BEAR™ Operation and SDK Manual

Westwood Robotics® Corporation

v 0.3.3

Contents

1 Introduction

1.1 About This Manual

- Type of a data is enclosed with "<>". For example, <list>is a data type in Python.
- **BEAR** is also referred to as BEAR motors, BEAR actuators or BEAR modules.
- **CAUTION** labels contain advises and instructions that, if not properly followed, can possibly lead to damage or malfunction on your BEARs.

CAUTION

This is an example of **CAUTION** label.
CAUTION labels contain advice and instructions that, if not properly followed, can possibly lead to damage or malfunction on your BEARs.

- **WARNING** labels contain restrictions and instructions that, if not properly followed, will definitely lead to severe damage on your BEARs, and can result in dangerous situations.

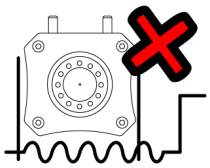
WARNING

This is an example of **WARNING** label.
WARNING labels contain restrictions and instructions that, if not properly followed, will definitely lead to severe damage on your BEARs, and can result in dangerous situations.

1.2 Warnings

WARNING

To avoid structural deformation, please NEVER clamp your BEAR from the side.



WARNING

To avoid leakage, please do not loosen or tighten the screws on the cooling channel cap or disassemble the cooling channel cap. The gasket must be replaced once disassembled.

WARNING

In Direct Force Mode, the output speed of BEAR is not limited by the limit_velocity_max setting.

WARNING

Pay extra attention when changing mode while BEAR is enabled. BEAR will remain enabled and execute the corresponding goal_xxx setting in the new mode immediately.

WARNING

Do not save configurations when the motor is enabled. The motor may not respond when it is writing flash memory.

1.3 Know Your BEAR

a) Feature Overview

For convenience and comparison, basic mechanical, electrical and performance properties of all current available BEAR products are listed in Table. 1 and 2 as below.

Property	**Koala BEAR**	**Koala BEAR Muscle Build**
Current Version	KB02	KBMB01
Weight	250g	285g
Supply Voltage	$9 \sim 33.6$V ($3 \sim 8$S)	
Power Connector	XT30	
Signal Connector	Molex PicoBlade 53047 6Pin	
Reflected Inertia	$1.82 \times 10^{-3}\,\mathrm{kg/m^2}$	
Speed Constant K_V	27.3 RPM/V	9 RPM/V
Torque Constant K_T	0.35 Nm/A	1.16 Nm/A
Stall Torque 15sec	3.5 Nm	8 Nm
Stall Torque 15sec(LC)	4.2 Nm	N.A.[2]
Stall Torque 1.5sec	10.5 Nm	20 Nm

Table 1: Koala Series Specification[1]

Property	**Panda BEAR**	**Panda BEAR Plus**	**Kodiak BEAR**
Current Version	PB02	PB02P	CB01
Weight	685g	925g	2500g
Supply Voltage	$9 \sim 50.4$V ($3 \sim 12$S)		
Power Connector	XT60		XT90
Signal Connector	Molex PicoBlade 53047 6Pin		
Reflected Inertia	$7.44 \times 10^{-3}\,\mathrm{kg/m^2}$		
Speed Constant K_V	14.3 RPM/V	7.1 RPM/V	
Torque Constant K_T	0.67 Nm/A	1.3 Nm/A	
Stall Torque 15sec	13.4 Nm	26.5 Nm	180 Nm
Stall Torque 15sec(LC)	16.8 Nm	33 Nm	240 Nm
Stall Torque 1.5sec	33.5 Nm	67 Nm	$\geq$ 350 Nm

Table 2: Panda and Kodiak Series Specification[1]

b) Koala BEAR™ V2 (KB02)

Koala BEAR V2(KB02) is a small actuator designed for highly dynamic applications with relatively low loads, such as all kinds of small mobile robots, robot hands or robot manipulators. The mechanical and electrical as well as the thermal management features are introduced as following, with the help of figure 1.

- Mechanical Features The dimensions and locations of mounting features are anno-

[1] Stall Torque is the maximum torque BEAR can deliver under given time period, while temperature rise is within 55°C; LC stands for Liquid Cooled.

[2] Liquid cooling option is not available on KBMB01.

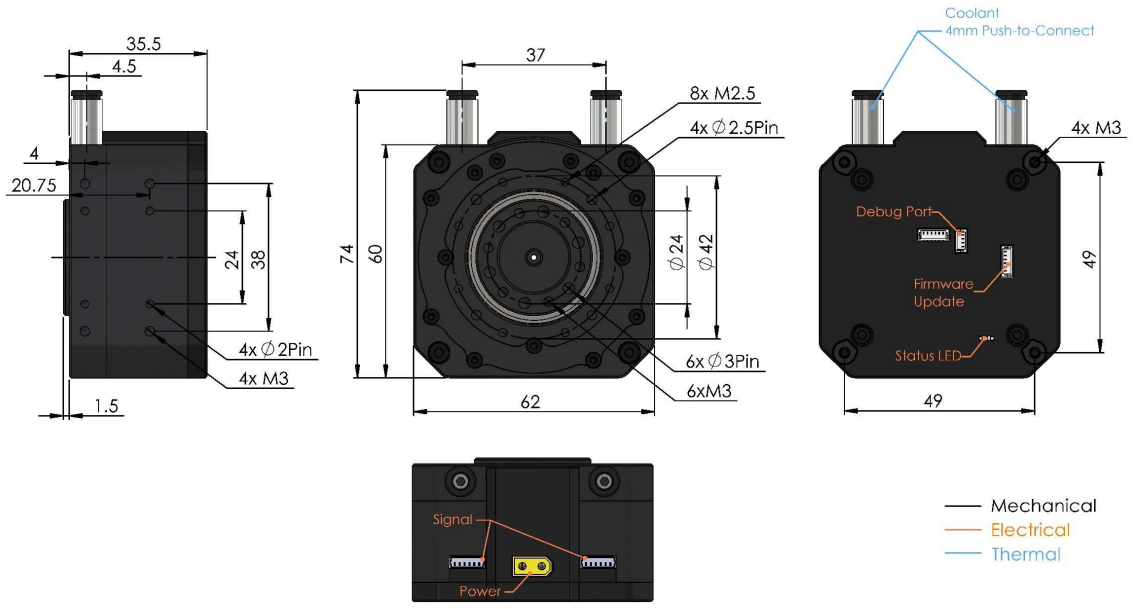


Figure 1: Koala BEAR V2 Specs

tated as in figure 1. There are six M3 screw holes and six 3mm pin holes on the output shaft for locating and connecting the payload. The eight M2.5 screw holes, four 2.5mm pin holes located on the front side as well as the four M3 screw holes and four 2mm pin holes on three sides can all be used to locate and mount KB02 to its application. Besides, there are also four M3 screw holes on the back that are axisymmetric about the output shaft, and these four M3 screw holes can also be used as mounting points or to mount additional bearings under certain application.

Please refer to Table. 1 for overall mechanical and performance properties of KB02.

> **CAUTION**
>
> When high axial load is expected at the application joint, additional axial support is required instead of directly use Koala BEAR for the axial support on the joint.

- Electrical Features KB02 has one XT30 power port, and two Molex PicoBlade 53047 6Pin signal ports. The two signal ports make it convenient to connect multiple BEAR modules in serial. Power supply voltage for KB02 ranges from 9V to 33.6V.

CAUTION

When driving the load dynamically, a non-negligible back EMF will be generated by BEAR upon back-drive motions or impacts. In such applications, it is highly recommended to use Li-Po batteries as power supply or add a back-EFM absorbing circuit with significant capacitance to the power supply circuit to protect the power supply.

- Liquid Cooling The headers of the liquid cooling channel on KB02 are push-to-connect headers for 4mm OD tubes. The applied coolant pressure in the cooling channel can be up to 1MPa. It is recommended to use DI water mixed with appropriate amount of biological inhibitors as coolant. It is not recommended to add other additives or dyes into the coolant.

CAUTION

. To prevent damage to the circuit from coolant leakage, please apply/replace PTFE sealant tape on the thread of the cooling channel headers before attaching them onto your BEAR and check for leakage carefully.

CAUTION

Refrain from using Copper (II) Sulphate ($CuSO_4$) additive – common trade name “Nuke Cu” or “Biocide Cu” – due to its tendency to react with metals usually found in the liquid cooling loop, especially radiators (Zn, Cu, Sn) as well as BEAR (Al) thus promoting corrosion. Using $CuSO_4$ also accelerates visually discouraging copper tarnishing phenomena.

CAUTION

In the case of regularly liquid cooled applications, please check for leakage in the liquid cooling loop at least weekly.

WARNING

To avoid leakage, please do not loosen or tighten the screws on the cooling channel cap or disassemble the cooling channel cap. The gasket must be replaced once disassembled.

c) Koala BEAR™ Muscle Build V1 (KBMB01)

Koala BEAR Muscle Build V1(KBMB01) is the strongest actuator in the Koala Series. It remains the same compact and extra-low weight design of Koala Series, but offers as high as 3 times the torque of standard Koala BEAR V2. It is ideal for dynamic applications that are very restrict on size and weigh, while prefer higher load capacity than high speed, such as wearable robotic devices, rehabilitation robots and devices, robot hands or table-top robot manipulators. Liquid cooling is not available on KBMB01.

The mechanical and electrical as well as the thermal management features are introduced as following, with the help of figure 2.

