The iCub CAN protocol for the foc motor board

This document describes the iCub CAN protocol with focus on the foc motor board.

Approval History

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Version | Author | | Date | Approved | | Date |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 0.5 | 12 Apr 21 | M. Accame | First WIP version. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1 Introduction 1](#_Toc69309169)

[2 The messages of the motor polling class 2](#_Toc69309170)

[2.1 Class polling motor control 3](#_Toc69309171)

[2.1.1 Messages used for general management 3](#_Toc69309172)

[2.1.2 Messages used for configuring the motion control 3](#_Toc69309173)

[3 The messages of the motor streaming class 16](#_Toc69309174)

[3.1 Class periodic motor control 16](#_Toc69309175)

[3.1.1 Messages for debug 16](#_Toc69309176)

[3.1.2 Messages which stream status 17](#_Toc69309177)

[3.1.3 Messages which send setpoints 18](#_Toc69309178)

[4 Use of the messages inside the ems and foc boards 20](#_Toc69309179)

[4.1 Configuration of the motor 20](#_Toc69309180)

[4.2 Control mode 21](#_Toc69309181)

[4.3 Runtime behaviour 22](#_Toc69309182)

[4.4 Other messages 22](#_Toc69309183)

[5 References 24](#_Toc69309184)

# Introduction

This document describes the iCub CAN protocol specifically used by the foc motor board.

We refer to foc board as one half of the 2foc board. This latter is a PCB with two MPUs each with one different CAN address where each MPU can manage a single motor.

For more information about the iCub CAN protocol refer to the document *TSD-ICUB-CAN-protocol-sensor-boards* (see [1])

The messages of the motor polling class

Here is description of the messages of the motor polling class.

## Class polling motor control

The messages of class polling motor control (CLS = 000b) are used mostly for motor control boards (MOT). However, some messages of this class have a general use and are also supported by the sensor board.

### Messages used for general management

They are messages originally developed for MC boards that have similar versions for sensor boards.

They are the messages with CMD in the following table.

|  |  |
| --- | --- |
| CMD | Description |
|  | … others |
| 0x32 | SET\_BOARD\_ID |
|  | … others |
| 0x52 | GET\_FIRMWARE\_VERSION |
|  | … others |

**Table 1** – The CMDs of the polling motor control class (supported by some sensor boards).

The above messages are supported also by the foc board. To avoid duplications of information, for details refer to the document TSD-ICUB-CAN-protocol-sensor-boards which originally treated them.

### Messages used for configuring the motion control

These messages are used to configure the motion control service in boards which manage one or two joints.

The protocol rules in class polling motor control tell that each board can manage at most two joints / motors. The info of the motor number is contained in the most significant bit of the field CMD. As such, this rule limits the possible commands specified by the CMD field to 128 values from 0x00 to 0x7F.

The messages we use for the foc board are however a small subset of the complete set of CAN messages of the class polling motor control.

Some messages are generic (also managed by the mc4can board) and only one is dedicated to the foc board (the SET\_MOTOR\_CONFIG).

In here is a table of all the messages which are supported by the foc board with some comments.

|  |  |
| --- | --- |
| CMD | Description |
| 9 | SET\_CONTROL\_MODE  The command GET\_CONTROL\_MODE = 0x07 exists but is not managed by foc / ems. |
| 72 | SET\_CURRENT\_LIMIT  A command GET\_CURRENT\_LIMIT does not exist in the iCub CAN protocol. |
| 101 | SET\_CURRENT\_PID  The command GET\_CURRENT\_PID = 102 exists but is not managed by foc / ems. |
| 105 | SET\_VELOCITY\_PID  The command GET\_VELOCITY\_PID = 106 exists but is not managed by foc / ems. |
| 119 | SET\_MOTOR\_CONFIG  This message is only for the foc board and is not used by other MOT boards. |
| 4 | CALIBRATE\_ENCODER  I suspect that the command is not managed correctly by the pair ems / 2foc because the function in Motor.c which use passes a NULL pointer instead of the arguments expected by the iCub CAN protocol.  As a result of that, I suspect that an undefined calibration type is used. |
| 121 | SET\_TEMPERATURE\_LIMIT  A command GET\_TEMPERATURE\_LIMIT does not exists in the iCub CAN protocol. |

