparente.

En resumen, la elección de std::map como estructura de datos en este proyecto ha sido fundamentada en su capacidad para proporcionar un acceso eficiente, organizado y estructurado a la información clave de los motores Dynamixel. Su naturaleza asociativa, ordenada y eficiente en términos de operaciones la convierte en una herramienta invaluable para el desarrollo de sistemas robóticos complejos y de alto rendimiento.

3.2.2. Campos de clase

A continuación, se nombrarán y se explicarán los atributos que pertenecen a un objeto cualquiera de la clase dynamixelMotor. Cabe destacar que, por razones de seguridad para evitar un manejo incorrecto de los valores por parte del usuario, todos los campos de clase utilizan la etiqueta private. Esto significa que no se puede acceder directamente a ellos desde fuera de la clase. Para interactuar con estos atributos, se proporcionan sus respectivos métodos getters (para consultar el valor) y setters (para cambiarlo), los cuales serán detallados en el capítulo 3.2.4.

- **ID:** atributo de tipo básico int, almacena el número de identificación correspondiente al motor. Su importancia radica en que todas las funciones implementadas dentro de la librería que interactúan con los registros del motor requieren este ID como parámetro, determinando así sobre qué motor realizar la acción. Cabe destacar que, aunque es posible asignar el mismo ID a varios dynamixels, esta práctica es altamente desaconsejada debido a que puede ocasionar errores difíciles de identificar y un funcionamiento incorrecto.

- **MODEL:** atributo de tipo básico int, contiene el número de modelo específico del dynamixel. Cada modelo de dynamixel tiene un número único proporcionado por el fabricante. El atributo se utiliza en múltiples métodos para distinguir entre los diferentes modelos existentes y, como resultado, ejecutar diferentes partes del código. Se ha elegido el tipo int para este atributo en lugar de std::string por su mayor conveniencia en el uso de sentencias como switch o if.

- **IDENTIFICATOR:** atributo de tipo std::string, ha sido creado exclusivamente para la comodidad y uso del usuario. Su función es identificar el motor como una pieza dentro de un sistema más amplio compuesto por varios dynamixels.

- **CONTROL_TABLE:** es un elemento de tipo std::map que representa la tabla de control específica de cada Dynamixel, la cual varía según el modelo del motor. Este atributo se inicializa cuando el usuario invoca la función setControlTable() (cuya explicación detallada se encuentra en secciones posteriores). Esta tabla es esencial para la correcta interacción con el Dynamixel, ya que contiene las direcciones de los registros que controlan sus diferentes funcionalidades y parámetros. El uso de un std::map permite un acceso eficiente a estas direcciones durante la ejecución de los métodos de la librería, facilitando así la configuración y lectura de los valores necesarios para el control del motor Dynamixel. El proceso de inicialización de esta tabla se basa en el modelo específico del Dynamixel, determinando las direcciones correspondientes a cada función y parámetro relevante para su correcto funcionamiento.

3.2.3. Atributos estáticos de dynamixelMotor

Los atributos estáticos descritos a continuación, incluidas las constantes de clase, son elementos relacionados con la clase dynamixelMotor que poseen una utilidad específica y constante. Es importante destacar que al ser estáticos, estos atributos pertenecen a la clase en sí misma, no a una instancia particular de la misma.

- **DMXL_MODELS:** Es un std::map que establece una relación entre el número de modelo proporcionado por Dynamixel y un modelo específico del mismo. Este atributo es fundamental dentro de la librería, ya que internamente se manejan los números de modelo para facilitar su gestión. Sin embargo, al mostrar esta información al usuario, es preferible y más comprensible devolver el nombre del modelo en lugar de su número correspondiente.

- **ADDR_DMXL22:** Este elemento del tipo std::map se utiliza en el código para representar el área EEPROM de la tabla de control de los Dynamixels con 22 parámetros en la misma. Su función principal radica en la configuración de la tabla de control de la instancia de dynamixelMotor sobre la cual se ha aplicado el método setControlTable(). Esta versión con 22 parámetros es compatible con los modelos XW, XD430, XH430 y XM430 de Dynamixels.

- **ADDR_DMXL25:** Este miembro del tipo std::map desempeña un papel similar al de ADDR_DMXL22, pero está diseñado específicamente para los modelos de Dynamixels con una tabla de control de 25 parámetros en la EEPROM. Al utilizar la función setControlTable(), este elemento se encarga de configurar la tabla de control de la instancia de dynamixelMotor. La versión con 25 parámetros es compatible con los modelos XD540, XH540 y XM540 de Dynamixels.