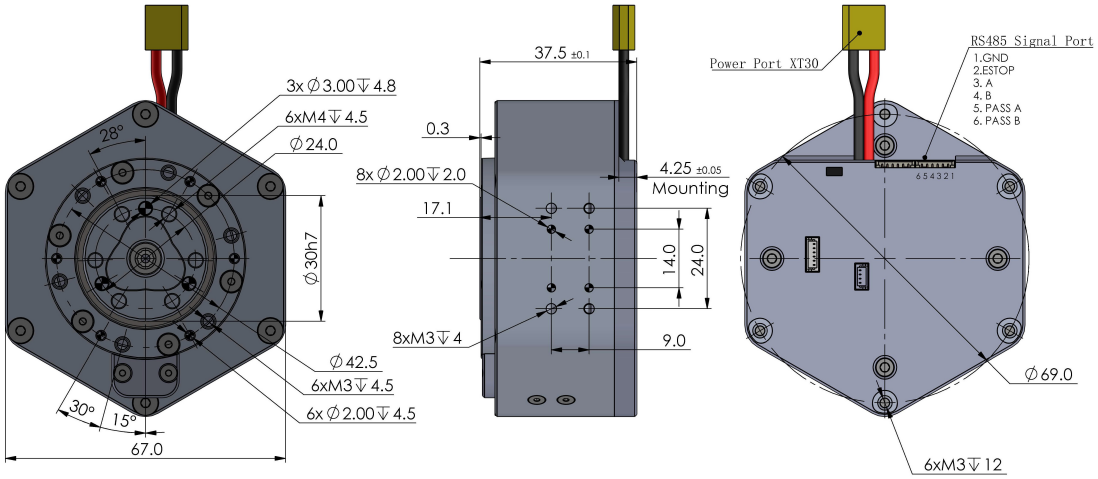


Figure 2: Koala BEAR Muscle Build V1 Specs

- Mechanical Features The dimensions and locations of mounting features are annotated as in figure 2. There are six M4 screw holes and three 3mm pin holes on the output shaft for locating and connecting the payload.

  The six M3 screw holes, six 2mm pin holes located on the front side as well as the four M3 screw holes and four 2mm pin holes on two sides can all be used to locate and mount KBMB01 to its application. Besides, there are also six M3 screw holes on the back that are axisymmetric about the output shaft, and these six M3 screw holes can also be used as mounting points or to mount additional bearings under certain application.

  Please refer to Table. 1 for overall mechanical and performance properties of KBMB01.

CAUTION

When high axial load is expected at the application joint, additional axial support is required instead of directly use Koala BEAR for the axial support on the joint.

- Electrical Features KBMB01 has one XT30 power port, and two Molex PicoBlade 53047 6Pin signal ports. The two signal ports make it convenient to connect multiple BEAR modules in serial. Power supply voltage for KB02 ranges from 9V to 33.6V.

CAUTION

When driving the load dynamically, a non-negligible back EMF will be generated by BEAR upon back-drive motions or impacts. In such applications, it is highly recommended to use Li-Po batteries as power supply or add a back-EFM absorbing circuit with significant capacitance to the power supply circuit to protect the power supply.

- Liquid Cooling Liquid cooling is not an option for KBMB01.

d) Panda BEAR™ V2 (PB02) and Panda BEAR™ Plus V2 (PB02P)

With the right balance of torque, weight, and form factor, Panda BEAR V2(PB02) and Panda BEAR Plus V2(PB02P) are our most versatile units. Their excellent dynamic performance and payload capability make them well suited for diverse applications ranging from legged mobile robots to service and entertainment robots. Both PB02 and PB02P share identical mechanical, electrical, and thermal management features, as detailed in the following, with the help of figure 3.

- Mechanical Features The dimensions and locations of mounting features are annotated as in figure 3. There are eight M3 screw holes and eight 3mm pin holes on the output shaft for locating and connecting the payload. The eight M3 screw holes and 3mm pin holes located on the front side as well as the six M3 screw holes and six 2mm pin holes on each of the three sides can all be used to locate and mount PB02 to its application. Besides, there are also eight M3 screw holes on the back that are axisymmetric about the output shaft, and these eight M3 screw holes can also be used as mounting points or to mount additional bearings under certain application.

CAUTION

When high axial load is expected at the application joint, additional axial support is required instead of directly use Panda BEAR for the axial support on the joint.

Please refer to Table. 2 for overall mechanical and performance properties of PB02.

- Electrical Features PB02 has one XT60 power port, and two Molex PicoBlade 53047 6Pin signal ports. Pry open the back cap to access the signal ports, as instructed in figure 4. The two signal ports make it convenient to connect multiple BEAR modules in serial. Power supply voltage for PB02 ranges from 9V to 48V.

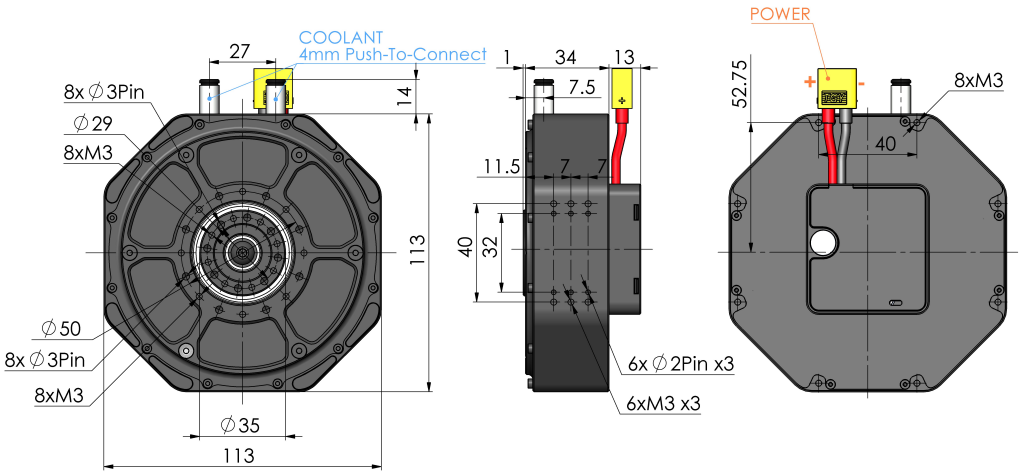


Figure 3: Panda BEAR V2 Specs

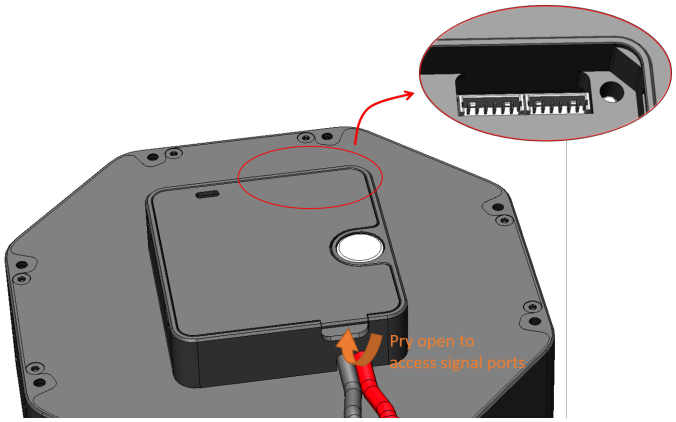


Figure 4: Panda BEAR Signal Ports

CAUTION

When driving the load dynamically, a non-negligible back EMF will be generated by BEAR upon back-drive motions or impacts. In such applications, it is highly recommended to use Li-Po batteries as power supply or add a back-EFM absorbing circuit with significant capacitance to the power supply circuit to protect the power supply.

- Liquid Cooling The headers of the liquid cooling channel on PB02 are push-to-connect headers for 4mm OD tubes. The applied coolant pressure in the cooling channel can be up to 1MPa, but it is recommended to regulate your coolant pressure under 0.7MPa if your system's coolant is pressurized at all time. It is recommended to use DI water mixed with appropriate amount of biological inhibitors as coolant. It is not recommended to add other additives or dyes into the coolant.

CAUTION

. To prevent damage to the circuit from coolant leakage, please apply/replace PTFE sealant tape on the thread of the cooling channel headers before attaching them onto your BEAR and check for leakage carefully.

CAUTION

Refrain from using Copper (II) Sulphate ($CuSO_4$) additive – common trade name "Nuke Cu" or "Biocide Cu" – due to its tendency to react with metals usually found in the liquid cooling loop, especially radiators (Zn, Cu, Sn) as well as BEAR (Al) thus promoting corrosion. Using $CuSO_4$ also accelerates visually discouraging copper tarnishing phenomena.

CAUTION

In the case of regularly liquid cooled applications, please check for leakage in the liquid cooling loop at least weekly.

WARNING

To avoid leakage, please do not loosen or tighten the screws on the front cap or disassemble the front cap. The gasket inside must be replaced once disassembled.

e) Kodiak BEAR™ V1 (CB01)

Kodiak BEAR V1(CB01) is our strongest line-up specifically built to provide maximum torque for big applications ranging from walking humanoids to industrial manipulators, while maintain agile torque sensing and control capabilities.

CAD models, mechanical, electrical, and thermal management features of CB01 are available for existing customers of Kodiak BEAR actuator. Contact us for these files once your purchase is complete.

2 Using BEAR

2.1 Power & Signal

a) Power

The power port on Koala Series is a male XT30 connector, the power port on Panda Series is a male XT60 connector, and XT90 male connector on Kokdiak. The power port polarity follows the regular convention of XT30/XT60/XT90 connectors: the pole near flat side is positive and the pole near the rounded/chamfered side is ground, as shown in fig. 5

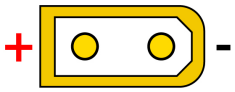


Figure 5: BEAR power connector and polarity

> **WARNING**
>
> Be cautious and never reverse power polarity.

Power supply voltage for different BEARs are as listed in Table. 3. Connect BEARs getting same voltage in parallel when using multiple BEARs.

> **WARNING**
>
> To avoid fire hazard, please estimate nominal current consumption on each BEAR when chaining multiple BEAR in parallel and select power cable with appropriate AWG, especially higher current consumption is expected.

