User Guide Motion Protocols CPR-CAN and CPR-CAN-V2

May 27th, 2019
Version 06
christian.meyer@commonplacerobotics.de

Changelog:
V06: Added information: where to find status of referencing, see section 5
V05: Added referencing command. Added "separate DIO-Board" description

1. Introduction

Motor control boards developed by Commonplace Robotics are commanded via the CAN field bus at 500 kBaud with one of the following protocols:

- CPR-CAN
  A simple protocol that allows to define joint position setpoints or joint velocities. The positions are transferred as 16 bit value. This protocol is used in the Mover4 and Mover6 robots.
- CPR-CAN-V2
  An extension of the above protocol with 32 bit position values. This protocol is used in for the igus robolink and drylin robots, and the SRA robots. Mover6 robots from 2016 on understand both protocols.

Furthermore, the boards can be configured via CAN, e.g. regarding their PID parameter or error limits.

2. Tips and Hints

- Most of these hints extend the documentation in the following part
- The PID parameter of the position and velocity controller on the board should be adapted to the application using the "ModuleControl" software, see www.cpr-robots.com → Media → Wiki → ConfigSoftware
- If the boards stops then the reason for stopping is transmitted in the first byte of the boards answer, see section "Error Codes". If it happens in the normal application (without collision) then the board parameter should be adapted. E.g. if a "Position lag" error occurs repeatedly, the "maxLag" parameter should be increased.
- The on-board test for position lag and overcurrent can be switched off. To do this set the according parameter "maxLag" and "maxCurrent" to zero.
- Digital IO are only set when the board is in error state 0x00, no error. See section 8.
- It is possible to set a "Rollover"-flag. This means that the board is not limited in the position area (16 or 32bit), but can jump from the highest position to the lowest position without interrupting the motion. But it is advisable to use the 32bit protocol instead of 16bit with rollover.

Please refer also to the C++ implementations found on www.githu.com/CPR-Robots for code examples. The Mover4 has the standard CAN IDs 0x10, 0x20, 0x30 and 0x40 for the four joint modules, the Mover6 has the additional ID 0x50 and 0x60.

Please also see how the available controller (CPRog, …) communicates with the boards.

3. Main Loop

The CPR robot arms do have the standard CAN IDs 0x10, 0x20, 0x30, 0x40, (and 0x50, 0x60) for the joint modules. The joint modules answer with the incremented ID, e.g. 0x11, 0x21, …

To run the modules a constant main loop has to be set up with e.g. 20 Hz. In this main loop the motion command has to be send, normally the SetJoint command for position control. If these commands are not received in constant time spans then the controller board will get into an error state because the master control might be dead. The main loop has to be independent of any graphical user elements to reach the necessary stability.

To get the motors running the following steps are necessary:

1. Connect the CAN bus
2. Start the main loop and send position command cyclically
3. Sync to hardware position: Store the current joint positions as setpoint positions in the control software. The current positions are part of the answers to the position command.
4. Reset all joints. The error codes should be 0x04 after reset: "Motor not enabled".
5. Enable all joints. The error codes should be 0x00 now.
6. Now you can increment or decrement the setpoint position according to the desired motion.
7. During this process the main motion loop still has to be active with only short interruptions to send the additional commands.

When sending the commands to all joints (e.g. reset or position command) there should be a small wait of 1 to 2 ms in between the CAN messages.

4. Set Joints to Zero

The joint electronics does not have a electrical or mechanical end switch, but stores the last position in non-volatile memory. This means that the joints should not be moved without electricity on the boards, because then the position is lost.

To set the joints position to zero you can use the command 'SetToZero' see section 11. But please be aware that the motors are not enabled when resetting the position, because this might lead to unintended motion! The procedure should be:

- Disable the motors so that the error state is != 0x00
- Wait for a short time > 100 ms
- Send the 'SetToZero' command

- Send the command a second time to verify, see the command description
- Wait for a short time > 100 ms
- Reset the joint
- Load the joint position and align with your setpoint position in the robot control

5. Referencing

The igus robolink robots work without referencing, but the position may not be accurate. Referencing the joints after each startup allows to get the robot into a precise and known position.

Referencing is available in the modular DIN-Rail control and the robolink-DCi control. It is not available for the Mover robots.

To start the referencing:

- Connect to the joint
- Reset and enable the joint so that the error status is 0x00
- Send the 'StartReferencing' command, see section "Commands"
- Send the command a second time to verify, see the command description
- Now the joint starts with the referencing motion.
- When the referencing motion has finished the joint changes into an error state with the motor not enabled and position lag errors.
- The position is now set to the offset value set in the joint electronics

The referencing type and several parameters can be set using the module control software.

The referencing process can be stopped with a "Reset" command.

