



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



Trabajo de Final de Grado
ROBÓTICA: Modelado de Cinemática
Directa e Inversa basado en el Algoritmo de
Denavit-Hartenberg – Desacoplamiento de
Subproblemas

Documentación Técnica
Desarrollo Teórico - Casos Previos y
Aplicación en otras Arquitecturas



Presentado por Jaime Sáiz de la Peña
en Universidad de Burgos — 19/01/2023

Tutores:

José Manuel Sáiz Diez
Raúl Marticorena Sánchez



Índice de contenido

1 Casos a resolver.....	6
1.1 Caso 10-0 – RMD X8 PRO.....	7
1.1.1 Prueba de motor.....	7
1.1.2 Biblioteca para motor RMD-X-8.....	10
1.1.3 Pruebas de funcionalidad - Comunicaciones CAN.....	20
1.1.3.1 Comunicar un Arduino UNO a un motor RMD-X8-PRO por CAN	20
1.1.3.2 Biblioteca RMDx8ArduinoUBU y uso de Encoders en la comunicación de Arduinos UNO y motores RMD-X8-PRO por CAN	31
1.1.3.3 Error de la controladora y ejemplo de uso de Encoders en la comunicación de Arduinos UNO y motores RMD-X8-PRO por CAN	38
1.1.4 Bípido a escala humana.....	44
2 Referencias Bibliográficas.....	46

Illustration Index

Ilustración 1: Valores del ID de motor RMD-X8-PRO activando/desactivando los switches (RMD X8 Pro DC de GYEMS [WWWaliexpressProd165]).....	7
Ilustración 2: Adaptador de USB a UART con cables de conexión Molex 1.25 (RMD X8 Pro DC de GYEMS [WWWaliexpressProd165]).....	7
Ilustración 3: Interfaz de configuración y conexión de RMD V2.0.exe.....	8
Ilustración 4: Interfaz de prueba de RMD V2.0.exe.....	8
Ilustración 5: Lecturas del Encoder en 0° y 1°.....	9
Ilustración 6: Conectores CAN y UART del motor (RMD X8 Pro DC de GYEMS [WWWaliexpressProd165]).....	9
Ilustración 7: Interconexión en serie (RMD X8 Pro DC de GYEMS [WWWaliexpressProd165]).....	9
Ilustración 8: Escudo CAN-BUS. Compatible con Arduino. MCP2515 (CAN-controller) y MCP2551 (CAN-transceptor). Conexión GPS. Lector de tarjetas MicroSD. [WWWaliexpressProd191].....	11
Ilustración 9: Componentes del Escudo CAN-BUS. [WWWamazonProd13].....	12
Ilustración 10: Placa de expansión de BUS CAN-Bus Shield V2, IIC I2C y UART para Arduino [WWWaliexpressProd192].....	12
Ilustración 11: MCP2515 (Bus CAN) [WWWarduinoDoc31].....	13
Ilustración 12: CANBed - Arduino CAN-BUS Development Kit (ATmega32U4 with MCP2515 and MCP2551) [WWWseedstudioProd23].....	13
Ilustración 13: Listado de funciones del fabricante (RMD X8 Pro DC de GYEMS [WWWaliexpressProd165]).....	14
Ilustración 14: Prueba de uso de la biblioteca RMDx8ArduinoUBU ("JaimeSaiz/RMDx8ArduinoUBU" [WWWgithubDrivers105]) y de las funciones del fabricante (RMD X8 Pro DC de GYEMS [WWWaliexpressProd165]).....	15
Ilustración 15: Prueba de giro del motor de 1°.....	16
Ilustración 16: Prueba de giro a velocidad constante.....	17
Ilustración 17: Prueba de giro completo con cambio de valor en el Encoder.....	17
Ilustración 18: Características del motor RMD X8 Pro ("Servomotor de CC sin escobillas RMD X8 Pro, motor de engranaje, controlador foc con codificador de 18 bits, 24V, 48V, 10A" [WWWaliexpressProd188])	18
Ilustración 19: Prueba de giro del motor con modificación del Encoder en 10 unidades.....	19
Ilustración 20: Pinout de CAN Bus [WWWarduinoDoc31].....	21
Ilustración 21: MCP2515 (Bus CAN) [WWWelectronicshubDoc1].....	21
Ilustración 22: Pinout de CAN Bus [WWWarduinoDoc31].....	22
Ilustración 23: Prueba de codificación mediante <RMDx8Arduino.h> sobre el motor "RMD X8 PRO" con Emisor "CAN-BUS Shield" y "mcp-can.h/cpp" y sin utilizar Receptor.....	27
Ilustración 24: Prueba de codificación directa de comandos sobre el motor "RMD X8 PRO" con Emisor "CAN-BUS Shield" y "mcp-can.h/cpp" y sin utilizar Receptor.....	29
Ilustración 25: Prueba de lectura del Encoder sobre el motor "RMD X8 PRO" con Emisor "CAN-BUS Shield" y "mcp-can.h/cpp" y sin utilizar Receptor.....	33
Ilustración 26: Prueba de cálculo de variables para la toma de datos desde el motor "RMD X8 PRO"	37

Ilustración 27: Prueba de visualización de variables para para un ángulo dado del motor "RMD X8 PRO"	38
Ilustración 28: Prueba del ejemplo del uso de Encoders en la comunicación de Arduinos UNO y motores RMD-X8-PRO por CAN para determinar sus datos una vez colocado en su posición.....	39
Ilustración 29: Error entre los valores del Encoder devueltos por el motor a través de la función "readEncoder()" y el valor del ángulo en una vuelta, también devuelto desde el motor por la función "readSingleCircleAngle()" de la biblioea <RMDx8ArduinoUBU.h> - MOTOR 1	40
Ilustración 30: Error entre los valores del Encoder devueltos por el motor a través de la función "readEncoder()" y el valor del ángulo en una vuelta, también devuelto desde el motor por la función "readSingleCircleAngle()" de la biblioea <RMDx8ArduinoUBU.h> - MOTOR 2	41
Ilustración 31: Nuevo error entre los valores del Encoder devueltos por el motor a través de la función "readEncoder()" y el valor del ángulo en una vuelta, también devuelto desde el motor por la función "readSingleCircleAngle()" de la biblioea <RMDx8ArduinoUBU.h>.....	42
Ilustración 32: Prueba de solución de errores entre los valores del Encoder devueltos por el motor a través de la función "readEncoder()" y el valor del ángulo en una vuelta, también devuelto desde el motor por la función "readSingleCircleAngle()" de la biblioea <RMDx8ArduinoUBU.h>.....	43
Ilustración 33: Análisis de DH para la pierna de un Humanoide de tamaño natural.....	44

Index of Tables

Tabla 1: Uso de las bibliotecas "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) y "Seeed-Studio/Seeed_Arduino_CAN – v2.0.0" [WWWgithubDrivers84] con "MCP2515" y "CAN-BUS Shield".....	30
Tabla 2: Uso de las bibliotecas "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) y "Seeed-Studio/Seeed_Arduino_CAN – v2.0.0" [WWWgithubDrivers84] con "CAN-BUS Shield".....	31
Tabla 3: Uso de las bibliotecas "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) y "Seeed-Studio/Seeed_Arduino_CAN – v2.0.0" [WWWgithubDrivers84] con "MCP2515".....	32
Tabla 4: Uso de la biblioteca "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) con dispositivos "MCP2515" como Emisor y "CAN-BUS Shield" como Receptor.....	33
Tabla 5: Uso de la biblioteca "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) con dispositivo "CAN-BUS Shield" como Emisor.....	34
Tabla 6: Uso de la biblioteca "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) con dispositivo "MCP2515" como Emisor.....	34
Tabla 7: Uso de las bibliotecas "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) y "Seeed-Studio/Seeed_Arduino_CAN – v2.0.0" [WWWgithubDrivers84] con "CAN-BUS Shield" y el Ejemplo de codificación directa de comandos sobre el motor "RMD X8 PRO".....	35

1 Casos a resolver

Casos analizados – Motor de cada articulación respecto al anterior y a la base

	Arti-1	Arti-2	Arti-3	Arti-4	Arti-5	
Caso 3-0	Perpen	Perpen				DH
Caso 3-1	Perpen	Perpen	Paralelo			DH+Geométrico
Caso 4-0	Paralelo	Perpen	Paralelo			Cuadrúpodo resuelto por DH
Caso 4-1	Paralelo	Perpen	Paralelo			Cambio de ejes sobre Caso 4-0
Caso 4-2	Paralelo	Paralelo	Paralelo			
Caso 5-0	Perpen	Perpen	Perpen	Paralelo		
Caso 1-0...	Perpen	Perpen	Perpen	Paralelo	Paralelo	
... Arti. 1	Perpen	Fija	Fija	Fija	Fija	DH
... Arti. 2 *	Fija	Perpen	Fija	Fija	Fija	DH-Sin giro de 90°
... Arti. 2	Fija	Perpen	Fija	Fija	Fija	DH-Con giro de 90°
... Arti. 3	Fija	Fija	Perpen	Fija	Fija	DH
... Arti. 4	Fija	Fija	Fija	Paralelo	Fija	DH
... Arti. 5	Fija	Fija	Fija	Fija	Paralelo	DH
... Arti. 1-2	Perpen	Perpen	Fija	Fija	Fija	DH
... Arti. 2-3 *	Fija	Perpen	Perpen	Fija	Fija	DH-Sin giro de 90°
... Arti. 2-3	Fija	Perpen	Perpen	Fija	Fija	DH-Con giro de 90°
... Arti. 3-4	Fija	Fija	Perpen	Paralelo	Fija	DH
... Arti. 4-5	Fija	Fija	Fija	Paralelo	Paralelo	DH
... Arti. 1-2-3	Perpen	Perpen	Perpen	Fija	Fija	DH
... Arti. 1-3-4	Perpen	Fija	Perpen	Paralelo	Fija	DH
... Arti. 2-3-4	Fija	Perpen	Perpen	Paralelo	Fija	DH
... Arti. 3-4-5	Fija	Fija	Perpen	Paralelo	Paralelo	DH
... Arti. 1-2-3-4	Perpen	Perpen	Perpen	Paralelo	Fija	DH
... Arti. 2-3-4-5	Fija	Perpen	Perpen	Paralelo	Paralelo	DH
... Arti. Desacopladas 1-2 + 3-4-5	Perpen	Perpen	Perpen	Paralelo	Paralelo	Brazo resuelto por DH y Acoplamiento, con plano y ángulo de ataque
Caso 7-0-Pierna	Perpen	Paralelo	Paralelo	Paralelo	Perpen	Humanoide resuelto por DH
Caso 7-0-Brazo	Perpen	Paralelo	Paralelo			Humanoide resuelto por DH
Caso 8-0-Mano	Perpen					Mano
Caso 9-0-Brazo2						

1.1 Caso 10-0 – RMD X8 PRO

1.1.1 Prueba de motor

Se comenzará por probar el motor con la aplicación de test suministrado en la página web de la empresa (RMD V2.0.exe).

Antes de comenzar a trabajar con el motor se debe determinar el identificador del motor (ID) con los tres switches situados en la placa de control unida al motor, con los que se podrá identificar de forma única a cada uno de los motores conectados.



Switch1	Switch2	Switch3	ID
OFF	OFF	OFF	#1
ON	OFF	OFF	#2
OFF	ON	OFF	#3
ON	ON	OFF	#4
OFF	OFF	ON	#5
ON	OFF	ON	#6
OFF	ON	ON	#7
ON	ON	ON	#8

Ilustración 1: Valores del ID de motor RMD-X8-PRO activando/desactivando los switches (RMD X8 Pro DC de GYEMS [WWWaliexpressProd165])

Ahora se conectará el motor a un ordenador mediante un "Adaptador de USB a UART con cables de conexión Molex 1.25". Y se conecta el cable de corriente a una fuente de 24V (también puede conectarse a 48V).

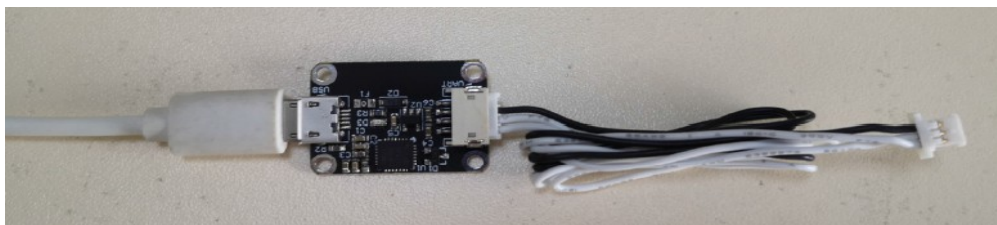


Ilustración 2: Adaptador de USB a UART con cables de conexión Molex 1.25 (RMD X8 Pro DC de GYEMS [WWWaliexpressProd165])

Una vez conectado físicamente, se instalará el driver que permita conectar el motor a través de un puerto USB. Para ello se instala "CP210x_Windows_Drivers" (<http://www.gyems.cn/support/download>).

Ahora, se ejecutará la aplicación, se configura el puerto USB al que está conectado mediante el campo "Select COM", la velocidad de la conexión serie (115200), se configura el ID que previamente haya sido configurado en el motor en la pestaña "ID" y se presiona sobre "CONNECT".

Lo normal es que se encendiera un LED en la placa del adaptador y que funcionara sin problemas.

Por último, se podrán definir varios parámetros para comprobar el correcto funcionamiento del motor, por ejemplo a través de la pestaña "Test", o se pueden comprobar otros mediante las otras pestañas.

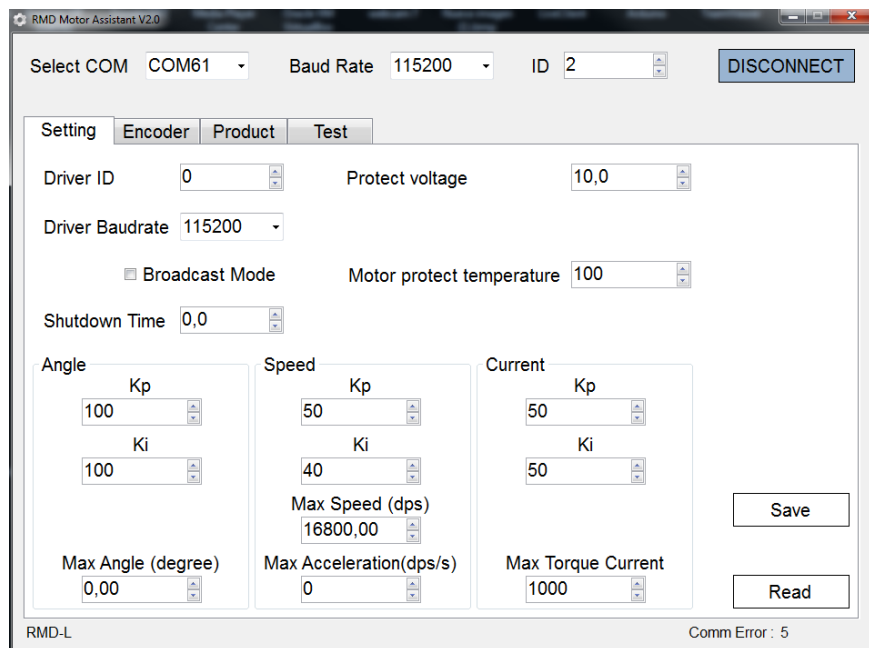


Ilustración 3: Interfaz de configuración y conexión de RMD V2.0.exe

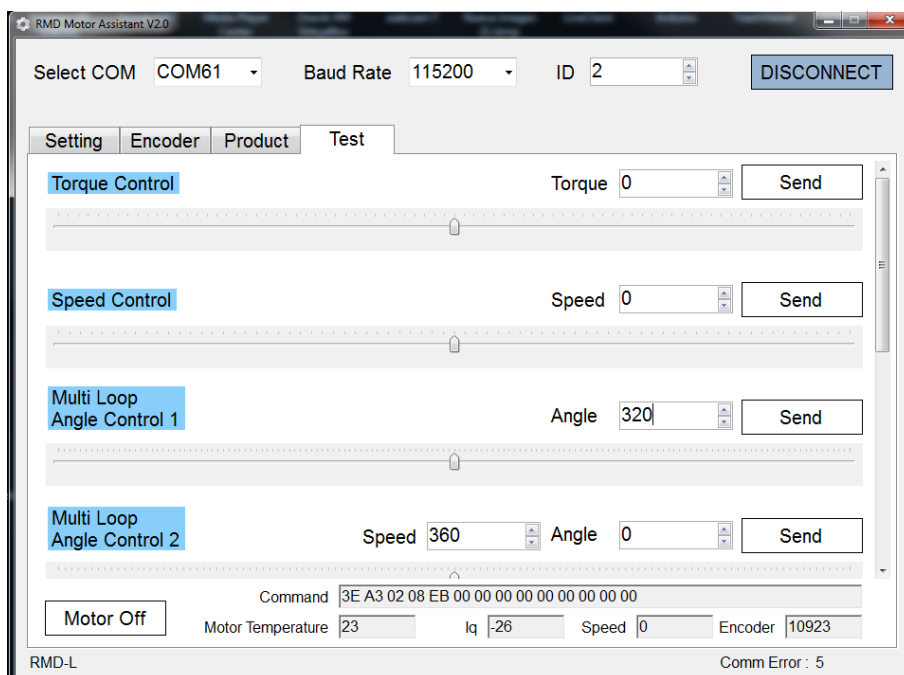


Ilustración 4: Interfaz de prueba de RMD V2.0.exe

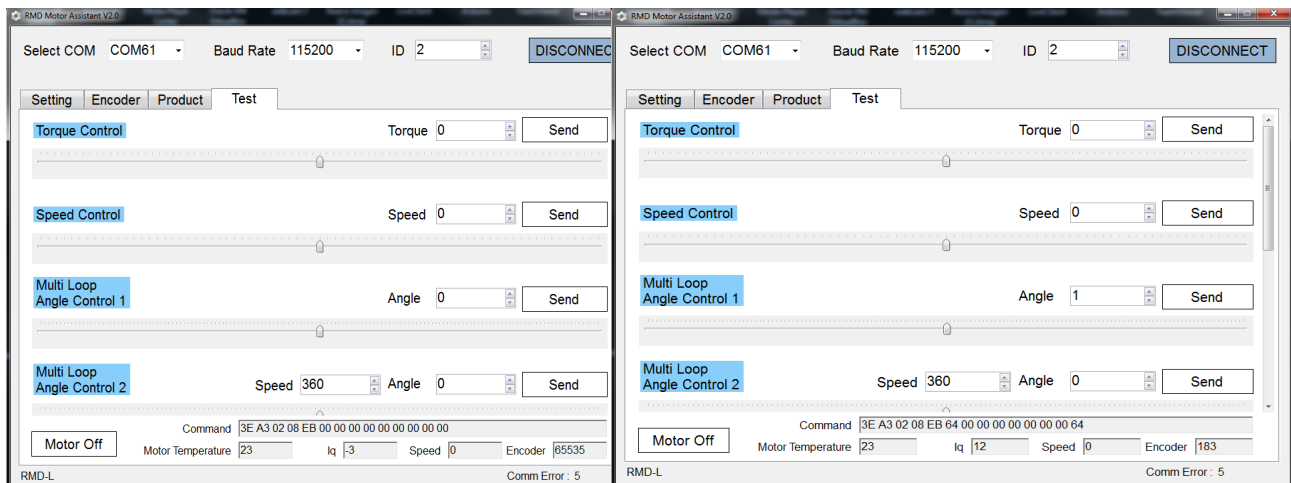


Ilustración 5: Lecturas del Encoder en 0° y 1°.

Una vez probado el motor, se puede montar en la instalación correspondiente. Para ello, se conectará el cable CAN desde el motor hasta el dispositivo de control, teniendo en cuenta que deben estar interconectados todos los cables de los motores en serie para su correcto funcionamiento.