**Table 2** – The CMDs of the polling motor control class sent by the ems to the 2foc board plus some comments on them.

|  |  |
| --- | --- |
| CMD | Description |
| **0x04**  **0x84** | **CALIBRATE\_ENCODER**  **In brief**  Used to calibrate the encoder for motor m = 0 or 1. This message models the struct icubCanProto\_calibrator\_t which has a calibration type and some parameters which depends on it.  **Parsed by**    appl<MC>  mc-generic  foc  **Format of PAYLOAD**    M|0x04  CTYPE  CPARS   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 7. * CTYPE is the type of calibration. Its value is in icubCanProto\_calibration\_type\_t and can assume values such as: type0\_hard\_stops = 0, type1\_abs\_sens\_analog = 1, type2\_hard\_stops\_diff = 2, type3\_abs\_sens\_digital = 3, type4\_abs\_and\_incremental = 4, type6\_mais = 6, type7\_hall\_sensor = 7. * CPARS are the calibration parameters which depend on the calibration type. Here are some examples:   type0\_hard\_stops: contains the struct icubCanProto\_calibrator\_params\_type0\_hard\_stops\_t    PWMLIMIT  VELOCITY  ZEROPAD  - PWMLIMIT is the value pwmlimit expressed in int16\_t mapped in little endian order  - VELOCITY is the velocity expressed in int16\_t mapped in little endian order  - ZEROPAD is a field with value 0x0000.  …  …  **Actions on reception**    appl<MC>  mc-generic  foc  It applies the specified calibration to motor m.  **Reply**    app<MC>  It does not reply. |

FRAME-calibencoder: command that start a calibration procedure

ID

This frame imposes a calibration type 0 (hard stop) to motor m = 0 of the board with PWM 0x0100 (256) and velocity 0x0101 (257).

{CLS = POLLING-MC = 000b} {SRC = 0000b} {DST = 0001b} = 0x001

CMD

ARG

[ 0x00, 0x00, 0x01, 0x01, 0x01, 0x00, 0x00 ]

CALIBRATE\_ENCODER = 0x04

sizeof(ARG) = 7

**Figure 1**: The frame that starts a calibration of type 0 hard stops to motor m = 0 of board at address 1.

|  |  |
| --- | --- |
| CMD | Description |
| **0x09**  **0x89** | **SET\_CONTROL\_MODE**  **In brief**  Used to impose a control mode to the motor m = 0 or 1.  **Parsed by**    appl<MC>  mc-generic  foc  **Format of PAYLOAD**    M|0x09  CTRL   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 1. * CTRL is the required control mode. Its value is in icubCanProto\_controlmode\_t.   Values sent by the ems board to the foc are however only idle, openloop, speed\_voltage, current:  icubCanProto\_controlmode\_idle = 0x00,  icubCanProto\_controlmode\_openloop = 0x50,  icubCanProto\_controlmode\_speed\_voltage = 0x0A,  icubCanProto\_controlmode\_current = 0x06  **Actions on reception**    appl<MC>  mc-generic  foc  It applies the control mode to motor m.  **Reply**    app<MC>  It does not reply. |

FRAME-ctrlmode: command that imposes the IDLE control mode

ID

This frame imposes the IDLE (0x00) control mode to motor m = 0 of the board.

{CLS = POLLING-MC = 000b} {SRC = 0000b} {DST = 0001b} = 0x001

CMD

ARG

[ 0x00 ]

SET\_CONTROL\_MODE = 0x09

sizeof(ARG) = 1

**Figure 2**: The frame that imposes control mode IDLE to motor m = 0 of board at address 1: [0x001 | 0x09, 0x00]. For motor m =1 the message would be: [0x001 | 0x89, 0x00].

|  |  |
| --- | --- |
| CMD | Description |
| **0x48**  **0xC8** | **SET\_CURRENT\_LIMIT**  **In brief**  Used to configure current limits of motor m = 0 or 1.  **Parsed by**    appl<MC>  mc-generic  foc  **Format of PAYLOAD**    M|0x48  X  NOMINAL  PEAK  OVERLOAD   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 7. * X is an undocumented and un-used value (the ems board uses 0). * NOMINAL is the nominal current contained inside eOmc\_current\_limits\_params\_t and is expressed in mA units contained inside a int16\_t with little endian ordering. * PEAK is the peak current contained inside eOmc\_current\_limits\_params\_t * OVERLOAD is the overload current inside eOmc\_current\_limits\_params\_t   **Actions on reception**    appl<MC>  mc-generic  foc  It applies the limits to motor m.  **Reply**    app<MC>  It does not reply.  **NOTES**  I suspect that the foc does not respect this protocol or the function Motor\_config\_max\_currents\_2FOC() has a bug because the CAN frame is formed in a different way. |

