PiezoMotor

PiezoMotor Microstep Driver 206/236

Technical Manual

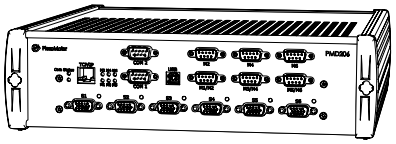
Imprint

Revision 02
05.25.2013
Document number 150087-02

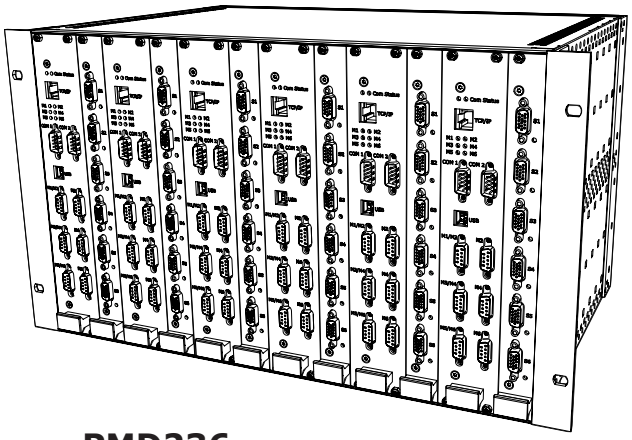
Overview

This document is a guide to the technical aspects of installing and using the ***PiezoMotor Microstep Driver 206*** and the ***PiezoMotor Microstep Driver 236*** (hereafter referred to as PMD206 and PMD236). The PMD206 is an advanced 6-axis driver for *Piezo LEGS* linear and rotary motors from PiezoMotor. The PMD236 is basically six PMD206's packaged in a rack, forming a 36-axis driver. The drivers will give the motors resolution down in the sub-nanometer/ sub-microradian range by dividing every full wfm-step into small and precise microsteps.

Detailed information about the different Piezo LEGS motors can be found on our homepage:
www.piezomotor.com



PMD206



PMD236

Contents

1 – Important Information

1.1 Symbols Used in this Document 1
1.2 Safety Instructions 2

2 – Delivery

2.1 Packaging and Delivery Contents 3

3 – Functional Principle

3.1 Piezo LEGS Linear and Rotary Motors 5
3.2 Motor Drive Principle 5
3.3 PiezoMotor Microstepping Driver 6
3.4 Waveforms and Voltages 6

4 – Quick Start

4.1 Getting Started 7
4.2 Installation 8
- 4.2.1 Cooling
- 4.2.2 Power Supply
- 4.2.3 Host Communication
- 4.2.4 Motor Connection
- 4.2.5 Sensor Connection
- 4.2.6 Example of System Configuration

5 – Detailed Technical Description

6 – Commands

6.1 Introduction 13
6.2 Power On 13
6.3 Host Communication 14
6.4 Basic Settings 15
6.5 Controller Parameter Settings 16
6.6 Sensor Board Parameter Settings 18
6.7 Run Commands 20
6.8 Controller Status 22
6.9 Error Codes 24
6.10 Firmware Version 24

7 – Installation and Assembly

7.1 Installation 25
7.2 Main Dimensions 25
- 7.2.1 Main Dimensions PMD206
- 7.2.2 Main Dimensions PMD236

7.3 Motor and Sensor Connectors 26
- 7.3.1 Pinout on the Motor Ports
- 7.3.2 Pinout on the Sensor Ports

7.4 Communication 28
- 7.4.1 Serial Communication
- 7.4.2 Communication via TCP/IP

7.5 Miscellaneous 29
- 7.5.1 USB Port
- 7.5.2 Power Supply

8 – Operation

8.1 Driving a Piezo LEGS Motor 31
8.2 Closed Loop 31
8.3 Open Loop 32
8.4 Homing 32

9 – Settings

9.1 Closed Loop Target Mode Parameters 33
- 9.1.1 StepsPerCount Parameter
- 9.1.2 Target Mode Position Limits A and B
- 9.1.3 Target Mode Minimum and Maximum Speed
- 9.1.4 Target Mode Speed Ramp Up
- 9.1.5 Target Mode Speed Ramp Down
- 9.1.6 Target Mode Stop Range
- 9.1.7 Encoder Direction

9.2 Other Settings 35
- 9.2.1 Parking and Initialization
- 9.2.2 Encoder Type
- 9.2.3 External Limits
- 9.2.4 Position Offset

10 – Maintenance

10.1 General Maintenance Instructions 37
10.2 Troubleshooting 37
10.3 Firmware Updates 37

11 – Warranty

11.1 Warranty Conditions 39

1 – Important Information

1.1 Symbols Used in this Document

Caution!

This pictogram with the wording “Caution!” indicates an imminent danger which can result in slight physical injuries or material damage.

►This arrow points out the appropriate precautions.

Regulation!

This pictogram with the wording “Regulation!” indicates a statutory regulation, guideline or directive which must be observed in respective context of the text.

Note!

This pictogram with the wording “Note” provides tips and recommendations for use and handling of the component.

1.2 Safety Instructions

Note!

The PMD206 / PMD236 driver is a high-end product intended for use with PiezoMotor's *Piezo LEGS* product line. In order to get best performance and reliability it is important that the driver unit is handled according to the instructions given in this manual and other delivery documents.

Caution!

The piezoceramic elements in a Piezo LEGS motor act as capacitors and can sometimes hold substantial electrical charge.

► Make sure motors are discharged through suitable discharge resistors.

Caution!

Incorrect installation using improper mounting materials or methods can cause damage to the driver unit.

► Observe the installation instructions.

Caution!

Depending on its use the driver unit can get very hot.

► The unit should be installed in a clean and dry environment with access to proper ventilation. On installation, ensure that air can flow around the unit without obstruction. The driver is intended for indoor operation.

Caution!

Electrostatic discharges at the driver unit connectors can cause irreparable damage to the electronics.

► Note and follow the ESD protective measures

Caution!

Incorrect connection of motor leads may cause irreparable damage to both motor and driver unit electronics.

► Connect in accordance with the specified pin assignment (see section 7.3 on page 26).

2 – Delivery

2.1 Packaging and Delivery Contents

The PMD206 driver unit has been cleaned prior to packaging, but does not carry any clean room classifications. The unit is shipped in a cardboard box with foam packaging material.

These items are included in the box at delivery:

1 x PMD206 driver unit

1 x 2 meter power cord with CEE 7/7 plug

1 x CD with *PiezoMotor Motion System* software and pdf-version of this document (latest version of software and manual is available on the PiezoMotor homepage)

3 – Functional Principle

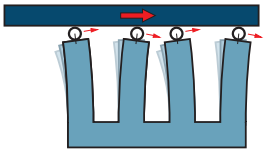
3.1 Piezo LEGS Linear and Rotary Motors

The PMD206 / PMD236 driver unit is made to control PiezoMotor's linear and rotary Piezo LEGS motors. The Piezo LEGS technology is characterized by its outstanding precision and high resolution. The motors are backlash free because of the friction based direct drive. There is no need for gearboxes or linear screws, and the motor is very strong compared to its size.

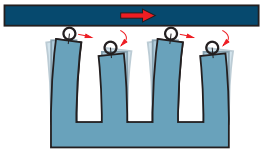
The performance of a Piezo LEGS motor is different from that of a DC or stepper motor in several aspects. A Piezo LEGS motor is friction based, meaning motion is transferred through contact friction between the drive leg and the drive rod (Piezo LEGS Linear), or between the drive leg and drive disc (Piezo LEGS Rotary). You cannot rely on each step being equal to the next. This is especially true if the motor is operated under varying loads/torques.

3.2 Motor Drive Principle

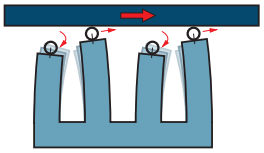
You need to feed 4 electrical drive signals to the motor in order to activate the legs and start moving. The actuation of the legs will lift and push the drive rod forward, or in case of a rotary motor, lift and spin the drive disc and axle around. For each waveform signal period the legs will complete one full step (see illustration below). To alter the speed of the motor the frequency of the electrical signals is changed. The motor speed will depend on the magnitude of external force/torque. The maximum speed is limited to a few thousand steps per second, which translates to movements in the range of millimeters per second (linear motion).



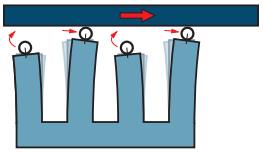
1 *When all legs are electrically activated they are elongated and bending. As we shall see below, alternate legs move as pairs. Arrows show the direction of motion of the tip of each leg.*



2 *The first pair of legs maintains contact with the rod and moves towards the right. The second pair retracts and their tips begin to move left.*



3 *The second pair of legs has now extended and repositioned in contact with the rod. Their tips begin moving right. The first pair retracts and their tips begin to move left.*



4 *The second pair of legs has moved right. The first pair begins to elongate and move up towards the rod.*

Definition

One ***waveform-step*** (***wfm-step***) is the step taken by the drive legs for each waveform signal period.

Note!

The wfm-step length/angle is not constant. Step length/angle will depend on external force/torque, and therefore the drive rod/disc will not always travel the same distance for one wfm-step.

Example

Piezo LEGS Linear: When taking a wfm-step the drive rod will move several micrometers.
Piezo LEGS Rotary: When taking a wfm-step the drive disc will rotate. The step angle depends on the diameter of the drive disc.

If there is an external force/torque present and the force/torque is opposing the direction of motion, the wfm-step length/angle will be decreased (shorter step). If the force/torque is working in the same direction as the direction of motion, the wfm-step length/angle will be increased (longer step). The speed of a Piezo LEGS motor is a product of the actual wfm-step length/angle (long or short) and the frequency of the drive signals. For varying loads/torques, using constant waveform frequency will not give a constant speed, simply because the wfm-step length is not constant.

To exemplify, the Piezo LEGS Linear 20N motor has a typical step length of ~5 µm when there is no external load present. Running 2000 wfm-steps per second will generate a speed of ~5 µm • 2000 Hz = ~10 mm/s. When the motor is loaded with 10 N of force opposing the direction of motion, the wfm-step length is decreased to ~4 µm, and resulting speed when driving at 2 kHz is of course ~4 µm • 2000 Hz = ~8 mm/s. When the motor is loaded with 10 N of force in the same direction as the direction of motion, the wfm-step length is increased to ~6 µm. Resulting speed for 2 kHz drive frequency is ~6 µm • 2000 Hz = ~12 mm/s. To conclude, with an unbalanced load the resulting speed will be different for forward and reverse motion.