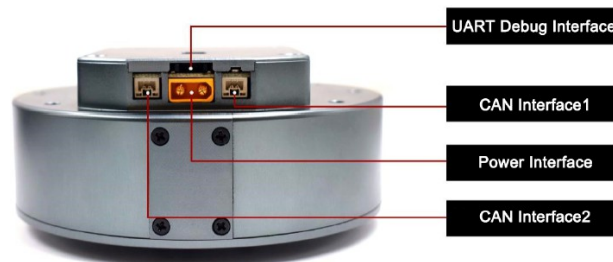


Ilustración 6: Conectores CAN y UART del motor (RMD X8 Pro DC de GYEMS [WWWaliexpressProd165])

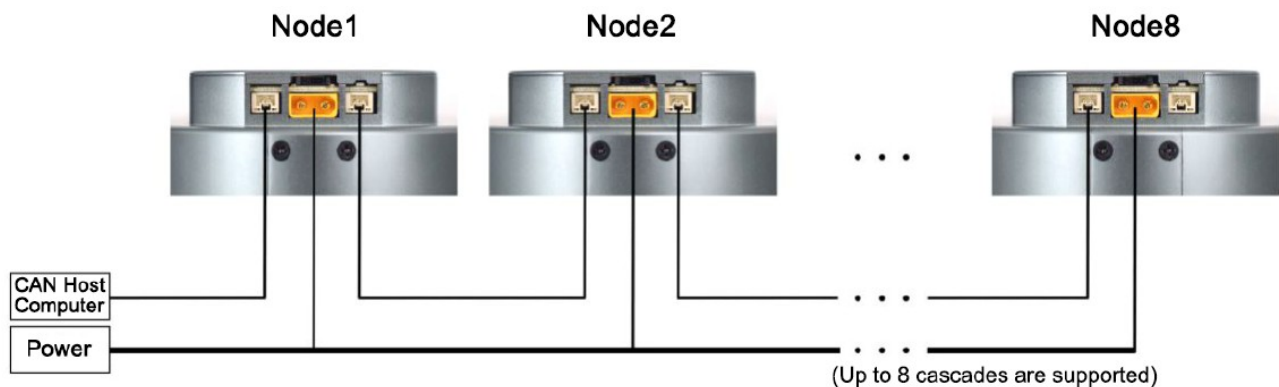


Ilustración 7: Interconexión en serie (RMD X8 Pro DC de GYEMS [WWWaliexpressProd165])

1.1.2 Biblioteca para motor RMD-X-8

A medida que se iban realizando las pruebas y desarrollo de código necesarios para comprobar las características y funcionalidad del motor RMD-X-8, se ha ido desarrollando la biblioteca correspondiente a este motor, dado que la biblioteca original ("bump5236/RMDx8Arduino" [WWWgithubDrivers76]) carecía de muchas de las funciones definidas por el fabricante.

Esta nueva biblioteca se ha publicado como "**JaimeSaiz/RMDx8ArduinoUBU**" [WWWgithubDrivers105].

Se ha cambiado la nomenclatura de la versión original para adaptarla a la nomenclatura del fabricante, por lo que no se podía probar el código de ejemplo original sin modificar esa nomenclatura. Además se han incluido gran parte de las funciones del fabricante, porque tampoco estaban en la biblioteca original.

Así, la funcionalidad total de la biblioteca estaría definida en el fichero "RMDx8ArduinoUBU.h" y su desarrollo, en "RMDx8ArduinoUBU.cpp".

En el fichero "RMDx8ArduinoUBU.h" se definen las funciones a utilizar:

```
#ifndef RMDx8ArduinoUBU_h
#define RMDx8ArduinoUBU_h
#include "Arduino.h"
#include <mcp_can.h>
class RMDx8ArduinoUBU {
    public:
        ...
        RMDx8ArduinoUBU(MCP_CAN &CAN, const uint16_t motor_addr); // Maneja el
            // controlador si tiene el mismo nombre que la clase
        ...
        // Commands
        void canSetup();
        void readPID();
        void writePID(uint8_t anglePidKp, uint8_t anglePidKi, uint8_t speedPidKp, uint8_t
            speedPidKi, uint8_t iqPidKp, uint8_t iqPidKi); // Write PID to RAM parameter command
            // (one frame)
        void writePIDROM(uint8_t anglePidKp, uint8_t anglePidKi, uint8_t speedPidKp, uint8_t
            speedPidKi, uint8_t iqPidKp, uint8_t iqPidKi); // Write PID to ROM parameter command
            // (one frame)
        void readAccel(); // Read acceleration data command (one frame)
        void writeAccel(uint32_t Accel); // Write acceleration data command (one frame)
        void readEncoder(); // The host sends the command to read the current position of the
            // encoder
        void writeEncoderOffset(uint16_t encoderOffset);
        void writePositionROMMotorZero(); // Write current position to ROM as motor zero
            // position command(one frame)
        void readMotorAngle(); // Read multi turns angle command
        void readSingleCircleAngle(); // The host sends command to read the single circle angle of
            // the motor.
        void clearState(); // Turn off motor, while clearing the motor operating status and previously
            // received control commands
        void stopMotor(); // Stop motor, but do not clear the motor operating state and previously
            // received control commands
        void resumeStopMotor(); // Resume motor operation from motor stop command (Recovery
            // control mode before stop motor)
        void readStatus1(); // This command reads the motor's error status and voltage, temperature
            // and other information.
        void clearErrorFlag(); // This command clears the error status of the current motor.
        void readStatus2(); // This command reads motor temperature, voltage, speed, encoder
            // position
```

```

void readStatus3(); // This command reads the phase current status data of the motor.
void writeCurrent(int16_t iqControl);
void writeSpeed(int32_t speedControl);
void writePosition3(int32_t angleControlMT);
void writePosition4(int32_t angleControlMT, uint16_t maxSpeed);
void writePosition5(uint16_t angleControlST, uint8_t spinDirection);
void writePosition6(uint16_t angleControlST, uint16_t maxSpeed, uint8_t spinDirection);
// General function
void serialWriteTerminator();

private:
    MCP_CAN_CAN;

...

```

Cabe destacar que la biblioteca "mcp_can.h" será la correspondiente a la biblioteca que implementa la comunicación del Bus CAN "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]). Y por otra parte el Arduino UNO se podría conectar al motor a través del dispositivo MCP251, y sin embargo, se va a utilizar "**CAN-BUS Shield**", porque se considera que el cableado resultará más sencillo y seguro, dado que cuenta con un conector directo a los dos cables CAN-L y CAN-R, reduciendo mucho el cableado respecto al MCP2515 con hasta 8 cables. Otra posibilidad interesante sería CANBed por razones similares.

- **CAN-BUS Shield - RobotDyn** - "Escudo CAN-BUS. Compatible con Arduino. MCP2515 (CAN-controller) y MCP2551 (CAN-transceptor). Conexión GPS. Lector de tarjetas MicroSD." [WWWaliexpressProd191] - "Seeed-Studio/CAN_BUS_Shield" [WWWgithubDrivers51] - "Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]

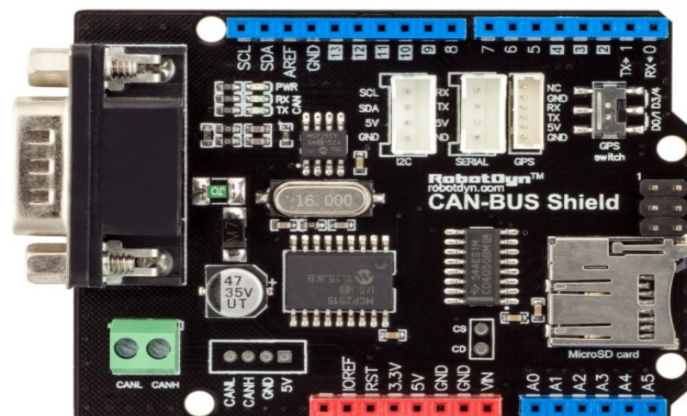


Ilustración 8: Escudo CAN-BUS. Compatible con Arduino. MCP2515 (CAN-controller) y MCP2551 (CAN-transceptor). Conexión GPS. Lector de tarjetas MicroSD. [WWWaliexpressProd191]

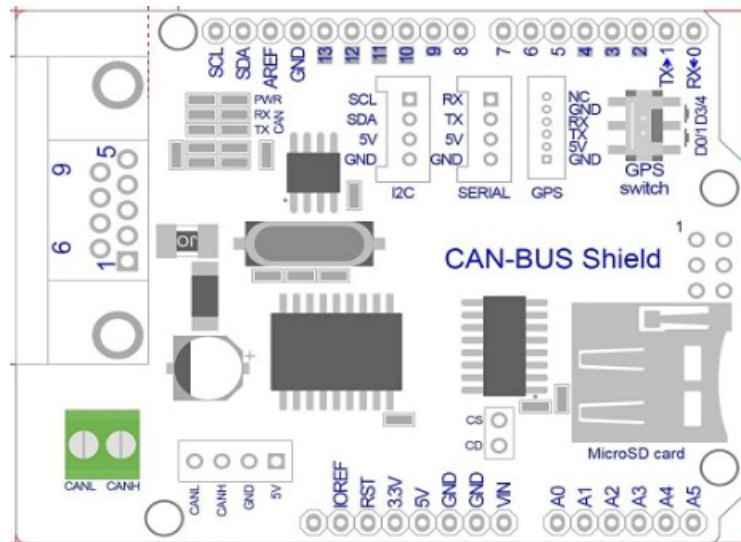


Ilustración 9: Componentes del Escudo CAN-BUS. [WWWamazonProd13]

Características:

- CAN V2.0B con velocidad de hasta 1 Mb/s
- Alta Velocidad de interfaz SPI (10MHz)
- Identificadores estándar (11 bits) y expandido (29 bits)
- Conector de 9 pines
- CANH, CANL
- Lector de tarjetas MicroSD
- Conectores I2C y serial UART
- Conexión GPS

Otra versión: "Placa de expansión de BUS CAN-Bus Shield V2, IIC I2C y UART para Arduino" [WWWaliexpressProd192], "Can-Bus Shield V2" [WWWamazonProd18], "CAN-BUS Shield V2 adopts MCP2515 and MCP2551" [WWWseedstudioProd24]

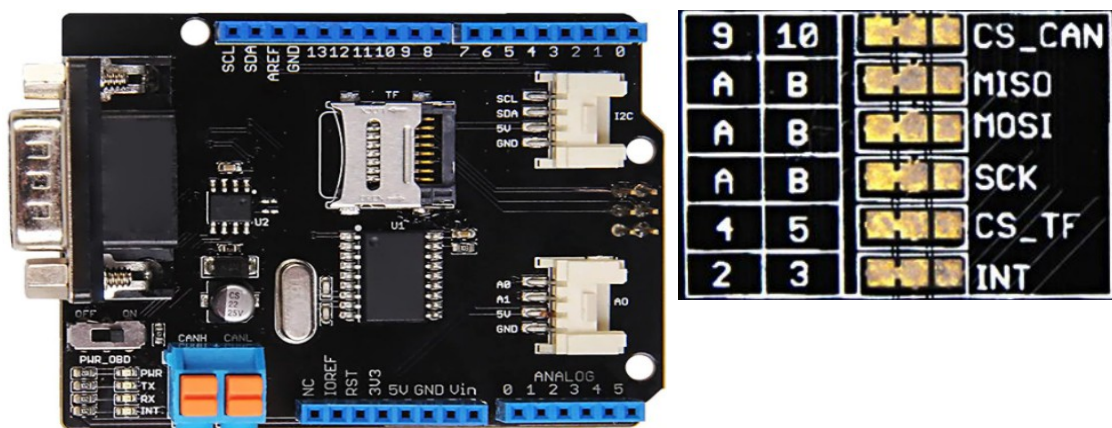


Ilustración 10: Placa de expansión de BUS CAN-Bus Shield V2, IIC I2C y UART para Arduino [WWWaliexpressProd192]

- **MCP2515 - NiRen** - "Módulo de Bus CAN A5, MCP2515, receptor TJA1050, módulo SPI" [WWWaliexpressProd193], "CAN Bus Using Arduino" [WWWarduinoDoc31], "Módulo de Bus CAN MCP2515, receptor TJA1050 SPI para 51 MCU, controlador de brazo D71" [WWWaliexpressProd194], "Tutorial CAN de Arduino – Módulo MCP2515"

CAN BUS de interfaz con Arduino – Comparación de CAN sobre SPI e I2C" [WWWecuarobotDoc0], "autowp/arduino-mcp2515" [WWWgithubDrivers52], "autowp/arduino-can hacker" [WWWgithubDrivers53], "spirilis/mcp2515" [WWWgithubDrivers54], "macchina/mcp2515" [WWWgithubDrivers49], "collin80/can_common" [WWWgithubDrivers50], "Seeed-Studio/CAN_BUS_Shield" [WWWgithubDrivers51], "Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]

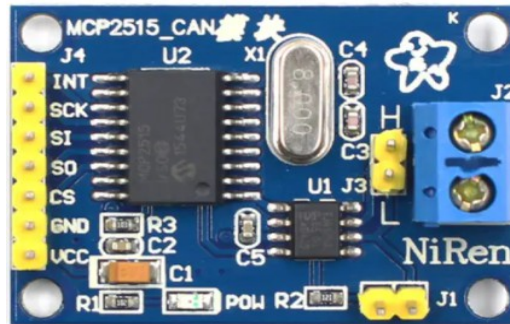


Ilustración 11: MCP2515 (Bus CAN) [WWWarduinodoc31]

- "CANBed - Arduino CAN-BUS Development Kit (ATmega32U4 with MCP2515 and MCP2551)" [WWWseedstudioProd23]

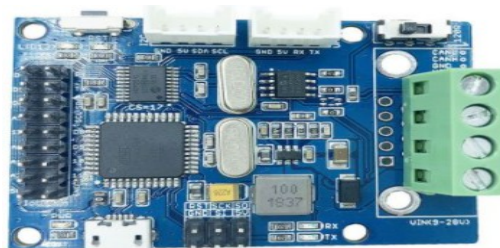


Ilustración 12: CANBed - Arduino CAN-BUS Development Kit (ATmega32U4 with MCP2515 and MCP2551) [WWWseedstudioProd23]

Y en el fichero "RMDx8ArduinoUBU.cpp" se definen las variables a las que se tiene acceso mediante las funciones definidas en "RMDx8ArduinoUBU.h". Un ejemplo del mismo podría ser la consulta de las variables relacionadas con el Encoder:

```
/**
 * The host sends the command to read the current position of the encoder
 * The motor responds to the host after receiving the command, the frame data contains:
 * Encoder current position (uint16_t type, 14bit encoder value range 0~16383), which is the encoder original
 * position minus the encoder zero offset value
 * Encoder original position (encoderRaw) (uint16_t type, 14bit encoder value range 0~16383)
 * EncoderOffset(uint16_t type, 14bit encoder value range 0~16383)
 */

void RMDx8ArduinoUBU::readEncoder() {
    cmd_buf[0] = 0x90;
    cmd_buf[1] = 0x00;
    cmd_buf[2] = 0x00;
    cmd_buf[3] = 0x00;
    cmd_buf[4] = 0x00;
    cmd_buf[5] = 0x00;
    cmd_buf[6] = 0x00;
    cmd_buf[7] = 0x00;

    // Send message
    writeCmd(cmd_buf);
```



```

readBuf(cmd_buf);

encoder = ((uint16_t)reply_buf[3] << 8) + reply_buf[2];
encoderRaw = ((uint16_t)reply_buf[5] << 8) + reply_buf[4];
encoderOffset = ((uint16_t)reply_buf[7] << 8) + reply_buf[6];
}

```

Ahora se puede comprobar que efectivamente se corresponde con lo definido por el fabricante, como puede comprobarse en el fichero "RMD servo motor control protocol (CAN BUS)V1.61.pdf" ("GYEMS MOTOR CONTROL PROTOCOL (CAN BUS)" [WWWdropboxDoc0] ("GYEMS MOTOR CONTROL PROTOCOL (CAN BUS) V1.61" [WWWdownload.gyemsDoc1])).

De esta forma, ahora esta biblioteca permite utilizar todas las funciones del fabricante, lo que, a su vez, permite implementar el código necesario para poder desarrollar la funcionalidad de un robot.

Desde la aparición de este motor se han producido nuevos productos ("MYACTUATOR - SERVICE AND SUPPORT" [WWWmyactuatorDoc0]) y se han desarrollado nuevas versiones de documentación en las que se definen nuevas funcionalidades, que aún no han sido incluidos en la biblioteca, por no haber sido probadas.

En las siguientes imágenes se puede ver el listado de órdenes que el fabricante ha implementado en la controladora del motor y la ejecución del nuevo código.

SN	COMMAND NAME	COMMAND DATA
1.	Read PID data command	0x30
2.	Write PID to RAM command	0x31
3.	Write PID to ROM command	0x32
4.	Read acceleration data command	0x33
5.	Write acceleration data to RAM command	0x34
6.	Read encoder data command	0x90
7.	Write encoder offset command	0x91
8.	Write current position to ROM as motor zero command	0x19
9.	Read multi turns angle command	0x92
10.	Read single circle angle command	0x94
11.	Read motor status 1 and error flag commands	0x9A
12.	Clear motor error flag command	0x9B
13.	Read motor status 2	0x9C
14.	Read motor status 3	0x9D
15.	Motor off command	0x80
16.	Motor stop command	0x81
17.	Motor running command	0x88
18.	Torque closed-loop command	0xA1
19.	Speed closed-loop command	0xA2
20.	Position closed-loop command 1	0xA3
21.	Position closed-loop command 2	0xA4
22.	Position closed-loop command 3	0xA5
23.	Position closed-loop command 4	0xA6

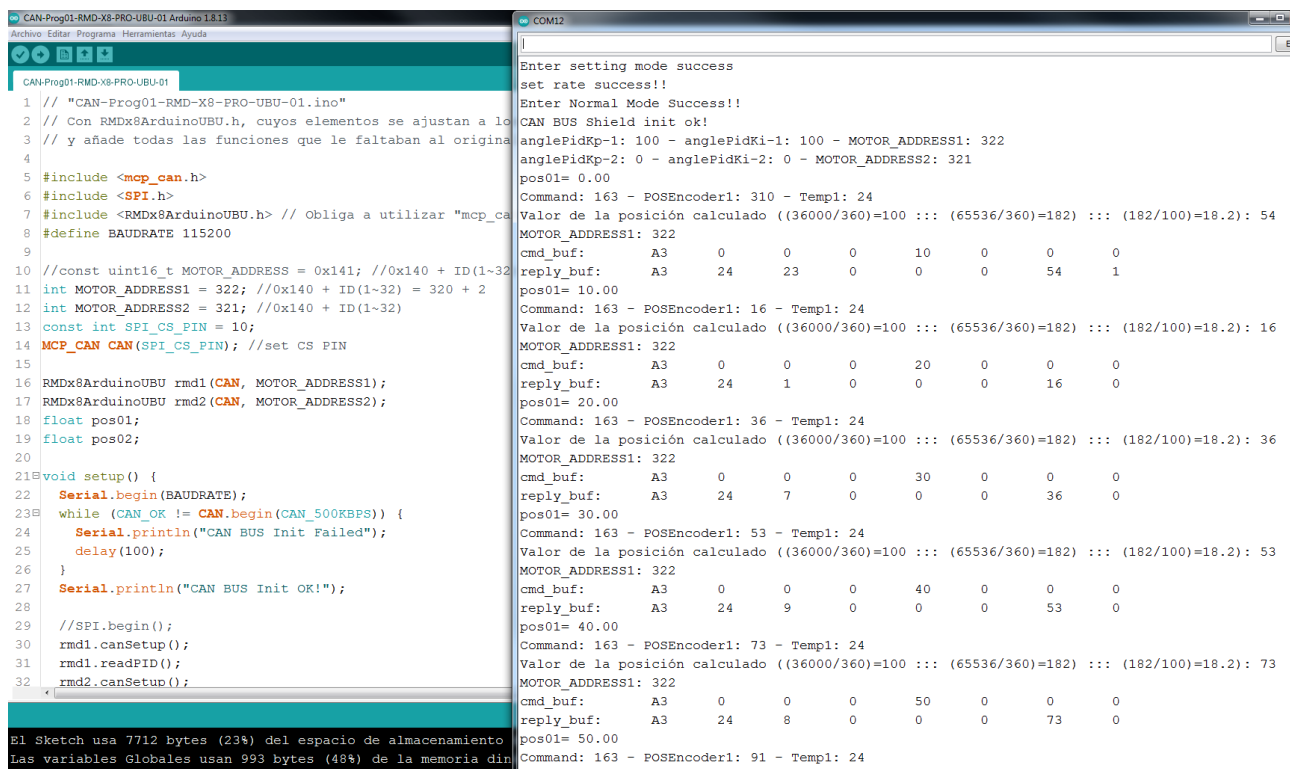
Ilustración 13: Listado de funciones del fabricante (RMD X8 Pro DC de GYEMS [WWWaliexpressProd165])

En cuanto a la funcionalidad se refiere, este motor da la posibilidad de realizar varios tipos de

controles: Posición, velocidad y par. Esto hace que el control del motor se pueda adaptar a las necesidades del robot dependiendo de la funcionalidad del mismo.