Product Series	**Koala**	**Panda**	**Kodiak**
Supply Voltage	9 $\sim$ 33.6V (3 $\sim$ 8S)	9 $\sim$ 50.4V (3 $\sim$ 12S)	9 $\sim$ 50.4V (3 $\sim$ 12S)
Connector	XT30	XT60	XT90

Table 3: BEAR Power Supply Specs

b) Indicator

Each BEAR has an LED indicator on the back side. There are three(3) LEDs with different colors on the indicator that indicates the status of the actuator: Grean, Blue and Red.

The Grean light comes on once BEAR is powered and initialization is complete; The Blue LED lights up as soon as the torque output is enabled, and turns off once disabled; The Red light indicates an existing Error.

c) USB2BEAR

It is recommended to use Westwood Robotics USB2BEAR™ high speed RS485 USB dangle to connect BEARs to your computer.

CAUTION

Use at own risk when using other generic RS485 dangles.

The USB2BEAR dangle is a USB2.0 device and its RS485 connector is a 4-pin Molex Mini-SPOX 5268 male header. (Mating female housing: 4-pin Molex Mini-SPOX 5264, part number: 0050375043) It's pin-out and function of the switches are as shown in fig. 6.

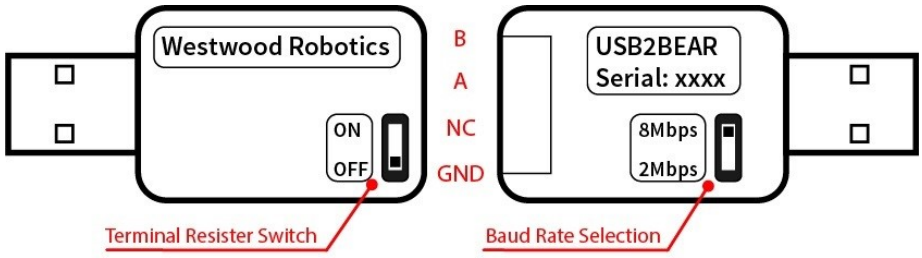


Figure 6: USB2BEAR pin-out and switches.

It is recommended to add a terminal resister of 120 Ohms at the end of RS485 chain when using long signal cables and the impedance of the signal line is high enough to result in noisy communication. Put the Terminal Resister Switch at 'ON' when a terminal resister is applied at the end of the chain, but be sure to have it at 'OFF' when no terminal resister is applied at the end of the chain. USB2BEAR can also be used to communicate with generic RS485 devices other than BEAR. Make sure the matching communication Baud Rate is selected. It is recommended to use 8Mbps for fast communication when paired with BEARs.

d) Chaining BEAR Signal Ports

When having multiple BEARs in your system, properly chaining their signal lines not only can make your wire management easy and clean, but also contribute to system reliability and robustness. Just like all other RS485 devices, you can simply chain your BEARs in a daisy chain, as shown in fig. 7. It is recommended to add a terminal resistor of 120 Ohm at the end of the signal chain, especially when the signal line is relatively long or whenever exceptional signal noise is observed.

There could also be multiple chains of BEARs, such as in the application of dual-arm manipulators or legged robots. In this type of situation, traditional solutions are either using a long signal cable to connect the end of one chain with the start of another, or using multiple RS485 adapters, one for each chain. The former solution could lead to high impedance in the signal line thus noisy communication, while the latter could result in control complication and a demand of too many USB ports on the controller.

In the above situations, the unique pass-thru channels on all BEAR signal ports become very handy. Fig. 8 illustrates a simple example of using the pass-thru channels to achieve a fork chain. Again, it is recommended to add a terminal resistor of 120 Ohm at the end of the signal chain, especially when the signal line is relatively long or whenever exceptional signal noise is observed.

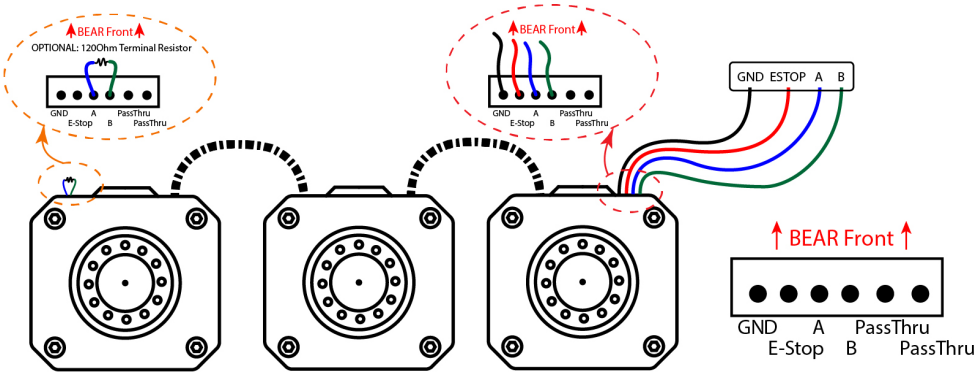


Figure 7: Daisy chain BEARs.

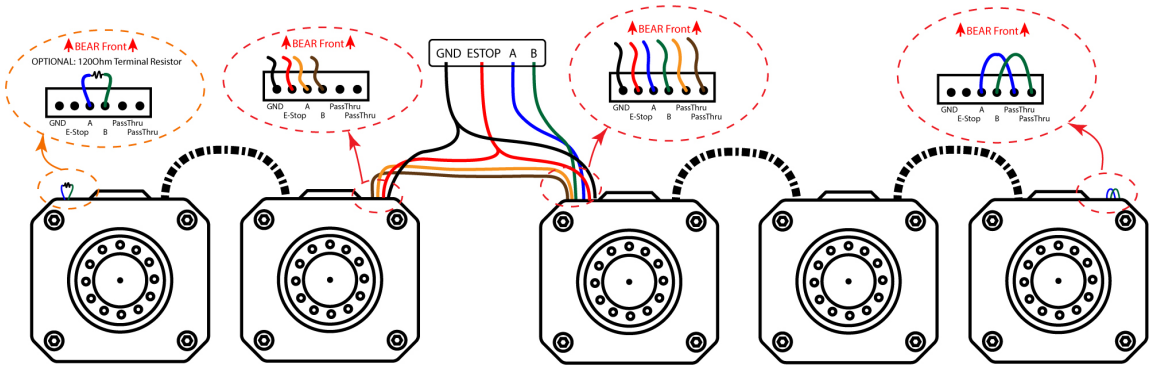


Figure 8: Fork chain BEARs.

It is fine to chain different type of BEARs together, but be careful to make sure that all BEARs get the correct power supply voltage. All BEARs in the same chain should have the same baud rate setting, and there is no ID conflict as well.

e) Connect to USB2BEAR

There are three ways of connecting the signal line from BEARs to the USB2BEAR dangle, as shown in fig. 9. The GND, A and B terminals are always connected to corresponding pins on the USB2BEAR dangle, and the only difference between these three configurations is how the ESTOP signal terminal is handled. The minimum connection is to connect the E-STOP terminal to signal GND, but doing this will eliminate the function of E-STOP protection thus depreciated. A basic configuration which adds a E-STOP switch between the ESTOP and the signal GND terminal is preferred over the minimum configuration. Disconnecting the ESTOP terminal from the signal GND terminal via the E-STOP switch triggers the E-STOP protection on all connected BEARs.

Our recommended configuration is to connect the GND and ESTOP terminal to a Westwood Robotics Wireless ESTOP module, which enables the user to trigger the E-STOP protection

on all connected BEARs remotely, which we consider to be an extremely important safety feature for high power or high complicity systems, or any system that works around a human.

Please refer to section. **f)** for detailed explanation on E-STOP protection.

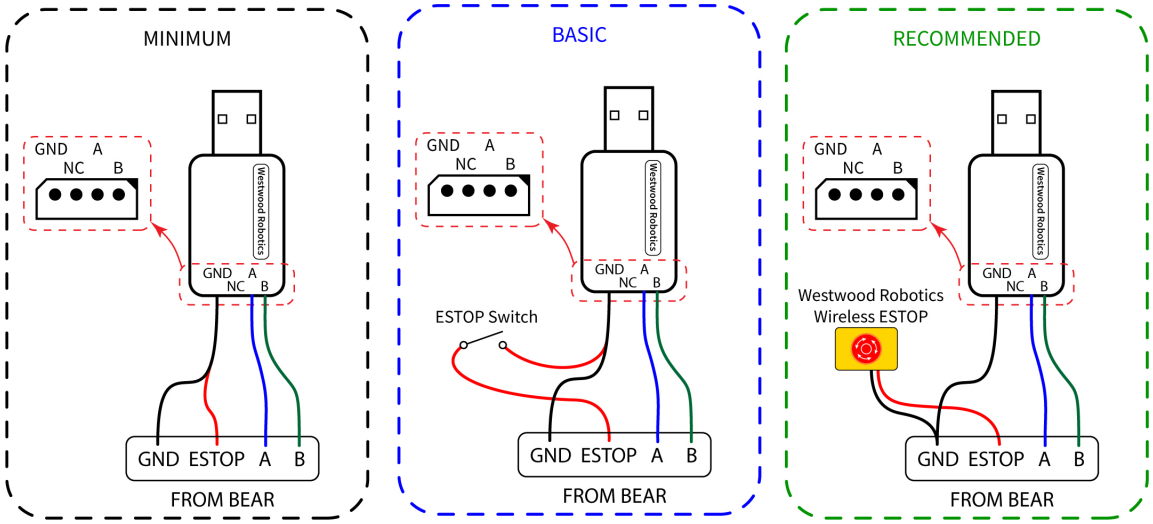


Figure 9: Connecting BEARs to USB2BEAR.