3.3 PiezoMotor Microstepping Driver

The PMD206 / PMD236 is a high resolution microstepping driver made for Piezo LEGS linear and rotary motors. The driver generates suitable waveforms to control six individual motors. Each waveform signal is constituted of numerous predefined discrete points. These points are voltage targets occurring at specific times along the waveform. When going from one point to the next, the legs of the motor are bent just a little bit, and the drive rod or drive disc is moved by so called *microsteps*. The PMD206 / PMD236 driver has a resolution of 8192 microsteps per 1 wfm-step. With typical wfm-step length of ~5 µm (at no load), the microstep length at maximum resolution is ~5 µm / 8192 = ~0.6 nm.

Definition

One ***microstep*** is a small fraction of a full wfm-step. The theoretical hardware limit is 65536 microsteps within 1 wfm-step, however with the currently implemented firmware code the actual resolution is limited to 8192 microsteps/wfm-step.

Note!
The microstep length is not constant. Microstep length will depend on external forces/torques.

Note!
The resolution is currently limited to 8192 microsteps/wfm-step, but the *Run Steps* command (RS=) operate according to the microstep theoretical range of 65536 microsteps/wfm-step. Therefore, you always need to take at least 8 microsteps (i.e. 65536/8192=8) to make an actual movement.

Example

Piezo LEGS Linear:	The drive rod will not always travel the same distance for one microstep. Microsteps are close to 0.6 nm (with a resolution of 8192 microsteps/wfm-step).
Piezo LEGS Rotary:	The drive disc will not always rotate the same distance for one microstep. The angular rotation of the drive axle for one microstep will depend on diameter of the drive disc.

3.4 Waveforms and Voltages

The driver unit delivers approximately 44 V_{PP} drive signals with optimal waveforms to operate the Piezo LEGS motors. Each repeating waveform signal period is constituted of numerous predefined discrete voltages. The maximum resolution of the PMD206 / PMD236 is 8192 target points per signal period, giving the Piezo LEGS motor resolution in the sub-nanometer range (linear motion).

The microstep function means the motor can stop anywhere in the motion, with sub-nanometer resolution, and hold position. When stopped on a specific microstep, the driver must be powered and the motor can of course not be disconnected. If you want to power down in a controlled manner, there is a parking command which will give little positional disruption. Still the position might change by fractions of a wfm-step (depending on external loads/torques) when going from an unparked to a parked state. When parked, the motor can be disconnected and still hold position.

4 – Quick Start

4.1 Getting Started

1. Unpack the PMD206 / PMD236 unit and verify that no damage has occurred during transportation. Place unit so that it gets sufficient cooling.

2. Run SetupMotionSystem.msi from the CD to install *PiezoMotor Motion System* software and the FTDI communication drivers. Latest version of the software can always be found online at: ***www.piezomotor.com***

3. Connect your Piezo LEGS motor(s) to the driver. If you are using a Piezo LEGS Linear Twin motor, it needs to be connected with two cables in parallel (for example using the *Twin Connect Card*).

4. If you want to be able to run motor in closed loop, you will need to connect an encoder. The PMD206 / PMD236 supports quadrature and serial SSi encoders directly connected to the internal sensor board (via connectors S1-S6 on the front panel). The PMD206 also supports some sensors connected via the serial interface (COM2), and Ethernet sensors (transferring data via TCP/IP).

5. Connect to the driver unit using RS485 or TCP/IP.

6. Connect the power cord and switch on the driver.

7. Start *PiezoMotor Motion System* software and scan for driver units. Software will detect and list all connected driver units.

8. Double click on the driver unit in the list to start wizard. The Wizard will help you configure the basic settings.

9. When the wizard is completed, the motor can be tested in open loop and/or in closed loop operation (closed loop only if an encoder is connected). The software can also be used to send commands directly to the driver in order to verify function and configure parameter settings. The software can also run command sequences with one or several drivers to perform simple automation tasks.

4.2 Installation

4.2.1 Cooling

The driver unit is cooled by convection with internal fans, and therefore needs free space for this purpose. If the driver gets overheated it will automatically stop running the motor.

4.2.2 Power Supply

The driver unit is powered with 90-264 V AC, 50/60 Hz. The PMD206 is fused at 5 A, and the PMD236 at 10 A.

4.2.3 Host Communication

The driver unit connects to a host (e.g. a PC) via serial RS485 or Ethernet. See chapter 7 – Installation and Assembly, on page 25.

4.2.4 Motor Connection

The connection to the motor is via a straight five pole cable with D-sub connectors on driver side. All Piezo LEGS twin motors need a *Twin Connect Card* in order to split the signal to the two motor connectors. Recommended motor cable is shielded with multistrand wires 0.34 mm^2 (22 AWG). For short distances a smaller cable area is possible. The shield should normally be connected to the connector housing.

Caution!

Failure to connect all motor phases, or switching phases, can permanently damage the motor. Functional wiring can be verified by measuring motor phase capacitances through the motor cable from the driver side before attempting to run motor.

4.2.5 Sensor Connection

Encoder for position feedback may be connected to the dedicated sensor port, or sensor data can be transferred to the unit via serial interface (RS422) or via Ethernet (TCP/IP). Read more about supported encoder types and connection options in chapter 7 – Installation and Assembly, on page 25.

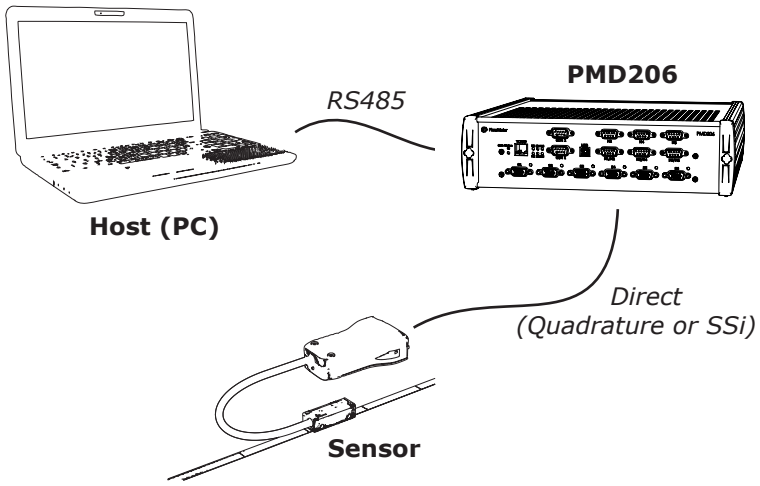
It is recommended to study the sensor manufacturer's installation guidelines.

4.2.6 Example of System Configuration

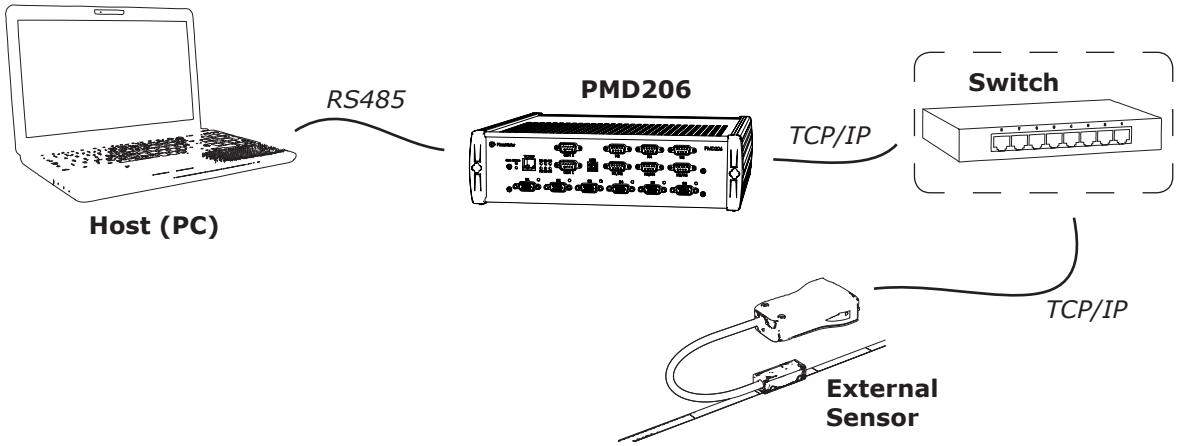
The following schematic illustrations show different ways to establish host and sensor communication. The motor is of course always connected to one of the motor ports on the driver front panel (and is not included in these illustrations). The illustrations show:

1. Host connection via RS485, and sensor (quadrature or SSi) connected directly to the internal sensor board.
2. Host connection via RS485, and external sensor via TCP/IP.
3. Host connection via RS485, and external sensor via RS422.
4. Host connection via TCP/IP, and sensor (quadrature or SSi) connected directly to the internal sensor board.
5. Host connection via TCP/IP, and external sensor via TCP/IP.
6. Host connection via TCP/IP, and external sensor via RS422.