Así, se va a transformar el código "**bump5236/RMDx8Arduino**" [WWWgithubDrivers76] (origen de la biblioteca ahora creada) para implementar el código necesario para utilizar el motor RMD-X8-PRO con la biblioteca creada y probar, inicialmente, el acceso al mismo. En la misma prueba también se va a probar que funciona con más de un motor al mismo tiempo. Y de igual forma, se van a comprobar los datos solicitados o cambiados en el motor mostrando el comando enviado con su acuse de recibo, y la respuesta recibida .

Programa 1 (Código de conexión de un Dispositivo Arduino UNO a dos motores "RMD X8 PRO" por mediación de la biblioteca <RMDx8ArduinoUBU.h> - "...Código\Com CAN\CAN-Prog01-RMD-X8-PRO\CAN-Prog01-RMD-X8-PRO-UBU-01\CAN-Prog01-RMD-X8-PRO-UBU-01.ino" – Derivado de "bump5236/RMDx8Arduino" [WWWgithubDrivers76]).



```
1 // "CAN-Prog01-RMD-X8-PRO-UBU-01.ino"
2 // Con RMDx8ArduinoUBU.h, cuyos elementos se ajustan a lo
3 // y añade todas las funciones que le faltaban al origina
4
5 #include <mcp_can.h>
6 #include <SPI.h>
7 #include <RMDx8ArduinoUBU.h> // Obliga a utilizar "mcp_ca
8 #define BAUDRATE 115200
9
10 //const uint16_t MOTOR_ADDRESS = 0x141; //0x140 + ID(1~32
11 int MOTOR_ADDRESS1 = 322; //0x140 + ID(1~32) = 320 + 2
12 int MOTOR_ADDRESS2 = 321; //0x140 + ID(1~32)
13 const int SPI_CS_PIN = 10;
14 MCP_CAN CAN(SPI_CS_PIN); //set CS PIN
15
16 RMDx8ArduinoUBU rmd1(CAN, MOTOR_ADDRESS1);
17 RMDx8ArduinoUBU rmd2(CAN, MOTOR_ADDRESS2);
18 float pos01;
19 float pos02;
20
21 void setup() {
22   Serial.begin(BAUDRATE);
23   while (CAN_OK != CAN.begin(CAN_500KBPS)) {
24     Serial.println("CAN BUS Init Failed");
25     delay(100);
26   }
27   Serial.println("CAN BUS Init OK!");
28
29   //SPI.begin();
30   rmd1.canSetup();
31   rmd1.readPID();
32   rmd2.canSetup();
33 }
34
35 El Sketch usa 7712 bytes (23%) del espacio de almacenamiento
36 Las variables Globales usan 993 bytes (48%) de la memoria din
```

```
Enter setting mode success
set rate success!!
Enter Normal Mode Success!!
CAN BUS Shield init ok!
anglePidKp-1: 100 - anglePidKi-1: 100 - MOTOR_ADDRESS1: 322
anglePidKp-2: 0 - anglePidKi-2: 0 - MOTOR_ADDRESS2: 321
pos01= 0.00
Command: 163 - POSEncoder1: 310 - Temp1: 24
Valor de la posición calculado ((36000/360)=100 ::: (65536/360)=182) ::: (182/100)=18.2): 54
MOTOR_ADDRESS1: 322
cmd_buf:      A3      0      0      0      10      0      0      0
reply_buf:    A3      24      23      0      0      0      54      1
pos01= 10.00
Command: 163 - POSEncoder1: 16 - Temp1: 24
Valor de la posición calculado ((36000/360)=100 ::: (65536/360)=182) ::: (182/100)=18.2): 16
MOTOR_ADDRESS1: 322
cmd_buf:      A3      0      0      0      20      0      0      0
reply_buf:    A3      24      1      0      0      0      16      0
pos01= 20.00
Command: 163 - POSEncoder1: 36 - Temp1: 24
Valor de la posición calculado ((36000/360)=100 ::: (65536/360)=182) ::: (182/100)=18.2): 36
MOTOR_ADDRESS1: 322
cmd_buf:      A3      0      0      0      30      0      0      0
reply_buf:    A3      24      7      0      0      0      36      0
pos01= 30.00
Command: 163 - POSEncoder1: 53 - Temp1: 24
Valor de la posición calculado ((36000/360)=100 ::: (65536/360)=182) ::: (182/100)=18.2): 53
MOTOR_ADDRESS1: 322
cmd_buf:      A3      0      0      0      40      0      0      0
reply_buf:    A3      24      9      0      0      0      53      0
pos01= 40.00
Command: 163 - POSEncoder1: 73 - Temp1: 24
Valor de la posición calculado ((36000/360)=100 ::: (65536/360)=182) ::: (182/100)=18.2): 73
MOTOR_ADDRESS1: 322
cmd_buf:      A3      0      0      0      50      0      0      0
reply_buf:    A3      24      8      0      0      0      73      0
pos01= 50.00
Command: 163 - POSEncoder1: 91 - Temp1: 24
```

Ilustración 14: Prueba de uso de la biblioteca RMDx8ArduinoUBU ("JaimeSaiz/RMDx8ArduinoUBU" [WWWgithubDrivers105]) y de las funciones del fabricante (RMD X8 Pro DC de GYEMS [WWWaliexpressProd165])

Como se puede ver en la imagen, ambas controladoras contestan (identificadores CAN de las controladoras 322 y 321, al comienzo de la imagen de datos, y definidas en el código como MOTOR_ADDRESS), y los datos enviados y devueltos se corresponden con alguna de las órdenes definidas por el propio fabricante (A3: Asignación de posición del motor).

Como ejemplo, en las líneas que comienzan por cmd_buf, y que viene precedida por MOTOR_ADDRESS1 (dirección CAN del motor), el primer campo indica la orden enviada. Se podría haber solicitado la consulta de la posición a través de la orden 90, comando de lectura del Encoder, pero en este caso, no hace falta, ya que, como se ve en el documento del fabricante, se produce un envío de acuse de recibo como se muestra a continuación, como reply_buf.

Una vez probado que funciona, se puede utilizar la biblioteca para el control del motor. Por ejemplo, se puede modificar un grado el giro del motor para hacer la lectura correspondiente. Para ello, se podría modificar el ángulo a través de la aplicación para hacer una lectura posterior de la posición, o se puede hacer la modificación del ángulo desde la herramienta de testeo y hacer la lectura. De cualquier forma, se verá la relación entre el Encoder de la controladora del motor y el giro correspondiente.

En un primer intento, se hará a través de la herramienta para hacer el seguimiento desde la aplicación. Y el giro a realizar será de 1°.

Programa 2 (Código de conexión de un Dispositivo Arduino UNO a un motor "RMD X8 PRO" por mediación de la biblioteca <RMDx8ArduinoUBU.h>" para la prueba de precisión - "...\\Código\\Com CAN\\CAN-Prog01-RMD-X8-PRO\\CAN-Prog01-RMD-X8-PRO-UBU-01\\CAN-Prog01-RMD-X8-PRO-UBU-01.ino" – Derivado de "bump5236/RMDx8Arduino" [WWWgithubDrivers76]).

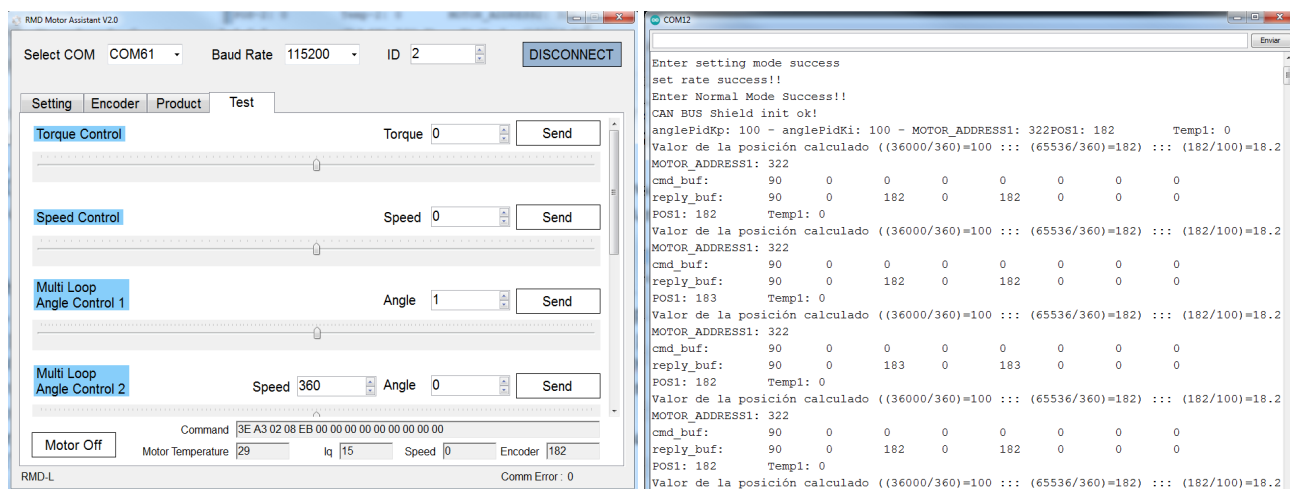


Ilustración 15: Prueba de giro del motor de 1°

Puede comprobarse que la modificación del Encoder es de 0 a 182 con una corrección de la misma muy pequeño de $\pm 1^\circ$.

Se debe a que los 360° suponen un total de 65536 unidades, lo que da una idea de la precisión de la medición, y de la precisión en la ejecución del posicionamiento.

Ahora se puede hacer una prueba de control de velocidad y ver la modificación de los valores de la posición al girar con una velocidad constante para la que se puede utilizar el "Programa2".

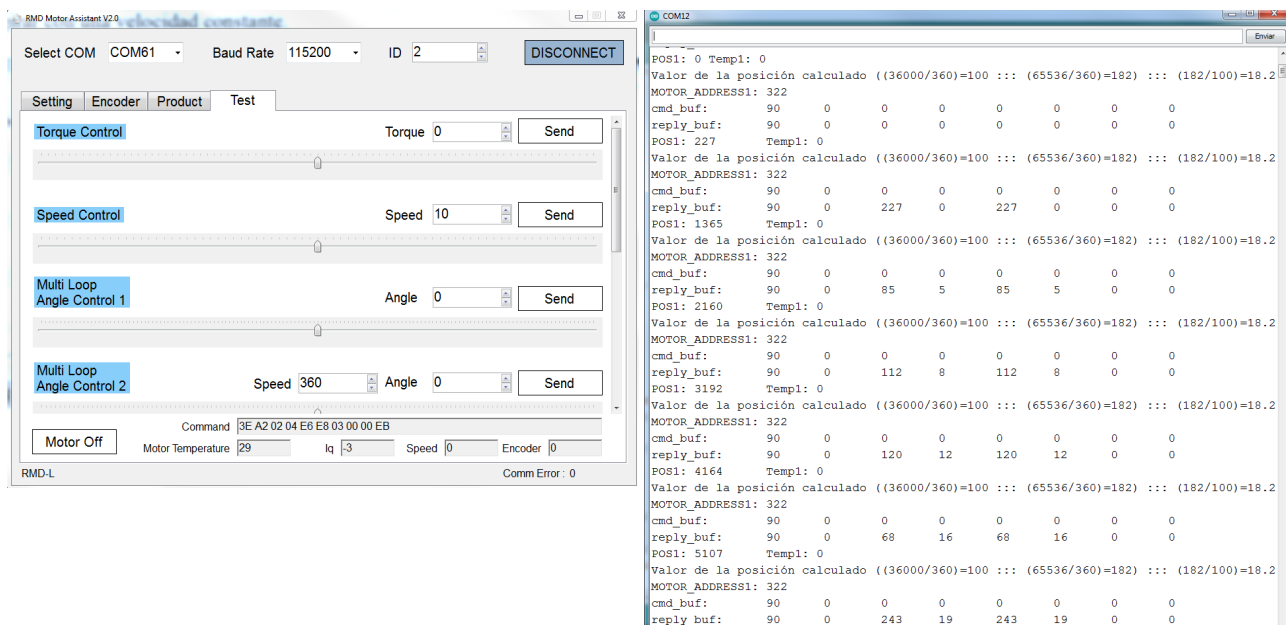


Ilustración 16: Prueba de giro a velocidad constante

Ahora se puede ver el incremento inicial de la posición irregular (en la cuarta columna de las tres primeras tomas de datos) hasta alcanzar la velocidad definida, para continuar con una velocidad estable como se muestra a continuación. Además, se ve que al llegar a la posición correspondiente a una vuelta completa, el Encoder se inicia de nuevo desde 0. Este comportamiento podría dar problemas a la hora de controlar el giro, pero la controladora de este motor permite hacer una lectura de posición multi-vuelta, lo que supone permitir su control aún habiendo dado giros completos.

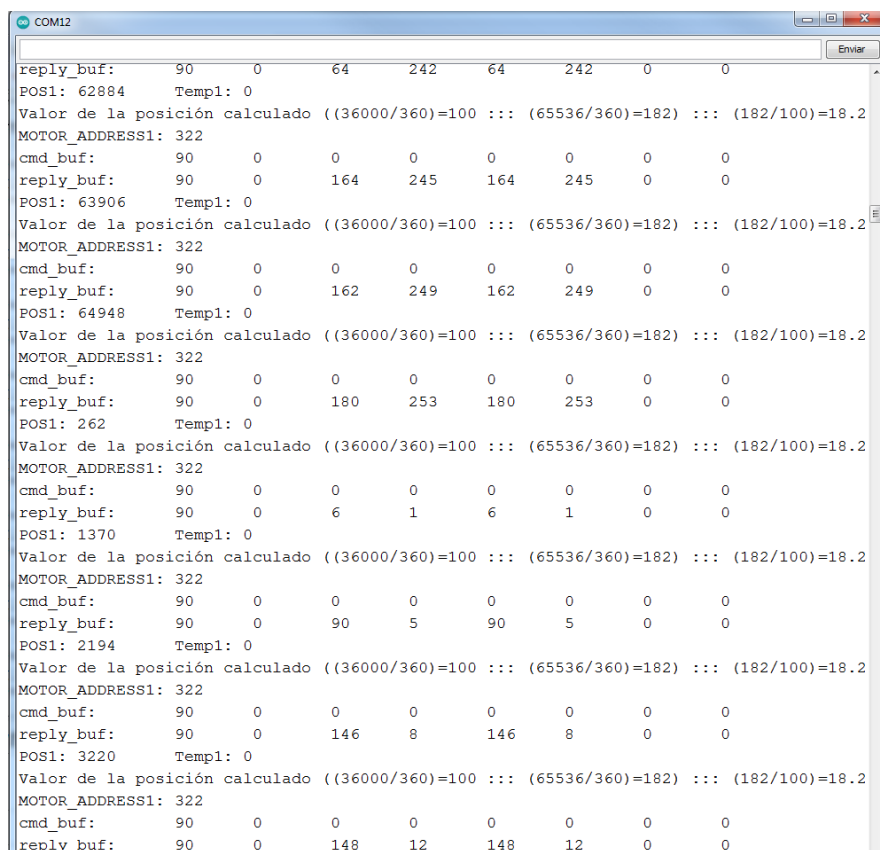


Ilustración 17: Prueba de giro completo con cambio de valor en el Encoder

Resulta interesante ver esa modificación del valor del Encoder para determinar cómo controlar el motor en un movimiento real, una vez montado sobre un robot.

En cuanto a su control de posición desde programa, el propio "Progama1", ya estaba ejecutado para realizar este control cambiando la posición de 10 en 10 unidades de Encoder.

El último de los controles, el par ejercido, no puede ser mostrado ya que se necesita de un vídeo para comprobar la resistencia al movimiento ejercido por el motor al modificar el par del mismo. Sin embargo resultaría útil, por ejemplo, en caso de un salto para amortiguar de forma activa el aterrizaje.

En este sentido, es interesante resaltar los datos incluidos en las especificaciones dado que, a la hora de realizar saltos, se debe considerar el número de vueltas por minuto. Este motor puede llegar a 155rpm, velocidad suficiente como para permitir saltar al robot.

产品参数 PRODUCT PARAMETERS	
产品型号 (Item name)	RMD-X8 PRO
匝数(Turns)	13
额定电压 (Nominal voltage)	V
空载转速 (No-load Speed)	rpm
额定转速 (Nominal Speed)	rpm
额定电流 (Nominal current)	A
额定功率 (Nominal power)	W
额定扭矩 (Nominal torque)	N.M
最大瞬时电流 (Max stall current)	A
最大瞬时扭矩 (Max stall current)	N.M
线电阻 (resistance)	
接线方式 (wire connect)	Y
相间电感 (Phase to phase inductance)	mH
转速常数 (Speed constant)	rpm/v
扭矩电流常数 (Torque constant)	N.M/A
径向负载(Radial load)	N
轴向负载(Axial load)	N
转子惯量 (Rotor inertia)	gc m²
极对数 (Number of pole pairs)	20
电机重量 (Motor weight)	g
工作温度范围 (working temperature)	°C
最高退磁温度 (Max demagnetize temperature)	°C
减速机电速比 (Reducer ratio)	
减速箱背隙 (Backlash)	Arc min
适配驱动类型 (Drive)	DRC20
驱动器输入电压范围 (input voltage)	V
驱动器电流范围 (Current)	A
额定功率 (Normminal power)	W
通信方式 (Communication)	CAN BUS
编码器 (endcoder)	16bit magnetic encoder
波特率 (baudrate)	bps
通信频率 (Communication frequency)	HZ
控制频率 (Control frequency)	
控制模式 (control mode)	
加速度曲线 (S-Curve)	
地址选择开关 (ID switch)	
接插件 (connector)	
温度传感器 (temperature sensor)	

Ilustración 18: Características del motor RMD X8 Pro ("Servomotor de CC sin escobillas RMD X8 Pro, motor de engranaje, controlador foc con codificador de 18 bits, 24V, 48V, 10A" [WWWaliexpressProd188])

Y por último, se puede hacer el control del motor desde la aplicación, por ejemplo, haciendo una modificación continuada de la posición, aumentando el Encoder en 10 unidades en cada lectura para realizar un control fino del giro.

```

COM12
pos01= 0.00
Command: 163 POSEncoder1: 6464 Temp1: 29
Valor de la posición calculado ((36000/360)=100 ::: (65536/360)=182) ::: (182/100)=18.2): 64
MOTOR_ADDRESS1: 322
cmd_buf: A3 0 0 0 10 0 0 0
reply_buf: A3 29 27 0 0 0 64 25
pos_buf: 0 0 0 0 0 0 0 0
pos01= 10.00
Command: 163 POSEncoder1: 21 Temp1: 29
Valor de la posición calculado ((36000/360)=100 ::: (65536/360)=182) ::: (182/100)=18.2): 21
MOTOR_ADDRESS1: 322
cmd_buf: A3 0 0 0 20 0 0 0
reply_buf: A3 29 8 0 0 0 21 0
pos_buf: 0 0 0 0 0 0 0 0
pos01= 20.00
Command: 163 POSEncoder1: 37 Temp1: 29
Valor de la posición calculado ((36000/360)=100 ::: (65536/360)=182) ::: (182/100)=18.2): 37
MOTOR_ADDRESS1: 322
cmd_buf: A3 0 0 0 30 0 0 0
reply_buf: A3 29 8 0 0 0 37 0
pos_buf: 0 0 0 0 0 0 0 0
pos01= 30.00
Command: 163 POSEncoder1: 56 Temp1: 29
Valor de la posición calculado ((36000/360)=100 ::: (65536/360)=182) ::: (182/100)=18.2): 56
MOTOR_ADDRESS1: 322
cmd_buf: A3 0 0 0 40 0 0 0
reply_buf: A3 29 13 0 0 0 56 0
pos_buf: 0 0 0 0 0 0 0 0
pos01= 40.00
Command: 163 POSEncoder1: 74 Temp1: 29
Valor de la posición calculado ((36000/360)=100 ::: (65536/360)=182) ::: (182/100)=18.2): 74
MOTOR_ADDRESS1: 322
cmd_buf: A3 0 0 0 50 0 0 0
reply_buf: A3 29 18 0 0 0 74 0
pos_buf: 0 0 0 0 0 0 0 0
pos01= 50.00

```

Ilustración 19: Prueba de giro del motor con modificación del Encoder en 10 unidades.