> **CAUTION**
>
> Do **NOT** leave the ESTOP terminal floating, as this keeps BEARs in their ESTOP status and prevents them from being enabled.

f) E-STOP

When the ESTOP terminal on a BEAR is not pulled low to signal GND, the BEAR module's E-STOP protection will be triggered. The 3rd bit of its error code will be come HIGH and the torque enable status will become 3. If the E-STOP protection is triggered while BEAR is enabled, such BEAR will go into hardware damping mode preventing any potential damage; if the E-STOP protection is triggered while BEAR is disabled, such BEAR will stay disabled and will not enter hardware damping mode.

The E-STOP protection can also be triggered by writing "3" to the torque enable status register and does not necessarily require the ESTOP terminal to be disconnected from signal GND.

To release a BEAR from its E-STOP protection, first make sure that the ESTOP terminal is pulled low to signal GND, then disable the BEAR by writing "0" to its torque enable status register. This will also reset the 3rd bit of its error code to LOW.

Refer to section. 2.2.2 for details on torque enable status register and section. 2.2.3 for details on error code.

2.2 Communication

Communication with BEAR is achieved by using BEAR SDK to interact with the Control Table. The Control Table is a structure that consists of multiple Registers to store status or to control the device. Users can check current status of the device by reading from specific Registers in the Control Table, or to control the device by writing specific data to some Registers.

The Control Table is explained in detail in this section as following. Please refer to Section. 3 for complete instruction of using BEAR SDK of various languages to interact with the Control Table.

2.2.1 Control Table

All Registers in the Control Table are divided into two groups: Configuration Registers(CONFIG) and Status Registers(STAT). All values in the Configuration Registers will be saved in the flash memory when "*save_config*" command is received by BEAR. All values in status registers and all unsaved configuration registers will be lost when power-off and reset to default or last-save at power-on.

WARNING

Do not save configurations when the motor is enabled. The motor may not respond when it is writing flash memory.

CAUTION

It is recommended to save config only when necessary. Internal flash guaranteed endurance is 10K write cycle.

Please refer to Table. 4 for complete lists of CONFIG and STAT Registers.

Configuration Registers							
Name	**Description**	**Access**	**Type**	**Unit**	**Default**	**Min**	**Max**
id	Unique motor ID	R/W	uint32		1	0	0xFC
mode		R/W	uint32				
baudrate		R/W	uint32	Mbps			
homing_offset		R/W	float32	rad			
p_gain_id	P gain for Id current loop	R/W	float32		0.001	0	10
i_gain_id	I gain for Id current loop	R/W	float32		0.0001	0	10
d_gain_id	D gain for Id current loop	R/W	float32		0	0	10
p_gain_iq	P gain for Iq current loop	R/W	float32		0.001	0	10
i_gain_iq	I gain for Iq current loop	R/W	float32		0.0001	0	10
d_gain_iq	D gain for Iq current loop	R/W	float32		0	0	10
p_gain_velocity	P gain for velocity loop	R/W	float32		0.2	0	1000
i_gain_velocity	I gain for velocity loop	R/W	float32		0.001	0	1000
d_gain_velocity	D gain for velocity loop	R/W	float32		0	0	1000
p_gain_position	P gain for position loop	R/W	float32		0.01	0	1000
i_gain_position	I gain for position loop	R/W	float32		2E-05	0	1000
d_gain_position	D gain for position loop	R/W	float32		0	0	1000
p_gain_direct_force	P gain for direct force loop	R/W	float32		0	0	1000
i_gain_direct_force	I gain for direct force loop	R/W	float32		0	0	1000
d_gain_direct_force	D gain for direct force loop	R/W	float32		0	0	1000
limit_acc_max	Maximum Acceleration	R/W	float32	rad/s^2	5	0	100000
limit_i_max	Maximum Iq (torque) and Id	R/W	float32	A	5	0	100
limit_velocity_max	Maximum absolute velocity	R/W	float32	rad/s	100	0	10000
limit_position_min	Position limit min.	R/W	float32	rad	-8π	-8π	8π
limit_position_max	Position limit max.	R/W	float32	rad	8π	-8π	8π
min_voltage		R/W	float32	V	6	6	60
max_voltage		R/W	float32	V	40	6	60
watchdog_timeout		R/W	uint32	μs	0	0	10000000
temp_limit_low	Limit power at this temperature	R/W	float32	°C	80	0	125
temp_limit_high	Shutdown at this temperature	R/W	float32	°C	100	0	125
Status Registers							
Name	**Description**	**Access**	**Type**	**Unit**	**Default**	**Min**	**Max**
torque_enable	Enable output	R/W	uint32		0	0	3
goal_id	Goal Excitation Current	R/W	float32	A			
goal_iq	Goal Torque Current	R/W	float32	A			
goal_velocity	-	R/W	float32	rad/s			
goal_position	-	R/W	float32	rad		-8π	8π
present_id	Present Excitation Current	RO	float32	A			
present_iq	Present Torque Current	RO	float32	A			
present_velocity	Present velocity	RO	float32	rad/s			
present_position	Present position	RO	float32	rad		-8π	8π
input_voltage	Present input voltage	RO	float32	V			
winding_temperature	Winding temperature in °C	RO	float32				
powerstage_temperature	Powerstage temperature in °C	RO	float32				
ic_temperature	IC temperature in °C	RO	float32				
error_status	*Not implemented yet	RO	float32				

*R/W: read and write RO: read only

Table 4: Table of Registers

2.2.2 Detailed Description

Config Registers

- **id** The ID of a BEAR. This should be unique for every BEAR in the same chain.
- **mode** Operating mode. Refer to Section. 2.3 for detailed explanation.
- **baudrate** Baud-rate for the RS-485 communication. Unless needed by system setup, it is recommended to leave it at default 8Mbps.
- **homing_offset** present_position = raw position + homing_offset. When setting up new homing offset, always take the existing homing_offset into account.
- **PID Gains** Refer to Section. 2.4 for more details.
- **limit_acc_max** Absolute maximum limit for acceleration (unit: rad/s^2). Effective in mode 2 (position mode) for trajectory generation.
- **limit_i_max** Absolute maximum limit for torque current Iq and excitation current Id (unit: A). Effective in all modes. Since Iq is proportional to torque, this is effectively the torque limit.
- **limit_velocity_max** Absolute maximum limit for velocity (unit: rad/s). Effective in modes 1(Velocity) and 2(Position).
- **limit_position_min/max** Lower/upper position limit for BEAR (unit: rad). Going out of bounds triggers internal damping mode and generates an error. Disable, then bring motor within limit physically or by limit adjustment clears the error. Effective in modes 2(Position) and 3(Direct Force).
- **min/max_voltage** When voltage goes below min_voltage or above max_voltage , hardware fault triggers with an error generated. Disable, then regulate the supply voltage within limits clears the error.
- **watchdog_timeout** Safety watchdog timeout value in micro seconds(μs). When communication times out, BEAR goes into internal damping mode and generates an error. Disable clears the error. This **ONLY** applys to modes 0(torque).
- **temp_limit_low** From this temperature(°C) to temp_limit_high, the Iq limit will start to decrease linearly from limit_i_max, and an error will be generated.
- **temp_limit_high** From this temperature(°C) above, the Iq limit will be reduced to 0. The temperature limit functionality uses the maximum between Winding temperature and Powerstage temperature.

Status Registers

- **torque_enable** Enable status and control of BEAR.
  When write:
  0 - Disable BEAR, also clear latching errors;
  1 - Enable BEAR torque output;
  3 - EStop protection triggered. If motor was enabled, motor goes into safe damping mode.
  When read:

0 - BEAR disabled
1 - BEAR enabled
2 - BEAR disabled and critical error preventing enabling the motor
3 - BEAR in safe damping mode due to non-critical error.

- **goal_id** Reference excitation current Id input. Leave it at 0 for normal operation.
- **goal_iq** Reference torque current Iq (unit: A). Can be written to when BEAR is in mode 0 and 3. Iq is roughly proportional to the output torque.
- **goal_velocity** Reference velocity (unit: rad/s). Can only be written to when BEAR is in mode 1.
- **goal_position** Reference position (unit: rad). Can be written to when BEAR is in mode 2 and 3.
- **present_id/iq/velocity/position** Present status value of BEAR. Read only.
- **input_voltage** Present power supply voltage to BEAR (unit: V). Read only.
- **winding_temperature** Winding temperature reading (unit: °C).
- **powerstage_temperature** MOSFETs temperature reading (unit: °C).
- **ic_temperature** Temperature reading (unit: °C) of the micro controllers.

2.2.3 Error Code

A BEAR will always return its present error status by returning an 8-bit error code together with every returned data. The highest bit of the error code is always 1. Refer to table. 5 for detailed explanation of every bit in the error code.

bit	Type	Name	Note
0	Warning	Communication	
1	Warning	Overheat	
2	Error	Absolute Position	
3	Error	Watchdog Timeout & ESTOP	
4	Error	Joint Limit	
5	Error	Hardware Fault	
6	Error	Initialization Error	
7	1		Always 1

Table 5: Table of Error Code (little-endian)

Detailed Description

- **Communication** A corrupted data packet was received. This warning resets automatically and is only associated with corresponding round of communication.
- **Overheat** The temperature of at least one component among IC, powerstage and winding in this BEAR has exceeded the value written to temperature_limit_low. This warning resets automatically when the temperature_limit_low value is higher than the highest temperature measured in this BEAR module.
- **Absolute Position** Absolute position reading error.
- **Watchdog Timeout & ESTOP** When in mode 0(torque mode), motor enabled and watchdog timer configured, watchdog timeout triggers this error.

