English

Petoi Bittle User Manuals

Rongzhong Li

> ⚠ **Under Construction**
>
> You may find some complementary materials at: https://www.yuque.com/tinkergen-help-en/bittle

---

Getting started

To keep these instructions simple to use, I'm focusing on the assembling rather than an in-depth explanation.

Petoi Bittle

- OpenCat framework
- Panning head
- Arduino compatible microcontroller
- clip
- wireless connectivity
- detachable neck
- Interlocking plastic frame
- 9 x Fast & powerful metal servos
- Shock reduction joints
- Programmable gaits and behaviors
- Rechargeable Li-ion battery

* Patent Pending

If you have specific questions on "why" rather than "how", please post on our forum at https://www.petoi.com/forum or write to support@petoi.com.

You can support us by shopping at www.petoi.com/store. Our social media (Instagram/Twitter/Facebook/GitHub) account is @**PetoiCamp**. Share your build by tagging **#bittle #petoi #opencat** so that we can repost for you!

1 🧔♂ Tool and Preparation

"A beard well lathered is half shaved." 🪒

1.1. Preparation

Prepare a clean desk and some small boxes to unzip the package. Take a picture of the kit contents in case you lost something later.

> ✓ It's better to work in a room without carpet or textured mosaic. Little screws and springs can magically disappear if dropped onto the ground.

1.2. Tools and accessories

Required

Tool	Notes
Flat and Phillips screwdrivers	For M2 (diameter = 2mm) screws
Computer with Arduino IDE	Install the latest Arduino IDE
USB charging port	5V 1A output should be enough

Optional

Tool	Note
Soldering iron w/ accessories	For modifying the PCB if you are a hacker!
3D printer w/ accessories	Add your special design

Arduino/Raspberry Pi kit	Add more gadgets to Bittle
Multimeter	Test and debug
Oscilloscope	Test and debug
Hot glue/super glue	Avoid using them. Bittle is designed to be ...

2 □ Open the Box

"life is like a box of Bittle." □

Unassembled Bittle Kit and Pre-assembled Bittle

2.1. Pre-assembled Bittle

If you received the pre-assembled Bittle, you need to insert the neck into the body and **bend the knees to natural angles**. Drag the curly wire from the knee side to the shoulder side to avoid squeezing when the

Long press the battery's button for 2~3 seconds to power on/off. **Unplug the insulation sheet** of the infrared remote's battery, then you can control Bittle to move. The robot will keep pausing its movements with beeping sounds when the battery is low. Then you need to charge the battery with a 5V micro-USB cable. Considering safty, the battery won't supply power during charging.

Buzzer beep meaning:

Sound type	Occasion	Explaination
Short melody	Power up or reboot	The program starts successfully
Short beep	During use	The program receives a command
Repetitive 3 beeps	During use and pausing the movements	The battery is low or unconnected

> ⚠ **The pre-assembled Bittle is only coarse-tuned.** You still need to setup Arduino IDE to calibrate Bittle's sensors and fine-tune its joints for the best performance.

> ⚠ Friction plays an important role in dynamic balancing during walking. Though the silicone toe

covers (socks) can improve grip, they will also amplify the differences of the unpredictable surface of your test environment. So for regular use, we recommend running Bittle without the socks, unless you can tune the gait or need friction to perform some specific tasks.

2.2. Unassembled Bittle Kit

If you ordered the unassembled Bittle kit, you will have to solve a challenging and fun puzzle. You will learn more about robotics after understanding the design logic behind Bittle. Please move to the following chapters for details.
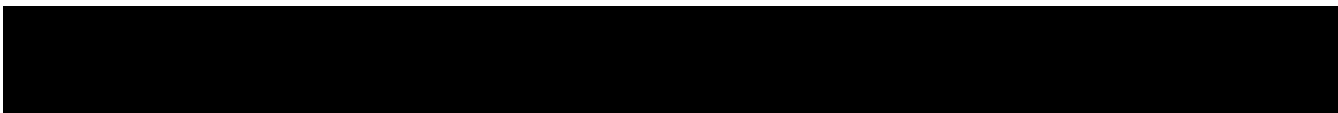
3 Assemble the Frame

"The whole is more than the sum of its parts."

3.1. Start with the Unassembled Kit

You can take a glance at this assembling animation for Bittle.

Here's a more detailed hands-on tutorial on building Bittle from the kit.