- **CURRENT_CONTROL_MODE:** Esta constante de clase está diseñada para referirse al modo de control del motor mediante la corriente suminis-

trada. Se ha creado con el propósito de facilitar al usuario el manejo de los métodos que requieren como parámetro un modo de control específico. Al utilizar dynamixelMotor.CURRENT_CONTROL_MODE, el usuario puede acceder de manera sencilla a los valores numéricos asignados a cada modo de control, sin necesidad de recordarlos o buscarlos manualmente.

- **VELOCITY_CONTROL_MODE:** Esta constante de clase se utiliza para referirse al modo de control del motor mediante una consigna de velocidad. Su propósito es proporcionar al usuario una forma fácil y directa de acceder a los valores numéricos asignados a cada modo de control de velocidad. Al utilizar dynamixelMotor.VELOCITY_CONTROL_MODE, el usuario puede especificar el modo de control deseado al llamar a los métodos relevantes, sin la necesidad de recordar los valores numéricos correspondientes.

- **POSITION_CONTROL_MODE:** Esta constante de clase está diseñada para referirse al modo de control del motor mediante una consigna de posición. Su objetivo es proporcionar al usuario una manera sencilla de acceder a los valores numéricos asignados a cada modo de control de posición. Al utilizar dynamixelMotor.POSITION_CONTROL_MODE, el usuario puede especificar el modo de control deseado al llamar a los métodos pertinentes, sin la necesidad de recordar los valores numéricos correspondientes.

- **EXTENDED_POSITION_CONTROL_MODE:** Este modo de control del motor está diseñado para manejar la posición mediante una consigna extendida. Para utilizar este modo de control en la clase dynamixelMotor, simplemente se hace referencia a la constante EXTENDED_POSITION_CONTROL_MODE, proporcionando así una forma conveniente y legible de configurar el motor para este tipo específico de control.

- **CURRENT_BASED_CONTROL_MODE:** Este modo de control del motor permite tanto el control basado en la corriente como el control mediante una consigna de posición. Proporciona una versatilidad adicional al usuario al permitir el ajuste dinámico del controlador para adaptarse a diferentes situaciones de funcionamiento. Al utilizar la constante CURRENT_BASED_CONTROL_MODE en la clase dynamixelMotor, los usuarios pueden seleccionar fácilmente este modo de control sin tener que preocuparse por los detalles técnicos asociados. Esto simplifica significativamente la interfaz y hace que sea más accesible para aquellos que buscan un control preciso y versátil del motor Dynamixel.

- **PWM_CONTROL_MODE:** Este modo de control del motor permite controlar el motor Dynamixel mediante una consigna de PWM (Modulación por Ancho de Pulso). En este modo, el usuario puede especificar la duración del pulso PWM enviado al motor para controlar su velocidad y posición. Esta funcionalidad es particularmente útil cuando se requiere un control preciso sobre la velocidad y posición del motor Dynamixel en diversas aplicaciones. Al utilizar la constante PWM_CONTROL_MODE

en la clase dynamixelMotor, los usuarios pueden seleccionar fácilmente este modo de control sin tener que preocuparse por los detalles técnicos asociados, lo que simplifica la interfaz y hace que sea más accesible para aquellos que buscan una forma intuitiva de controlar el motor Dynamixel.

3.2.4. Métodos de objeto dynamixelMotor

3.2.5. Métodos estáticos de dynamixelMotor

3.2.6. Interacción con el usuario

Todos los métodos que requieren informar al usuario sobre la ejecución exitosa de una acción, o sobre un error ocurrido, están equipados con código diseñado para tal propósito. En este contexto, y considerando que la librería está concebida completamente para su integración con ROS, se han empleado los métodos proporcionados por la propia librería ROS de C++ (ROS-ERROR, ROS-INFO, entre otros).

La ventaja de utilizar estos métodos radica en que ofrecen una presentación visual más adecuada dependiendo del tipo de mensaje que se desea comunicar (error, advertencia, información, etc.). Además, los mensajes que aparecen en la consola siempre mostrarán en primer lugar el número de identificación del Dynamixel, si la acción está relacionada con una instancia específica de la clase dynamixelMotor. Esto permite al usuario conocer el cambio que se ha producido y sus posibles consecuencias de manera clara y precisa.

3.3. Diagrama de uso

Aquí se presenta el diagrama de uso del sistema, que muestra la secuencia que el usuario final ha de seguir para hacer un uso apropiado de la librería y reducir la probabilidad de error.