  External ESTOP signals also triggers this error. Including physical signal and writing 0x03 to torque enable.
- **Joint Limit** Joint limit exceeded.
- **Hardware Fault** Input voltage out of range or MOSFET driver fault.
- **Initialization Error** Corrupted save file in flash, calibration needed.

2.3 Operating Modes

BEAR actuator can run in the following four different modes:

WARNING

Pay extra attention when changing mode while BEAR is enabled. BEAR will remain enabled and execute the corresponding goal_xxx setting in the new mode immediately.

0 - Torque Mode
The torque current(iq) in BEAR is directly controlled in this mode. Control commands instructing the torque current i_q is written to status register goal_iq, and the unit of the input is Amps. BEAR tracks goal Iq and goal Id using PID gains for Iq and Id.

The output torque T can be roughly estimated using the torque current i_q and the torque constant K_T of the BEAR module as shown in Eq. 1:

$$T = i_q \times K_T \tag{1}$$

CAUTION

When the current command goal_iq is higher than the limit_i_max setting, the BEAR module will not execute the command.

CAUTION

The actual maximum torque of BEAR is limited by the maximum current the power supply can provide as well as the limit_i_max setting.

1 - Velocity Mode

The output speed of BEAR is directly controlled by user input via status register goal_velocity. The unit of the input is rad/s. BEAR tracks goal velocity using PID gains for velocity loop, and the PID output feeds into Iq.

CAUTION

When the speed command goal_velocity is higher than the limit_velocity_max setting, the BEAR module will not execute the command.

CAUTION

The actual maximum speed of BEAR is limited by the supply voltage and the maximum speed it can achieve under the given load conditions; The actual maximum torque of BEAR is limited by the maximum current the power supply can provide as well as the limit_i_max setting.

2 - Position Mode

The output position of BEAR is directly controlled by user input via status register goal_position. The unit of the input is rad. BEAR tracks goal position using PID gains for position loop, and the PID output feeds into velocity.

CAUTION

When the position command goal_position is out of the range defined by the settings in limit_position_max and limit_position_min, the BEAR module will not execute the command.

CAUTION

The actual output speed of BEAR is limited by the supply voltage and the maximum speed it can achieve under the given load conditions as well as the limit_velocity_max setting; The actual maximum torque of BEAR is limited by the maximum current the power supply can provide and the limit_i_max setting.

3 - Direct Force Mode
In this mode, BEAR tracks goal position and velocity using PID gains, in addition to goal Iq. In another word, user can use this mode to command BEAR to track a trajectory that contains all of position, velocity and torque data. The diagram in fig. 10 shows how the signals are mixed internally.

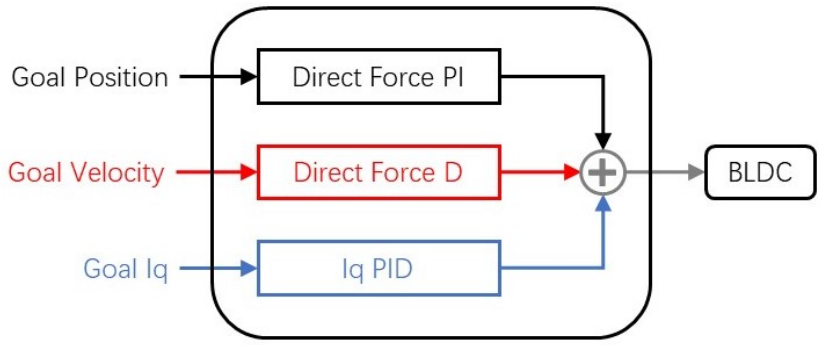


Figure 10: Direct Force Mode Diagram.

CAUTION

When the speed command goal_velocity is higher than the limit_velocity_max setting, or the goal_iq is higher than the limit_i_max setting, the BEAR module will not execute the command.
BEAR will still execute the command even if goal_position is out of the range defined by limit_position_max and limit_position_min in this particular mode.

CAUTION

The actual output speed of BEAR is **ONLY** limited by the supply voltage and the maximum speed it can achieve under the given load conditions; The actual maximum torque of BEAR is limited by the maximum current the power supply can provide and the limit_i_max setting.

WARNING

In Direct Force Mode, the output speed of BEAR is not limited by the limit_velocity_max setting.

2.4 PID Tunning

It is very important for the PID gains to be well tuned for a BEAR to function as desired, and there can be multiple sets of PID gains that need to be tuned to suit a BEAR into it's designated tasks, depending on the BEAR's operating mode.

0 - Torque Mode
Only the iq and id loops are involved in this mode, thus only the PID gains of these two loops need to be tuned. The PID gains of iq and id loops should always be the same in normal operation, thus there is actually only one set of PID gains to be tuned when BEAR is operating in this mode.

It is recommended to start with the settings as specified in table. 6 which should work just fine for most applications.

BEAR	P Gain	I Gain	D Gain
Koala V2	0.277	0.061	0
Koala Muscle Build V1	0.358	0.045	0
Panda V2	0.099	0.039	0
Panda Plus V2	0.184	0.065	0
Kodiak V1	0.25	0.017	0

Table 6: Table of Current Loop Gains

BEAR with correct id/iq PID gains shows great id/iq tracking and maintains relative low noise on both current loops. The P gain and I gain should not deviate from the above by a significant amount even when further tuning is needed, however, in some applications where exceptional noise is seen in the current loops, you can try scaling down these P gain and I gain settings, and the D gain should be kept at zero(0) all the time.

1 - Velocity Mode
When operating in Velocity Mode, the velocity PID loop result feeds to iq loop, thus the velocity PID gains are tuned on top of correct id/iq PID gains. See previous sector for how to tune the id/iq PID gains correctly.

While the actual settings could vary depending on the application and desired dynamic performance, it is recommended to start with the following settings:

Velocity $\quad P = 0.5 \sim 1, I = 0, D = 0$

A very small ($0 \sim 0.001$) I gain may be needed depending on the application, but it is recommended to keep the D gain at zero(0).

BEAR with correct id/iq PID gains and well tuned velocity PID gains shows good velocity tracking performance.

2 - Position Mode
Getting the correct PID settings for BEAR to run dynamically and accurately in Position Mode can be a little challenging, but it is simple and straight forward once it is tuned step by step in the correct sequence.

When in Position Mode, the position PID loop result feeds to velocity loop, and then the velocity PID loop result feeds to iq loop, thus all PID gains of the id/iq loop, velocity loop and position loop need to be correct. Therefore, a fairly reliable set of velocity loop gains is required before the position gains can be tuned. Refer to previous sector for how to tune the velocity PID gains correctly.

The actual PID setting for position loop varies dramatically between different types of BEAR and their applications. Always get the velocity and id/iq PID gains correct **BEFORE** spending time in tuning the position PID. Bad velocity and/or id/iq gains can be the reason why a BEAR is not outputting enough torque or not tracking goal position, no matter its position gains.

3 - Direct Force Mode

When operating in Direct Force Mode, all of position, velocity and iq can be involved as explained in section. 2.3, and eventually feeds into the internal Iq loop, thus a well tuned set of id/iq PID gain is fundamental.

In this mode, BEAR closely simulates a spring-damper system. The Direct Force P gain is multiplied by position error in rad and the Direct Force D gain is multiplied by velocity error in rad/s. It's good to start with D = 0.1P when tuning and any I gain is highly depreciate as it will likely lead to oscillation.

Depending on the application and desired trajectory to track, iq and velocity can also be part of the goal command sent to BEAR. In this case, velocity is fused into the loop with the Direct Force D gain, along with iq based on id/iq PID gain.

See previous sector for information on how to tune the id/iq PID gains.

3 SDK

3.1 PyBEAR

In this section, the **BEAR**™ actuator driver Python SDK, called **PyBEAR**™ is introduced. PyBEAR is compatible with Python3, supported functions are introduced as follows.

3.1.1 Getting Started

Follow these steps to get started with PyBEAR:

0) **Serial port access permission** Make sure you have the permissions to access the serial ports on your computer. Adding such permission to your account in Linux can be done with the following commands in terminal:

```
sudo chown -R your_username /usr/local
sudo usermod -a -G dialout your_username
```

1) **Unzip** After downloading the PyBEAR zip file, unzip the file into a preferred path, and we will refer to this path as `/usr_path` from this point forward.

2) **Install dependencies** PyBEAR requires the NumPy module for scientific computing and PySerial module to access the serial port. Install these two packages in bash as follows(Python 3):

```
pip3 install numpy pyserial
```

3) **Install PyBEAR** Cd into `/usr_path/PyBEAR` and run the installation(Python 3):

```
python3 setup.py install
```

Start enjoying PyBEAR!

3.1.2 Communication With BEAR

To use PyBEAR, import the driver first:

```
# Import Manager from PyBEAR
from pybear import Manager
```

0) **Connect to/Disconnect from BEAR** The serial port connection needs to be established before any command can be transferred to the BEAR actuators that are connected. The serial port can be connected by creating a serial port object.

Example:

```
# Create a serial port object
# A typical port will be /dev/ttyUSB*
# The default (and fastest) baud rate is 8000000 bps
bear = Manager.BEAR(port='/dev/ttyUSB0', baudrate=8000000)
```

To disconnect from BEAR(clear existing PyBEAR instance), simply call the *close()* function:

```
bear.close()
```

1) **Ping BEAR** The ping function can be used to detect if one or multiple BEARs are online, or to read their error codes. It will return firmware and hardware version information along with error code if successfully pinged, or 'None' for the specific BEAR(s).