FRAME-currlim: command that imposes the current limits

ID

This frame imposes currents to motor 0 in this way:

nominal = 0x0102 (258 mA), peak = 0x0203 (515 mA), overload = 0x0304 (772 mA)

{CLS = POLLING-MC = 000b} {SRC = 0000b} {DST = 0001b} = 0x001

CMD

ARG

[ 0x00 ] [0x02, 0x01] [0x03, 0x02] [0x04, 0x03]

SET\_CURRENT\_LIMIT = 0x48

sizeof(ARG) = 7

**Figure 3**: The frame that imposes current limits to motor m = 0 of board at address 1: [0x001 | 0x48, 0x00, 0x02, 0x01, 0x03, 0x02, 0x04, 0x03].

|  |  |
| --- | --- |
| CMD | Description |
| **0x65**  **0xE5** | **SET\_CURRENT\_PID**  **In brief**  Used to configure the current PID of motor m = 0 or 1.  **Parsed by**    appl<MC>  mc-generic  foc  **Format of PAYLOAD**    M|0x65  KP  KI  KD  KS   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 7. * KP is the proportional gain expressed in int16\_t with little endian ordering. * KI is the integral gain expressed in int16\_t with little endian ordering. * KD is the derivative gain expressed in int16\_t with little endian ordering. * KS is the shift factor expressed in uint8\_t.   **Actions on reception**    appl<MC>  mc-generic  foc  It applies the PID values to motor m.  **Reply**    app<MC>  It does not reply.  **NOTES**  The way the values are formatted is inside function xxxx(). |

FRAME-currpid: command that imposes the current PID

ID

This frame imposes a PID to motor m = 0 in this way:

Kp = 0x0102 (258), Ki = 0x0203 (515), Kd = 0x0304 (772), Ks = 0x01 (1)

{CLS = POLLING-MC = 000b} {SRC = 0000b} {DST = 0001b} = 0x001

CMD

ARG

[0x02, 0x01] [0x03, 0x02] [0x04, 0x03] [0x01]

SET\_CURRENT\_PID = 0x65

sizeof(ARG) = 7

**Figure 4**: The frame that imposes the current PID of motor m = 0 of board at address 1: [0x001 | 0x65, 0x02, 0x01, 0x03, 0x02, 0x04, 0x03, 0x01].

|  |  |
| --- | --- |
| CMD | Description |
| **0x69**  **0xE9** | **SET\_VELOCITY\_PID**  **In brief**  Used to impose the velocity PID of motor m = 0 or 1.  **Parsed by**    appl<MC>  mc-generic  foc  **Format of PAYLOAD**  M|0x69  KP  KI  KD  KS   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 7. * KP is the proportional gain expressed in int16\_t with little endian ordering. * KI is the integral gain expressed in int16\_t with little endian ordering. * KD is the derivative gain expressed in int16\_t with little endian ordering. * KS is the shift factor expressed in uint8\_t.   **Actions on reception**    appl<MC>  mc-generic  foc  It applies the PID values to motor m.  **Reply**    app<MC>  It does not reply.  **NOTES**  The way the values are formatted is inside function yyyy(). |

FRAME-velpid: command that imposes the velocity PID

ID

This frame imposes a PID to motor m = 0 in this way:

Kp = 0x0102 (258), Ki = 0x0203 (515), Kd = 0x0304 (772), Ks = 0x01 (1)

{CLS = POLLING-MC = 000b} {SRC = 0000b} {DST = 0001b} = 0x001

CMD

ARG

[0x02, 0x01] [0x03, 0x02] [0x04, 0x03] [0x01]

SET\_VELOCITY\_PID = 0x69

sizeof(ARG) = 7

**Figure 5**: The frame that imposes the velocity PID of motor m = 0 of board at address 1: [0x001 | 0x69, 0x02, 0x01, 0x03, 0x02, 0x04, 0x03, 0x01].