Y cuando se observa el giro enviado y el valor del Encoder devuelto por el propio comando, parece haber una cierta diferencia. De hecho, el programa determina aumentar de 10 en 10, y sin embargo, la aplicación devuelve 21, un valor notablemente diferente.

Si se mira la documentación del fabricante, se comenta que los 360° corresponden a 3600 unidades, y sin embargo, cuando se habla del Encoder, los 360° se corresponden con un valor de 65535 unidades. Por tanto, el valor enviado hay que convertirlo a la escala del Encoder para poder ser comparado.

Esa equivalencia se estima como:

$$\begin{aligned} (36000/360) &= 100 \equiv (65535/360) = 182 \\ 182/100 &= 18,2 \end{aligned}$$

Y esa es la relación entre ambas escalas.

Si ahora, se revisan el resto de datos, salvo los primeros datos que aún no hacen una conversión estable, parece que sí se corresponde con esta relación entre los datos introducidos y los mostrados por el Encoder.

Evidentemente, habría que tenerlo en cuenta a la hora de introducir este motor en un robot real. De esa forma los cálculos de Posiciones y Ángulos a través de Cinemática Directa y Cinemática Inversa serían exactos.

1.1.3 Pruebas de funcionalidad - Comunicaciones CAN

1.1.3.1 Comunicar un Arduino UNO a un motor RMD-X8-PRO por CAN

Documentos base:

- **"bump5236/RMDx8Arduino"** [WWWgithubDrivers76]
- **"hdh7485/Control rmd x8 using arduino with can-bus shield"** [WWWgist.githubDrivers1] (**"hdh7485/rmd_x8_control"** [WWWgithubDrivers99])
- **"GYEMS MOTOR CONTROL PROTOCOL (CAN BUS)"** [WWWdropboxDoc0] (**"GYEMS MOTOR CONTROL PROTOCOL (CAN BUS) V1.61"** [WWWdownload.gyemsDrivers0])
- **"RMD-X user manual V1.01-EN.pdf"** [WWWdropboxDoc1]
- **"RMD-X User Manual For Motion Actuator"** [WWWdownload.gyemsDoc0]
- **"RMD-S Servo Motor Manual V1.5"** [WWWmymobilemmsDoc0]
- **"RMD-L Series Servo Actuator User Manual Rev 1.01 (Release).pdf"** [WWWdropboxDoc2]
- Software de configuración de motores "RMD X8 PRO" vía USB:
 - **"RMD V2.0.exe"** [WWWdropboxSoft0]
 - **"CP210x_Windows_Drivers.zip"** [WWWdropboxSoft1]
 - **"RMD config V1.7(EN).exe"** [WWWdropboxSoft2]
- Biblioteca **"autowp/arduino-mcp2515"** [WWWgithubDrivers52] - Versión de **"mcp2515_can.h/cpp"**
- **"Paul112110/CAN_Bus_Shield-master"** [WWWgithubDrivers77] - Versión de **"mcp_can.h/cpp"**.
- **"Seeed-Studio/Seeed_Arduino_CAN"** [WWWgithubDrivers82] - Varias versiones con diferentes resultados:
 - **"Seeed-Studio/Seeed_Arduino_CAN - v1.3.0"** [WWWgithubDrivers83] - Versión de **"mcp_can.h/cpp"** obsoleta.
 - **"Seeed-Studio/Seeed_Arduino_CAN - v2.0.0"** [WWWgithubDrivers84] - Versión de **"mcp2515_can.h/cpp"** – Incluyen dos juegos principales de bibliotecas: **"mcp-can.h/cpp"** y **mcp2515_can.h/cpp** lo que obliga a realizar variaciones en los sketches para su uso al tener que definir la biblioteca **"mcp2515_can.h/cpp"** en los sketches a partir de esta versión y en detrimento de **"mcp_can.h/cpp"**.
 - **"Seeed-Studio/Seeed_Arduino_CAN - v2.0.1"** [WWWgithubDrivers85] - Sólo contiene código para las pruebas tanto para la conexión de dispositivos CAN de diferentes fabricantes como para el acceso a motores "RMD X8 PRO". Pero no contiene una nueva versión de la biblioteca para el manejo de Bus CAN.

Para la comunicación entre dispositivos por Bus CAN se pueden utilizar los dos últimos juegos de bibliotecas, entre otros juegos de bibliotecas.

Como se vio con anterioridad, algunas configuraciones de dispositivos pueden dar problemas dado que, diferentes tipos de dispositivos tienen problemas de interconexión con algún tipo determinado de configuración, dependiendo de qué papel juegue cada dispositivo (emisor/receptor), y de la biblioteca utilizada. Por ejemplo, se encuentran problemas de interconexión, cuando los modelos de dispositivos Bus CAN son distintos, y cuando se ejecutan sketches con llamadas a otras bibliotecas, como **<RMDx8Arduino.h>**, o cuando se utilizan ejemplos de interconexión a motores "RMD X8 PRO".

Se van a ver diferentes pruebas de interconexión entre diferentes dispositivos y con distintas

bibliotecas. E incluso, orientado o no a la conexión de motores "RMD X8 PRO" mediante la biblioteca <RMDx8Arduino.h> en "bump5236/RMDx8Arduino" [WWWgithubDrivers76], o directamente mediante el código del ejemplo "hdh7485/Control rmd x8 using arduino with can-bus shield" [WWWgist.githubDrivers1] ("hdh7485/rmd_x8_control" [WWWgithubDrivers99]).

Por tanto, además de usar los dos sketches Emisor y Receptor ya utilizados con anterioridad (Programa 1: Código transmisor de múltiples datos - "...\\Código\\Com CAN\\CAN-Prog01-Transmisor-Receptor-VariosMensajes\\CAN-Prog01-Transmisor-VariosMensajes\\CAN-Prog01-Transmisor-VariosMensajes .ino" - Programa 2: Código receptor de múltiples datos - "...\\Código\\Com CAN\\CAN-Prog01-Transmisor-Receptor-VariosMensajes\\CAN-Prog01-Receptor-Varios-Mensajes\\CAN-Prog01-Receptor-Varios-Mensajes .ino") se van a utilizar los siguientes dos sketches presentados a continuación.

El primer código es un ejemplo de uso de un motor "RMD X8 PRO" con una biblioteca intermedia que facilita el uso del formato de los mensajes del Bus CAN.

Nombre pin	UTILIZAR
VCC	Pin de entrada de alimentación de 5V
GND	Pin de tierra
CS	Pin SPI SLAVE select (bajo activo)
ENTONCES	SPI maestro entrada esclavo salida conductor
SI	Salida principal SPI cable de entrada esclavo
SCLK	Pin de reloj SPI
EN T	Pin de interrupción MCP2515

Ilustración 20: Pinout de CAN Bus [WWWarduinoDoc31]

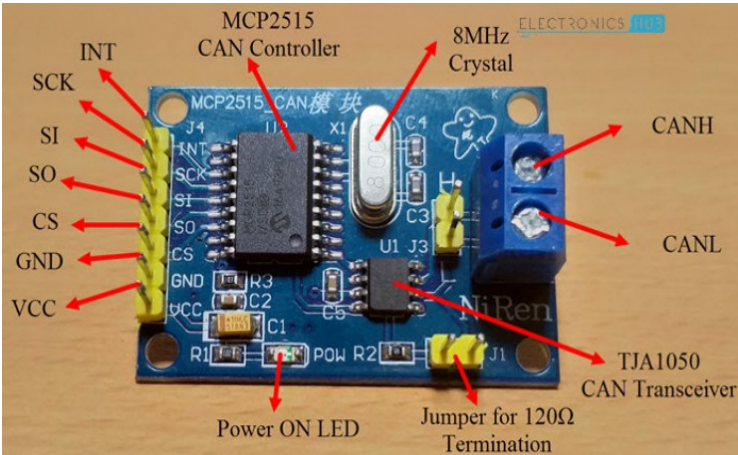
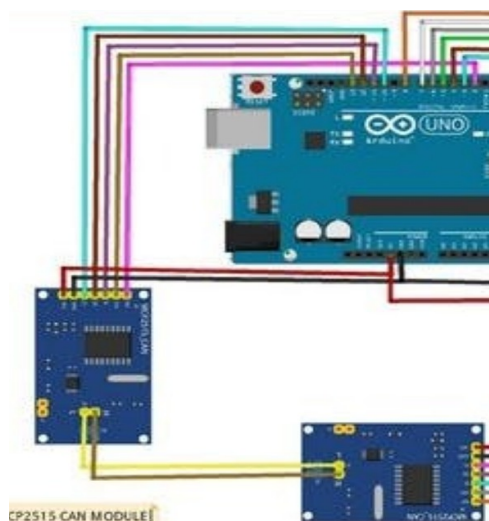


Ilustración 21: MCP2515 (Bus CAN) [WWWelectronicsHubDoc1]



Microchip MCP2515	Arduino
VCC	5V
GND	GND
SCK	SCK
SO	MISO
SI	MOSI
CS	10
INT	2

Ilustración 22: Pinout de CAN Bus [WWWarduinoDoc31]

Programa 3 (Código de conexión de un Dispositivo Arduino UNO a un motor "RMD X8 PRO" con mediación de la biblioteca <RMDx8Arduino.h> - "...\\Código\\Com CAN\\CAN-Prog01-RMD-X8-PRO\\CAN-Prog01-RMD-X8-PRO-02\\CAN-Prog01-RMD-X8-PRO-02.ino" – Derivado de "bump5236/RMDx8Arduino" [WWWgithubDrivers76]):

El siguiente código, en realidad es un ejemplo de uso de un motor "RMD X8 PRO" sin una biblioteca que facilite el uso del formato de los mensajes del Bus CAN.

Programa 4 (Código de conexión de un Dispositivo Arduino UNO a un motor "RMD X8 PRO" - "...\\Código\\Com CAN\\CAN-Prog01-RMD-X8-PRO\\Control_rmd_x8_using_arduino_with_can-bus_shield-3\\Control_rmd_x8_using_arduino_with_can-bus_shield-3.ino" – Derivado de "hdh7485/Control_rmd_x8_using_arduino_with_can-bus_shield" [WWWgist.githubDrivers1] ("hdh7485/rmd_x8_control" [WWWgithubDrivers99])):

Caso de uso de las bibliotecas con <RMDx8Arduino.h> y **Ejemplo:**

- "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) - Versión de "mcp_can.h/cpp"
- "Seeed-Studio/Seeed_Arduino_CAN - v2.0.0" [WWWgithubDrivers84] - Versión de "mcp2515_can.h/cpp"

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
MCP2515 + <RMDx8Arduino.h>	Emisor		
CAN-BUS Shield	Rerector		Se recibe

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
MCP2515 + <RMDx8Arduino.h>	Emisor		
CAN-BUS Shield		Rerector	Se recibe

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
MCP2515 + <RMDx8Arduino.h>		Emisor	Fallo de compilación
CAN-BUS Shield		Rerector	

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
MCP2515 + Ejemplo	Emisor		
CAN-BUS Shield	Rerector		No recibe

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
MCP2515 + Ejemplo	Emisor		
CAN-BUS Shield		Rerector	No recibe

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
MCP2515 + Ejemplo		Emisor	
CAN-BUS Shield		Rerector	No recibe

Tabla 1: Uso de las bibliotecas "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) y "Seeed-Studio/Seeed_Arduino_CAN – v2.0.0" [WWWgithubDrivers84] con "MCP2515" y "CAN-BUS Shield"

Caso de uso entre dispositivos del mismo modelo **CAN-BUS Shield** y con las bibliotecas con **<RMDx8Arduino.h>** y **Ejemplo**:

- "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) - Versión de "mcp_can.h/cpp"
- "Seeed-Studio/Seeed_Arduino_CAN - v2.0.0" [WWWgithubDrivers84] - Versión de "mcp2515_can.h/cpp"

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
CAN-BUS Shield + <RMDx8Arduino.h>	Emisor		
CAN-BUS Shield		Rerecotor	No recibe

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
CAN-BUS Shield + <RMDx8Arduino.h>	Emisor		
CAN-BUS Shield	Rerecotor		No recibe

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
CAN-BUS Shield + <RMDx8Arduino.h>		Emisor	Fallo de compilación
CAN-BUS Shield		Rerecotor	

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
CAN-BUS Shield + Ejemplo	Emisor		
CAN-BUS Shield		Rerecotor	Se recibe

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
CAN-BUS Shield + Ejemplo	Emisor		
CAN-BUS Shield	Rerecotor		Se recibe

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
CAN-BUS Shield + Ejemplo		Emisor	
CAN-BUS Shield		Rerecotor	Se recibe

Tabla 2: Uso de las bibliotecas "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) y "Seeed-Studio/Seeed_Arduino_CAN – v2.0.0" [WWWgithubDrivers84] con "CAN-BUS Shield"

Caso de uso entre dispositivos del mismo modelo **MCP2515** y con las bibliotecas con **<RMDx8Arduino.h>** y **Ejemplo**:

- "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) - Versión de "mcp_can.h/cpp"
- "Seeed-Studio/Seeed_Arduino_CAN - v2.0.0" [WWWgithubDrivers84] - Versión de "mcp2515_can.h/cpp"

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
MCP2515+ <RMDx8Arduino.h>	Emisor		
MCP2515		Rerecotor	No recibe

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
MCP2515 + <RMDx8Arduino.h>		Emisor	Fallo de compilación
MCP2515		Rerecotor	

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
MCP2515+ <RMDx8Arduino.h>	Emisor		
MCP2515	Rerecotor		No recibe

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
MCP2515 + Ejemplo	Emisor		
MCP2515		Rerecotor	Se recibe

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
MCP2515 + Ejemplo		Emisor	
MCP2515		Rerecotor	Se recibe

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
MCP2515 + Ejemplo	Emisor		
MCP2515	Rerecotor		Se recibe

Tabla 3: Uso de las bibliotecas "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) y "Seeed-Studio/Seeed_Arduino_CAN – v2.0.0" [WWWgithubDrivers84] con "MCP2515"

Luego como conclusión, parece razonable pensar que el dispositivo más adecuado para conectar motores "RMD X8 PRO" y dispositivos Arduinos será un dispositivo MCP2515 asociado a cada dispositivo Arduino con el que se accedería a otro dispositivo MCP2515 y de éste a un dispositivo Arduino, o a un motor "RMD X8 PRO". En este caso se podría usar "mcp_can.h/cpp" o "mcp2515_can.h/cpp" en cualquiera de los dispositivos Arduino.

Y en cuanto a la utilización de bibliotecas para acceder a los motores "RMD X8 PRO", parece aconsejable usar funciones dentro del propio sketch y no utilizar la biblioteca <RMDx8Arduino.h> hasta que alguna evolución no resuelva los problemas encontrados.

Y si se quiere usar <RMDx8Arduino.h> se debe usar CAN-BUS Shield pero, tiene la limitación de que no se puede usar "mcp2515_can.h/cpp" en el lado emisor ya que resulta ser el dispositivo donde se usa <RMDx8Arduino.h>, y ésta exige el uso de mcp_can.h/cpp.

Sin embargo, <RMDx8Arduino.h> da la posibilidad de usar comandos más sencillos en lugar de obligar a implementar todos los envíos dentro de funciones en el propio sketch.

Sin embargo, esta biblioteca tiene el problema de estar realizada por un alumno de postgrado ("Takuya Sanada" [WWWrah.web.nitech.acDoc0]) lo que implica la posibilidad de dejar de tener mantenimiento en cualquier momento.

Por otra parte, en la investigación realizada por su autor se incluyen otros proyecto de interés:

- Implementación de caminar y correr "bump5236/biped" [WWWgithubDrivers86]
- Aplicación de aprendizaje automático a los robots que caminan y corren "bump5236/ML_biped" [WWWgithubDrivers87]
- Relación de ángulos definido en cada pata al caminar "bump5236/Sim_biped" [WWWgithubDrivers88]. MATLAB.
- Un ensayo de salto "bump5236/JumpPhaseSimulation" [WWWgithubDrivers89]. MATLAB.

Y en "t-kamimura/arduino_can" [WWWgithubDrivers90] se encuentra una colección interesante de pruebas de su uso.

Una vez definido el identificador de la controladora, si fuera necesario, y probado el motor, se puede comprobar si funciona correctamente desde un dispositivo Arduino a través del sketch que usa <RMDx8Arduino.h> como biblioteca. Como comprobación se podría usar un receptor y, tanto el emisor como el receptor, usarán "mcp-can.h/cpp" ("CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77])), ya que la recepción entre dispositivos parecía funcionar correctamente.

Así, como receptor se usará el **Programa 1 (CAN-Prog01-Receptor-Varios-Mensajes-1.ino)** y como emisor el **Programa 3 (CAN-Prog01-RMD-X8-PRO-02.ino)**, que se corresponde con otro caso anterior como se ve en la siguiente tabla.

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
MCP2515 + <RMDx8Arduino.h>	Emisor		
CAN-BUS Shield	Receptor		Se recibe

Tabla 4: Uso de la biblioteca "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) con dispositivos "MCP2515" como Emisor y "CAN-BUS Shield" como Receptor

Tras realizar el montaje se observa que el motor no llega a recibir los datos. E incluso, al conectar el motor deja de llegar la información al Receptor. Por tanto, sigue habiendo ciertos problemas de compatibilidad entre dispositivos.

Dado que la opción más apropiada falla, ahora se puede probar sólo con un **"CAN-BUS Shield"** como Emisor, con el **Programa 3 (CAN-Prog01-RMD-X8-PRO-02.ino)** y sin Receptor, que se corresponde con otro caso anterior, como se ve en la siguiente tabla pero que, en aquella ocasión no parecía ser la más apropiada para asegurar la compatibilidad entre dispositivos.

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
CAN-BUS Shield + <RMDx8Arduino.h>	Emisor		
CAN-BUS Shield	Rerector		No recibe

Tabla 5: Uso de la biblioteca "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) con dispositivo "CAN-BUS Shield" como Emisor

En este caso, el motor recibe la información correctamente y ejecuta el comando enviado. Y a continuación se puede probar a usar sólo dispositivos **MCP2515** que tampoco parecía ser una opción apropiada.

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
MCP2515 + <RMDx8Arduino.h>	Emisor		
MCP2515	Rerector		No recibe

Tabla 6: Uso de la biblioteca "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) con dispositivo "MCP2515" como Emisor

En este caso parece que tampoco funciona correctamente. Además, de las pruebas anteriores también se sabe que el Emisor no puede utilizar la biblioteca "mcp2515_can.h/cpp" ya que devuelve un error de compilación. Por tanto, parece que sólo funciona correctamente cuando se usa un Emisor "CAN-BUS Shield" y con "mcp-can.h/cpp".