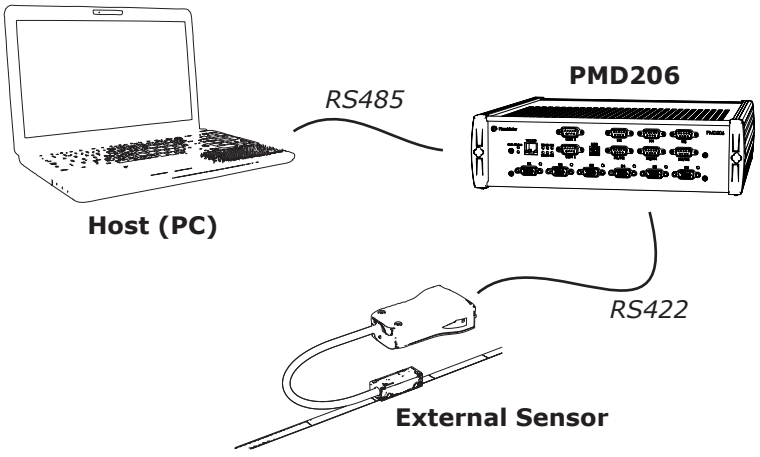
1


PMD206
RS485
Host (PC)
Direct
(Quadrature or SSi)
Sensor

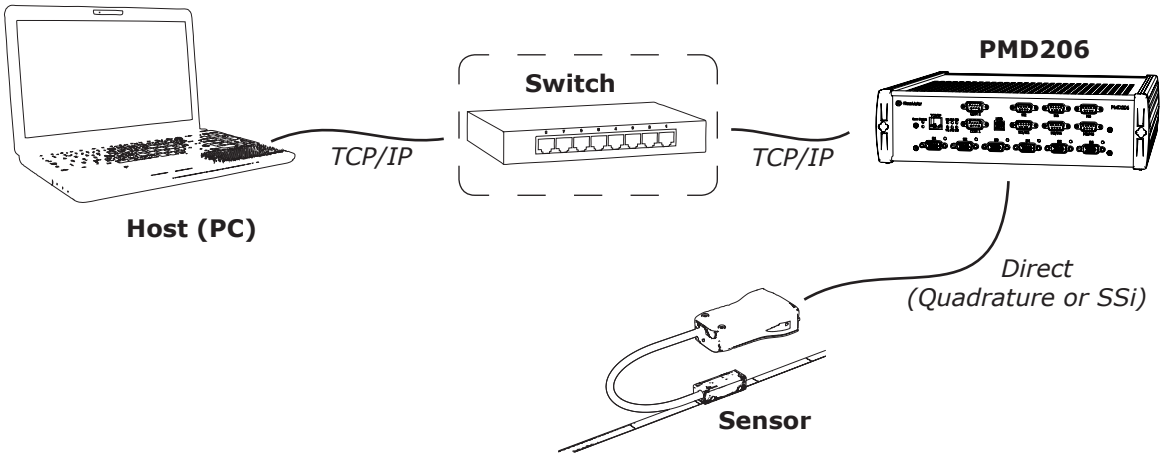
2


Switch
PMD206
RS485
TCP/IP
Host (PC)
TCP/IP
External
Sensor

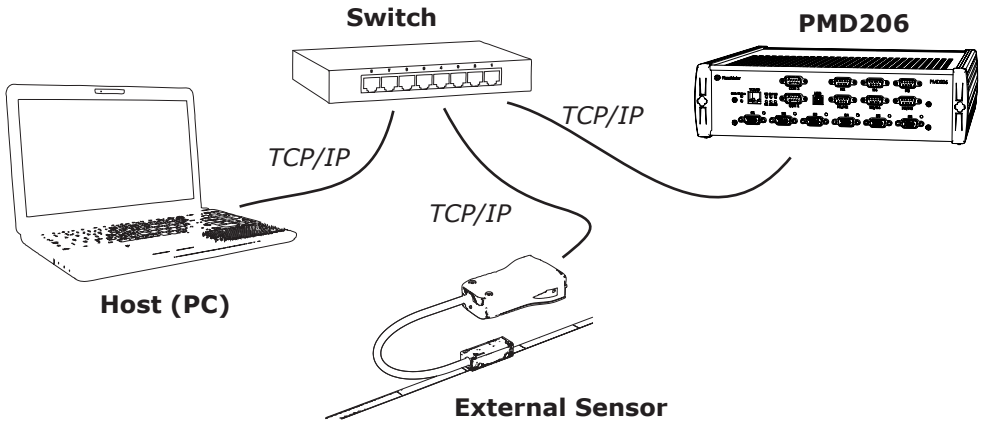
3


PMD206
RS485
Host (PC)
RS422
External Sensor

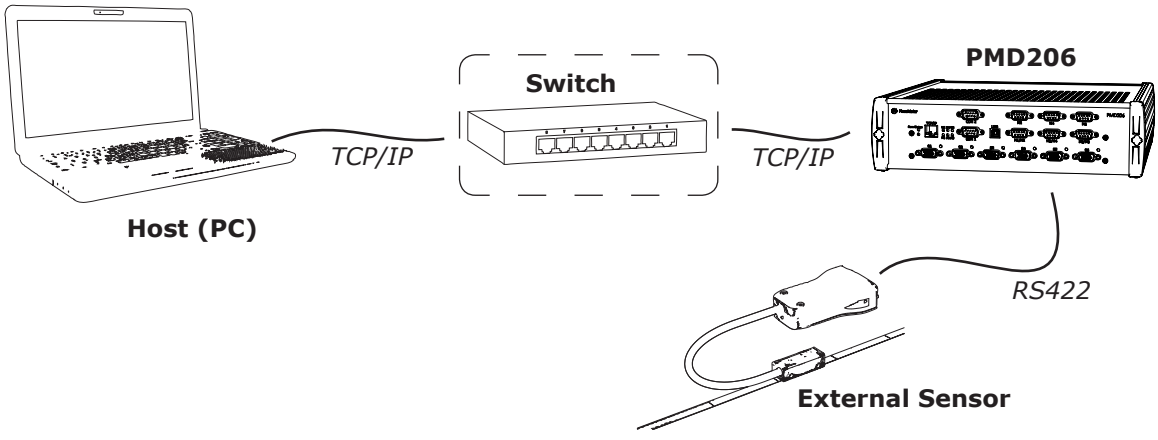
4


Switch
PMD206
TCP/IP
TCP/IP
Host (PC)
Direct
(Quadrature or SSi)
Sensor

5


Switch
PMD206
TCP/IP
TCP/IP
TCP/IP
Host (PC)
External Sensor

6


Switch
PMD206
TCP/IP
TCP/IP
Host (PC)
RS422
External Sensor

5 – Detailed Technical Description

Editorial note - this chapter will be included in a later revision of this document.

6 – Commands

6.1 Introduction

Each command to the driver starts with the letters **PM** followed by one digit **<ID>**, and one digit **<axis>**. All commands shall end with 'Carriage Return' character (CR), i.e. ASCII character 13. A command time-out (300 ms) silently rejects a command and sets an internal error code. Unknown header will be silently rejected. A matching header, but with wrong command or syntax, will return an error code. All characters are given in ASCII format. All values must always be given in hexadecimal form, (lower case 0...f). For example, if the returned value is 10001, the value is 0x10001=65537. The data is not signed, e.g. ffffffff = -1.

The PMD206 is by default identified by **<ID>** = 1. The user may assign a different identifier to the unit if necessary. The driver card has six axes, each addressed separately with a second digit **<axis>**, 1...6. Sending one command simultaneously to all axis on the PMD206 is possible by addressing axis 0 (broadcast).

The PMD236 rack has 6 separate driver modules, each with 6 individual axes. Each driver module has a unique identifier, or **<ID>**, which is related to the slot number on the common backplane. The identifier contains one digit 1...6. A second digit, **<axis>**, specifies the axis on the separate module (digits 1...6). Sending one command simultaneous to all axis on the same driver module can be done by addressing axis 0 (broadcast mode). There is no way to control all six driver modules at once. Each driver module needs to be controlled separately.

Examples of header strings

PM51 where 5 is **<ID>**, i.e. driver module with identifier 5,
and 1 is **<axis>**, i.e. axis number 1 on the addressed driver module.

PM20 where 2 is **<ID>**, i.e. driver module with identifier 2,
and 0 is **<axis>**, i.e. broadcast command to all broadcast enabled axis on the addressed driver module.

Examples of valid commands

PM12TP=f2 addressing axis 2 on module 1, and setting the target position to 242 (i.e. 0xf2).

PM60TP=f2 addressing all broadcast enabled axes (0) on module 6, and setting the target position to 242 (i.e. 0xf2)

Note!
In the following text all examples are given only for **<ID>** = 1. The header needs to be changed for addressing driver modules with other identifier.

Note!
All set commands sent to the driver will be echoed back with an identical string (if accepted).

6.2 Power On

At power on, controller parameter settings (CP2...b), encoder type setting (SB1), and latest TCP/IP settings are restored to values stored in non volatile memory (Flash/EEPROM). Controller mode (CM) is set to target mode enabled and motors are parked. If the controller parameters or the encoder type setting are changed during operation, be sure to save to Flash/EEPROM using the CC command.

6.3 Host Communication

Communication with the driver is via serial line (4-wire RS485), or via Ethernet (TCP/IP socket). Several PMD206 units may be connected on the same RS485 line, provided that they have different identifier **<ID>** and that responses are allowed to finish before addressing the next module. These are the settings for serial communication:

RS485 Communication Settings							
Baud Rate	**Start Bit**	**Data Bits**	**Parity**	**Stop Bits**	**Handshaking**	**Duplex**	**Text**
115200	1	8	none	1	none	Full duplex	ASCII

By default the driver is set to dynamic IP allocation (DHCP), so when connecting via Ethernet you will need to look for the driver module(s) in your local area network. You can assign a static IP address to the driver module(s) using the commands in the table below. Setting IP/GW/IM will immediately save the new settings to Flash/EEPROM.

TCP/IP Settings					
	Cmd	**Description**	**Examples**		**Notes**
IP Settings	**IP=**	Asigns a static IP address for the specific driver module, and the port number for host communication.	Static IP example: Dynamic IP example:	**PM10IP=c0,a8,0a,01,2620** Corresponding to IP address 192.68.10.1 and port 9760 **PM10IP=00,00,00,00,0000**	If the port number is set to **0000**, the default socket port number 9760 will be assigned, i.e. 0x2620. Static IP address is not valid until a gateway and a mask has been assigned (using GW and IM commands). If you use dynamic IP, the DHCP server will provide the IP mask and gateway. Use XV? command for current IP mode.
	IP?	This command returns the currently used IP address and port number of the driver module (hex values).	Command: *Response:*	**PM10IP?** *For example* ***PM10IP?:c0,a8,0a,01,2620***	
Gateway Settings	**GW=**	Sets the gateway address to be used for static IP address.	Example:	**PM10GW=c0,a8,0a,0a** Corresponding to gateway address 192.68.10.10	Should not be necessary to set when host is on the same subnet.
	GW?	This command returns the currently used gateway address (hex values).	Command: *Response:*	**PM10GW?** *For example* ***PM10GW?:c0,a8,0a,0a***	Will return the gateway for static IP, not for DHCP.
Mask Settings	**IM=**	Sets the mask address to be used for static IP address.	Example:	**PM10IM=ff,ff,ff,00** Corresponding to IP mask address 255.255.255.0	
	IM?	This command returns the currently used mask address (hex values).	Command: *Response:*	**PM10IM?** *For example* ***PM10IM?:ff,ff,ff,00***	Will return the mask for static IP, not for DHCP.

6.4 Basic Settings

The basic settings defines how the driver will interact in the system. For example, you need to enable closed loop operation, and assign the sensor interface (e.g. an encoder connected to the internal sensor board, or data collection from an external sensor via RS422 or Ethernet).