|  |  |
| --- | --- |
| CMD | Description |
| **0x77**  **0xF9** | **SET\_****MOTOR\_CONFIG**  **In brief**  Used to configure the motor of the foc. This message is used only by the foc board which has only one motor, so the associated CMD = 0xF7 is not used.  **Parsed by**    appl<MC>  foc  **Format of PAYLOAD**  0x77  FLAGS  ROTENCRES  ROTINDEXOFF  POLES  ENTLR   * sizeof(ARG) = 7. * FLAGS contains the following binary flags: HAS\_QE = 0x01, HAS\_HALL = 0x02, HAS\_TSENS = 0x04, USE\_INDEX = 0x08, HAS\_SPEED\_QE = 0x10, ENA\_VERBOSE = 0x20. * ROTENCRES contains the resolution of the rotor encoder, expressed in int16\_t with little endian ordering. * ROTINDEXOFF contains the offset of the rotor encoder, expressed in int16\_t with little endian ordering. * POLES contains the number of poles in the motor, expressed in uint8\_t. * ENTLR contains the encoder tolerance, expressed in uint8\_t.   **Actions on reception**    appl<MC>  foc  It applies the motor configuration.  **Reply**    app<MC>  It does not reply.  **NOTES**  For better description of the fields see struct eOmc\_motor\_config\_t and function yyyy(). |

FRAME-setmotorconfig: command that configures the foc

ID

This frame impose: HAS\_QE, HAS\_HALL, rotor encoder resolution = 0x0201 = 513, rotor index offset = 0x0403 = 1027, number of poles = 0x06 = 6, encoder tolerance = 0x07 = 7

{CLS = POLLING-MC = 000b} {SRC = 0000b} {DST = 0001b} = 0x001

CMD

ARG

[0x03] [0x01, 0x02] [0x03, 0x04] [0x06] [0x07]

SET\_MOTOR\_CONFIG = 0x77

sizeof(ARG) = 7

**Figure 6**: The frame that imposes the settings of the motor at address 1: [0x001 | 0x77, 0x03, 0x01, 0x02, 0x03, 0x04, 0x06, 0x07].

|  |  |
| --- | --- |
| CMD | Description |
| **0x79** | **SET\_TEMPERATURE\_LIMIT**  **In brief**  Used to configure temperature limits of motor m = 0 or 1.  **Parsed by**    appl<MC>  mc-generic  foc  **Format of PAYLOAD**    M|0x79  HARDWARE\_LIMIT   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 2. * HARDWARE\_LIMIT is the hardware temperature limit contained inside eOmeas\_temperature\_t and is expressed in raw value contained inside a int16\_t with little endian ordering.   **Actions on reception**    appl<MC>  mc-generic  foc  It applies the limits to motor m.  **Reply**    app<MC>  It does not reply. |

FRAME-currlim: command that imposes the temperature limit

ID

This frame imposes currents to motor 0 in this way:

hardware\_limit = 0x0102 (258 mV),

{CLS = POLLING-MC = 000b} {SRC = 0000b} {DST = 0001b} = 0x001

CMD

ARG

[ 0x00 ] [0x02, 0x01]

SET\_TEMPERATURE\_LIMIT = 0x79

sizeof(ARG) = 2

**Figure 3**: The frame that imposes current limits to motor m = 0 of board at address 1: [0x001 | 0x79, 0x00, 0x02].

The messages of the motor streaming class

Here is description of the messages of the motor streaming class.

## Class periodic motor control

The class periodic motor control (CLS = 001b) contains messages used by motor control boards to stream their status to the host or also to other motor control boards.

|  |  |
| --- | --- |
| TYP | Description |
| 0x0 | 2FOC |
| 0x1 | POSITION |
| 0x2 | PID\_VAL |
| 0x3 | STATUS |
| 0x4 | CURRENT |
| 0x5 | OVERFLOW |
| 0x6 | PRINT |
| 0x7 | VELOCITY |
| 0x8 | PID\_ERROR |
| 0x9 | DEBUG |
| 0xA | MOTOR\_POSITION |
| 0xB | MOTOR\_SPEED |
| 0xC | ADDITIONAL\_STATUS |
| 0xF | EMSTO2FOC\_DESIRED\_CURRENT |

**Table 3** – The TYPs of the periodic motor control class.

Full description of these messages is out of the scope of this document. In here, we shall describe only the messages supported by the foc board.

They are very few: 2FOC, EMSTO2FOC\_DESIRED\_CURRENT, PRINT and ADDITIONAL\_STATUS.