   Example:

```
# Ping BEAR with ID 1
rtn = bear.ping(1)[0]
if bool(rtn):
    pass
else:
    print("BEAR 1 not detected.")
```

2) **Read from BEAR** The value of configure and status registers of BEAR can be read by calling "*get_*" functions.

   a) Read from a single register

   The functions have a format of:

```
get_name-of-register(motor_id)
```

   Register value of multiple BEAR actuators can be read at one time, and the function return format is:

```
[([data_of_motor1], error_code_of_motor1),
 ([data_of_motor2], error_code_of_motor2) ...]
```

   Example:

```
# Read torque_enable status from BEAR 1, 2, 3
bear.get_torque_enable(1,2,3)

# Read p_gain_position from BEAR 1, 4
bear.get_p_gain_position(1,4)

# Read present_velocity from BEAR 2, 3
bear.get_present_velocity(2,3)
```

   If bad connection happens and nothing is heard from the BEAR actuator(s) or the return packets are damaged, PyBEAR automatically retries the read function, and a warning is generated:

```
[PyBEAR | WARNING] :: Read response timed out. Re-sending the same packet.
```

   b) Read from multiple registers

CAUTION

This function is previously known as get_bulk_status/config(), which is verbally misleading since this is in fact not using bulk communication protocol. get_bulk_status/config() is now depreciated.

CAUTION

Will not work for ID, Mode, Baudrate, Watchdog_timeout and Torque Enable

Multiple registers on a BEAR can be visited within one frame of communication using:

```
get_status/config((ID1, reg1, reg2...), (ID2, reg1, reg2 ...) ...)
```

Multiple target motors can be visited but PyBEAR will go through them one-by-one.

CAUTION

Do not use get_status() with config registers, nor to use get_config() with status registers

The function return format is:

```
[([data_from_motor1], error_code_of_motor1),
 ([data_from_motor2], error_code_of_motor2)...]
```

Example:

```
# Read present_iq and present_velocity from BEAR 1 and 2,
# and read present_velocity from BEAR 3
bear.get_status((1, 'present_iq', 'present_velocity'),
                (2, 'present_iq', 'present_velocity'),
                (3, 'present_velocity'))

# Read p_gain_position from BEAR 1, 4
bear.get_config((1, 'p_gain_position'), (4, 'p_gain_position'))
```

If bad connection happens and nothing is heard from the BEAR actuator(s) or the return packets are damaged, PyBEAR automatically retries the read function, and a warning is generated:

```
[PyBEAR | WARNING] :: Read response timed out. Re-sending the same packet.
```

3) **Write to BEAR** All configure registers and most status registers can be written to achieve proper control of BEAR modules by calling "*set_*" functions. The "*set_*" functions have no return.

a) Write to a single register

The functions have a format of:

```
set_name-of-register((motor_id, value))
```

Format of input should always be one or several instances of "*(motor_id, value)*" when writing to the same register of multiple BEAR actuators.

Example:

```
# Enable the torque of BEAR 1, 3 and disable BEAR 2
bear.set_torque_enable((1,1),(2,0),(3,1))

# set p_gain_velocity on BEAR 1 to 0.02, BEAR 4 to 0.05
bear.set_p_gain_position((1,0.02),(4,0.05))

# set goal_position of BEAR 1 to 0, BEAR 3 to 2.5
bear.set_goal_position((1,0),(3,2.5))
```

b) Write to multiple registers

CAUTION

This function is previously known as set_bulk_status/config(), which is verbally misleading since this is in fact not using bulk communication protocol. set_bulk_status/config() is now depreciated.

CAUTION

Will not work for ID, Mode, Baudrate, Watchdog_timeout and Torque Enable

Multiple registers on a BEAR can be visited within one frame of communication using:

```
bear.set_status/config((ID1, reg1, data1, reg2, data2...),
                       (ID2, reg1, data1...) ...)
```

Multiple target motors can be visited but PyBEAR will go through them one-by-one.

CAUTION

Do not use set_status() with config registers, nor to use set_config() with status registers

Example:

```
# set goal_position on BEAR 1 to 0, goal_velocity on BEAR 4 to 2
bear.set_status((1, 'goal_position', 0),
                (4, 'goal_velocity', 2))

# set BEAR 1 p_gain_position to 0.8 and d_gain_position to 0
bear.set_config((1, 'p_gain_position', 0.8, 'd_gain_position', 0))
```

4) **Bulk Communication** Writing to/reading from multiple **status** registers of one or more BEAR actuators within single communication frame can be done with bulk communication. It is highly recommended to use bulk communication functions when communicating with multiple BEARs or/and visiting multiple registers, especially the application is preferring very fast communication.

CAUTION

Bulk communication functions only support **status** registers and does **NOT** support Torque Enable.

CAUTION

Bulk communication functions can respectively read from and write to no more than 16 registers each time.

There are three functions for bulk communication:

CAUTION

Use LIST instead of TUPLE to construct commands when using bulk communication functions.

a) Read only

```
bear.bulk_read([ID1, ID2 ...], [reg1, reg2 ...])
```

Use this function to read the same cluster of status registers from multiple BEARs. The return has a format as:

```
[[[data_of_motor1], error_code_of_motor1],
 [[data_of_motor2], error_code_of_motor2] ...]
```

If BULK_COMM timed out before getting any reply, the following message will be displayed:

```
[PyBEAR | WARNING] :: BULK_COMM response timed out. Re-sending the same packet.
```

If BULK_COMM timed out before getting all requested data, the following message will be displayed:

```
[PyBEAR | WARNING] :: BULK_COMM return packet timed out. Retrying BULK_COMM...
```

Either way, bulk communication will retry for up to 3 times. If the communication fails after all retrial attempts, the function will return **None**. If there has been no reply at all, the following message will be displayed:

```
[PyBEAR | ERROR] :: BULK_COMM no return.
```

Otherwise, the following message will be displayed:

```
[PyBEAR | ERROR] :: BULK_COMM only get partial return.
```

If any of the target BEAR returned corrupted data, the returned data of such BEAR will be **None** and the corresponding error code will be **-99**. Note that this error is only associated with this particular communication attempt, rather than error code of BEAR as described in item. **6**).

b) Write only

```
bear.bulk_write([ID1, ID2 ...], [reg1, reg2 ...],
                [[ID1-data1, ID1-data2 ...], [ID2-data1, ID2-data2 ...]])
```

Use this function to write to the same cluster of status registers with different values to multiple BEARs. The function returns **True**.

a) Read then write

```
bear.bulk_read_write([ID1, ID2 ...], [registers to read],
                     [registers to write],
                     [[ID1-data1, ID1-data2 ...], [ID2-data1, ID2-data2 ...]])
```

This function will read data from the specified registers then write to the specified registers with given data. It's return behavior is the same as *bulk_read()*.

Examples:

```
# Set goal_position of BEAR 1, 2, 3 respectively to 0, 1, 2
bear.bulk_write([1, 2, 3], ['goal_position'], [[0], [1], [2]])

# Get goal_position and goal velocity of BEAR 1, 2
bear.bulk_read([1, 2], ['goal_position', 'goal_velocity'])

# Get winding_temperature of BEAR 1, 2, 3
# and write new values to their goal_iq and goal_id
bear.bulk_read_write([1, 2, 3], ['winding_temperature'],
                     ['goal_iq', 'goal_id'], [[1, 0], [2.5, 0], [-0.3, 0]])
```

5) **Configuration vs Status Registers** Certain registers are ***Configuration Registers*** which are persistent after shutdown, and need to be saved once they are changed, otherwise restored to last saved value after power-cycle. Whereas ***Status Registers*** are non-persistent (volatile) and will be restored to default values when the BEAR is powered on and get populated with corresponding real time status of BEAR.

After updating ***Configuration Registers***, the values can be saved with "*save_config(motor_id)*" function. An example of how to set the ID of a new BEAR module:

```
# Set the ID of motor (1) (default) to (4)
bear.set_id((1,4))
# Save the configuration using the new motor ID
bear.save_config(4)
```

CAUTION

"*save_config*" currently only works for a single ID. The following command with NOT work:

```
bear.save_config(1,2,3)
```

WARNING

Do not save configurations when the motor is enabled. The motor may not respond when it is writing flash memory.

Configuration Registers include:

- Motor ID
- Operation Mode
- Baud Rate
- Home offset
- Limits (position, velocity, current, voltage, temperature)
- PID gains

Status Registers include:

- Torque enable/disable
- Goal position/velocity/current
- Present position/velocity/current/voltage/temperature

Refer to Table.4 for a full list of supported registers.

6) **Error Code** All read functions will return with error codes, with the requested data and error code of each BEAR form a <tuple>.

Example:

```
# Get present position of BEAR 1
bear.get_present_position(1)

-> return: [([0.0855], 128)]
# 0.0855 is the present position and 128 is the error code
```

Error code 128 is 'Normal'. Refer to table.5 for detailed meanings of the error code.

7) **Timeout** The default communication timeout as well as the bulk communication timeout are set to be 0.001s(1ms). When PyBEAR displays these messages:

```
[PyBEAR | WARNING] :: Read response timed out. Re-sending the same packet.

[PyBEAR | WARNING] :: BULK_COMM response timed out. Re-sending the same packet.

[PyBEAR | WARNING] :: BULK_COMM return packet timed out. Retrying BULK_COMM...
```

it is most likely that there is a faulty signal connection or the target is offline. However, if you believe that your hardware setup is all normal, or simply for debug reasons, you can change the timeout settings of existing PyBEAR instance:

```
# Normal communication timeout:
bear.timeout = new_value
# Bulk communication timeout:
bear.bulk_timeout = new_value
```

or specify values when instancing PyBEAR:

```
# Create a serial port object at /dev/ttyUSB0 with baud rate of 8Mbps
# timeout set to 5ms and bulk_timeout set to 3ms
bear = Manager.BEAR(port='/dev/ttyUSB0', baudrate=8000000,
                    timeout=0.005, bulk_timeout=0.003)
```

3.2 LabBEAR

In this section, the **BEAR**™ actuator driver LabVIEW SDK, called **LabBEAR**™ is introduced as follows.