Basic Settings						
	Cmd	**Description**	**Examples**			**Notes**
Controller Mode	**CM=**	Target mode enable	Example:	**PM10CM=0** **PM10CM=1**	Target mode disabled Target mode enabled	When target mode is disabled (**0**) you will not be able to run closed loop run commands (***TP*** and ***TR***)
	CM?	Read currently used controller mode.	Command: *Response:*	**PM10CM?** *For example **PM10CM?:01** when target mode is enabled*		Response will be **2** when running homing command.
Broadcast Enable	**CE=**	Each axis on the unit can be enabled or disabled for broadcast run commands (TP, TR and RS).	Example:	**PM10CE=1,1,1,1,1,1**		**0** for broadcast disabled **1** for broadcast enabled The enable/disable function only applies for run commands (***TP***, ***TR*** and ***RS***)
	CE?	Reads the broadcast enable settings.	Command: *Response:*	**PM10CE?** *For example, when all axis enabled **PM10CE?:01,01,01,01,01,01***		
Sensor Interface	**SI=**	Defines the sensor interface. Options: • Internal sensor board • Serial sensor (RS422) • Ethernet sensor (TCP/IP)	Internal sensor board: Serial sensor: Ethernet sensor example:	**PM10SI=00,00,00,00,0001** **PM10SI=00,00,00,00,0000** **PM10SI=c0,a8,0a,02,2711** Corresponding to IP address 192.68.10.2 and port 10001		
	SI?	Report sensor interface setting.	Command: *Response:*	**PM10SI?** *For example **PM10SI?:c0,a8,0a,02,2711***		
Slot Identifier	**ID=**	A set command to change the **<ID>** identifier.	Example: Example:	**PM10ID=7** to change the identifier from 1 to 7. **PM70ID=1** to change the identifier from 7 to 1.		Can be 0...f (default is 1). The PMD236 has fixed identifiers 1...6 which are related to the physical backplane. These cannot be changed.

6.5 Controller Parameter Settings

The PMD206 / PMD236 must be configured correctly in order to get best performance out of the Piezo LEGS motor in your application. For instance, to run in closed loop you need to have ramping parameters set according to the actual external loads in your application. Most important for closed loop operation is to have the *StepsPerCount* properly set up. The Piezo LEGS motor will perform very differently depending on external load/torque because the step length/angle is not constant. With target mode parameters properly configured, you will get the desired function in closed loop operation. See table below for explanation of each setting. More detailed information given in chapter 9.1.

Controller Parameter Settings			
Cmd	**Description**	**Examples**	**Notes**
CP=	Sets various parameters in the driver. You can set parameters for a single axis, or on all broadcast enabled axis by using broadcast command.	Send to axis 1: **PM11CP=<par>,<value>** … Send to axis 6: **PM16CP=<par>,<value>** Send broadcast: **PM10CP=<par>,<value>**	**<par>** is the parameter identifier **<value>** is the parameter value
CP?	Reads various parameters in the driver.	Send to axis 1: **PM11CP?<par>** … Send to axis 6: **PM16CP?<par>** Send broadcast: **PM10CP?<par>** *Response:* *For example* ***PM11CP?<par>:<value>***	**<par>** is the parameter identifier Broadcast only works up to parameter 1c.

<par>	**<value>**	**Notes**
0	***Cycle counter*** There is a continuous counter which keeps track of microsteps and wfm-steps. With the set command the user can set the wfm-step counter. With the read command the user can read out the total microstep counter (1 wfm-step = 65536 µsteps).	Read (U32) Write (U16)
1	***Parking and initialization*** **<value>** for set: **<value>** for read: **0** to unpark **00** - unparked (ready to run) **1** to park **01** - parked (powered down) **2** to initiate from Flash/EEPROM * **11** - parking/unparking is ongoing **3** to initiate default values ** * CP-parameters 2, 3, 4, 5, 6, 7, 8, 9, a, b, SB-parameter 1. ** CP-parameters 3, 4, 5, 6, 7, 8, 9, a, b.	Any run command will also unpark, but stop command will not park. When issuing the park command (**1**) it will take around 300 ms to complete. Save to Flash/EEPROM with ***CC=4*** command.
2	***External limits*** **0** for limit switches disabled **1** for limit switches enabled See chapter 7.3.2 for information on which pins to connect!	(U16)
3	***Position limit A*** - target mode will stop when position <A	Default value: **ffffd8f0** (I32) (decimal -10000)
4	***Position limit B*** - target mode will stop when postition >B	Default value: **2710** (I32) (decimal +10000)
5	***Stop range*** - how many counts (±) from target position should the regulation stop	Default value: **0** (U16)
6	***Encoder direction*** - to define the expected encoder counting direction for target loop **0** for positive counting when running in forward direction **1** for negative counting when running in forward direction	Default value: **0** (U16)
7	***Minimum speed in target mode*** - wfm-steps per second	Default value: **2** (U16)
8	***Maximum speed in target mode*** - wfm-steps per second	Default value: **32** (U16)
9	***Speed ramp up in target mode*** - wfm-steps per second per millisecond	Default value: **30** (U16)

Controller Parameter Settings		
<par>	**<value>**	**Notes**
a	***Speed ramp down in target mode*** - wfm-steps/second at 1 wfm-step from target	Default value: **30** (U16)
b	***StepsPerCount*** - Target mode parameter which tells the controller how many wfm-steps to run to reach target. **CP=b,<SPC>** where SPC is a integer in hexadecimal form which needs to be calculated by the user. $SPC = \frac{2^{20} \cdot \text{encoder resolution [nm/count]}}{\text{distance per wfm-step [nm/wfm-step]}} = \frac{2^{20}}{\text{encoder counts per wfm-step}}$ Example: You have a linear encoder with 20 nm resolution mounted on the motion axis, and you know that for your application the distance travelled for one wfm-step is 4 µm (4000 nm). You calculate SPC = $2^{20} \cdot 20 / 4000 = 5243 = 0x147b$, and enter **CP=b,147b**. Example: You have a linear encoder with 20 nm resolution mounted on the motion axis, but you are uncertain of the travel distance for one wfm-step. Read encoder value (**MP?**) then run at least ten wfm-steps (**RS=3e8,a0000,0**) and read encoder again (**MP?**). You get a difference of 2000 counts, i.e. averaging 200 counts per wfm-step. You calculate SPC = $2^{20} / 200 = 6991 = 0x1b4f$, and enter **CP=b,147b**. Example: You have a rotary encoder with 8192 CPR (counts per revelation), and find an average of 1 count per wfm-step. You calculate the SPC = $2^{20} / 1 = 1048576 = 0x100000$ and enter **CP=b,100000**. **Note!** The average wfm-step may not be the same in both directions because the wfm-step length/angle is load dependant. You may use the average of both directions. Since the wfm-step length is rather approximate, so is the SPC value while still obtaining a stable target loop.	Default value: **147b** (U32) $2^{20}=16 \cdot 65536$
c...f	*Not in use*	
10	***Driver board temperature*** Reads temperature on the driver card. **<value>** is 0x8830 at 25ºC and 0x2740 at 75ºC.	Read only (U16)
11	*Reserved - do not use*	
12	***Driver board 48 V level*** - where 48 V is 0x9de0 and 0 V is 0x0	Read only (U16)
13	*Reserved - do not use*	
14	***Position*** - reports the encoder value	Read only (U30)
15...1c	*Reserved - do not use*	
1d	***Wfm point and voltages*** - current microstep point and phase DAC values	Read only (multiple csv)
1e	***Target mode parameters*** - comma separated values of parameters 2, 3, 4, 5, 6, 7, 8, 9, a, and b.	Read only (multiple csv)
1f	*Reserved - do not use*	

6.6 Sensor Board Parameter Settings

The internal sensor board can be used for quadrature encoders and SSI sensors. It also provides digital in/out of which some may be used for limit switch signal inputs. has an own set of parameters which needs to be configured if you intend to connect an encoder directly to the driver. Pinout of the sensor port is described in chapter 7.3.2.

Sensor Board Parameter Settings			
Cmd	**Description**	**Examples**	**Notes**
SB=	Sets parameters for the internal sensor board.	Send to axis 1: **PM11SB=<par>,<value>** … Send to axis 6: **PM16SB=<par>,<value>** Send broadcast: **PM10SB=<par>,<value>**	**<par>** is the parameter identifier **<value>** is the parameter value
SB?	Reads parameters for the internal sensor board.	Send to axis 1: **PM11SB?<par>** … Send to axis 6: **PM16SB?<par>** Send broadcast: **PM10SB?<par>** *Response:* *For example* ***PM11SB?<par>:<value>***	**<par>** is the parameter identifier

<par>	**<value>**	**Notes**
0	***Position*** - Sensor board encoder count (without offset).	The sensor board position may be read even if the primary sensor interface is RS422 or Ethernet. ***MP*** command will give position including position offset.
1	***Encoder type*** **0** no encoder **1** for quadrature encoder **2** for quadrature encoder (inverted counting direction) **3…7** no encoder (reserved) **8…1e** for 2-wire SSi 8…30 bit, extended to 32 bit by lap counting	Default value: **0**
2	***Index mode*** To find index mark on quadrature encoder scale, i.e. when A, B and Z are all high. After entering index mode, user must drive motor in open loop using ***RS***-command to look for index. **0** off **1** on - motor stops from running when the index mark has been found **2** on - position is set to zero at index **3** on - position is set to zero at index, and motor stops **Note!** When the index position has been found the driver will exit index mode. The motor will in most cases overshoot and come to a halt a few counts away from the index position. Issue target mode run command ***TP*** if you wish to return to the exact index position.	Default value: **0**
3	***Position offset*** **<value>** is a position offset that will add to the encoder position.	Default value: **0**
4	Not in use	

Sensor Board Parameter Settings

<table>
<tr><th><par></th><th><value></th><th>Notes</th></tr>
<tr><td>5</td><td>
I/O port - set command for output pins

Set command: PM11SB=5,<value_{out}>

Example: PM11SB=5,1 to set Out 0 high, and clear Out 1 and Out 2

<table>
<tr><th>value_{out}</th><th>Pin Out 2</th><th>Pin Out 1</th><th>Pin Out 0</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>2</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>3</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>4</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>5</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>6</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>7</td><td>1</td><td>1</td><td>1</td></tr>
</table>
</td><td></td></tr>
<tr><td></td><td>
I/O port - read command for status on input and output pins

Send to axis 1: PM11SB?5
...
Send to axis 6: PM16SB?5

Send broadcast: PM10SB?5

Response: For example
PM11SB?5:<value_{out}value_{in}>

<table>
<tr><th>value_{in}</th><th>Pin In 3</th><th>Pin In 2</th><th>Pin In 1</th><th>Pin In 0</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>2</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>3</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>4</td><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>5</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>6</td><td>0</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>7</td><td>0</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>8</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>9</td><td>1</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>a</td><td>1</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>b</td><td>1</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>c</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>d</td><td>1</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>e</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>f</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table>