3.4. Funcionalidades implementadas

Las funcionalidades implementadas en el proyecto son las siguientes:

- Funcionalidad 1: Breve descripción de la funcionalidad 1.
- Funcionalidad 2: Breve descripción de la funcionalidad 2.
- Funcionalidad 3: Breve descripción de la funcionalidad 3.

3.5. Restricciones

Durante el desarrollo del proyecto, se encontraron las siguientes restricciones:

- Restricción 1: Descripción de la restricción 1.
- Restricción 2: Descripción de la restricción 2.
- Restricción 3: Descripción de la restricción 3.

El proyecto tiene las siguientes dependencias:

- Dependencia 1: Descripción de la dependencia 1.
- Dependencia 2: Descripción de la dependencia 2.
- Dependencia 3: Descripción de la dependencia 3.

4. Ejecución

4.1. Problemáticas encontradas

Durante la ejecución del proyecto, se han manifestado las siguientes problemáticas:

1. **Implementación de las tablas de control**: Para garantizar la flexibilidad y la legibilidad de la librería por parte de los usuarios finales, se ha planteado la necesidad de incorporar las tablas de control de los dynamixel al código en C++. Es importante tener en cuenta que estas tablas pueden variar significativamente según el modelo del dynamixel en cuestión. El desafío principal ha sido encontrar una solución que cumpla con los siguientes criterios:
    - **Eficiencia de memoria:** La solución debe ser eficiente en cuanto al uso de la memoria del dispositivo, evitando cualquier derroche innecesario de recursos.
    - **Claridad y comprensión:** Es fundamental que la solución sea fácil de entender para los usuarios, permitiendo una interacción intuitiva con las tablas de control.
    - **Flexibilidad y escalabilidad:** La solución debe ser lo suficientemente flexible como para permitir futuras expansiones y modificaciones en las tablas de control, sin comprometer la estabilidad del código.
2. **Velocidades de comunicación**: El baudrate de todos los dynamixels y del controlador debería ser único para un trabajo cómodo. Sin embargo por defecto no es así y estas son las herramientas que se han utilziado para poner fronteras para el usuario.

4.2. Soluciones implementadas y alternativas existentes

Para resolver las problemáticas propuestas, se ha oprtado por:

1. **Implementación de las tablas de control:** Las tablas de control se han integrado finalmente mediante el uso del objeto ´map´ del espacio de nombres estándar de C++ (cuyo funcionamiento se detalla en el capítulo .ªstructura"). Se consideraron dos soluciones equivalentes como alternativas:

- **Uso de arrays del tipo std::string:** Esta solución implicaba definir arrays de std::string que contuvieran los nombres de los valores a modificar, con las posiciones en el array representando las direcciones en la tabla de control. Por ejemplo, si existe un parámetro llamado "Límite de PWMz su dirección en la tabla es "32", el array debería tener un std::string con el valor "Límite de PWM.en la posición "32". Esta aproximación no se implementó finalmente debido a que no es eficiente en cuanto al uso de memoria en el dispositivo. Las direcciones en las tablas no son consecutivas, sino que dependen del tamaño del parámetro previo. Por lo tanto, si el parámetro "Límite de PWM.ocupa 4 bytes, el siguiente parámetro estaría en la posición "36"del array, dejando las posiciones "33", "34z "35"sin utilizar y, por ende, ineficaces.
- **Uso de archivos del tipo .json:** Esta solución implicaba aprovechar el formato existente de .json para crear archivos que reflejaran las tablas de control y, utilizando una librería externa, manejarlos dentro del código en C++. Sin embargo, esta opción tampoco se implementó debido a que añade una dependencia adicional a la librería, la cual el usuario final no siempre tendría instalada previamente.

2. **Solución 2**: Descripción detallada de la solución 2.
3. **Solución 3**: Descripción detallada de la solución 3.

5. Experimentos y comprobaciones

6. Conclusiones

En conclusión, hemos discutido...

7. Referencias

- Autor A, et al. (2022) Título del artículo. Revista, vol(1), pp. 1-10.
- Autor B, et al. (2023) Título del libro. Editorial, Ciudad.

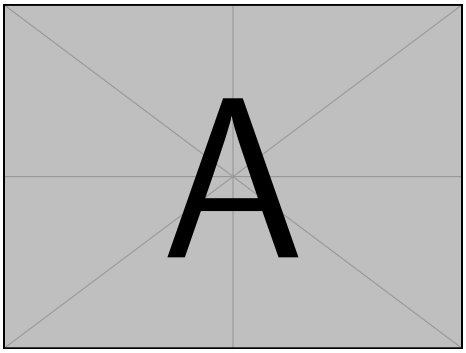

Figura 1: Ejemplo de imagen