### Messages for debug

The message PRINT is supported by the MOT boards and by some other sensor boards. Its full description can be found in document *TSD-ICUB-CAN-protocol-sensor-boards* [1].

### Messages which stream status

|  |  |
| --- | --- |
| TYP | Description |
| **0x0** | **2FOC**  **In brief**  It transports data from the foc board: current, velocity and position.  **Emitted by**  foc  Transmission is started by … TBD.  **Format of DATA**    CURRENT  VELOCITY  POSITION   * sizeof(DATA) = 8. * CURRENT is an int16\_t in little endian format which contains the values of current measured by the foc board. Unit is mA. * VELOCITY is an int16\_t in little endian format which contains the values of velocity measured by the foc board. Unit is ICUBDEG/ms. * POSITION is an int32\_t in little endian format which contains the values of position measured by the foc board. Unit is ICUBDEG. |

FRAME-2FOC: status values

ID

Explanation of the content of DATA.

* DATA[0, 1] contains the current expressed as int16\_t
* DATA[2, 3] contains the velocity expressed as int16\_t
* DATA[4, 5, 6, 7] contains the position expressed as int32\_t

{CLS = PERIODIC-MC = 001b} {SRC = 0001b} {TYP = 2FOC = 0x0} = 0x110

DATA

[LSB-CUR] [MSB-CUR] [LSB-VEL] [MSB-VEL] [B0-POS] [B1-POS] [B2-POS] [B3-POS]

sizeof(DATA) = 8

**Figure 7**: This frame contains the values measured and emitted by the foc board at address 0x1: [0x110 | 0x--, 0x--, 0x--, 0x--, 0x--, 0x--, 0x--, 0x--]. The frame emitted by board at address 0x2 would have ID = 0x120.

|  |  |
| --- | --- |
| TYP | Description |
| **0xC** | **ADDITIONAL\_STATUS**  **In brief**  It transports data from the foc board: current motor temperature retrieved from temperature sensor.  **Emitted by**  foc  Transmission happens from the foc board every 100+board\_id\*100 ms and then parsed by ems in MController.  **Format of DATA**    INTERACTION\_MODE  TEMPERATURE   * sizeof(DATA) = 4. * INTERACTION\_MODE are 2 nibbles contained in the first uint8\_t in little endian format which contains the interaction mode for the first and second joints whose possible values are defined in the enum typedef icubCanProto\_interactionmode\_t. The first 2 bytes are not managed by the foc board and therefore not overridden. Not only the first byte is skipped but 2 in order to not having parsing problems since temperature is int16\_t * TEMPERATURE is an int16\_t in little endian format which contains the value of motor temperature measured by the foc board. Unit is raw. |

FRAME-2FOC: status values

ID

Explanation of the content of DATA.

* DATA[0] contains the interaction mode for joint 1 and joint 2 expressed as uint8\_t (one per nibble)
* DATA[2, 3] contains the temperature expressed as int16\_t

{CLS = PERIODIC-MC = 001b} {SRC = 0001b} {TYP = 2FOC = 0x0} = 0x110

DATA

[INTMD-J2|INTMD-J1] [LSB-TEMP] [MSB-TEMP]

sizeof(DATA) = 4

**Figure 7**: This frame contains the values measured and emitted by the foc board at address 0x1: [0x110 | 0x--, 0x--, 0x--, 0x--, 0x--, 0x--, 0x--, 0x--]. The frame emitted by board at address 0x2 would have ID = 0x120.

### Messages which send setpoints

|  |  |
| --- | --- |
| TYP | Description |
| **0xF** | **EMSTO2FOC\_DESIRED\_CURRENT**  **In brief**  It transports the currents that the MController wants to send to every foc board it may have. As there can be a maximum of 4 foc boards, this message contains the four currents. Each foc board use the message by picking up the current which matches its CAN address. For instance, the foc at CAN address 0x1 will pick up the first current, the foc at can address 0x2 will pick up the second, etc.  **Emitted by**  ems  Transmission happens every 1 ms and is done inside the MController.  **Format of DATA**    DESIREDCUR1  DESIREDCUR2  DESIREDCUR3  DESIREDCUR4   * sizeof(DATA) = 8. * DESIREDCUR1 is the desired current for the foc at address 0x1 expressed an int16\_t in little endian format. * DESIREDCUR2 is the desired current for the foc at address 0x2 expressed an int16\_t in little endian format. * DESIREDCUR3 is the desired current for the foc at address 0x3 expressed an int16\_t in little endian format. * DESIREDCUR4 is the desired current for the foc at address 0x4 expressed an int16\_t in little endian format. |

FRAME- EMSTO2FOC\_DESIRED\_CURRENT: desired currents as requested by the ems

ID

Explanation of the content of DATA.