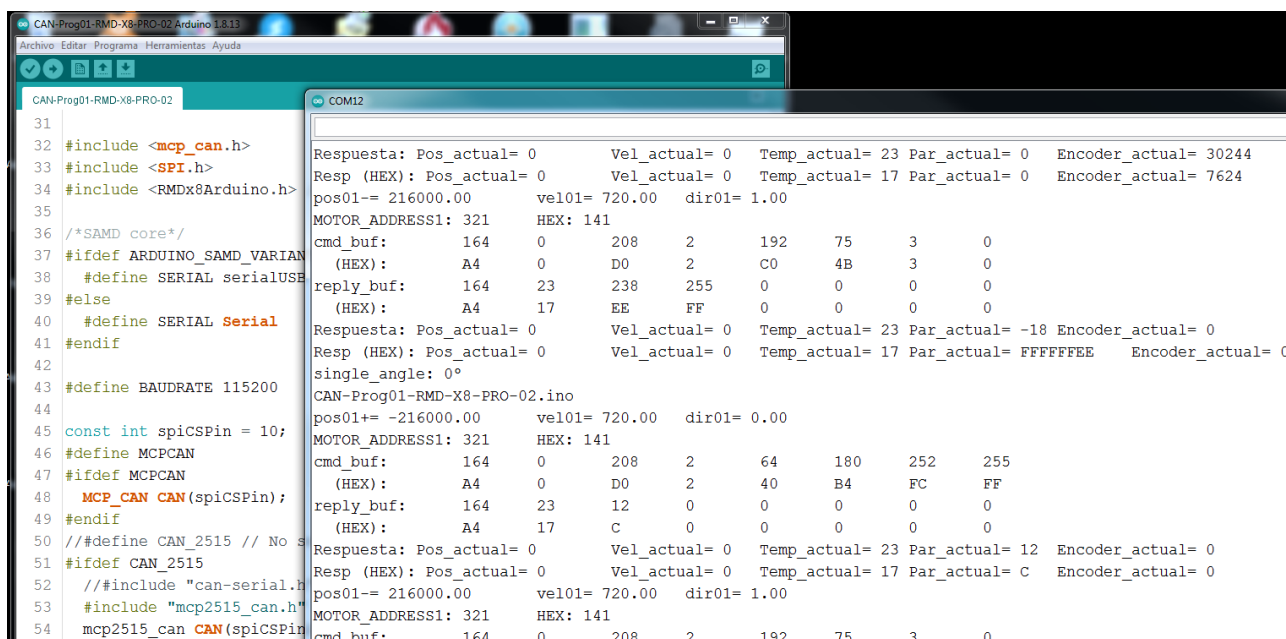


Ilustración 23: Prueba de codificación mediante `<RMDx8Arduino.h>` sobre el motor "RMD X8 PRO" con Emisor "CAN-BUS Shield" y "mcp-can.h/cpp" y sin utilizar Receptor

También sería interesante probar el acceso al motor usando como Emisor cualquiera de los dos tipos de dispositivos pero mediante "mcp2515-can.h/cpp" ("Seeed-Studio/Seeed_Arduino_CAN - v2.0.0" [WWWgithubDrivers84]) y ejecutando un acceso directo al motor mediante el Ejemplo en lugar de usar la biblioteca `<RMDx8Arduino.h>`. Como comprobación se usará un receptor y, tanto

el emisor como el receptor usarán "mcp-can.h/cpp" de "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) ya que la recepción entre dispositivos parecía funcionar correctamente.

Así, como receptor se usará el **Programa 1 (CAN-Prog01-Receptor-Varios-Mensajes-1.ino)** y como emisor el **Programa 4 (Control_rmd_x8_using_arduino_with_can-bus_shield-2.ino)**. Según las tablas iniciales de este mismo apartado, en caso de ser distintos los dispositivos Emisor y Receptor no se recibían mensajes. Por tanto sólo se van a hacer pruebas en las que los dispositivos sean idénticos.

Inicialmente se puede probar sólo con un **"CAN-BUS Shield"** como Emisor y Receptor, que se corresponde con otro caso anterior, como se ve en la siguiente tabla.

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
CAN-BUS Shield + Ejemplo	Emisor		
CAN-BUS Shield		Rereceptor	No recibe

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
CAN-BUS Shield + Ejemplo	Emisor		
CAN-BUS Shield	Rereceptor		No recibe

	mcp-can.h/cpp	mcp2515_can.h/cpp	Resultado
CAN-BUS Shield + Ejemplo		Emisor	
CAN-BUS Shield		Rereceptor	No recibe

Tabla 7: Uso de las bibliotecas "CAN_BUS_Shield-master" ("Paul112110/CAN_Bus_Shield-master" [WWWgithubDrivers77]) y "Seeed-Studio/Seeed_Arduino_CAN – v2.0.0" [WWWgithubDrivers84] con "CAN-BUS Shield" y el Ejemplo de codificación directa de comandos sobre el motor "RMD X8 PRO"

En este caso, el motor recibe la información correctamente y ejecuta el comando enviado. Sin embargo, en este caso el Receptor no recibe nada sea cual sea la biblioteca utilizada.

Por tanto, como resultado de las pruebas se observa que sólo funciona correctamente cuando se usa un Emisor **"CAN-BUS Shield"** y con **"mcp-can.h/cpp"**, y sin utilizar ningún Receptor como testigo de los envíos y, que pueda almacenar las órdenes enviadas al motor y las respuestas de éste.

```

Control_rmd_x8_using_arduino_with_can-bus_shield-3 Arduino 1.8.13
Archivo Editar Programa Herramientas Ayuda

Control_rmd_x8_using_arduino_with_can-bus_shield-3
4 #include <SPI.h>
5 #include <mcp_can.h>
6
7 /*SAMD core*/
8 #ifdef ARDUINO_SAMD_VARIANTE
9 #define SERIAL Serial
10 #else
11 #define SERIAL Serial
12 #endif
13
14 unsigned char len = 0;
15 byte sndStat;
16 const int spiCSpin = 10;
17 #define MCPCAN
18 #ifdef MCPCAN
19 MCP_CAN CAN(spiCSpin);
20 #endif
21 // #define CAN_2515 //
22 #ifdef CAN_2515
23 #include "can-serial.h"
24 #include "mcp2515_can.h"
25 mcp2515_can CAN(spiCSpin);
26 #endif
27
28 void Recibir() {

```

```

COM12
164 0 208 2 64 176 252 255
Control_rmd_x8_using_arduino_with_can-bus_shield-3.ino
-----
Data from ID: 0x141
164 27 240 255 0 0 223 117
Data from Mode: 0x0
164 27 240 255 0 0 223 117
Message Sent Successfully!
164 0 208 2 192 64 0 0
Control_rmd_x8_using_arduino_with_can-bus_shield-3.ino
-----
Data from ID: 0x141
164 27 250 255 0 0 184 248
Data from Mode: 0x0
164 27 250 255 0 0 184 248
Message Sent Successfully!
164 0 208 2 64 176 252 255
Control_rmd_x8_using_arduino_with_can-bus_shield-3.ino
-----
Data from ID: 0x141
164 27 234 255 0 0 223 117

```

Ilustración 24: Prueba de codificación directa de comandos sobre el motor "RMD X8 PRO" con Emisor "CAN-BUS Shield" y "mcp-can.h/cpp" y sin utilizar Receptor

Por otra parte, el Programa 3 (Código de conexión de un Dispositivo Arduino UNO a un motor "RMD X8 PRO" con mediación de la biblioteca <RMDx8Arduino.h> - "...Código\Com CAN\CAN-Prog01-RMD-X8-PRO\CAN-Prog01-RMD-X8-PRO-02\CAN-Prog01-RMD-X8-PRO-02.ino" – Derivado de "bump5236/RMDx8Arduino" [WWWgithubDrivers76]) es una evolución que ha necesitado de la mejora de la biblioteca utilizada.

Esta modificación introduce varias nuevas funciones y la modificación de alguno de los tipos de datos utilizados ya que no se ajustaban a las necesidades de su ejecución.

Las nuevas funciones y las ya existentes serían ("GYEMS MOTOR CONTROL PROTOCOL (CAN BUS)" [WWWdropboxDoc0]):

- Anteriores:
 - canSetup(); // Ejecuta una inicialización con parametrizaciones como "CAN_OK != CAN.begin(CAN_1000KBPS)"
 - readPID(); // Read PID data command
 - writePID(anglePidKp, anglePidKi, speedPidKp, speedPidKi, iqPidKp, iqPidKi); // Write PID to RAM parameter command.
 - writeEncoderOffset(offset); // Write encoder offset command
 - readPosition();
 - writeCurrent(current); // Torque current control command
 - writeVelocity(velocity); // Speed control command

- writePosition3(position); // Position control command. Devuelve la temperatura, voltage, velocidad y posición del encoder. Se ha añadido el número de la función y cambiado el tipo de dato, dependiente del número de bytes.
- writePosition4(position, velocity); // Position control command. También se pasa la velocidad límite. Devuelve la temperatura, el par, la velocidad y la posición actuales. Se ha añadido el número de la función y cambiado el tipo de dato, dependiente del número de bytes.
- writePosition5(position, spin); // Position control command. También se pasa la dirección del movimiento. Devuelve la temperatura, el par, la velocidad y la posición del encoder actuales. Se ha añadido el número de la función y cambiado el tipo de dato, dependiente del número de bytes.
- writePosition6(position, velocity, spin); // Position control command. También se pasa la dirección del movimiento y la velocidad límite. Devuelve la temperatura, el par, la velocidad y la posición del encoder actuales. Se ha añadido el número de la función y cambiado el tipo de dato, dependiente del número de bytes.
- Añadidas:
 - writePIDROM(anglePidKp, anglePidKi, speedPidKp, speedPidKi, iqPidKp, iqPidKi); // PUEDE DAÑAR EL CHIP SI SE REPITE MUCHAS VECES) Write PID to ROM parameter command.
 - readAccel(); // Read acceleration data command
 - writeAccel(accel); // Write acceleration data command
 - readEncoder(); // Read encoder data command
 - writePositionROM(); // PUEDE DAÑAR EL CHIP SI SE REPITE MUCHAS VECES) Write current position to ROM as motor zero position command
 - readSingleCircleAngle(); // Read single circle angle command
 - clearState(); // Turn off motor, while clearing the motor operating status and previously received control commands
 - stopMotor(); // Stop motor, but do not clear the motor operating state and previously received control commands
 - resumeStopMotor(); // Resume motor operation from motor stop command (Recovery control mode before stop motor)
 - readStatus1(); // Reads the motor's error status and voltage, temperature
 - clearErrorFlag1(); // This command clears the error status of the current motor. Da el voltage y el Status. The error flag cannot be cleared when the motor status does not return to normal.
 - readStatus2(); // Reads motor temperature, voltage, speed, encoder position.
 - readStatus3(); // Contains three-phase current data

En las bibliotecas se deben considerar los cambios de las variables al obedecer a la traducción de los diferentes tipos de datos y longitudes. Se utiliza la notación y semántica:

`g = (valor >> 8) & 0xFF;`

Se queda con los 8 bits mas altos: "valor >> 8" desplaza los bits 8 lugares a la derecha. Y a ese resultado le sigue una operación con una mascara de 8 bits (0xFF = 0b11111111)

Para entender toda la operación: valor es de 32 bits (unsigned long valor)

Supongamos	1001 0001 0111 1010 1001 0001 0111 1010 => 32 bits
>> 8	0000 0000 1001 0001 0111 1010 1001 0001 desplaza 8 lugares
& 0xFF	& 0000 0000 0000 0000 1111 1111 1111 1111

0000 0000 0000 0000 0111 1010 1001 0001 resultado

"&FF" SE llama máscara binaria, elimina la parte alta del longitud y se queda con el último

byte.

Un ejemplo:

```
void RGB(unsigned long valor) {  
    r = valor >> 16;  
    g = (valor >> 8) & 0xFF;  
    b = valor & 0xFF;  
    "valor" en binario sería: rrrrrrrrggggggggbbbbbbbbb
```

Por tanto, separa cada uno en una variable. De hecho, se manipulan los 24 menos significativos (rgb) de los 32 bits (unsigned long).

Datos utilizados (rangos) por las diferentes variables según la biblioteca:

- Cur: (-2000, 2000) = (-12,5A, 12,5A)
- Pos: 0.01 degree/LSB, 36000 represents 360° (LSB = least-significant bit - "degree/LSB" = "degree per LSB" = "degrees per least significant bit" (0.01 degree/LSB = 0.01 degree per unity))
- Vel: 1dps/s // 360 = 360°/s ("dps per LSB" = "dps/LSB" = "degrees per second per least significant bit")

Unidades y rangos por tipo de variables:

- Torque (int16): (-2048, 2048 (HEX800)) = (-33A, 33A)
- Encoder Position (uint16 - 14bits): (0, 16383 (HEX3FFF))
- Angle (int32): 0.01grados/LSB = 36000 (HEX8CA0) = 360°
- Angle (uint16): 0.01grados/LSB = 36000 = 360°

Y como resumen:

- 1 vuelta = 6 grupos de 60° = 360° = 216000 unidades del Encoder

1.1.3.2 Biblioteca RMDx8ArduinoUBU y uso de Encoders en la comunicación de Arduinos UNO y motores RMD-X8-PRO por CAN

Un aspecto interesante de los motores "RMD-X8-PRO" es el uso de los Encoders para la toma de datos. De hecho, es una forma perfecta de definir topes o posiciones deseadas de los brazos de un robot atendiendo a los valores que toma el Encoder de la posición de cada uno de los motores que lo componen.

Para ello, una vez definidas las nuevas funciones de la biblioteca "bump5236/RMDx8Arduino" [WWWgithubDrivers76] y modificados varios nombres de parámetros para ajustarlos al fabricante ("GYEMS MOTOR CONTROL PROTOCOL (CAN BUS)" [WWWdropboxDoc0]), se puede forzar la posición deseada de un motor "RMD-X8-PRO" y realizar la lectura del Encoder.

A continuación se muestra la biblioteca creada en GitHub ("**JaimeSaiz/RMDx8ArduinoUBU**" [WWWgithubDrivers105]), derivada de "bump5236/RMDx8Arduino" [WWWgithubDrivers76] y del documento "GYEMS MOTOR CONTROL PROTOCOL (CAN BUS)" [WWWdropboxDoc0], con la que se desarrollará la última parte de las pruebas.

"RMDx8ArduinoUBU.cpp"

"RMDx8ArduinoUBU.h"

Un aspecto interesante en la biblioteca "bump5236/RMDx8Arduino" [WWWgithubDrivers76] ("**RMDx8ArduinoUBU**" ("**JaimeSaiz/RMDx8ArduinoUBU**" [WWWgithubDrivers105])) es la utilización de diferentes tipos de variables. De hecho, dependiendo de la versión del controlador, se usan diferentes tipos de variables en alguna de las funciones definidas. Por ejemplo, en la función

multivuelta "readMotorAngle()" del fichero "RMDx8ArduinoUBU.cpp" y que en este caso utiliza una variable de tipo "uint32" cuando, en realidad y por la definición de la propia marca RMD debería usar un tipo "uint64" con una notable diferencia de valores ("UInt8, UInt16, UInt32, UInt64, Int8, Int16, Int32, Int64" [WWWclickhouseDoc0]):

UInt32 - [0 : 4294967295]

UInt64 - [0 : 18446744073709551615]

Además, en el propio texto de la biblioteca ya se incluyen una serie de acciones a realizar sobre la misma para resolver algunas carencias o debilidades:

HAY QUE REVISAR 0X92

DEBE SER MULTI-VUELTA Y POR TANTO CON 64 BITS.

SI SE CAMBIA DE TIPO DE DATOS, HAY QUE MIRAR ESA RESTA DERIVADA DEL TIPO DE DATO

PARA QUE SEA CORRECTA (VER <https://clickhouse.tech/docs/es/sql-reference/data-types/int-uint/>).

SU USO DEPENDE DEL ENCODER (DE 12bits A 18bits)

// -----

HAY QUE DEFINIR LA FUNCIÓN "MULTI MOTORS CONTROL COMMAND" CON ID 0X280

// -----

HAY QUE REVISAR LOS TIPOS DE DATOS int16_t, uint16_t, uint32_t, int32_t.

// -----

Para el primer ejemplo las funciones a utilizar podrían ser:

- writeEncoderOffset(offset); // The host sends the command to set encoder offset
- readEncoder(); // The host sends the command to read the current position of the encoder
- readStatus3(); // Devuelve la temperatura, voltage, velocidad y posición del encoder.

La primera función (writeEncoderOffset()), ya existente, permite definir el punto de partida del motor desde donde comenzará a tomarse medidas de posición diferenciales.

Cualquiera de las otras dos funciones (readEncoder(), readStatus3()), de nueva creación, permitirán tomar el dato desde el Encoder una vez colocado el motor en la posición deseada.

Programa 1 (Código de conexión de un Dispositivo Arduino UNO a un motor "RMD X8 PRO" con mediación de la biblioteca <RMDx8ArduinoUBU.h> ("RMDx8ArduinoUBU" ("JaimeSaiz/RMDx8ArduinoUBU" [WWWgithubDrivers105])) - "...Código\Com CAN\CAN-Prog01-RMD-X8-PRO\CAN-Prog01-RMD-X8-PRO-03\CAN-Prog01-RMD-X8-PRO-03.ino" – Derivado de "bump5236/RMDx8Arduino" [WWWgithubDrivers76] - Tras mejorar la biblioteca para ajustarla al fabricante del motor "RMD X8 PRO").

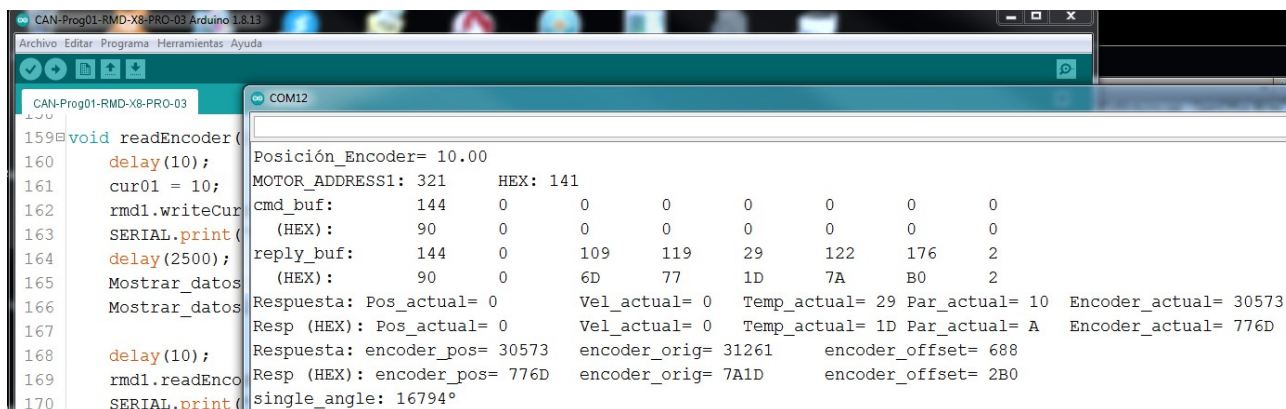


Ilustración 25: Prueba de lectura del Encoder sobre el motor "RMD X8 PRO" con Emisor "CAN-BUS Shield" y "mcp-can.h/cpp" y sin utilizar Receptor

El Encoder devuelve valores entre 0 y 65000. Y la relación será:

$$\text{encoder_orig} (65324) = \text{encoder_pos} (64636) + \text{encoder_offset} (688)$$

Sin embargo, se ven ciertos problemas con la equivalencia de diferentes valores obtenidos y definidos en el código.