3.2.1 Getting Started

Follow these steps to get started with LabBEAR:

0) **Check your LabVIEW version** LabBEAR requires LabVIEW 2016 or higher versions. So please upgrade your LabVIEW if you have earlier versions.

1) **Download** Download LabBEAR SDK zip package and unzip it into a preferred work directory.

2) **Install dependencies** LabBEAR requires NI VISA module to communicate with BEAR via USB ports on your computer. The installation can easily be down with NI Package Manager.

Start enjoying LabBEAR!

3.2.2 Communication With BEAR

0) **Open a VISA port** Use *VISA Configure Serial Port* followed by *VISA Set I/O Buffer Size* and *VISA Flush I/O Buffer* to properly establish communication with your BEAR via USB.

Find *VISA Configure Serial Port* subVI under: *Instrument I/O* → *Serial* as shown in figure 11.

Then Find *VISA Set I/O Buffer Size* and *VISA Flush I/O Buffer* under: *Instrument I/O* → *VISA* → *Advanced* → *Bus Specific* as shown in figure 12.

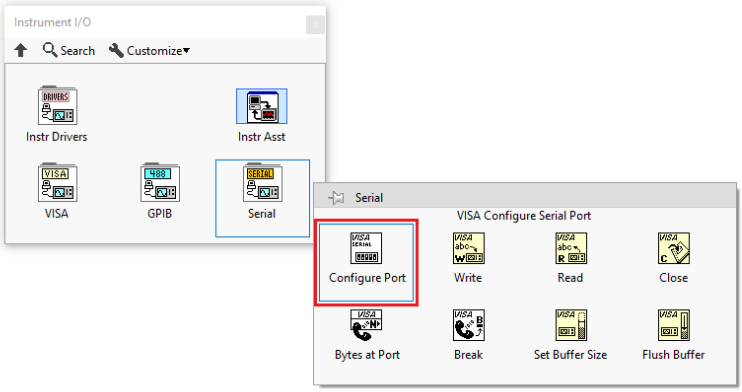


Figure 11: VISA Configure Serial Port

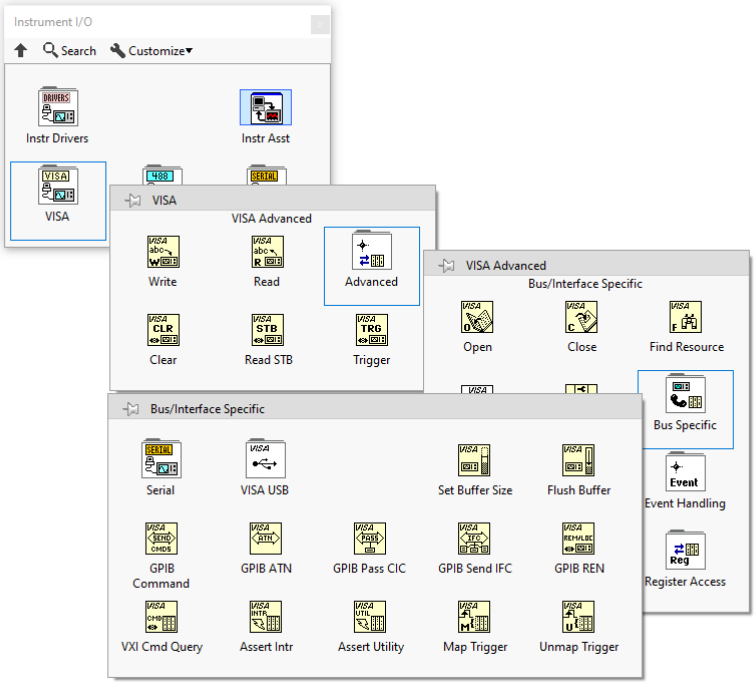


Figure 12: VISA Set I/O Buffer Size and Flush I/O Buffer

Connect the above subVIs as shown in figure 13, connect controls for VISA resource name and baud rate on *VISA Configure Serial Port* and specify 200ms for timeout and FALSE for termination char. Pass on the VISA resource name and error to *VISA Set I/O Buffer Size* and

specify an I/O Receive Buffer of 10000 bytes. Finally pass on the VISA resource name and error to *VISA Flush I/O Buffer*, and VISA resource name and error lines are ready for use.

Note that if the com port name and baud rate is predetermined, you can use constants instead of controls for them. Otherwise, specify the corresponding port name and baud rate in the front panel as shown in the example in figure 13.

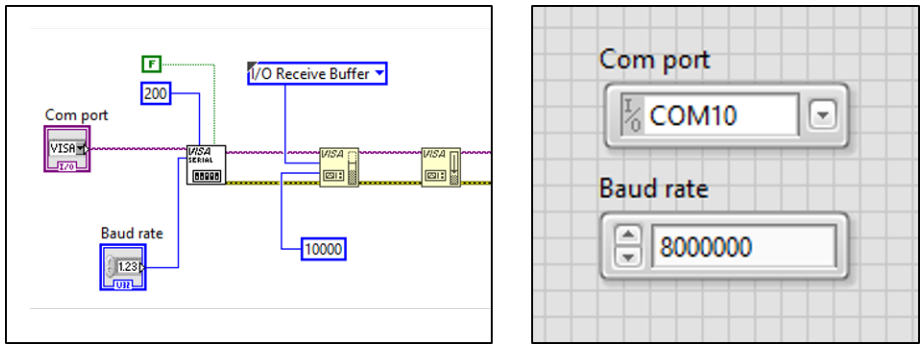


Figure 13: Open a VISA port to communicate with BEAR.

1) **Read from BEAR** The value of configure and status registers of BEAR can be read by respectively using the *Read_Config* and *Read_Stat* subVIs, as shown in figure 14.

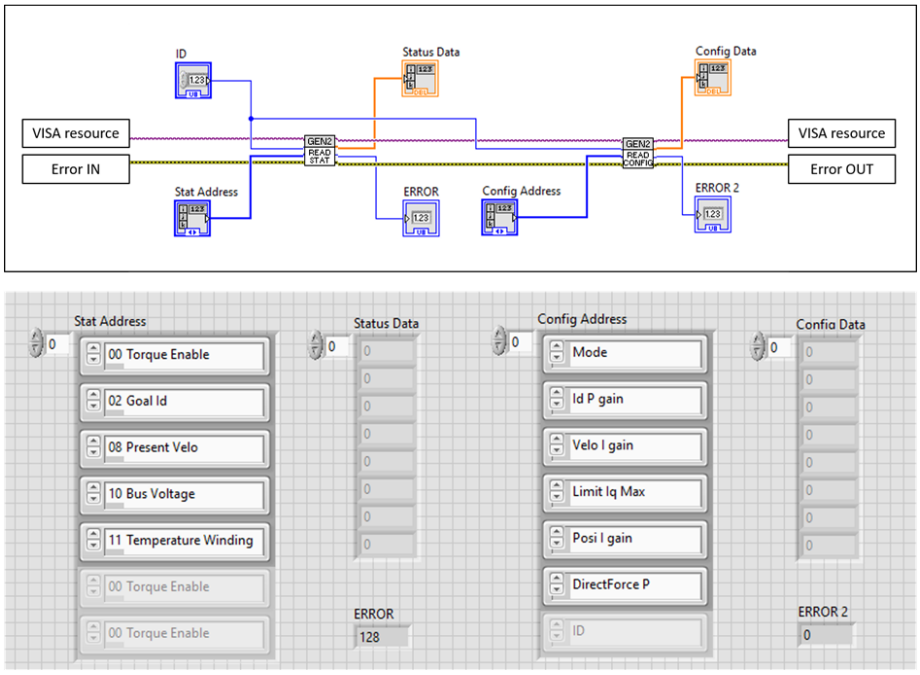


Figure 14: Read from BEAR.

To use the read subVIs, connect VISA resource and Error lines to corresponding terminals, then specify the ID of the target BEAR as well as the address of the target registers to read from. This can be done by creating controls from the corresponding terminals. Multiple registers can be read at once as shown in figure 14. Both subVIs return the requested data as well as error code from the target BEAR.

2) **Write to BEAR** You can write to supported configure and status registers by respectively using the *Write_Config* and *Write_Stat* subVIs, as shown in figure 15.

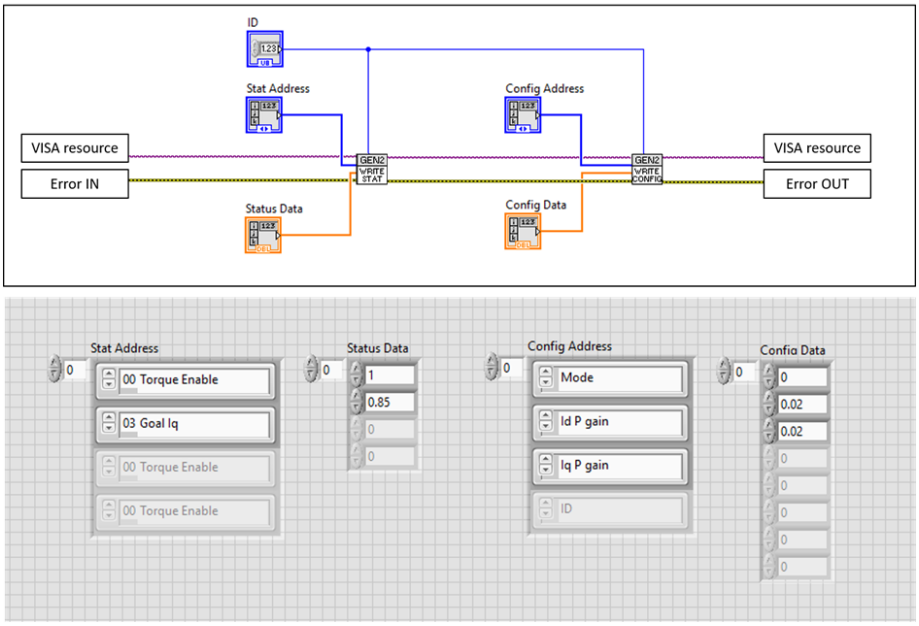


Figure 15: Write to BEAR.