* DATA[0, 1] contains the current requested to board w/ address 0x1 expressed as int16\_t
* DATA[2, 3] contains the current requested to board w/ address 0x2 expressed as int16\_t
* DATA[4, 5] contains the current requested to board w/ address 0x3 expressed as int16\_t
* DATA[6, 7] contains the current requested to board w/ address 0x4 expressed as int16\_t

{CLS = PERIODIC-MC = 001b} {SRC = 0000b} {TYP = EMSTO2FOC\_... = 0xF} = 0x10F

DATA

[LSB-C1] [MSB-C1] [LSB-C2] [MSB-C2] [LSB-C3] [MSB-C3] [LSB-C4] [MSB-C4]

sizeof(DATA) = 8

**Figure 8**: This frame contains the values of current that the ems request to all the foc boards. As the frame is emitted by the ems which has CAN address 0x0, the ID is always 0x10F. A typical frame is: [0x10F | 0x--, 0x--, 0x--, 0x--, 0x--, 0x--, 0x--, 0x--].

Use of the messages inside the ems and foc boards

Here is an analysis of how the messages are used inside the ems and the foc board.

## Configuration of the motor

This section deals of messages which flow from the ems towards the foc board and are used to configure some motor parameters before any movement of the motor.

The ETH message handler MController\_config\_motor() inside the ems board is called when yarprobotinterface sends the configuration of the whole motor inside struct eOmc\_motor\_config\_t.

This handler calls function MController\_config\_motor() which in turns calls Motor\_config() which fills data inside the MController and then calls Motor\_config\_2FOC() which sends several CAN frames to the foc board:

* SET\_CURRENT\_PID from inside Motor\_config\_current\_PID\_2FOC();
* SET\_VELOCITY\_PID from inside Motor\_config\_velocity\_PID\_2FOC();
* SET\_CURRENT\_LIMIT from inside Motor\_config\_2FOC();
* SET\_MOTOR\_CONFIG from inside Motor\_config\_2FOC();
* SET\_TEMPERATURE\_LIMIT from inside Motor\_config\_2FOC();

Some of the above CAN frames can be transmitted also separately when triggered by other ETH handlers or inside the MController. However, this behaviour mode may not be used by yarprobotinterface which always use the global configuration mode. PLS VERIFY.

Details in the following table.

|  |
| --- |
| CMD |
| **SET\_CURRENT\_PID**  The message is sent by the ems to the foc inside Motor.c, function Motor\_config\_current\_PID\_2FOC() and is triggered by the ETH message handler eoprot\_fun\_UPDT\_mc\_motor\_config() which configures the entire motor.  It can also be triggered by another ETH message handler in case yarprobotinterface wanted to configure the current PID alone: eoprot\_fun\_UPDT\_mc\_motor\_config\_pidcurrent(). |
| **SET\_VELOCITY\_PID**  The message is sent by the ems to the foc inside Motor.c, function Motor\_config\_velocity\_PID\_2FOC() and is triggered by the ETH message handler eoprot\_fun\_UPDT\_mc\_motor\_config() which configures the entire motor.  It can also be triggered by another ETH message handler in case yarprobotinterface wanted to configure the velocity PID alone: eoprot\_fun\_UPDT\_mc\_motor\_config\_pidspeed (). |
| **SET\_CURRENT\_LIMIT**  The message is sent by the ems to the foc inside Motor.c, function Motor\_config\_2FOC() and is triggered by the ETH message handler eoprot\_fun\_UPDT\_mc\_motor\_config() which configures the entire motor.  It can also be triggered by another ETH message handler in case yarprobotinterface wanted to configure the current limits alone: eoprot\_fun\_UPDT\_mc\_motor\_config\_currentlimits(). However, in this case the CAN frame is not formed correctly both inside the handler and inside Motor\_config\_max\_currents\_2FOC(). |
| **SET\_MOTOR\_CONFIG**  The message is sent by the ems to the foc inside Motor.c, function Motor\_config\_2FOC() and is triggered by the ETH message handler eoprot\_fun\_UPDT\_mc\_motor\_config() which configures the entire motor.  It can also be triggered by another ETH message handler in case yarprobotinterface wanted to configure the resolution of the encoder alone: eoprot\_fun\_UPDT\_mc\_motor\_config\_rotorencoder(). This function calls Motor\_config\_encoder() which effectively transmits the CAN frame. |
| **SET\_TEMPERATURE\_LIMIT**  The message is sent by the ems to the foc inside Motor.c, function Motor\_config\_2FOC() and is triggered by the ETH message handler eoprot\_fun\_UPDT\_mc\_motor\_config() which configures the entire motor.  It can also be triggered by another ETH message handler in case yarprobotinterface wanted to configure the temperature limit alone: eoprot\_fun\_UPDT\_mc\_motor\_config\_temperaturelimitconfig().  This function calls Motor\_config\_max\_temperature(), which then calls Motor\_config\_motor\_max\_temperature\_2FOC() that does the same operations as in Motor\_config\_2FOC() for the temperature part. |