Para comprobarlo se pueden tomar diferentes lecturas desde el sketch "...Código\Com CAN\CAN-Prog01-RMD-X8-PRO\CAN-Prog01-RMD-X8-PRO-04\CAN-Prog01-RMD-X8-PRO-04.ino" en las que se ve el problema:

```
void readMAXEncoder() { // Current se refiere al par (torque) // Define un par
  pos01 = 0;
  vel01 = 720;
  for(int i = 0; i<1200; i++) {
    SERIAL.println("-----");
    pos01 = pos01 + 100; //
    delay(10);
    rmd1.writePosition4(pos01, vel01);
    delay(10);
    SERIAL.print("pos01+= "); SERIAL.print(pos01); SERIAL.print("\t");
    SERIAL.print("vel01= "); SERIAL.print(vel01); SERIAL.print("\t");
    SERIAL.print("dir01= "); SERIAL.print(dir01); SERIAL.println("\t");
    rmd1.readSingleCircleAngle(); // Single-turn - Cero al pasar por cero
    delay(10);
    SERIAL.print("circleAngle= "); SERIAL.println(rmd1.circleAngle);
    delay(10);
    rmd1.readEncoder();
    delay(200);
    Mostrar_datos1();
    Mostrar_datos2();
    Mostrar_datos3();
  }
}
```

Obteniendo como datos:

```
pos01= 35800.00 vel01= 720.00   dir01= 0.00
circleAngle= 35802
MOTOR_ADDRESS1: 321          HEX: 141
cmd_buf:      144    0    0    0    0    0    0    0
(HEX):        90    0    0    0    0    0    0    0
reply_buf:    144    0    150   254   70    1   176    2
(HEX):        90    0    96    FE    46    1    B0    2
Respuesta: Pos_actual= 0 Vel_actual= 0   Temp_actual= 25 Par_actual= 20   Encoder_actual= 65174
Resp (HEX): Pos_actual= 0   Vel_actual= 0   Temp_actual= 19 Par_actual= 14   Encoder_actual= FE96
Respuesta: encoder= 65174   encoderRaw= 326   encoderOffset= 688
Resp (HEX): encoder= FE96   encoderRaw= 146   encoderOffset= 2B0
```

```
pos01= 35900.00 vel01= 720.00   dir01= 0.00
```

```

circleAngle= 35906
MOTOR_ADDRESS1: 321      HEX: 141
cmd_buf:      144    0    0    0    0    0    0    0
(HEX):      90    0    0    0    0    0    0    0
reply_buf:    144    0    82    255    2    2    176    2
(HEX):      90    0    52    FF    2    2    B0    2
Respuesta: Pos_actual= 0 Vel_actual= 0 Temp_actual= 25 Par_actual= 14 Encoder_actual= 65362
Resp (HEX): Pos_actual= 0 Vel_actual= 0 Temp_actual= 19 Par_actual= E Encoder_actual= FF52
Respuesta: encoder= 65362 encoderRaw= 514 encoderOffset= 688
Resp (HEX): encoder= FF52 encoderRaw= 202 encoderOffset= 2B0

```

```

pos01= 36000.00 vel01= 720.00 dir01= 0.00
circleAngle= 7
MOTOR_ADDRESS1: 321      HEX: 141
cmd_buf:      144    0    0    0    0    0    0    0
(HEX):      90    0    0    0    0    0    0    0
reply_buf:    144    0    9    0    185    2    176    2
(HEX):      90    0    9    0    B9    2    B0    2
Respuesta: Pos_actual= 0 Vel_actual= 0 Temp_actual= 25 Par_actual= 13 Encoder_actual= 9
Resp (HEX): Pos_actual= 0 Vel_actual= 0 Temp_actual= 19 Par_actual= D Encoder_actual= 9
Respuesta: encoder= 9 encoderRaw= 697 encoderOffset= 688
Resp (HEX): encoder= 9 encoderRaw= 2B9 encoderOffset= 2B0

```

```

pos01= 36100.00 vel01= 720.00 dir01= 0.00
circleAngle= 103
MOTOR_ADDRESS1: 321      HEX: 141
cmd_buf:      144    0    0    0    0    0    0    0
(HEX):      90    0    0    0    0    0    0    0
reply_buf:    144    0    194    0    114    3    176    2
(HEX):      90    0    C2    0    72    3    B0    2
Respuesta: Pos_actual= 0 Vel_actual= 0 Temp_actual= 25 Par_actual= 6 Encoder_actual= 194
Resp (HEX): Pos_actual= 0 Vel_actual= 0 Temp_actual= 19 Par_actual= 6 Encoder_actual= C2
Respuesta: encoder= 194 encoderRaw= 882 encoderOffset= 688
Resp (HEX): encoder= C2 encoderRaw= 372 encoderOffset= 2B0

```

Y otros dos valores intermedios:

```

pos01= 13600.00 vel01= 720.00 dir01= 0.00
circleAngle= 13610
MOTOR_ADDRESS1: 321      HEX: 141
cmd_buf:      144    0    0    0    0    0    0    0
(HEX):      90    0    0    0    0    0    0    0
reply_buf:    144    0    196    96    116    99    176    2
(HEX):      90    0    C4    60    74    63    B0    2
Respuesta: Pos_actual= 0 Vel_actual= 0 Temp_actual= 25 Par_actual= 8 Encoder_actual= 24772
Resp (HEX): Pos_actual= 0 Vel_actual= 0 Temp_actual= 19 Par_actual= 8 Encoder_actual= 60C4
Respuesta: encoder= 24772 encoderRaw= 25460 encoderOffset= 688
Resp (HEX): encoder= 60C4 encoderRaw= 6374 encoderOffset= 2B0

```

```

pos01= 25700.00 vel01= 720.00 dir01= 0.00
circleAngle= 25698
MOTOR_ADDRESS1: 321      HEX: 141
cmd_buf:      144    0    0    0    0    0    0    0
(HEX):      90    0    0    0    0    0    0    0
reply_buf:    144    0    190    182    110    185    176    2
(HEX):      90    0    BE    B6    6E    B9    B0    2
Respuesta: Pos_actual= 0 Vel_actual= 0 Temp_actual= 25 Par_actual= -20 Encoder_actual= 46782
Resp (HEX): Pos_actual= 0 Vel_actual= 0 Temp_actual= 19 Par_actual= FFFFFFFEC Encoder_actual= B6BE
Respuesta: encoder= 46782 encoderRaw= 47470 encoderOffset= 688
Resp (HEX): encoder= B6BE encoderRaw= B96E encoderOffset= 2B0

```

Desde estos datos se extraen los siguientes resultados:

pos01= 35800.00
circleAngle= 35802
Respuesta: encoder= 65174 encoderRaw= 326 encoderOffset= 688

pos01= 35900.00
circleAngle= 35906
Respuesta: encoder= 65362 encoderRaw= 514 encoderOffset= 688

pos01= 36000.00
circleAngle= 7
Respuesta: encoder= 9 encoderRaw= 697 encoderOffset= 688

pos01= 36100.00
circleAngle= 103
Respuesta: encoder= 194 encoderRaw= 882 encoderOffset= 688

...

...

pos01= 13600.00
circleAngle= 13610
Respuesta: encoder= 24772 encoderRaw= 25460 encoderOffset= 688

pos01= 35900.00
circleAngle= 35906
Respuesta: encoder= 65362 encoderRaw= 47470 encoderOffset= 688

Por tanto:

circleAngle= 7	pos01= 36000.00	encoder= 9
circleAngle= 13610	pos01= 13600.00	encoder= 24772
circleAngle= 25698	pos01= 25700.00	encoder= 46782
circleAngle= 35906	pos01= 35900.00	encoder= 65362

Y mientras "pos01" sigue creciendo al seguir girando el motor a razón de 36000 unidades por vuelta, "circleAngle" pasaría a 0, para volver a comenzar.

Y, por otra parte, cuando "circleAngle" comienza de nuevo, "encoder" pasaría a 0, para volver a comenzar.

Sin embargo, los rangos son diferentes:

circleAngle = (0, 36000) == encoder = (0, 65535)

Y si se busca una linealidad mediante una regla de tres:

circleAngle1/circleAngle2 = circleAngle2/circleAngle3

encoder1/ encoder2 = encoder2/ encoder3

Los resultados obtenidos serán para "circleAngle"/"encoder" en su rango:

$65535/36000 = 1,820416666$

Y en el resto de datos obtenidos:

$24772/13610 = 1,820132$

$46782/ 25698 = 1,82045$

Como el error sobre la medición puede ser, según especificaciones, de 5 unidades, el cálculo puede ser considerado correcto, más aún si se toman sólo dos decimales.

Por tanto, se puede determinar la posición actual mediante el valor del Encoder, mediante la variable "encoder", o también a través de la variable "circleAngle".

Y de igual forma se podría tomar a través de la posición del motor enviada desde el sketch. Pero en este caso, se puede ordenar el movimiento necesario mediante comando pero podría no llegar a

ejecutarse por la razón que fuese, por lo que la posición real no tiene por qué ser la deseada. De la misma forma, esas dos variables serían adecuadas si se hace un control del par y se forzara la posición para tomar las lecturas de las posiciones alcanzadas para determinar los límites o la posición adecuada para algún tipo de movimiento que interesara.

En este segundo caso, se puede usar una función como la siguiente:

```
void readPosEncoder() { // Para tomar valores con posicionamiento manual del
    // motor
    cur01 = 1;
    rmd1.writeCurrent(cur01);
    delay(100);
    for(int i = 0; i<1200; i++) {
        SERIAL.println("-----");
        delay(10);
        rmd1.readSingleCircleAngle(); // Single-turn - Cero al pasar por cero
        delay(100);
        rmd1.readEncoder();
        delay(1000);
        SERIAL.print("circleAngle= "); SERIAL.println(rmd1.circleAngle);
        Mostrar_datos3();
    }
}
```

Y al ejecutar el sketch se posicionaría el motor en la posición deseada y se tomarían los datos de cualquiera de las dos variables.

Ahora se encuentra un nuevo problema y es que, cada 60° se pasa por una posición cero y vuelve a comenzar el conteo. Por tanto las variable "encoder" y "circleAngle" vuelven a comenzar desde cero cada 60°.

Además, los valores obtenidos no se mueven en el rango (0, 16282) como se define en las especificaciones del motor.

Para resolver este problema se debe realizar un cierto cálculo que permita utilizar ciertos parámetros del motor como fuente de datos. Este cálculo se implementa en la siguiente función (incluida en "...\\Código\\Com CAN\\CAN-Prog01-RMD-X8-PRO\\CAN-Prog01-RMD-X8-PRO-04\\CAN-Prog01-RMD-X8-PRO-04.ino").

```

77 void readPosEncoder() { // Para tomar valores con posicionamiento manual del motor
78   pos01= 126000.00   vel01= 720.00   dir01= 0.00
79   vel01= 720.00
80   dir01= 0.00
81   cur01= 0.00
82   rmd1= 0.00
83   delay(1000)
84   // rmd1= 0.00
85   delay(1000)
86   for(i=0; i<6; i++)
87     SERI
88     // p
89     // p
90     pos01= 126000.00   vel01= 720.00   dir01= 0.00
91     pos01= 126000.00   vel01= 720.00   dir01= 0.00
92     delay(1000)
93     // Si no
94     // Sin d
95     delay(1000)
96     rmd1= 0.00
97     delay(1000)
98     rmd1= 0.00
99     delay(1000)
100    pos01= 132000.00   vel01= 720.00   dir01= 0.00
101    rmd1= 0.00
102    delay(1000)

```

COM12

```

Giros totales (decimal)= 1.97   Resto de giro (decimal)= 0.97
todoMotorAngle (unidades totales de giros)= 426299.00
restoMotorAngle (unidades dentro de un giro)= 210299.00 restoMotorAngle - Corregido (unidades dentro de un giro)= 210302.00
Grupos totales de 60° (dentro de un giro - decimal)= 5.84   Resto de grupo de 60° (decimal)= 0.84
circleAngle (grados dentro de un grupo de 60°): 50.45°
Ángulo real (dentro de un giro)= 350.45°
Respuesta: encoder= 55114   encoderRaw= 55802   encoderOffset= 688
Resp (HEX): encoder= D74A   encoderRaw= D9FA   encoderOffset= 2B0
-----
pos01= 126000.00   vel01= 720.00   dir01= 0.00
Giros totales (decimal)= 2.00   Resto de giro (decimal)= 0.00
todoMotorAngle (unidades totales de giros)= 432664.00
restoMotorAngle (unidades dentro de un giro)= 664.00   restoMotorAngle - Corregido (unidades dentro de un giro)= 667.00
Grupos totales de 60° (dentro de un giro - decimal)= 0.02   Resto de grupo de 60° (decimal)= 0.02
circleAngle (grados dentro de un grupo de 60°): 0.95°
Ángulo real (dentro de un giro)= 0.95°
Respuesta: encoder= 1143   encoderRaw= 1831   encoderOffset= 688
Resp (HEX): encoder= 477   encoderRaw= 727   encoderOffset= 2B0
-----
pos01= 132000.00   vel01= 720.00   dir01= 0.00
Giros totales (decimal)= 2.02   Resto de giro (decimal)= 0.02
todoMotorAngle (unidades totales de giros)= 435285.00
restoMotorAngle (unidades dentro de un giro)= 3285.00   restoMotorAngle - Corregido (unidades dentro de un giro)= 3288.00
Grupos totales de 60° (dentro de un giro - decimal)= 0.09   Resto de grupo de 60° (decimal)= 0.09
circleAngle (grados dentro de un grupo de 60°): 5.45°
Ángulo real (dentro de un giro)= 5.45°
Respuesta: encoder= 5966   encoderRaw= 6654   encoderOffset= 688
Resp (HEX): encoder= 174E   encoderRaw= 19FE   encoderOffset= 2B0
-----

```

Guardado.

Ilustración 26: Prueba de cálculo de variables para la toma de datos desde el motor "RMD X8 PRO"

Así, puede comprobarse que la posición varía 216.000 unidades para definir los 360° de cada vuelta. Y a su vez, se ve que las 216.000 unidades son equivalentes a 6 grupos de 60° y, por tanto, también a 6 grupos de otras variables como "encoder" o "circleAngle".

Una vez terminadas las pruebas, las últimas modificaciones pueden ser encontradas en el código "...Código\Com CAN\CAN-Prog01-RMD-X8-PRO\CAN-Prog01-RMD-X8-PRO-04\CAN-Prog01-RMD-X8-PRO-04.ino"

Y una última prueba interesante puede ser la generación de movimientos una vez calculados los ángulos, como realmente se hace a la hora de caminar.

En este caso se utilizan las funciones incorporadas en "...Código\Com CAN\CAN-Prog01-RMD-X8-PRO\CAN-Prog01-RMD-X8-PRO-Datos-02\CAN-Prog01-RMD-X8-PRO-Datos-02.ino".

```

173 void writePosAngDado() { // Para definir posicionamiento del motor dado un ángulo calculado.
174     float angini= 0.00    angfin= 0.00    angReal= 50.31
175     float pos01= 36000.00 vel01= 720.00  dir01= 0.00
176     float pos02= 36000.00 vel02= 720.00  dir02= 0.00
177     pos01= 36000.00    pos02= 36000.00
178     vel01= 720.00      vel02= 720.00
179     dir01= 0.00        dir02= 0.00
180     cur01= 0.16        cur02= 0.16
181     rmd1= 34637.00      rmd2= 34637.00
182     restoMotorAngle (unidades dentro de un giro)= 34637.00  restoMotorAngle - Corregido (unidades dentro de un giro)= 34640.00
183     while (true) {
184         Grupos totales de 60° (dentro de un giro - decimal)= 0.96      Resto de grupo de 60° (decimal)= 0.96
185         circleAngle (grados dentro de un grupo de 60°): 57.49°
186         Ángulo real (dentro de un giro)= 57.49°
187         Respuesta: encoder= 62942      encoderRaw= 63630      encoderOffset= 688
188         Resp (HEX): encoder= F5DE      encoderRaw= F88E      encoderOffset= 2B0
189         rmd1= 34637.00      rmd2= 34637.00
190         pos01= 36000.00    pos02= 36000.00    vel01= 720.00    vel02= 720.00    dir01= 0.00    dir02= 0.00
191         cur01= 0.17        cur02= 0.17
192         rmd1= 34637.00      rmd2= 34637.00
193         restoMotorAngle (unidades dentro de un giro)= 34637.00  restoMotorAngle - Corregido (unidades dentro de un giro)= 34640.00
194         Grupos totales de 60° (dentro de un giro - decimal)= 1.00      Resto de grupo de 60° (decimal)= 1.00
195         circleAngle (grados dentro de un grupo de 60°): 59.98°
196         Ángulo real (dentro de un giro)= 59.98°
197         Respuesta: encoder= 65516      encoderRaw= 668      encoderOffset= 688
198         Resp (HEX): encoder= FFEC      encoderRaw= 29C      encoderOffset= 2B0
199     }
200     angini= 0.00    angfin= 0.00    angReal= 59.98
201     pos01= 36000.00    pos02= 36000.00    vel01= 720.00    vel02= 720.00    dir01= 0.00    dir02= 0.00
202     cur01= 0.17        cur02= 0.17
203     rmd1= 34637.00      rmd2= 34637.00
204     restoMotorAngle (unidades dentro de un giro)= 34637.00  restoMotorAngle - Corregido (unidades dentro de un giro)= 34640.00
205     Grupos totales de 60° (dentro de un giro - decimal)= 1.00      Resto de grupo de 60° (decimal)= 1.00
206     circleAngle (grados dentro de un grupo de 60°): 59.98°
207     Ángulo real (dentro de un giro)= 59.98°
208     Respuesta: encoder= 65516      encoderRaw= 668      encoderOffset= 688
209     Resp (HEX): encoder= FFEC      encoderRaw= 29C      encoderOffset= 2B0
210 }

```

Ilustración 27: Prueba de visualización de variables para para un ángulo dado del motor "RMD X8 PRO"

Y como último código a utilizar, y que incluya las funciones ya mencionadas, puede utilizarse "...Código\Com CAN\CAN-Prog01-RMD-X8-PRO\CAN-Prog01-RMD-X8-PRO-05\CAN-Prog01-RMD-X8-PRO-05.ino". En este caso se ha modificado el código para incorporar las modificaciones y mejoras aportadas durante las pruebas además de hacerlo sobre dos motores.

Programa 2 (Código de conexión con varias funciones de un Dispositivo Arduino UNO a un motor "RMD X8 PRO" con mediación de la biblioteca <RMDx8ArduinoUBU.h> ("RMDx8ArduinoUBU" ("JaimeSaiz/RMDx8ArduinoUBU" [WWWgithubDrivers105])) - "...Código\Com CAN\CAN-Prog01-RMD-X8-PRO\CAN-Prog01-RMD-X8-PRO-05\CAN-Prog01-RMD-X8-PRO-05.ino" – Derivado de "bump5236/RMDx8Arduino" [WWWgithubDrivers76] - Tras mejorar la biblitoeca para ajustarla al fabricante del motor "RMD X8 PRO").