3.2. Step-by-step Instruction (under construction)

You can finish the following parts then put them together in later chapters.

- Neck
- Body
- Upper leg
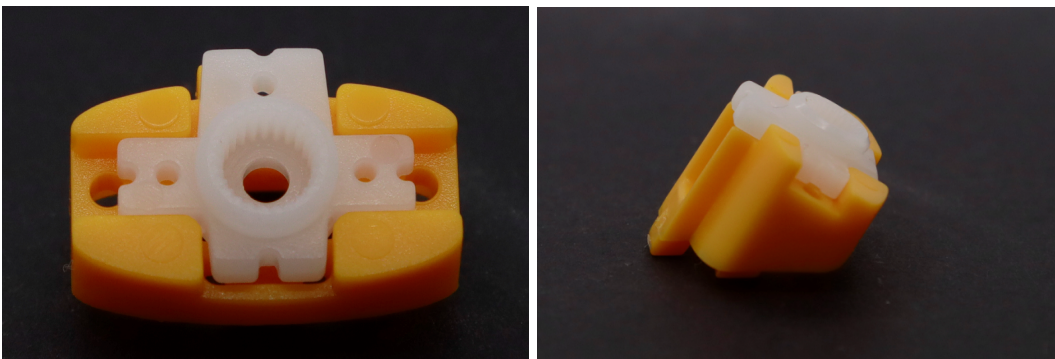- Lower leg
- Head
- Shoulder servo

3.2.1. Neck

Materials：

- Neck x 1
- Servo arm x 1
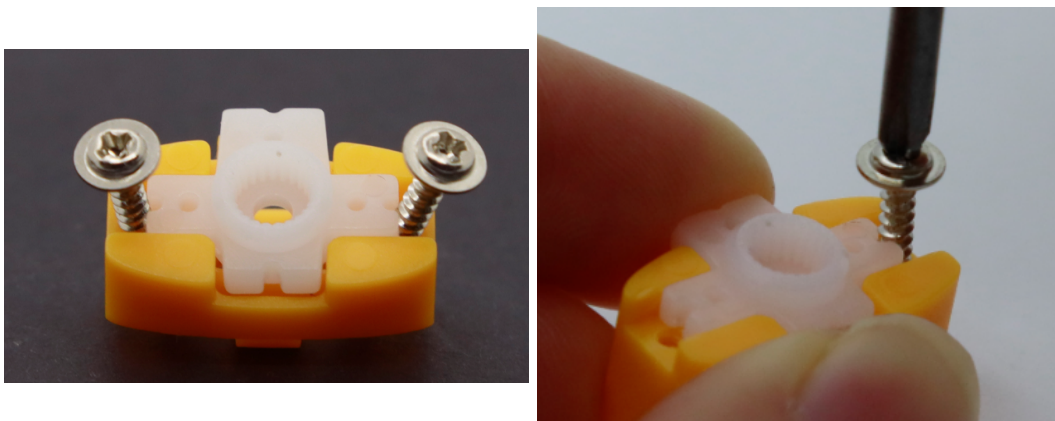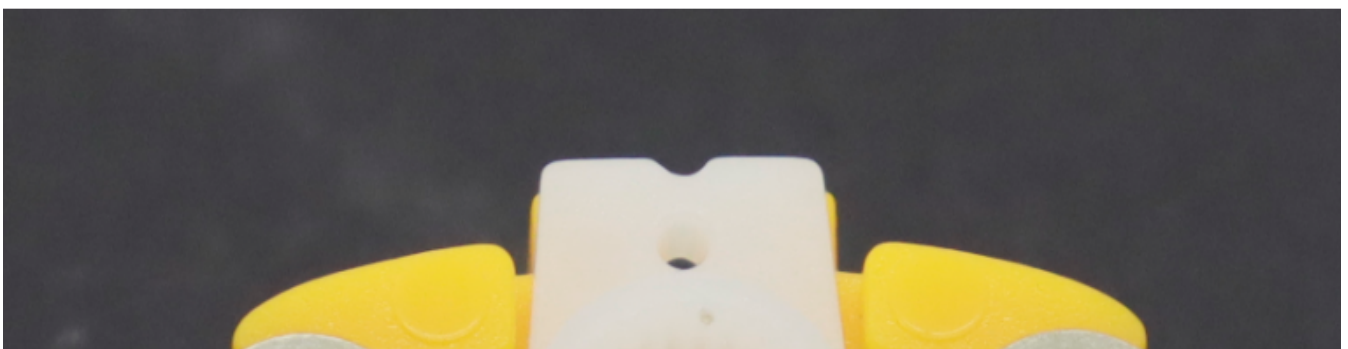- M2x8 sharp-end self-tapping screws x 2

Put the servo arm in the neck part like the figure below. The teeth of the servo arm should face upward.