**Table 4** – The CMDs used for configuration of the foc.

**NOTE**: there are not messages which reads back the values hopefully written in the foc board.

## Control mode

This section deals of messages which flow from the ems towards the foc board and are used change the control mode.

It is used only one command, the SET\_CONTROL\_MODE.

This command is sent by the MController whenever it is necessary to set the motor idle or in the different running modes.

The message is sent by function Motor\_set\_control\_mode\_2FOC() inside Motor.c and in turn is called by Motor\_set\_idle(), Motor\_force\_idle() and Motor\_set\_run(). All the above are called by JointSet.c and Calibrators.c.

|  |
| --- |
| CMD |
| **SET\_CONTROL\_MODE**  The message is sent by the ems to the foc from function Motor\_set\_control\_mode\_2FOC() inside Motor.c  It uses modes such as: icubCanProto\_controlmode\_openloop, icubCanProto\_controlmode\_speed\_voltage or icubCanProto\_controlmode\_current, icubCanProto\_controlmode\_idle, icubCanProto\_controlmode\_forceIdle.  The function Motor\_set\_control\_mode\_2FOC() is sent from several places inside the MController where it is necessary to start o stop the motors (calibration etc). |

**Table 5** – The CMD used to change the control mode.

## Runtime behaviour

During runtime, every millisecond, the ems:

* decodes the CAN frame 2FOC which contains the status of the foc board (current, velocity and position), uses them inside the MController and
* sends the message EMSTO2FOC\_DESIRED\_CURRENT to every foc board with the desired currents.

Here are details.

|  |
| --- |
| CMD |
| **2FOC**  This message is transmitted by the foc to the ems to report its current, velocity and position.  It is parsed by the CAN message handler eocanprotMCperiodic\_parser\_PER\_MC\_MSG\_\_2FOC() at the beginning of every cycle of the control loop and the inside value are:   * copied inside the motor status which is streamed up to yarprobotinterface; * used by the MController with call to function MController\_update\_motor\_odometry\_fbk\_can(). |
| **EMSTO2FOC\_DESIRED\_CURRENT**  This message is sent by the ems to every foc board to impose the desired current. The calling functions is Motor\_actuate() which is in turn called by eo\_motioncontrol\_Tick() every millisecond. |

**Table 6** – The messages exchanged between the ems and the foc board at every control cycle.

## Other messages

The message CALIBRATE\_ENCODER is used in function Motor\_uncalibrate() inside Motor.c but the result is very likely a malformed frame because the former function is wrongly passed a NULL pointer to the required parameters.

Here are details.

|  |
| --- |
| CMD |
| **CALIBRATE\_ENCODER**  The message is sent by the ems to the foc from function Motor\_uncalibrate() inside Motor.c  The former of this message is called with wrong parameters and it uses a NULL pointer which may end up in undefined behaviour. The above does not cause any crash of code in the ems but surely produces a malformed CAN frame.  The function Motor\_uncalibrate() is called in cases such as:   * Calibration timeout, function JointSet\_do\_wait\_calibration(), * Start of calibration of tripod or of hands in R1, function JointSet\_calibrate(). |
|  |

**Table 7** – A malformed CAN message which is probably not necessary.

References

[1] TSD-ICUB-CAN-protocol-sensor-boards, <https://github.com/robotology/icub-firmware/blob/master/emBODY/eBdocs/arch-arm/TSD-ICUBUNIT-canprotocol-sensorboards.docx>