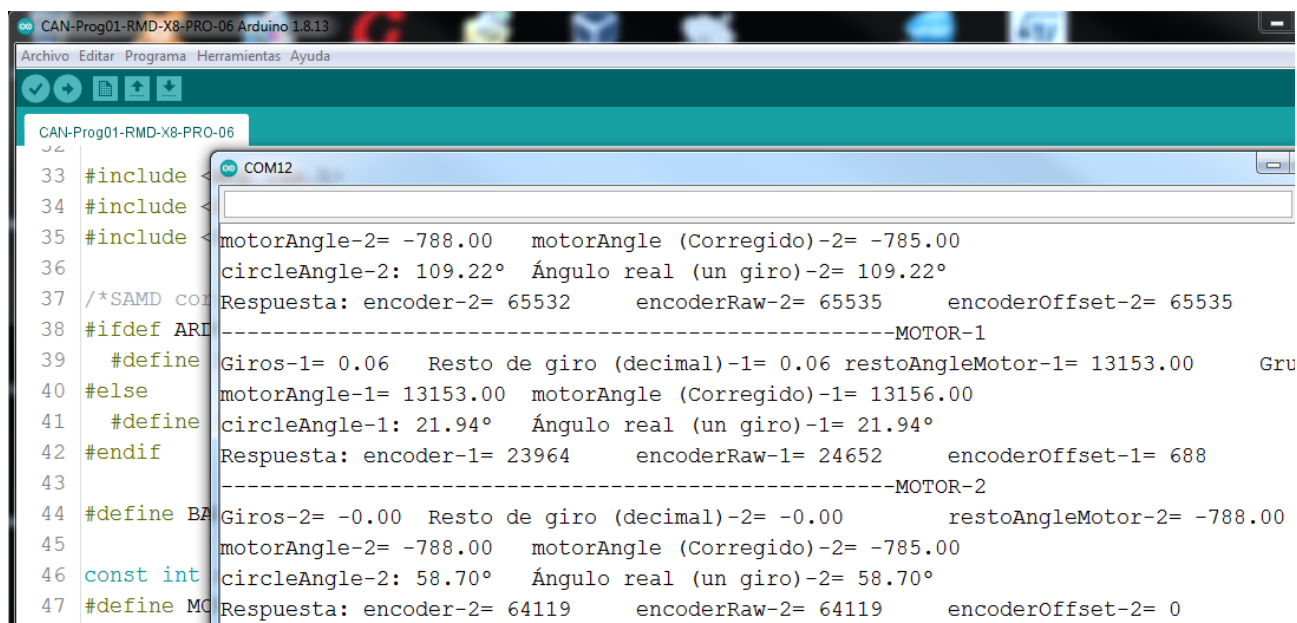
1.1.3.3 Error de la controladora y ejemplo de uso de Encoders en la comunicación de Arduinos UNO y motores RMD-X8-PRO por CAN

Otro problema interesante es la toma de datos desde un prototipo que permita determinar los ángulos límite de un motor o el ángulo más acertado para una posición determinada, cuya solución puede ser utilizada en muchas situaciones.

Para ello, se puede utilizar un código como el siguiente, definiendo las generalizaciones necesarias y modificando, por tanto, el código hasta adecuarse a la situación en particular a resolver.

Programa 1 (Código de conexión de un Dispositivo Arduino UNO a dos motores "RMD X8 PRO" con mediación de la biblioteca <RMDx8ArduinoUBU.h> ("RMDx8ArduinoUBU" ("JaimeSaiz/RMDx8ArduinoUBU" [WWWgithubDrivers105])) para la toma de datos desde

los dos Encoders- "...\\Código\\Com CAN\\CAN-Prog01-RMD-X8-PRO\\CAN-Prog01-RMD-X8-PRO-06\\CAN-Prog01-RMD-X8-PRO-06.ino" – Derivado de "bump5236/RMDx8Arduino" [WWWgithubDrivers76] - Tras mejorar la biblioteca para ajustarla al fabricante del motor "RMD X8 PRO").



```

33 #include <
34 #include <
35 #include <
36
37 /*SAMD co
38 #ifdef AR
39 #define
40 #else
41 #define
42 #endif
43
44 #define BA
45
46 const int
47 #define MC

```

```

COM12
motorAngle-2= -788.00  motorAngle (Corregido)-2= -785.00
circleAngle-2: 109.22°  Ángulo real (un giro)-2= 109.22°
Respuesta: encoder-2= 65532      encoderRaw-2= 65535      encoderOffset-2= 65535
-----MOTOR-1
Giros-1= 0.06  Resto de giro (decimal)-1= 0.06  restoAngleMotor-1= 13153.00  Gru
motorAngle-1= 13153.00  motorAngle (Corregido)-1= 13156.00
circleAngle-1: 21.94°  Ángulo real (un giro)-1= 21.94°
Respuesta: encoder-1= 23964      encoderRaw-1= 24652      encoderOffset-1= 688
-----MOTOR-2
Giros-2= -0.00  Resto de giro (decimal)-2= -0.00      restoAngleMotor-2= -788.00
motorAngle-2= -788.00  motorAngle (Corregido)-2= -785.00
circleAngle-2: 58.70°  Ángulo real (un giro)-2= 58.70°
Respuesta: encoder-2= 64119      encoderRaw-2= 64119      encoderOffset-2= 0

```

Ilustración 28: Prueba del ejemplo del uso de Encoders en la comunicación de Arduinos UNO y motores RMD-X8-PRO por CAN para determinar sus datos una vez colocado en su posición

Sin embargo, cuando se ejecuta este código y de forma no esperada, se produce un error de forma esporádica y sin ningún tipo de movimiento en el motor, entre los valores del Encoder devueltos por el motor a través de la función "readEncoder()" de la biblioteca "RMDx8ArduinoUBU" ("JaimeSaiz/RMDx8ArduinoUBU" [WWWgithubDrivers105]) y el valor del ángulo en una vuelta, también devuelto desde el motor por la función "readSingleCircleAngle()" de la misma biblioteca.

Debido a los cálculos, y dado que en las divisiones se toman sólo dos decimales, se podrá llegar a tener un error que podría llegar a ser de 0,1 grados en las mediciones correspondientes a los ángulos en una vuelta. Sin embargo, como se ve en la imagen, el error de los valores devueltos por el motor entre ambas funciones, puede ser de muchos grados.

Además, este error se produce en ambos motores de prueba luego, se determina que es un error generalizado de la controladora.


```

74 delay(300)
75 for(int i
76 SERIAL.
77 rmd1.re
78 delay(1)
79 rmd1.re
80 delay(1)
81 rmd1.re
82 delay(1)
83 float a
84 //
85 //
86 //
87 float g
88 #includ
89 double
90 double
91 SERIAL.
92 float r
93 if(ang
94 resto
95 }else{
96 resto
97 }
98 SERIAL.
99 SERIAL.

```

```

-----MOTOR-1
Giros-1= 67.95 Resto de giro (decimal)-1= 0.95 restoAngleMotor-1= 204884.00
#####Ángulo corregido equivalente a rmd1.circleAngle-1= 41.47
Grupos de 60°-1= 5.69
motorAngle-1= 14676884.00 motorAngle (Corregido)-1= 204887.00
***** ERROR1111111111111111
circleAngle-1: 0.00° Ángulo real (un giro)-1= 300.00°
Respuesta: encoder-1= 45306 encoderRaw-1= 45994 encoderOffset-1= 688
-----MOTOR-2
Giros-2= 1.62 Resto de giro (decimal)-2= 0.62 restoAngleMotor-2= 134384.00
#####Ángulo corregido equivalente a rmd1.circleAngle-1= -76.03
Grupos de 60°-2= 3.73
motorAngle-2= 350384.00 motorAngle (Corregido)-2= 134387.00
circleAngle-2: 43.98° Ángulo real (un giro)-2= 223.98°
Respuesta: encoder-2= 48037 encoderRaw-2= 48037 encoderOffset-2= 0
-----MOTOR-1
Giros-1= 67.95 Resto de giro (decimal)-1= 0.95 restoAngleMotor-1= 204884.00
#####Ángulo corregido equivalente a rmd1.circleAngle-1= 41.47
Grupos de 60°-1= 5.69
motorAngle-1= 14676884.00 motorAngle (Corregido)-1= 204887.00
***** ERROR1111111111111111
circleAngle-1: 0.00° Ángulo real (un giro)-1= 300.00°
Respuesta: encoder-1= 45306 encoderRaw-1= 45994 encoderOffset-1= 688
-----MOTOR-2

```

Ilustración 29: Error entre los valores del Encoder devueltos por el motor a través de la función "readEncoder()" y el valor del ángulo en una vuelta, también devuelto desde el motor por la función "readSingleCircleAngle()" de la biblioeca <RMDx8ArduinoUBU.h> - MOTOR 1

```

74 delay(300)
75 for(int i=0; i<2; i++)
76 {
77   SERIAL.println("-----MOTOR-1");
78   delay(1000);
79   rmd1.readEncoder();
80   delay(1000);
81   rmd1.readSingleCircleAngle();
82   delay(1000);
83   float a1 = rmd1.circleAngle-1;
84   //
85   //
86   //
87   float g1 = rmd1.Giros-1;
88   #include <RMDx8ArduinoUBU.h>
89   double e1 = rmd1.encoderRaw-1;
90   double r1 = rmd1.encoderOffset-1;
91   SERIAL.println("Respuesta: encoder-1= 45306 encoderRaw-1= 45994 encoderOffset-1= 688");
92   float r1 = rmd1.circleAngle-1;
93   if (ang1 < 0)

```

```

COM12
Respuesta: encoder-2= 48037 encoderRaw-2= 48037 encoderOffset-2= 0
-----MOTOR-1
Giros-1= -6626.12 Resto de giro (decimal)-1= -0.12 restoAngleMotor-1= -1431242240.00
#####Ángulo corregido equivalente a rmd1.circleAngle-1= -43.73
Grupos de 60°-1= -39756.73
motorAngle-1= -1431242240.00 motorAngle (Corregido)-1= -1431242240.00
circleAngle-1: 41.48° Ángulo real (un giro)-1= -2385318.50°
Respuesta: encoder-1= 45306 encoderRaw-1= 45994 encoderOffset-1= 688
-----MOTOR-2
Giros-2= 1.62 Resto de giro (decimal)-2= 0.62 restoAngleMotor-2= 134384.00
#####Ángulo corregido equivalente a rmd1.circleAngle-1= 2385584.00
Grupos de 60°-2= 3.73
motorAngle-2= 350384.00 motorAngle (Corregido)-2= 134387.00
***** ERROR2222222222222222
circleAngle-2: 43.98° Ángulo real (un giro)-2= 223.98°
Respuesta: encoder-2= 48037 encoderRaw-2= 48037 encoderOffset-2= 0
-----MOTOR-1
Giros-1= 67.95 Resto de giro (decimal)-1= 0.95 restoAngleMotor-1= 204884.00

```

Ilustración 30: Error entre los valores del Encoder devueltos por el motor a través de la función "readEncoder()" y el valor del ángulo en una vuelta, también devuelto desde el motor por la función "readSingleCircleAngle()" de la biblioteca <RMDx8ArduinoUBU.h> - MOTOR 2

El código de comprobación se puede ver a continuación.

Programa 2 (Localización de error en código de conexión de un Dispositivo Arduino UNO a dos motores "RMD X8 PRO" con mediación de la biblioteca <RMDx8ArduinoUBU.h> ("RMDx8ArduinoUBU" ("JaimeSaiz/RMDx8ArduinoUBU" [WWWgithubDrivers105])) para la toma de datos desde los dos Encoders- "...Código\Com CAN\CAN-Prog01-RMD-X8-PRO\CAN-Prog01-RMD-X8-PRO-07\CAN-Prog01-RMD-X8-PRO-07.ino" – Derivado de "bump5236/RMDx8Arduino" [WWWgithubDrivers76] - Tras mejorar la biblioteca para ajustarla al fabricante del motor "RMD X8 PRO").

Y para resolverlo, se debería tomar el dato obtenido a través del Encoder ya que, aunque exige la ejecución de cierto código, el dato obtenido no vendrá afectado por el error de la controladora localizado.

Sin embargo, también se ha observado otro tipo de error sin causa aparente. En este caso los datos del Encoder tienen un error de proporciones aún mayores como puede verse en la siguiente imagen.

```
136 // Se divide por 6 para que divida la vuelta en 6 partes, el número de veces
137
138
139 circleAngle-2: -14.78° Ángulo real (un giro)-2= -14.78°
140 Respuesta: encoder-2= 49398 encoderRaw-2= 49398 encoderOffset-2= 0
141 -----MOTOR-1
142 Giros-1= -3193.32 Resto de giro (decimal)-1= -0.32 restoAngleMotor-1= -1379444736.00
143 #####Ángulo corregido equivalente a rmd1.circleAngle-1= -4598094.50
144 Grupos de 60°-1= -38317.91
145 motorAngle-1= -689756672.00 motorAngle (Corregido)-1= -1379444736.00
146 ***** ERROR11111111111111111111
147 circleAngle-1: -6.76° Ángulo real (un giro)-1= -2299026.75°
148 Respuesta: encoder-1= 58150 encoderRaw-1= 58838 encoderOffset-1= 688
149 -----MOTOR-2
150 Giros-2= -0.04 Resto de giro (decimal)-2= -0.04 restoAngleMotor-2= -8867.00
151 #####Ángulo corregido equivalente a rmd1.circleAngle-1= 2299005.25
152 Grupos de 60°-2= -0.25
153 motorAngle-2= -8867.00 motorAngle (Corregido)-2= -8864.00
154 ***** ERROR22222222222222222222
155 circleAngle-2: -14.78° Ángulo real (un giro)-2= -14.78°
156 Respuesta: encoder-2= 49398 encoderRaw-2= 49398 encoderOffset-2= 0
157 -----MOTOR-1
158 Giros-1= -0.00 Resto de giro (decimal)-1= -0.00 restoAngleMotor-1= -1379444736.00
159 ***** ERROR11111111111111111111
```

Ilustración 31: Nuevo error entre los valores del Encoder devueltos por el motor a través de la función "readEncoder()" y el valor del ángulo en una vuelta, también devuelto desde el motor por la función "readSingleCircleAngle()" de la biblioteca <RMDx8ArduinoUBU.h>

Después de diversas pruebas variando las condiciones de la prueba como los tiempos entre las ejecuciones de los comandos para dar tiempo a que se produzcan las respuestas, se comprueba que estos errores persisten.

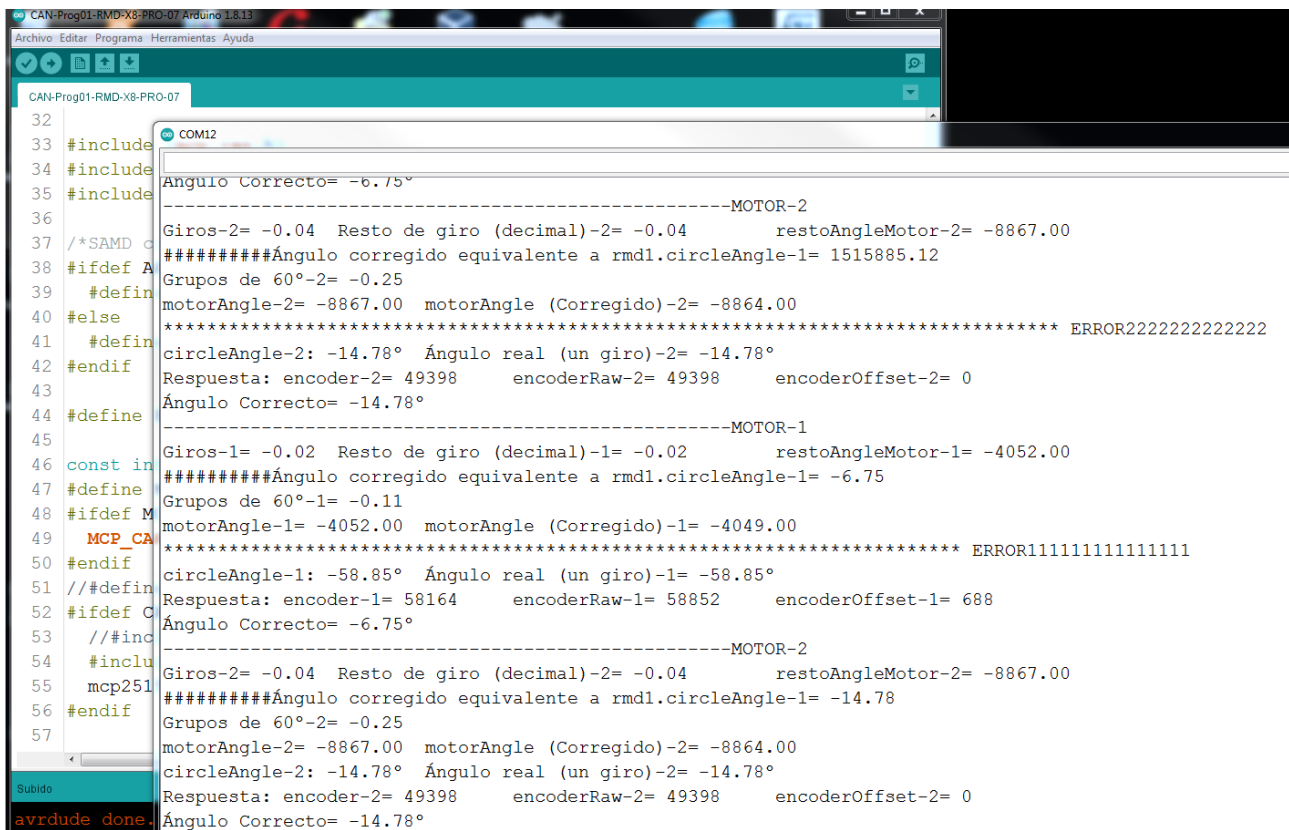
Sin embargo, los dos errores se producen en un rango de unidades muy preciso. El primero con una diferencia siempre menor a 200 unidades, siendo el correcto el tomado desde la función "readSingleCircleAngle()" y, el segundo siempre muy superior a esa cantidad y siendo el correcto el tomado desde la función "readEncoder()".

De hecho no se ha encontrado casuística que indique otro comportamiento.

Por tanto, se puede aplicar la corrección por código. Después se intentará reducir el tiempo entre lecturas y comprobar en un rango de tiempo más largo que no se produzca el citado error o cualquier otra variación.

El código de comprobación se puede ver a continuación.

Programa 2 (Localización de error en código de conexión de un Dispositivo Arduino UNO a dos motores "RMD X8 PRO" con mediación de la biblioteca <RMDx8ArduinoUBU.h> ("RMDx8ArduinoUBU" ("JaimeSaiz/RMDx8ArduinoUBU" [WWWgithubDrivers105])) para la toma de datos desde los dos Encoders- "...Código\Com CAN\CAN-Prog01-RMD-X8-PRO\CAN-Prog01-RMD-X8-PRO-08\CAN-Prog01-RMD-X8-PRO-08.ino" – Derivado de "bump5236/RMDx8Arduino" [WWWgithubDrivers76] - Tras mejorar la biblioteka para ajustarla al fabricante del motor "RMD X8 PRO").



```
32
33 #include
34 #include
35 #include
36
37 /*SAMD c
38 #ifdef A
39 #define
40 #else
41 #define
42 #endif
43
44 #define
45
46 const in
47 #define
48 #ifdef M
49 MCP_CA
50 #endif
51 // #defin
52 #ifdef C
53 // #inc
54 #inclu
55 mcp251
56 #endif
57
Subido
avrdude done. Angulo Correcto= -6.75°
-----MOTOR-2
Giros-2= -0.04 Resto de giro (decimal)-2= -0.04 restoAngleMotor-2= -8867.00
#####Ángulo corregido equivalente a rmdl.circleAngle-1= 1515885.12
Grupos de 60°-2= -0.25
motorAngle-2= -8867.00 motorAngle (Corregido)-2= -8864.00
***** ERROR22222222222222222222
circleAngle-2: -14.78° Ángulo real (un giro)-2= -14.78°
Respuesta: encoder-2= 49398 encoderRaw-2= 49398 encoderOffset-2= 0
Ángulo Correcto= -14.78°
-----MOTOR-1
Giros-1= -0.02 Resto de giro (decimal)-1= -0.02 restoAngleMotor-1= -4052.00
#####Ángulo corregido equivalente a rmdl.circleAngle-1= -6.75
Grupos de 60°-1= -0.11
motorAngle-1= -4052.00 motorAngle (Corregido)-1= -4049.00
***** ERROR11111111111111111111
circleAngle-1: -58.85° Ángulo real (un giro)-1= -58.85°
Respuesta: encoder-1= 58164 encoderRaw-1= 58852 encoderOffset-1= 688
Ángulo Correcto= -6.75°
-----MOTOR-2
Giros-2= -0.04 Resto de giro (decimal)-2= -0.04 restoAngleMotor-2= -8867.00
#####Ángulo corregido equivalente a rmdl.circleAngle-1= -14.78
Grupos de 60°-2= -0.25
motorAngle-2= -8867.00 motorAngle (Corregido)-2= -8864.00
circleAngle-2: -14.78° Ángulo real (un giro)-2= -14.78°
Respuesta: encoder-2= 49398 encoderRaw-2= 49398 encoderOffset-2= 0
Ángulo Correcto= -14.78°
```

Ilustración 32: Prueba de solución de errores entre los valores del Encoder devueltos por el motor a través de la función "readEncoder()" y el valor del ángulo en una vuelta, también devuelto desde el motor por la función "readSingleCircleAngle()" de la biblioteca <RMDx8ArduinoUBU.h>

En esta situación, la biblioteca "RMDx8ArduinoUBU" ("RMDx8ArduinoUBU" ("JaimeSaiz/RMDx8ArduinoUBU" [WWWgithubDrivers105])) y el código con las correcciones realizadas, parece funcionar correctamente. Una vez resuelto y después de reproducir las pruebas con éxito, se suben todos los ficheros e información necesaria a GitHub ("JaimeSaiz/RMDx8ArduinoUBU" [WWWgithubDrivers105])).