Example response: PM11SB?5:7f
for all outputs high (=7), and all inputs high (=f)
</td><td></td></tr>
<tr><td>6</td><td>Voltages - Reports voltage levels of the sensor board
5V, 3V3, 5V_{S1}, 5V_{S2}, 5V_{S3}, 5V_{S4}, 5V_{S5}, 5V_{S6}</td><td>Read only.
Broadcast only.</td></tr>
<tr><td>7</td><td>Firmware - Reports firmware version of sensor board.</td><td>Read only.
Broadcast only.</td></tr>
</table>

6.7 Run Commands

Run Commands				
	Cmd	**Description**	**Examples**	**Notes**
Controller Command	**CC=**	A set command to park and unpark motor, but also to initiate parameters from flash memory, or save parameters to flash memory.	Send to axis 1: **PM11CC=<value>** … Send to axis 6: **PM16CC=<value>** Send broadcast: **PM10CC=<value>**	Where **<value>** is: **0** to unpark motor **1** to park motor **2** to initiate parameters* from Flash/EEPROM **3** to initiate parameters** from factory default **4** to save parameters* to Flash/EEPROM (only broadcast) **5** to reboot (only broadcast) * CP 2, 3, 4, 5, 6, 7, 8, 9, a, b, and SB 1. ** CP 3, 4, 5, 6, 7, 8, 9, a, b.
Target Position	**TP=**	A set command to run motor in target mode (i.e. closed loop with encoder). Sets the target position for one axis or for multiple axis (broadcast).	Send to axis 1: **PM11TP=<value>** … Send to axis 6: **PM16TP=<value>** Send broadcast: **PM10TP=<value$_1$>,<value$_2$>,<value$_3$>,<value$_4$>,<value$_5$>,<value$_6$>** Example: **PM11TP=41a** will run motor on drive module 1, axis 1, to encoder count 1050 (=0x41a).	Where **<value>** is the encoder count in signed *hexadecimal* form. A broadcast disabled axis will not be affected.
	TP?	A read command that reads last set target position.	Send to axis 1: **PM11TP?** … Send to axis 6: **PM16TP?** Send broadcast: **PM10TP?** *Response:* *For example **PM11TP?:00000001***	Returened **<value>** is the encoder count in *hexadecimal* form. Broadcast response is comma separated values.
Target Relative	**TR=**	To run motor a fixed number of encoder counts (in closed loop) from the current target position.	Send to axis 1: **PM11TR=<value>** … Send to axis 6: **PM16TR=<value>** Send broadcast: **PM10TR=<value$_1$>,<value$_2$>,<value$_3$>,<value$_4$>,<value$_5$>,<value$_6$>** Example: **PM11TR=b** if the previous target position was encoder count 1050 (=0x41a), this command will move motor forward 11 counts (=0xb) relative the previous position, and the new target position will be encoder count 1061 (=0x425).	If there is no active target position, the relative value will be based on the current ***MP*** value.
	TR?	A command that reads last set relative target value.	Send to axis 1: **PM11TR?** … Send to axis 6: **PM16TR?** Send broadcast: **PM10TR?** *Response:* *For example **PM11TR?:b***	Returened **<value>** is the count value in *hexadecimal* form. Broadcast response is comma separated values.

Run Commands

	Cmd	Description	Examples	Notes
Run Step	**RS=**	This is a set command to run a number of microsteps in open loop.	Send to axis 1: **PM11RS=<freq>,<µstep>,<dir>** … Send to axis 6: **PM16RS=<freq>,<µstep>,<dir>** Send broadcast: **PM10RS=<freq>,<µstep>,<dir>** Example: **PM11RS=3e8,c0000,0** sets the wfm-step freqency to 1000 Hz (=0x3e8), and runs 12 wfm-steps (e.g. 12 • 65536 = 0xc0000 µsteps) in the forward direction (0).	**<freq>** wfm-step frequency [wfm-steps/second] **<µstep>** number of microsteps **<dir>** **0** forward / CW **1** reverse / CCW **10** forward / CW * **11** reverse / CCW * * only runs if status ***stopi*** is high The actual resolution is limited to 8192 µsteps/wfm-step so you may need to take at least 8 microsteps (i.e. 65536/8192=8) to see movement. 1 wfm-step = 65536 µsteps = 0x10000 µsteps
Controller Status	**CS=**	Stop single motor from running, or stop motors from running on all axis.	Send to axis 1: **PM11CS=0** … Send to axis 6: **PM16CS=0** Send broadcast: **PM10CS=0**	**0** to stop running motor Broadcast will stop all motors regardless of broadcast enable settings. **Note!** The *Controller Status* read command is defined in 6.8.
Motor Position	**MP?**	A command that reads the encoder value(s).	Send to axis 1: **PM11MP?** … Send to axis 6: **PM16MP?** Send broadcast: **PM10MP?** *Response:* *For example* ***PM10MP?:<value$_1$>,<value$_2$>,<value$_3$>,<value$_4$>,<value$_5$>,<value$_6$>***	Returned value(s) will be motor position including offset
Homing Command	**HO=**	A command for finding quadrature encoder index position. The homing sequence will run in two directions and stop and zero position when the index marker is found.	Send to axis 1: **PM11HO= <freq>,<µstep$_{dir1}$>,<counts$_{dir1}$>, <dir1>,<µstep$_{dir2}$>,<counts$_{dir2}$>** … Send to axis 6: **PM16HO= <freq>,<µstep$_{dir1}$>,<counts$_{dir1}$>, <dir1>,<µstep$_{dir2}$>,<counts$_{dir2}$>** Send broadcast: **PM10HO= <freq>,<µstep$_{dir1}$>,<counts$_{dir1}$>, <dir1>,<µstep$_{dir2}$>,<counts$_{dir2}$>**	**<freq>** wfm-step frequency [wfm-steps/second] **<µstep>** maximum number of microsteps in (direction 1 and 2) **<counts>** maximum number of encoder counts (direction 1 and 2) **<dir1>** first direction: **0** forward **1** reverse
	HO?	Check status of homing sequence.	Send to axis 1: **PM11HO?** … Send to axis 6: **PM16HO?** Send broadcast: **PM10HO?** *Response:* *For example* ***PM10HO?:<hh$_1$>,<hh$_2$>,<hh$_3$>,<hh$_4$>,<hh$_5$>,<hh$_6$>***	**<hh>** status of homing sequence **0c** initiating **0b** starting index mode 3 **0a** starting direction 1 **09** running direction 1 **08** starting direction 2 **07** running direction 2 **06** is encoder counting? **05** running direction 2 **04** end direction 2 **03** stopped with error **02** index not found **01** index was found **00** not started

Run Commands				
	Cmd	**Description**	**Examples**	**Notes**
Data Recorder	**DR=**	The data recorder will store 100 motor position values for all six axis.	Send broadcast: **PM10DR=0** to enable and arm auto trig **PM10DR=1** to enable and start recordings now.	When using auto trig mode (**0**), the unit will wait for ***TP***, ***TR*** or ***RS*** commands before start of recordings. The alternative (**1**) will start recording immediately.
	DR?	A read command that reports recorded data.	Send broadcast: **PM10DR?0** or **PM10DR?1** to report position data and time *Response:* ***PM10DR?0:<numLines>,<tStart>,<tStop><LF>*** ***<val_1>,<val_2>,<val_3>,<val_4>,<val_5>,<val_6><LF>*** ***[...maximum 99 lines...]<CR>***	**<numLines>** number of lines to follow **<tStart>** time for first position **<tStop>** time for last position Times given as ticks of 0.5 ms. **<$val_{1\text{-}6}$>** position value for axis 1-6

6.8 Controller Status

Controller Status				
Cmd	**Description**	**Examples**		**Notes**
CS?	Reads controller status.	Send broadcast: **PM10CS?** Response:	***PM10CS?:<nnnn>,<mm_1>,<mm_2>,<mm_3>,<mm_4>,<mm_5>,<mm_6>***	**<nnnn>** is controller status **<mm>** is motor status
XS?	Reads extended controller status, also including status of I/O port and sensor board.	Send broadcast: **PM10XS?** *Response:*	***PM10XS?:<nnnn>,<ssiimm$_1$>,<ssiimm$_2$>,<ssiimm$_3$>,<ssiimm$_4$>,<ssiimm$_5$>,<ssiimm$_6$>***	**<nnnn>** is controller status. **<ssiimm>** is sensor board, I/O port, and motor status

	Status Code	**<nnnn>**	**Description**	**Com Status LED**
Internal Com	**otherErr**	**8** x x x	Abnormal reset detected.	Orange
	picComErr	**4** x x x	ComPic cannot communicate internally with MotorPic1 or MotorPic2.	Red
	pic2respErr	**2** x x x	MotorPic2 has sent an unexpected response (internal error).	Orange
	pic1respErr	**1** x x x	MotorPic1 has sent an unexpected response (internal error).	Orange
Power Supply	**ADCErr**	x **8** x x	Error reading ADC. A voltage could not be read.	Red
	v48Err	x **4** x x	48V level low or high current draw detected.	Orange
	v5Err	x **2** x x	5V level on driver board outside limits.	Red
	v3Err	x **1** x x	3V3 level on driver board outside limits.	Red
Encoder Interface	**xboardComErr**	x x **8** x	Sensor board communication error detected.	Orange.
	sensorComErr	x x **4** x	Parity or frame error RS422 sensor UART. Check cable/termination.	Red
	sensorDataErr	x x **2** x	Wrong data format on external sensor (RS422 or TCP/IP).	Red
	sensorNoReply	x x **1** x	External sensor not detected (RS422 and TCP/IP).	Orange
Host Com	**hostComErr**	x x x **8**	Problem with UART on host. Check cable and termination.	Red
	cmdErr	x x x **4**	Host command error - driver board (e.g. buffer overrun).	Red
	cmdTimeout	x x x **2**	300 ms command timeout ocurred.	Orange
	cmdWarning	x x x **1**	Command execution gave a warning.	Orange