To use the write subVIs, connect VISA resource and Error lines to corresponding terminals, then specify the ID of the target BEAR as well as the address and value of the target registers to write to. This can be done by creating controls from the corresponding terminals. Multiple registers can be written at once as shown in figure 15.

Configuration vs Status Registers

Certain registers are Configuration Registers which are persistent after shutdown, and need to be saved once they are changed. Status Registers are non-persistent (volatile) and will be restored to default values when the BEAR is powered on and get populated with corresponding real time status of BEAR.

After updating Configuration Registers, the values can be saved with *Save_Config* subVI as shown in figure 16.

WARNING

Do not save configurations when the motor is enabled. The motor may not respond when it is writing flash memory.

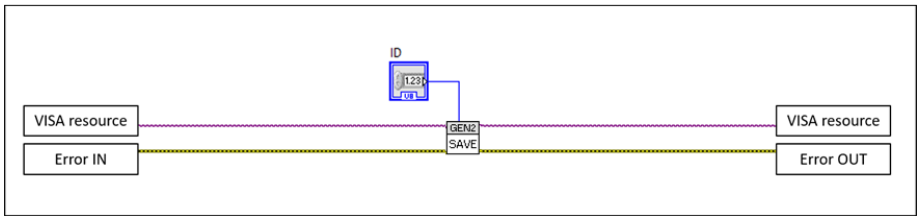


Figure 16: Save configuration registers.

Configuration Registers include:

- Motor ID
- Operation Mode
- Baud Rate
- Home offset
- Limits (position, velocity, current, voltage, temperature)
- PID gains

Status Registers include:

- Torque enable/disable
- Goal position/velocity/current
- Present position/velocity/current/voltage/temperature

Refer to Table.4 for a full list of supported registers.

3) **Bulk Communication** Writing to/reading from multiple **status** registers of one or more BEAR actuators can be done with the *Bulk_Stat* subVI conventionally all at once. Use this function as shown in figure 17.

CAUTION

Bulk communication with *Bulk_Stat* subVI can only be used with supported **status** registers.

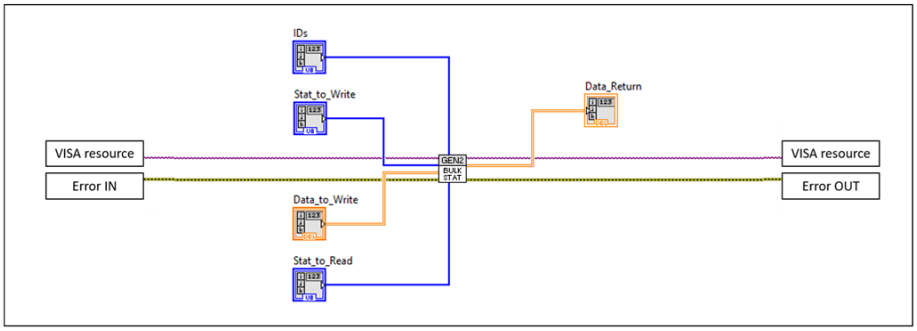


Figure 17: Bulk read/write with status registers.

4) **Examples** You can find the following examples in the LabBEAR package:

- **SIMPLE_Read_Stat** A simple example of reading status registers from one BEAR.
- **SIMPLE_Read_Config** A simple example of reading configuration registers from one BEAR.
- **SIMPLE_Write_Stat** A simple example of writing data to status registers on one BEAR.
- **SIMPLE_Write_Config** A simple example of writing data to configuration registers on one BEAR.
- **SIMPLE_Bulk_Stat** A simple example of bulk communication.

5) **Tuning Interface: RS485_Data_Ctrl** When it comes tuning or debugging your BEAR using LabVIEW, the **RS485_Data_Ctrl** VI is a very handy tool. The front panel of Data_Ctrl is as shown in figure 18, and it is introduced section by section as below.

a. Port setting Configure the port settings here to match your BEAR before you run the VI.

b. ID, Enable and Error You can chain multiple BEARs and tune them one by one with the ID setting. You can change the ID while the VI is running. Use the **Enable** and **DISABLE** button to enable/disable the current BEAR and use the **ESTOP** button to put the current BEAR into E-stop mode. Use the **STOP** button to stop the VI. The error code of the current BEAR is displayed in the ERROR indicator and Error count indicator displays how many times an error has accrued since the VI started.

c. Mode, Limits and Goals Use this section to write mode, limits and goals to the current connected BEAR, and you the **SAVE** button to save the current config register values.

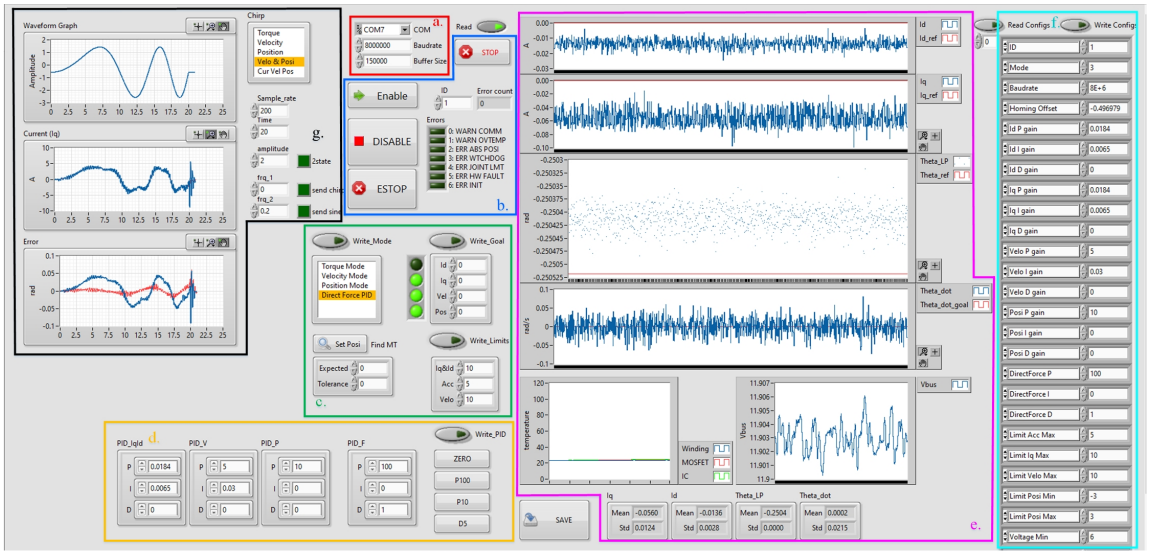


Figure 18: The front panel of Data_Ctrl VI.

> **CAUTION**
> The VI may freeze for a brief second when saving the config registers.

d. PID setting Use this section to change the PID settings of the current connected BEAR.

e. Real-time Status Status of the current connected BEAR is plotted in real-time in this section, including Excitation Current id, Torque Current iq, Position Theta, Velocity Theta_dot, Temperatures and Present Supply Voltage Vbus.

f. Read Config Click the **Read Config** button to read all config registers once. Returned data will be displayed in the indicator. You can also change specific config settings and hit **Write Config** button to update the current connected BEAR.

g. Chirp Signal A chirp, sinusoidal or 2-state(square) wave can be sent to the current connected BEAR for tuning purpose. Define desired wave to send in the controls, and it can be sent as goal_torque, goal_velocity or goal_position by choosing the corresponding selection. Click on the corresponding green square button to start sending the wave. This function is extremely helpful if a BEAR needs to be fine-tuned for a certain band of response frequency.

4 Version History

— Update Log: —

0.1.2 Complete Python section, Introduction now contains everything about KB02

0.1.3 Add LabBEAR

0.1.4 Add Warning section in Intro

0.1.5 Modified KT value and torque specs.

0.1.6 Add details on PB02; Add connection instruction.

0.2.0 A couple of updates:

a. Move connection, PID tuning and control table all into a new section: Using BEAR.

b. Add units to regs and correct reg value range.

c. Add more details to PID tuning.

d. Update file structure.

0.2.1 Update Ping function output: it outputs [([List of firmware and hardware versions], error), ...]

0.2.2 Update Specification

0.2.3 Officially support BULK_COMM in PyBEAR.

a. Add detailed description and examples of related functions.

b. Depreciate get/set_bulk_status/config() as these are in fact not bulk_comm and replace with get/set_status/config().

c. User can specify timeout and bulk_timeout when instancing pybear.

0.2.4 Update LabBEAR to 2.1: Update Introduction section to include the usage of ESTOP

0.2.5 Update PB02 Spec figure and signal port access instruction

0.2.6 Small Bug fix:

a. Fix typo in PyBEAR multi-communication.

b. Fix daisy chain fig missing Estop wire.

0.2.7 Minor improvements.

0.2.8 Update recommended current loop PID gains.

0.2.9 Fix a wrong statement about velocity in DF mode

0.3.1 Fix a wrong labeling about U2B pinout

0.3.2 Add recommended current loop PID gains for Kodiak BEAR V1(CB01) and Koala BEAR Muscle Build V1(KBMB01), and attach version history at the end of manual. Add Kodiak voltage range. Officially stopped support for python2, Section 3. Update BEAR spec table as well as

introduction to include KBMB01 and CB01. (CB01 related information only available to existing customers)

0.3.3 Modify Figure 6: USB2BEAR pin-out and switches for better understanding.