The referencing status of the joint can be found in the Joint-Motion answer, in byte 8:

- Value byte 8: zero means that the joint is not referenced
- Value byte 8: 0x80 means that the joint is referenced
- Other values: see below, this means a combination of digital inputs and reference status.

Byte 8 can contain the digital input states in bits 0 to 6, e.g. for the Mover robots.
And it can contain the reference status in bit 7, e.g. for the robolink stepper motor driver.
A combination is also possible.

6. CPR-CAN Motion Commands

It is recommended to use the CPR-CAN-V2 protocol because of its 32bit position value length.

6.1 Position Mode: SetJoint

The Joints are normally driven in the position mode. The controller provides the setpoint position oft he joint, and the joint electronics care about reaching this position. If it is not possible or if there are errors during motion then the active motion is aborted with the according error code.

To set a new joint value messages with the first byte 0x04 are used:

Message ID: board id
Protocol: Command Velocity posH posL timeStamp digitalout
Command: 0x04
Velocity: not used
Position: 16 bit unsigned int position value. Zero position is 32000.
TimeStamp: Arbitrary number, the module will copy this code in the answer
Example: 0x20 - 0x04 0x80 0x7D 0x00 0x51 0x02
This command sets joint 0x20 to position 0x7d00 (zero position). The second digital output is set to high. The time stamp of the message is 0x51.

The boards answer provides information on the actual position, the moment and motor current. The message ID used is ID+1
Protocol: ErrorCode Velocitiy posH posL shunt timeStamp divValue digitalInputs
Example: 0x21 - 0x04 0x7D 0x00 0x51 0xF1 0x00 0x00 0x00
This answer means that joint 0x20 is not active ("motor not enabled" and the current position is 0x7d00. The time stamp of the command was 0x51.
The contents of the last two bytes depends on the implementation. They can contain the motor current, the measured joint torque or digital input / output values.

6.2 Velocity Mode: SetVelocity

It is also possible to move the joints in velocity mode, but this is only recommended for servicing the arm. The answer from the module is the same as in 3.1. To set a velocity to the motor the first byte 0x05 is used:

Message ID: board id
Protocol: Command Velocity TimeStamp
Command: 0x05
Velocity: Velocity value from -max (0) to +max (255). 127 is zero.
TimeStamp: Arbitrary number, the module will answer with this code
Example: 0x20 - 0x05 0x90 0x51
The boards answer follows the SetJoints answer.

7. CPR-CAN-V2 Motion Commands

7.1 Position Mode: SetJoint

The Joints are normally driven in the position mode. The controller provides the setpoint position oft he joint, and the joint electronics care about reaching this position. If it is not possible or if there are errors during motion then the active motion is aborted with the according error code.

To set a new joint value messages with the first byte 0x14 are used:

Message ID: board id
Protocol: Command Velocity pos0 pos1 pos2 pos3 timeStamp digitalout
Command: 0x14
Velocity: not used
Position: 32 bit signed long position value. Pos3 is the least important byte, pos0 the most important byte.
TimeStamp: Arbitrary number, the module will copy this code in the answer
Example: 0x20 - 0x14 0x04 0x00 0x00 0x83 0xF1 0x51 0x02
This command sets joint 0x20 to position 0x000083F1. The second digital output is set to high. The time stamp of the message is 0x51.

The boards answer provides information on the actual position, the moment and motor current. The message ID used is ID+1

Protocol: ErrorCode pos0 pos1 pos2 pos3 timeStamp shunt digitalInputs
Example: 0x21 - 0x04 0x00 0x00 0x83 0xF1 0x51 0x00 0x00
This answer means that joint 0x20 is not active ("motor not enabled" and the current position is 33777. The time stamp of the command was 0x51.
The contents of the last two bytes depends on the implementation. They can contain the motor current, the measured joint torque or digital input / output / referencing values.

7.2 Velocity Mode: SetVelocity

It is also possible to move the joints in velocity mode, but this is only recommended for servicing the arm. The answer from the module is the same as in 3.1. To set a velocity to the motor the first byte 0x15 is used:

Message ID: board id
Protocol: Command Velocity TimeStamp
Command: 0x15
Velocity: Velocity value from -max (0) to +max (255). 127 is zero.
TimeStamp: Arbitrary number, the module will answer with this code
Example: 0x20 - 0x15 0x90 0x51#
The boards answer follows the SetJoints answer.

8. Digital Input / Output

Depending on the robot type digital inputs and outputs are available. The Mover robots in the current version provide 3 digital inputs via optocoupler (12 – 24V in) and 4 relay outputs on the base module 0x10, and two digital outputs (TTL level) on modules 0x40 to operate the gripper.

As standard method the digital output command is part of the SetJoint CAN message. Only in older Mover robots (until 2014, with a SetJoint-message length of 5 byte) separate commands are used.