Controller Status					
	Status Code		**<ssiimm>**	**Description**	**S1-6 LED**
Sensor Board	**v3Err**		**8** x x x x x	3V level on sensor board outside limits.	Red
	comErr		**4** x x x x x	An internal communication error was detected.	Orange
	cmdErr		**2** x x x x x	A command error was detected - sensor board.	Orange
	v5sErr		**1** x x x x x	5 V sensor supply voltage error detected.	Red
Encoder	**encErr**		x **8** x x x x	Encoder error detected.	Red
	imode		x **4** x x x x	Index mode active (zero at index).	-
	stopi		x **2** x x x x	Index mode active (stop at index).	-
	indexDetected		x **1** x x x x	Index has been detected since last report.	-
Output Pins	**unused**		x x **8** x x x	unused = 0	-
	Out 2		x x **4** x x x	'Out 2' high	-
	Out 1		x x **2** x x x	'Out 1' high	-
	Out 0		x x **1** x x x	'Out 0' high	-
Input Pins	**In 3**		x x x **8** x x	'In 3' high	-
	In 2		x x x **4** x x	'In 2' high	-
	In 1		x x x **2** x x	'In 1' high	-
	In 0		x x x **1** x x	'In 0' high	-
	Status Code	**<mm>**	**<ssiimm>**	**Description**	**M1-6 LED**
Motor Drive Status	**DriverErr**	**8** x	x x x x **8** x	48V low or other critical error, motor is stopped.	Red
	Overheat	**4** x	x x x x **4** x	Temperature limit reached, motor is stopped.	Orange
	Parked	**2** x	x x x x **2** x	Motor is parked.	Orange
	Tlimit	**1** x	x x x x **1** x	Max or min encoder limit was reached in target mode, or motor was stopped due to external limit signal.	-
Motor Run Status	**Tmode**	x **8**	x x x x x **8**	Target mode is active.	-
	Tstop	x **4**	x x x x x **4**	Target position was reached (if target mode is active). Also set during parking/unparking.	-
	Direction	x **2**	x x x x x **2**	Motor direction.	-
	Running	x **1**	x x x x x **1**	Motor is running.	Green

6.9 Error Codes

If the driver receives a command which is not valid, the response will be an error code. The table below explains all error codes.

Example response: ***??=<errCode>,<pos>,<ASCII>,<errString>***

where **<errCode>** is the identifier according to the table below, **<pos>** is the position of the incorrect character , **<ASCII>** is the ASCII code of the incorrect character, and **<errString>** is the error description in plain text.

Error Codes			
<errCode>	**<errString>**	**Description**	**Solution**
01	BAD COMMAND	The given command does not exist	Commands and values are case sensetive
02	BAD SYNTAX	The complete command does not have a valid header or syntax	Commands and values are case sensetive
03	BAD PARAM	The parameter or parameters has wrong amount or illegal values	Check parameter values or amount of parameters
04	WRONG ID	The <ID> and <axis> does not form a valid identifier for this command	Check that the <axis> is correct for this command
05	WRONG STATE	The *Controller Mode* is not in the correct state for receiving ***TP***- or ***TR-***command (closed loop run commands)	Set *Controller Mode* to 'target mode' using the ***CM***-command
06	CMD FAILED	An internal error occured during command execution	Check error codes
07	NOT DONE	Command or parameter not implemented, or the command has been deactivated	Check that the command is correct. Some commands are read only, others set only, some parameters may be unimplemented.

6.10 Firmware Version

Firmware Version			
Cmd	**Description**	**Examples**	**Notes**
SV?	Reads the firmware version of the PMD.	Send broadcast: **PM10SV?** *Response:* ***PM10SV?:<com>,<pic$_1$>,<pic$_2$>***	Where the returned values are the firmware revision numbers of the communication controller, and the two drive control pic's: <**com**>, <**pic$_1$**>, and <**pic$_2$**>.
XV?	Reads extended version information, and also some TCP/IP related information.	Send broadcast: **PM10XV?** *Response:* ***PM10XV?:<com>,<pic$_1$>,<pic$_2$>,<sen>,<2xx>,<MAC>,<IPmode>***	**<com>**, **<pic$_1$>**, and **<pic$_2$>** as above. **<sen>** is the sensor board firmware revision number. **<2xx>** is the type of driver (206 or 236). **<MAC>** is the TCP/IP MAC address of the driver. **<IPmode>** is **00** for DHCP mode, and **01** for static IP.

7 – Installation and Assembly

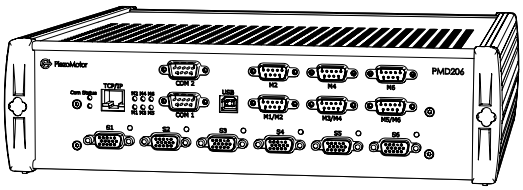
7.1 Installation

The PMD206 / PMD236 driver unit should be used in a clean and dry environment with proper ventilation. On installation, ensure that air can flow around the unit without obstruction. The PMD206 / PMD236 is intended for indoor operation.

7.2 Main Dimensions

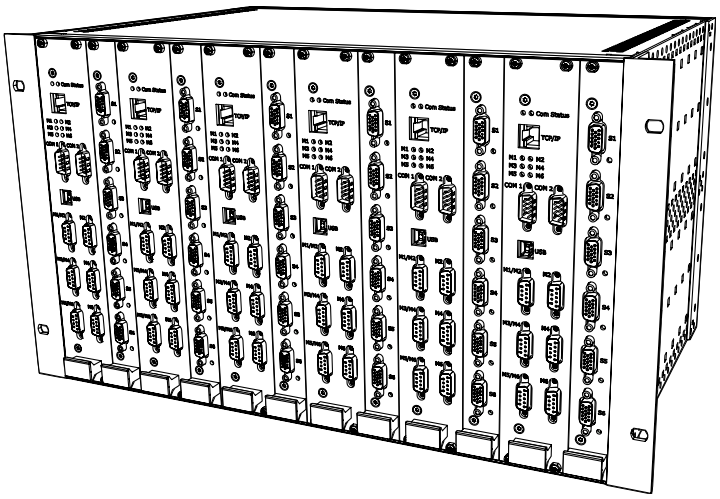
7.2.1 Main Dimensions PMD206

The standalone 6-axis driver PMD206 is packaged in a box of size 328 x 298 x 83 mm.



7.2.2 Main Dimensions PMD236

The 36-axis driver PMD236 is housed in a standard shielded 19” rack, 6HE, 84HP, 295 mm depth.



7.3 Motor and Sensor Connectors

The PMD206 / PMD236 has several ports for connecting motors and encoders, as well as I/O signals. The port pinouts are listed in the tables below.

Note!

Host communication with the PMD236 must be over 6 separate lines (Ethernet or serial), one to each driver module, or via a single serial line that is patched to the COM1 ports on each driver module.

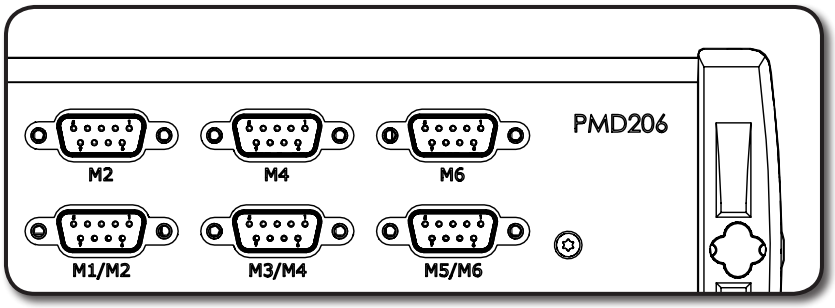
Note!

All I/O digital signals are 5 V logic and inputs have internal pull-up.

7.3.1 Pinout on the Motor Ports

The motor connectors on the driver unit are D-sub 9S, labeled **M1**...**M6**, and corresponding to axis 1...6. There is one connector for each axis, however, the user can reduce the number of cables in his system by accessing all 6 axis on the 3 double connectors. Recommended motor cable is shielded with multistrand wires 0.34 mm2 (22 AWG). For short distances a smaller cable area is possible. The shield should normally be connected to the connector housing.

MotorX/MotorY Connector	
Pin	**Description**
1	Common GND, Motor X and Y
2	Phase 4, Motor X
3	Phase 3, Motor X
4	Phase 2, Motor X
5	Phase 1, Motor X
6	Phase 4, Motor Y
7	Phase 3, Motor Y
8	Phase 2, Motor Y
9	Phase 1, Motor Y



MotorY Connector	
Pin	**Description**
1	GND, Motor Y
2	Phase 4, Motor Y
3	Phase 3, Motor Y
4	Phase 2, Motor Y
5	Phase 1, Motor Y
6	-
7	-
8	-
9	-

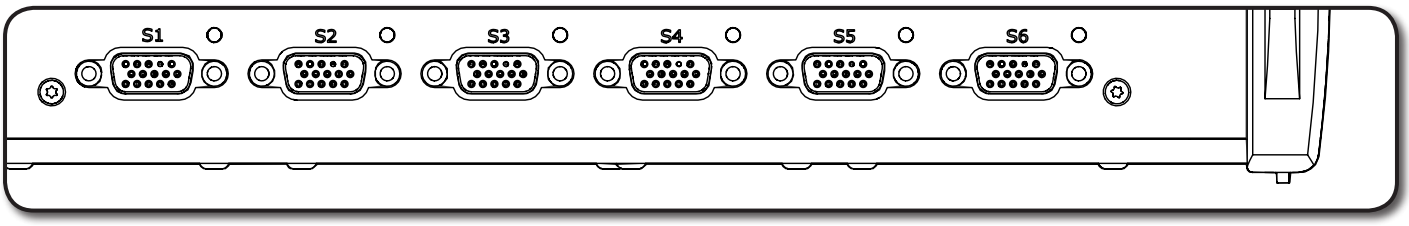
Caution!

Switching motor phases, or failure to connect all motor phases, can permanently damage the motor. Functional wiring can be verified by measuring motor phase capacitances through the motor cable from the driver side before attempting to run motor.

7.3.2 Pinout on the Sensor Ports

The sensor connectors on the driver unit are D-sub HD 15S, labeled **S1**...**S6**, and corresponding to axis 1...6. The internal sensor board needs to be activated using the ***SI***-command (see section 6.4 on page 15). The sensor type connected to the internal sensor board also needs to be defined, using the ***SB***-command (see section 6.6 on page 18). The table below defines the pinout depending on chosen sensor type.