1.1.4 Bípedo a escala humana

Sin embargo, no todos los datos son positivos.

El par máximo es de 35Nm con 30A y a 48V. Eso supone un consumo de energía muy alto, lo que obliga a utilizar una fuente con gran capacidad.

Si se piensa que esto debe multiplicarse por el número de motores, y aunque no se estén usando todos los motores a su máximo par, en el peor de los ángulos, la cantidad de energía para un bípedo en un esfuerzo máximo de levantarse y usando para ello motores en tobillos, rodillas y caderas, y por tanto, un mínimo de 6 motores, podría ser de alrededor de 180A, que a 48V, darían un mínimo de 8640W. Una enorme cantidad de energía.

Además, hay un problema añadido, el número de motores para el desarrollo de un bípedo no se corresponde con esa idea inicial. Para ello, se puede analizar la arquitectura de un humanoide con la idea clásica de un motor en cada tobillo, rodilla y conexión con la cadera para caminar hacia adelante, y dos motores más para el movimiento lateral del tobillo y para el movimiento lateral de la cadera.

Como se puede ver en la siguiente imagen, esta arquitectura puede representar un problema.

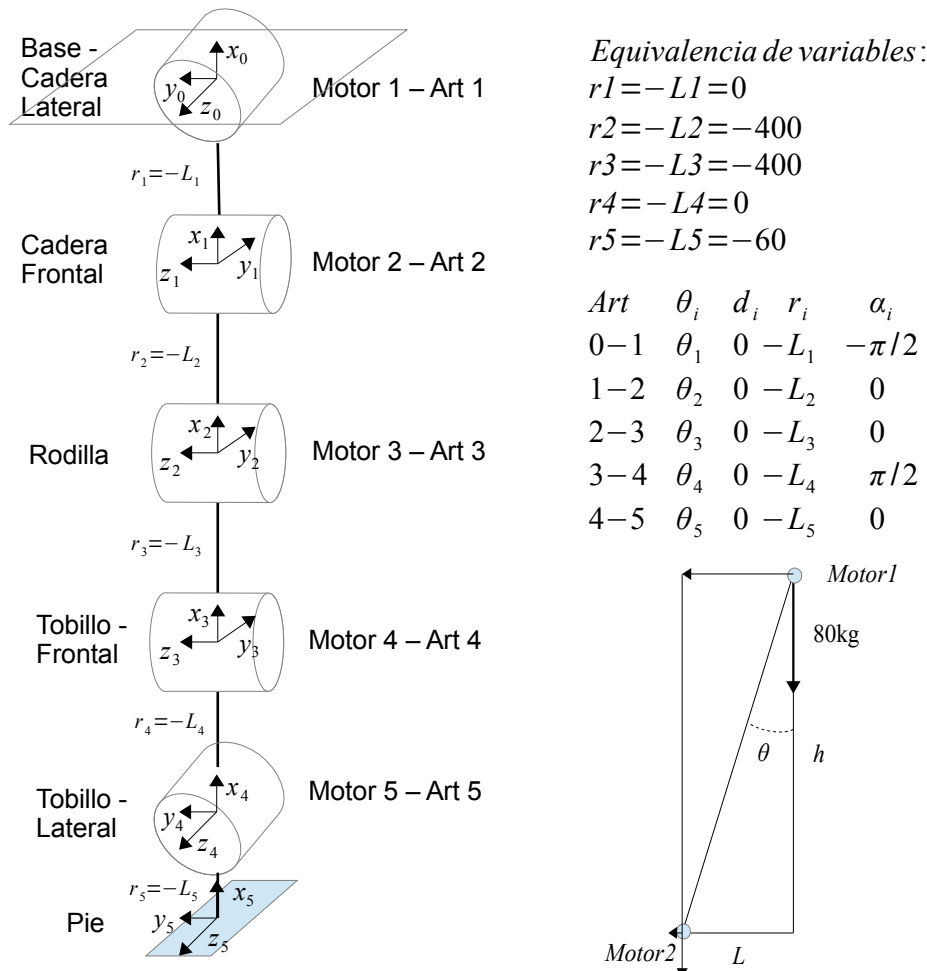


Ilustración 33: Análisis de DH para la pierna de un Humanoide de tamaño natural

Como puede verse, el Motor1 tiene una componente vertical de 80kg.

Mientras la pierna se mantenga completamente estirada y en vertical, no habrá problema. Sin embargo al caminar, se van produciendo una serie de avances mediante la generación de pasos, que

provocan un ángulo y un reparto de fuerzas sobre diferentes ejes.

Así, en el peor de los casos, el primer segmento de la pierna podría llegar a estar en horizontal para subir un escalón de 400mm, mientras que el segundo segmento estaría en vertical.

En ese caso, todo el peso del humanoide se encontraría sustentado sobre la pierna de apoyo en el escalón anterior, pero al iniciar el ascenso pasaría a la pierna más elevada con los ángulos descritos. Por tanto, los 80kg, estarían apoyados sobre ese motor ejerciendo un par bastante elevado.

Con un cálculo básico ($\text{Par} = \text{Peso} * \text{Distancia} = (80 * 9,8) * 0,4$) se llegaría a los 320Nwm.

Como los motores tratados tienen un par máximo de 39Nwm, el problema hace imposible este movimiento.

Sin embargo, se podría calcular, cuál es el ángulo máximo posible que un caminante podría usar para caminar sin que se alcanzara el máximo de los motores. De esa forma:

$$\begin{aligned} \text{Par} &= \text{masa} * \text{gravedad} * \text{distancia} * \sin\theta \\ \theta &= \text{asin}(\text{Par} / \text{masa} * \text{gravedad} * \text{distancia}) = \text{asin}(39/320) \approx 7 \end{aligned}$$

$$\text{Distancia a recorrer} = L * \sin\theta = L * \sin(39/320) \approx 0,048 \text{ m}$$

Como la distancia recorrida se considera a la altura de la rodilla, en realidad, en el suelo se traduce a unos 10cm, en cada paso.

Por tanto, resulta demasiado reducido, aún considerando que el peso pueda repartirse sobre las dos piernas hasta alcanzar una posición de equilibrio sobre la pierna apoyada en el suelo, solución que podría llevar a duplicar la longitud del paso máximo hasta los 20cm.

La siguiente opción es aumentar la capacidad de los motores o sumar más motores en cada articulación a través de sistemas de transmisión dentados (correas,...)

Cualquiera de las dos opciones aumentaría considerablemente las necesidades de energía, pero serían posibles.

Pero duplicando el número de motores o la potencia de cada motor, se volvería a duplicar la longitud del paso máximo hasta los 40cm.

Y en caso de aplicar las dos soluciones, el paso podría llegar a ser considerado, normal (menor a 80cm) distancia que de no alcanzar, iría en beneficio de la estabilidad del humanoide, las diferencias de cálculo, demasiado al límite de sus posibilidades, y de la seguridad de los sistemas.

Aún así, no parece desafortunado pensar que la posibilidad de caminar, aunque sea con limitaciones, resulte muy atractiva.

2 Referencias Bibliográficas

1. WWWaliexpressProd165: Empresa, RMD X8 Pro DC de GYEMS, 20-03-2020, <https://es.aliexpress.com/i/10000018030781.html>
2. WWWgithubDrivers76: Empresa, bump5236/RMDx8Arduino, 20-03-2020, <https://github.com/bump5236/RMDx8Arduino>
3. WWWgithubDrivers105: Empresa, JaimeSaiz/RMDx8ArduinoUBU, 20-03-2020, <https://github.com/JaimeSaiz/RMDx8ArduinoUBU>
4. WWWgithubDrivers77: Empresa, Paul112110/CAN_Bus_Shield-master, 20-03-2020, https://github.com/Paul112110/CAN_Bus_Shield-master
5. WWWaliexpressProd191: Empresa, Escudo CAN-BUS. Compatible con Arduino. MCP2515 (CAN-controller) y MCP2551 (CAN-transceptor). Conexión GPS. Lector de tarjetas MicroSD., 20-03-2020, https://es.aliexpress.com/item/32569554666.html?spm=a2g0o.productlist.0.0.4265363csbW8ha&algo_pvid=3c636977-3a33-4231-8ffc-e427cafabe9e&algo_expid=3c636977-3a33-4231-8ffc-e427cafabe9e-24&btsid=2100bdf016055334328875543e4505&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_
6. WWWgithubDrivers51: Empresa, Seeed-Studio/CAN_BUS_Shield, 20-03-2020, https://github.com/Seeed-Studio/CAN_BUS_Shield
7. WWWamazonProd13: Empresa, RobotDyn - CAN-BUS Shield. Compatible for Arduino. MCP2515 (CAN-controller) and MCP2551 (CAN-transceiver).GPS connect. MicroSD-card reader., 20-03-2020, <https://www.amazon.in/RobotDyn-Compatible-CAN-controller-CAN-transceiver-MicroSD-card/dp/B072JH4XB4>
8. WWWaliexpressProd192: Empresa, Placa de expansión de BUS CAN-Bus Shield V2, IIC I2C y UART para Arduino, 20-03-2020, https://es.aliexpress.com/item/1005001459656600.html?spm=a2g0o.detail.1000014.40.7a1f3449XhsP5O&gps-id=pcDetailBottomMoreOtherSeller&scm=1007.13338.183347.0&scm_id=1007.13338.183347.0&scm-url=1007.13338.183347.0&pvid=84e348fe-c3b1-4e30-8135-708917cca596&t=gps-id:pcDetailBottomMoreOtherSeller,scm-url:1007.13338.183347.0,pvid:84e348fe-c3b1-4e30-8135-708917cca596,ttp_buckets:668%230%23131923%2349_668%23808%234094%23192_668%23888%233325%2312_3338%230%23183347%230_3338%233142%239890%2310_668%234328%2319927%23260_668%232846%238110%23377_668%232717%237558%23179_668%231000022185%231000066058%230_668%233422%2315392%23720
9. WWWamazonProd18: Empresa, Can-Bus Shield V2, 20-03-2020, <https://www.amazon.es/SeeedStudio-Can-Bus-Shield-V2/dp/B07F2KD9BB>
10. WWWseedstudioProd24: Empresa, CAN-BUS Shield V2 adopts MCP2515 and MCP2551, 20-03-2020, <https://www.seedstudio.com/CAN-BUS-Shield-V2.html>
11. WWWaliexpressProd193: Empresa, Módulo de Bus CAN A5, MCP2515, receptor TJA1050, módulo SPI, 20-03-2020, https://es.aliexpress.com/item/4000375197962.html?spm=a2g0o.productlist.0.0.4265363csbW8ha&algo_pvid=3c636977-3a33-4231-8ffc-e427cafabe9e&algo_expid=3c636977-3a33-4231-8ffc-e427cafabe9e-12&btsid=2100bdf016055334328875543e4505&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_

12. WWWarduinoDoc31: Empresa, CAN Bus Using Arduino, 20-03-2020,
<https://create.arduino.cc/projecthub/maurizfa-13216008-arthur-jogy-13216037-agma-maretha-13216095/can-bus-using-arduino-9ce7ba>
13. WWWaliexpressProd194: Empresa, Módulo de Bus CAN MCP2515, receptor TJA1050 SPI para 51 MCU, controlador de brazo D71, 20-03-2020,
https://es.aliexpress.com/item/32799922806.html?spm=a2g0o.productlist.0.0.4265363csbW8ha&algo_pvid=3c636977-3a33-4231-8ffc-e427cafabe9e&algo_expId=3c636977-3a33-4231-8ffc-e427cafabe9e-28&btsid=2100bdf016055334328875543e4505&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_
14. WWWecuarobotDoc0: Empresa, Tutorial CAN de Arduino – Módulo MCP2515 CAN BUS de interfaz con Arduino – Comparación de CAN sobre SPI e I2C, 20-03-2020,
<http://ecuarobot.com/2020/05/30/tutorial-can-de-arduino-modulo-mcp2515-can-bus-de-interfaz-con-arduino/>
15. WWWgithubDrivers52: Empresa, autowp/arduino-mcp2515, 20-03-2020,
<https://github.com/autowp/arduino-mcp2515>
16. WWWgithubDrivers53: Empresa, autowp/arduino-can Hacker, 20-03-2020,
<https://github.com/autowp/arduino-can Hacker>
17. WWWgithubDrivers54: Empresa, spirilis/mcp2515, 20-03-2020,
<https://github.com/spirilis/mcp2515>
18. WWWgithubDrivers49: Empresa, macchina/mcp2515, 20-03-2020,
<https://github.com/macchina/mcp2515>
19. WWWgithubDrivers50: Empresa, collin80/can_common, 20-03-2020,
https://github.com/collin80/can_common
20. WWWseedstudioProd23: Empresa, CANBed - Arduino CAN-BUS Development Kit (ATmega32U4 with MCP2515 and MCP2551), 20-03-2020,
<https://www.seedstudio.com/CANBed-Arduino-CAN-BUS-Development-Kit-Atmega32U4-with-MCP2515-and-MCP2551-p-4365.html>
21. WWWdropboxDoc0: Empresa, GYEMS MOTOR CONTROL PROTOCOL (CAN BUS), 20-03-2020, <https://www.dropbox.com/s/2yzt90i10d6dn27/RMD%20servo%20motor%20control%20protocol%20%28CAN%20BUS%20%29V1.61.pdf?dl=0>
22. WWWdownload.gyemsDoc1: Empresa, GYEMS MOTOR CONTROL PROTOCOL (CAN BUS) V1.61, 20-03-2020, <http://download.gyems.cn/RMD%20servo%20motor%20control%20protocol%20%28CAN%20BUS%20%29V1.61.pdf>
23. WWWmyactuatorDoc0: Empresa, MYACTUATOR - SERVICE AND SUPPORT, 10-01-2023, <https://www.myactuator.com/download>
24. WWWaliexpressProd188: Empresa, Servomotor de CC sin escobillas RMD X8 Pro, motor de engranaje, controlador foc con codificador de 18 bits, 24V, 48V, 10A, 20-03-2020,
https://es.aliexpress.com/item/10000018030781.html?spm=a2g0o.productlist.0.0.792252b8OMMOie&algo_pvid=781e805e-3e16-46e3-8cc1-11ceb624c7f4&algo_expId=781e805e-3e16-46e3-8cc1-11ceb624c7f4-0&btsid=2100bdca16055540558828059e0f68&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_
25. WWWgist.githubDrivers1: Empresa, hdh7485/Control rmd x8 using arduino with can-bus shield, 20-03-2020, <https://gist.github.com/hdh7485/eaff036ba54d6e559fef43da0ca8ec5>

26. WWWgithubDrivers99: Empresa, hdh7485/rmd_x8_control, 20-03-2020,
https://github.com/hdh7485/rmd_x8_control
27. WWWdownload.gyemsDrivers0: Empresa, GYEMS MOTOR CONTROL
PROTOCOL (CAN BUS) V1.61, 20-03-2020, <http://download.gyems.cn/RMD%20servo%20motor%20control%20protocol%20%28CAN%20BUS%20%29V1.61.pdf>
28. WWWdropboxDoc1: Empresa, RMD-X user manual V1.01-EN.pdf, 20-03-2020,
<https://www.dropbox.com/s/ulac3ssiemyx949/RMD-X%20user%20manual%20V1.01-EN.pdf?dl=0>
29. WWWdownload.gyemsDoc0: Empresa, RMD-X User Manual For Motion Actuator, 20-03-2020, <http://download.gyems.cn/RMD-X%20user%20manual%20V1.01-EN.pdf>
30. WWWmymobilemmsDoc0: Empresa, RMD-S Servo Motor Manual V1.5, 20-03-2020,
<http://mymobilemms.com/OFFTHEGRIDWATER.CA/RMD-S-servo-motor-instruction-V1.5.pdf>
31. WWWdropboxDoc2: Empresa, RMD-L Series Servo Actuator User Manual Rev 1.01
(Release).pdf, 20-03-2020, <https://www.dropbox.com/s/4ur0aosv25kv7s8/RMD-L%20Series%20Servo%20Actuator%20User%20Manual%20Rev%201.01%20%28Release%29.pdf?dl=0>
32. WWWdropboxSoft0: Empresa, RMD V2.0.exe, 20-03-2020,
<https://www.dropbox.com/s/beqp6889q6lkd13/RMD%20V2.0.rar?dl=0>
33. WWWdropboxSoft1: Empresa, CP210x_Windows_Drivers.zip, 20-03-2020,
https://www.dropbox.com/s/bjih2p0sj1zd6yv/CP210x_Windows_Drivers.zip?dl=0
34. WWWdropboxSoft2: Empresa, RMD config V1.7(EN).exe, 20-03-2020,
<https://www.dropbox.com/s/h8ynt7vuhtqjzyl/RMD%20config%20V1.7%28EN%29.rar?dl=0>
35. WWWgithubDrivers82: Empresa, Seeed-Studio/Seeed_Arduino_CAN, 20-03-2020,
https://github.com/Seeed-Studio/Seeed_Arduino_CAN/releases
36. WWWgithubDrivers83: Empresa, Seeed-Studio/Seeed_Arduino_CAN - v1.3.0, 20-03-2020,
https://github.com/Seeed-Studio/Seeed_Arduino_CAN/releases/tag/v1.3.0
37. WWWgithubDrivers84: Empresa, Seeed-Studio/Seeed_Arduino_CAN - v2.0.0, 20-03-2020,
https://github.com/Seeed-Studio/Seeed_Arduino_CAN/releases/tag/v2.0.0
38. WWWgithubDrivers85: Empresa, Seeed-Studio/Seeed_Arduino_CAN - v2.0.1, 20-03-2020,
https://github.com/Seeed-Studio/Seeed_Arduino_CAN/releases/tag/v2.1.0
39. WWWelectronicshubDoc1: Empresa, ELECTRONICS HUB - Arduino MCP2515 CAN
Bus Interface Tutorial, 20-03-2020, <https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/>
40. WWWrah.web.nitech.acDoc0: Empresa, Takuya Sanada, 20-03-2020,
<http://rah.web.nitech.ac.jp/people>
41. WWWgithubDrivers86: Empresa, bump5236/biped, 20-03-2020,
<https://github.com/bump5236/biped>
42. WWWgithubDrivers87: Empresa, bump5236/ML_biped, 20-03-2020,
https://github.com/bump5236/ML_biped
43. WWWgithubDrivers88: Empresa, bump5236/Sim_biped, 20-03-2020,
https://github.com/bump5236/Sim_biped
44. WWWgithubDrivers89: Empresa, bump5236/JumpPhaseSimulation, 20-03-2020,
<https://github.com/bump5236/JumpPhaseSimulation>

45. WWWgithubDrivers90: Empresa, t-kamimura/arduino_can, 20-03-2020, https://github.com/t-kamimura/arduino_can
46. WWWclickhouseDoc0: Empresa, UInt8, UInt16, UInt32, UInt64, Int8, Int16, Int32, Int64, 20-03-2020, <https://clickhouse.tech/docs/es/sql-reference/data-types/int-uint/>