8.1 Standard Method

Byte 6 of the SetJoint message contains the digital out value, see section 6.1:

Protocol: CANID - Command Velocity posH posL timeStamp digitalout
Example: 0x10 - 0x04 0x80 0x7D 0x00 0x51 0x02

This example sets digital out 2 on module 0x10. This protocol / example shows the CPR-CAN commands, CPR-CAN-V2 works accordingly.

The base digital outputs are 1, 2, 3, 4 on module 0x10, the gripper outputs are 1, 2 on module 0x40.
The byte value is binary coded, to set outputs 2 and 3 the value has to be 0x06.
The digital outputs are only set if the module is in 'no error' statue, that means the first by of the answer to the SetJoint message has to be 0x00. If there is an error all digital outputs are set to zero.

To operate the gripper the digital ouputs 1, 2 of module 0x40 have to be set. Byte 5 of the SetJoint message has to be set to:

Not enabled:	0x00
Enabled, closed:	0x02
Enabled, open:	0x03

8.2 Separate DOut Command

This method is valid for Mover robots build until 2014. The SetJoints message has to be send without the digital out byte. The digital outputs are set by a separate command. The following CAN message switches the first d-out on joint 4:

Message ID: 0x40 Length: 3 Data: 0x01 0x20 0x01
Explanation:
Data byte 1: 0x01 - Command byte
Data byte 2: 0x20 - First channel (0x21 for the second, 0x22 for the third, 0x23 for the fourth channel)
Data byte 3: 0x01 - Switch on (0x00 switches off)

The joint has to be enabled for this command to take effect. The length must be 3, otherwise the joint controller will ignore the message.

To open the gripper the following commands are necessary:

Activate the gripper: ID 0x40 Length 3 Data 0x01 0x21 0x01
Open the gripper: ID 0x40 Length 3 Data 0x01 0x20 0x01
Close the gripper: ID 0x40 Length 3 Data 0x01 0x20 0x00
The activation needs only to be done once after enabling, and again when a re-enable is necessary.

8.3 Digital Inputs and Outputs with separate Boards

The control for igus robolink and drylin robots provides separate digital input / output boards with seven inputs and seven outputs. Up to three boards can be connected.

These boards are connected as separate CAN nodes to the bus. The first one normally starts with the CAN ID 0x70, depending on the switch on the module.

The modules need constant CAN communication with the position profile (command 0x04 or 0x14), then the digital outputs are part of the message to the board (byte 8), and the digital inputs are part of the answer (byte 8).

9. Error Codes

If the single LED on the controller board is not blinking, then the board does not have power.

The error byte on the CAN answers provide more detailed status information:

Error	Bit in error byte	Meaning	Possible action
Brown Out or Watch Dog	Bit 1	Microcontroller restarted after a brown out. Supply voltage was too low or µC got stuck.	Increase stability of supply voltage. Reset errors.
Velocity Lag	Bit 2	Velocity changes too fast	Reset errors, enable again. Slower acceleration.
Motor not enabled	Bit 3	Not an error. Motor needs to be enabled by explicit command	Enable motor when appropriate.
Comm Watch Dog	Bit 4	Interval without command was too long	Provide the position or velocity commands in a reliable and short enough time interval. Increase maxMissedCom.
Position Lag	Bit 5	Position is too far away from the setpoint position	Provide setpoint positions reachable to the current motor position. Increase maxLag.
Encoder Error	Bit 6	The sequence of the quadrature encoder pulses did not fit.	Check connection cable motor – motor controller
Over Current	Bit 7	Current value too high	Decrease applied load on motor. Increase maxCurrent.
CAN Error	Bit8	CAN error occured	CAN bus to crowded? All connectrions ok?

The error are bit-coded, e.g. an error 0x44 means "Overcurrent" and "Motor not enabled"
After reset the error code is 0x04 "Motor not enabled"
After enabling the motor the error code is 0x00. Only with this code the motor is enabled to move.

10. Additional Messages

On Startup the module sends a message on board ID+2:

```
0x01 0x02  0x03 0x04  0x00 0x00 0x00 0x00
```

Only on some boards: Some commands are answered with acknowledge messages on ID+2:

```
0x06 0x00  func1H func1L  func2H func2L  0x00 0x00
```

Only on some boards: In some cases error messages are send on ID+2:

```
0x07 0x00  err1H err1L  err2H  err2L  err3H  err3L
```

11. Commands

Command and paramter messages are identical for both protocols.

Commands are CAN messages that change the state of the controller board, e.g. to reset error.
It is important to send the commands with the correct length, otherwise the board controller will ignore them!