SensorX Connector			
Pin	**Type**	**Quadrature Sensor**	**Serial Sensor**
1	Input	In 2 / Limit A	In 2 / Limit A
2	Output	Out 2	Out 2
3	-	Ground (GND)	Ground (GND)
4	Input	Quadrature Index Z+	Serial Data In+
5	Input	Quadrature Signal A-	Serial Clock-
6	Input	In 3 / Limit B	In 3 / Limit B
7	Input	In 0	In 0
8	Output	Out 0	Out 0
9	Input	Quadrature Index Z-	Serial Data In-
10	Input	Quadrature Signal B+	-
11	Input	In 1	In 1
12	Output	Out 1	Out 1
13	Output	+5V (max XXX mA)	+5V (max XXX mA)
14	Input	Quadrature Signal B-	-
15	Input	Quadrature Signal A+	Serial Clock+

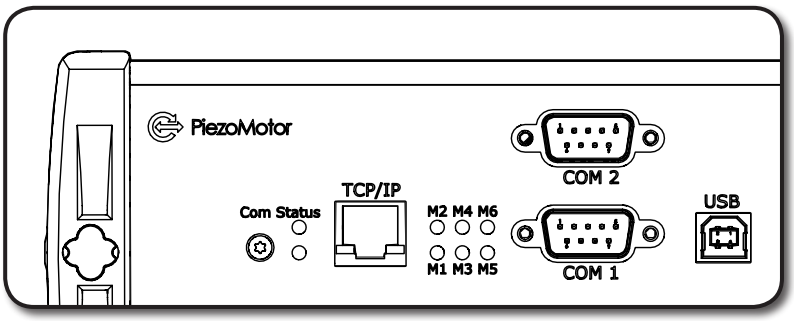


7.4 Communication

7.4.1 Serial Communication

It is possible to connect the host (e.g. a PC) to the driver via serial RS485 interface, using the connector labeled **COM1**. The connector is D-sub 9P. It is recommended to use shielded twisted pair cable. The twisted pairs are the complementary transmit and receive signals. The wiring between the driver and host/sensor is not by straight cables. The receiver is equipped with AC termination. See schematic illustrations of possible connection configurations in section 4.2.6 on page 8.

COM1 and COM2 Connectors		
Pin	**Signal**	**Description**
1	GND	Ground
2	TX+	Transmit data+
3	RX+	Receive data+
4	-	Not connected
5	-	Not connected
6	TX-	Transmit data-
7	RX-	Receive data-
8	-	Not connected
9	-	Not connected



The second serial communication port, labeled **COM2**, is for connecting with external sensors using serial RS422 interface. The connector for COM2 is also D-sub 9P. The receiver is equipped with AC termination.

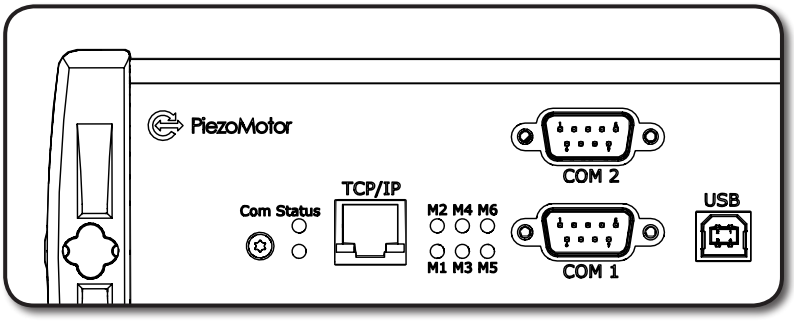
The table below show the communication settings to use when connecting host or sensor via serial line.

Serial Communication Settings								
	Port	**Baud Rate**	**Start Bit**	**Data Bits**	**Parity**	**Stop Bits**	**Handshaking**	**Duplex**
Host	COM1	115.2 kbit/s	1	8	none	1	none	Full duplex
Sensor	COM2	5 Mbit/s	1	8	even	1		

7.4.2 Communication via TCP/IP

You can connect the host (e.g. a PC) to the driver via Ethernet, and/or transmit sensor data from a sensor with Ethernet interface. The connector on the driver is of type RJ45, and is labeled **TCP/IP**. See schematic illustrations of possible connection configurations in section 4.2.6 on page 8.

TCP/IP Connector		
Pin	**Signal**	**Description**
1	TX+	Transmit data+
2	TX-	Transmit data-
3	RX+	Receive data+
4	-	Not connected
5	-	Not connected
6	RX-	Receive data-
7	-	Not connected
8	-	Not connected



By default, dynamic IP assignment is used for the driver and necessary information regarding IP address, gateway and bit mask is provided by a DHCP server on the network. For host communication the default port number is 9760 (i.e. 0x2620). The port number can be changed using the ***IP***-command (which is described in section 6.2 on page 13). A socket communication can be established with the driver board acting as a server. While a host is connected to the server socket, the driver board will not listen to commands on the serial port.

To use a static IP address, the driver must be given an address, a gateway and a bit mask using the settings commands (***IP***, ***GW***, and ***IM***-commands) described in section 6.2 on page 13.

Note!

If a static IP address is assigned, e.g.192.168.10.10, and the bit mask is set to 255.255.255.0, it is assumed that all addresses in the 192.168.10.X range are on the same subnet, and packets to those addresses will not be routed through the gateway.

To use TCP/IP for communication with an external sensor, the IP address and port number of the sensor must be set using the ***SI***-command (see section 6.4 on page 15). When an external sensor IP address is assigned, the driver will try to establish a socket communication with the sensor on the given port number.

Using TCP/IP, the external sensor will send a packet every 20 ms, containing all positions measured since last packet sent. This means that even with a faster sampling in the sensor, position information will not be available to the driver faster than 50 Hz. Still, it is advisable to set the highest possible sampling rate to ensure that the last set of measurements in the packet is as recent as possible. When a new packet is received from the sensor, the last positions for each channel will be extracted and used and the rest of the information will be ignored.

When using TCP/IP for external sensor communication, the driver will send a trig command to the sensor every 10 seconds and thus ensure that new measurements are made and sent automatically. This is different from the case of external serial sensor where the data is pulled from the sensor every millisecond.

7.5 Miscellaneous

7.5.1 USB Port

The USB port, labeled **USB** is not in use.

7.5.2 Power Supply

The driver unit is powered with 90-264 V AC, 50/60 Hz. The PMD206 is fused at 5 A, and the PMD236 at 10 A.

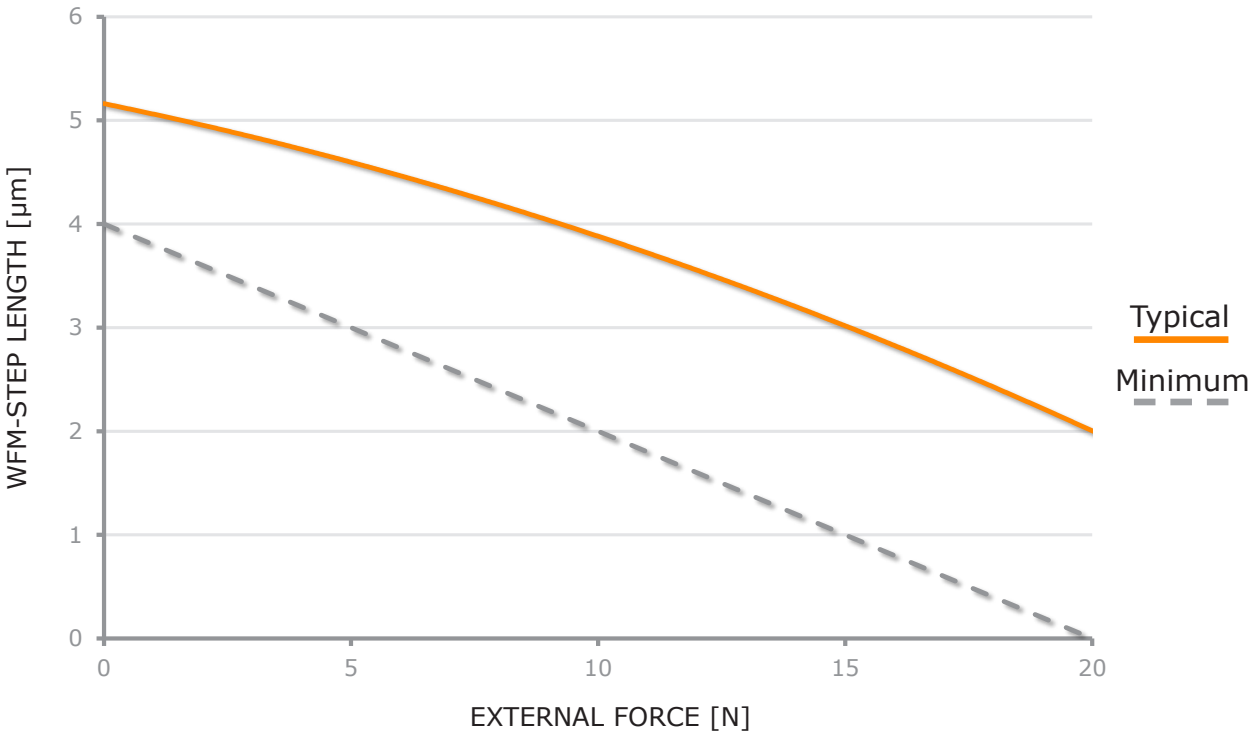
8 – Operation

8.1 Driving a Piezo LEGS Motor

When driving a Piezo LEGS motor, you cannot count wfm-steps or microsteps and know your position. Because of the friction drive you cannot rely on each step being equal to the next. For position control, you will have to use some form of feedback from positioning sensor or limit switches. The PMD206 / PMD236 allows for open or closed loop operation. There is no way to control the actual speed, only the number of wfm-steps per second.

Example

The motor wfm-step length is affected by the external force it is acting against. A typical behavior for waveform *Rhomb* is exemplified in the graph below (for a *Piezo LEGS Linear 20N* motor). Graph shows only behavior when working against the external force. When working with the external force the step length will increase accordingly.



Note!

No opposing force will for this specific motor type (and *Rhomb* waveform) give typical no load wfm-step length ~5 µm. With waveform *Delta* (only waveform used in the PMD206 / PMD236) the typical wfm-step length will be shorter. The stall force for this motor is defined to 20 N. For optimal microstep performance, and best life time, you should design your system so that the motor works at maximum 50% of its stall force.

8.2 Closed Loop

Closed loop operation is the preferred way of running the Piezo LEGS motor. In closed loop mode, the motor will be guided to a certain position when the "go to target" command is issued (***TP***-command). Of course, the closed loop mode of operation will only work when the driver is receiving feedback from an encoder. The PMD206 / PMD236 will read quadrature encoders and serial SSi encoders via the internal sensor board(s). It is also possible to communicate with a sensor with Ethernet or serial interface. If a special encoder is not supported, one may have the host (PC) report the current position so that the driver can make corrections based on each report.