Command	CAN Command	Comments
Reset Error	0x01 0x06	Sets Error Code to 0x04 (Motor not enabled) Length has to be 2. Acknowledge message 0x0106 0x001 is sent.
SetZero Position	0x01 0x08	Send posH in byte 3, posL in byte 4 Typical: 0x01 0x08 0x00 0x00 Currently the provided position data are not used, the joint is set to zero (0x7D00 for CPR-CAN, 0x0000 for CPR-CAN-V2) To ensure data integrity this command has to be send twice within the time of 50 ms to take effect. Length has to be 4. On some boards: Two acknowledge message are sent: 0x0208 0x001 after the first step, 0x0208 0x0002 after the successful second step.
StartReferencing (only for some boards)	0x01 0x0B	To ensure data integrity this command has to be send twice within the time of 50 ms to take effect. Length has to be 2. On some boards: Two acknowledge message are sent: 0x010B 0x001 when the message arrived, 0x020B 0x0001 after the first message and after 0x020B 0x0002 the second message.
Enable Motor	0x01 0x09	Also resets errors Length has to be 2. On some boards: Acknowledge message 0x0109 0x001 is sent.
Disable Motor	0x01 0x0A	Length has to be 2. On some boards: Acknowledge message 0x010A 0x001 is sent.
Set parameter	0x02 …	See following table for parameter and standard values
Get Parameter	0x03 0x50	Answer: 8 bytes with 0x50, maxMissedCom-H, maxMissedCom-L, maxLag-H, maxLag-L, maxCurrent, maxLagVel-H, maxLagVel
	0x03 0x51	Answer: parameter of the position control loop 0x51, (P*1000)-H, (P*1000)-L, (I*10000)-H, (I*10000)-L, (D*1000)-H, (‘D*1000)-L, antiWindUp
	0x03 0x52	Answer: parameter of the velocity control loop 0x51, (P*1000)-H, (P*1000)-L, (I*10000)-H, (I*10000)-L, (D*1000)-H, (‘D*1000)-L, antiWindUp
	0x03 0x54	Answer: Working hours. 0x54, workinghours-H, workinghours-L, workingMinutes, workingSeconds, firmwareVersion1, firmwareVersion2, 0x00
	0x03 0x55	Answer: supply voltage (when implemented in hardware) 0x55, battery-H, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
	0x03 0x59	Answer: Flags 0x59, flagActiveStop, flagRollOver, flagSwapEncoder, flagHerkulex, debugParameter, flagDebugMessages, ticScale
Set Joint Pos	0x14	See section Position Control
Set Velocity	0x05	See section Velocity Control

12. Parameter

The Operation can be adopted by a variety of parameters listed below. These parameters can be set using the 0x02 command, see section before. An exemplary command can be to set the maximal current is: 0x02 0x32 0x70 sets the allowed current to 127.

The parameter are set for the board. A two motor board does have only one set of parameter.
Some parameter changes require a reboot of the board to take place.

Parameter	CAN Command	Standard Value	Comment
maxMissedCom	0x02 0x30 data-H data-L	1000	Number of cycles without incoming CAN message before COM error When value is 0 then this test is switched of. Value ist saved in EEPROM
maxLag	0x02 0x31 data-H data-L	1200	Allowed distance between current position and setpoint position in encoder tics. When value is 0 then this test is switched of. Value ist saved in EEPROM
maxCurrent	0x02 0x32 data 0x00	0x80	0-255. Length has to be 4. When value is 0 then this test is switched of. Value ist saved in EEPROM
Position control – proportional	0x02 0x40 pos-H pos-L	0.1	PPID_P = (256posH + posL) / 1000 Value ist saved in EEPROM
Position control – integral	0x02 0x41 data-H data-L	0.0	PPID_I = (256posH + posL) / 10000 Value ist saved in EEPROM
Position control – differential	0x02 0x42 data-H data-L	0.0	PPID_D = (256posH + posL) / 1000 Value ist saved in EEPROM
Position control – anti-windUp	0x02 0x43 data 0x00	60	Value ist saved in EEPROM
Velocity control – proportional	0x02 0x44 pos-H pos-L	0.2	VPID_P = (256posH + posL) / 1000 Value ist saved in EEPROM
Velocity control – integral	0x02 0x45 data-H data-L	0.0	VPID_I = (256posH + posL) / 10000 Value ist saved in EEPROM
Velocity control – differential	0x02 0x46 data-H data-L	0.0	VPID_D = (256posH + posL) / 1000 Value ist saved in EEPROM
Velocity control – anti-windUp	0x02 0x47 data	30	Value ist saved in EEPROM
Rollover flag	0x02 0x66 0x01 0x9B	0	Value ist saved in EEPROM
Tic scaling factor Scales the encoder tics before CAN communication	0x02 0x69 ticScale 0x9E	1	Value ist saved in EEPROM