When the *Target Position* command is issued (***TP***-command), the motor starts moving and the PMD206 / PMD236 runs a target loop. A new estimate of the distance to target is made every millisecond. The target loop will take in

to account all applicable settings including ramping parameters. It is possible to set encoder limits to stop motion if position comes outside of a certain range. In addition, external limit switches can be connected to the sensor port.

There is also a command *Target Relative* (***TR***-command). With this command you enter the distance (in encoder counts) that you want to move relative the previous target position. If the command is issued when the driver is not in target mode, it will use the current encoder position as reference when making the move. It will hold the new target position just as the ***TP***-command does.

Note!

To get closed loop operation working properly it is important to set the target loop parameters for the actual system. Carefully read section 9.1 on page 33 to learn more about all important settings for closed loop operation. Most important is to set the *StepsPerCount* parameter which holds information about encoder resolution and wfm-step length.

8.3 Open Loop

To drive a motor in open loop means sending commands to the driver and not using encoder feedback in a control loop. You can command the driver to take a precise number of microsteps, but you cannot say beforehand in what position the motor will stop. The command for open loop operation is called *Run Steps* (***RS***-command), and allows user to set speed (wfm-steps/second), number of microsteps to run, and direction. External limit switches can be connected to the sensor port.

Note!

By definition (see chapter 3.3), the number of microsteps in one wfm-step is 65536 (hardware limit). The resolution in the current firmware is limited to 8192 microsteps/wfm-step, but the *Run Steps* command (***RS***-command) operates according to the microstep theoretical range of 65536 microsteps/wfm-step. Therefore, you always need to take at least 8 microsteps (i.e. 65536/8192=8) to make an actual movement.

8.4 Homing

Quadrature encoder scales are often equipped with an index marker to reference against. Since the quadrature encoder position is not remembered at power off, and because there is a risk of loosing encoder counts when the system is subjected to disturbance, you want to calibrate to the reference point (index marker) every once and again. The so called “homing” can be done in different ways. Either you use the semi-automatic homing command (***HO***-command), or else you perform the index search manually by setting the driver in index mode (***SB***-parameter 2) and running open loop (with ***RS***-command) until you find the index.

The ***HO***-command is used like this: You enter speed (wfm-steps/second) and the first motion direction to check. You also enter the maximum number of microsteps and the maximum number of encoder counts in each direction. When the command is executed the motion starts in direction one, and the controller starts looking for the index marker. If the index marker has not been found when either the maximum number of steps or maximum number of encoder counts is reached, the direction will be reversed and the search will continue. Whenever the index marker is reached the position will be zeroed immediately where after the motor will be stopped. Because of speed and impulse of the external load, the motor will most often not stop exactly at the index position. In other words, when the homing command has finished without error (when the index marker has been found), the motor will most often be at stand still some distance away from encoder count zero. The homing status will read **1** when index marker is found, and **2** when the sequence is completed without finding index at all. Status **3** indicates stop with error, for example if the controller is not detecting any encoder counts.

9 – Settings

9.1 Closed Loop Target Mode Parameters

To set up your system to run in closed loop target mode you need to use an encoder for position feedback. Quadrature as well as serial SSi encoders are supported. If you use the *PiezoMotor Motion System* software, most settings can be set in the options tab. If you are writing your own software or if you are running the driver from a terminal program, you have to use the settings commands to set parameters. Read more about command syntax and how to connect to the driver in chapter 6 – Commands, on page 13.

9.1.1 StepsPerCount Parameter

Most important is to set the *Target Mode StepsPerCount* (SPC) parameter. The parameter is used in the driver unit target loop to calculate the number of wfm-steps to run to reach target. The SPC value is a calculated relationship between these two. When the SPC is correctly set, the driver unit has all information necessary to perform an optimized target mode run. Default value is 5243 (0x147b).

Linear motor: $$SPC = \frac{2^{20} \bullet \text{encoder resolution [nm/count]}}{\text{wfm-step length [nm/wfm-step]}} = \frac{2^{20}}{\text{encoder counts per wfm-step}}$$

Rotary motor: $$SPC = \frac{2^{20} \bullet \text{encoder resolution [mrad/count]}}{\text{wfm-step angle [mrad/wfm-step]}} = \frac{2^{20}}{\text{encoder counts per wfm-step}}$$

Example Linear

You have a linear encoder with 20 nm resolution mounted on the motion axis, and you know (or can estimate from the graph in the motor data sheet) that for your application the wfm-step length is about 4 µm (= 4000 nm) given the external load. You can calculate SPC = 2^{20} • 20 / 4000 = 5243.

Example Linear

You have a linear encoder with 20 nm resolution mounted on the motion axis, but you are uncertain of the travel distance for one wfm-step. The wfm-step length can be estimated experimentally in the actual application. Run at least ten wfm-steps (open loop) to get a good average value. With an average of 200 counts per wfm-step you can calculate SPC = 2^{20} / 200 = 5243.

Example Rotary

You have a rotary encoder with 8192 CPR (counts per revolution). The encoder resolution is 2π/8192 ≈ 0.77 mrad. You know from your application (or can estimate from the graph in the motor data sheet) that the wfm-step angle is 0.9 mrad given the external load. You can calculate SPC = 2^{20} • 0.77 / 0.9 = 897115.

Example Rotary

You have a rotary encoder with 8192 CPR (counts per revolution). The encoder resolution is 2π/8192 ≈ 0.77 mrad. You are however uncertain of the wfm-step angle and need to measure it experimentally. Run at least ten wfm-steps (open loop) to get a good average value. With an average of 1.17 counts per wfm-steps you can calculate the SPC = 2^{20} / 1.17 = 896219.

Note!

The average wfm-step may not be the same in both directions because the wfm-step length/angle is load dependant. Recommended procedure is to run open loop in both directions to measure mean distance and calculate the average SPC.

9.1.2 Target Mode Position Limits A and B

In addition to external limit switches connected to the sensor port, you can set digital limits using the setting parameters *TargetMode Position Limit A* and *Target Mode Position Limit B*. When running in target mode the motor will stop when passing either of these limits. When motor is outside of the digital limits, it is no longer possible to

run the motor in target mode. User must use open loop run command (RS-command to return back to within limits. Default setting for limit A is -10,000 counts (0xffffd8f0), and limit B is +10,000 counts (0x2710).

9.1.3 Target Mode Minimum and Maximum Speed

When running in target mode, the setting parameters called *Target Mode Minimum Speed* and *Target Mode Maximum Speed* defines the minimum and maximum speed of the run. The parameters are given in the unit wfm-steps per second. Default settings are minimum 2 wfm-step per second (0x2), maximum 50 wfm-steps per second (0x32).

9.1.4 Target Mode Speed Ramp Up

The setting parameter *Target Mode Speed Ramp Up* will change the behavior of the target run. At the start of the target run, acceleration will be according to what is set in the ramp up parameter. Ramp up parameter is defined as the wfm-step frequency acceleration rate (Hz/ms). Default value is 48 wfm-steps per second per millisecond (0x30).

9.1.5 Target Mode Speed Ramp Down

The setting parameter *Target Mode Speed Ramp Down* will change the behavior of the target run. At the end of the target run, motion will slow down according to what is set in the ramp down parameter. Ramp down parameter is defined as the wfm-step frequency (Hz) at the estimated distance of one wfm-step from target. Default value is 48 wfm-steps per second (0x30).

9.1.6 Target Mode Stop Range

The setting parameter *Target Mode Stop Range* defines where it is acceptable for the driver to stop when closing in on target. Default value is 0 encoder count (0xa), meaning there is no stop range, and the target loop will only stop at the exact target count. If you enter a value, e.g. 10 encoder counts (0xa), the motion will stop in the range ±10 encoder count from target. It will remain stopped until a position value outside the range is encountered, and will at that time restart target loop.

9.1.7 Encoder Direction

The *Encoder direction* parameter is essential to get proper function in target mode, because if the counting direction is wrong, the target loop will fail (motor will be running further and further away from target instead of towards target).

9.2 Other Settings

In this chapter, some of the more important settings are more thoroughly explained.

9.2.1 Parking and Initialization

The *Parking and Initialization* setting allows the user to park (power down) or unpark (power up) motor. The same command is also used for initiating various saved parameters from Flash/EEPROM or to initiate the factory set default values. To save parameters to flash, user must use the *Controller command* (i.e. ***CC***-command).

9.2.2 Encoder Type

You have to use the *Sensor interface* command (i.e. ***SI***-command) to define the sensor interface: either via internal sensor board, via serial interface (RS422), or via Ethernet (TCP/IP). If the internal sensor board is chosen, the *Encoder type* setting defines what type of encoder is used. The internal sensor board will work with quadrature encoder (with ABZ index), and serial encoders (2-wire SSi, 8 to 31 bit, always extended to 32 bit by lap counting).

9.2.3 External Limits

The *External limits* setting will enable input pins 1 and 6 for use with external limit switches. External limit switches will work in closed loop target mode and in open loop. When the setting is enabled, low levels on the pins will stop motion. Pin 1 low will stop reverse motion and pin 6 low will stop forward motion.

9.2.4 Position Offset

The *Position Offset* parameter setting defines the encoder count offset position (for quadrature encoders). Default offset value is 0 counts (0x0). The offset value cannot be stored in Flash/EEPROM, and hence is not restored after power off.

10 – Maintenance

10.1 General Maintenance Instructions

The PMD206 / PMD236 does not require any regular maintenance. Follow given safety instructions!

10.2 Troubleshooting

If problem arise the status command error codes might provide useful information. Check wirings to motor and encoder. Try driving in open loop to see if you can get movement. To get closed loop operation in order, make sure the SPC setting is correct. Check that you are calculating the SPC using the right encoder resolution. If you are using a quadrature encoder you need to check the interpolation factor. If you get readings from the encoder you can see if you have the correct resolution by taking a number of wfm-steps and checking how many encoder counts you have traveled. A wfm-step is typically ~5 µm ± a few µm for a standard linear Piezo LEGS motor at zero load.

10.3 Firmware Updates

When new features are added, the user can update the firmware. Please check our homepage for updates: www.piezomotor.com.

When new firmware is installed, you should also look for the latest version of this manual to learn about the updates.

11 – Warranty

11.1 Warranty Conditions

General conditions for the supply of mechanical, electrical, and electronic products according to ***Orgalime S2012***.

150087-02

Visit our website for application examples, CAD files, videos and more...

www.piezomotor.com

PiezoMotor

PiezoMotor Uppsala AB
Stålgatan 14
SE-754 50 Uppsala, Sweden

Telephone: +46 18 489 5000
Fax: +46 18 489 5001

info@piezomotor.com
www.piezomotor.com