Screw two M2x8 self-tapping screws beside both sides of the servo arm. Don't over tighten the screws.
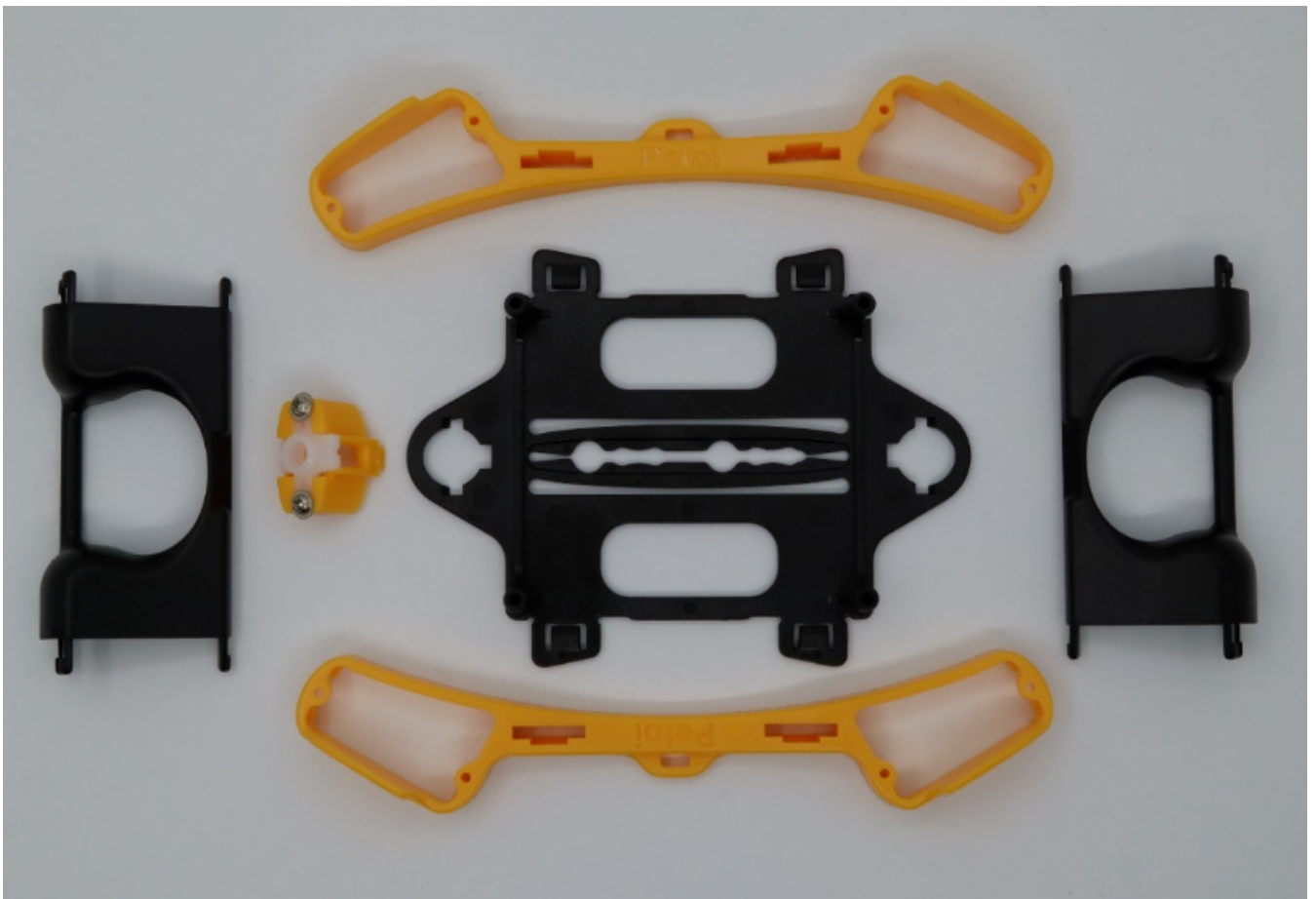

Finished:

### 3.2.2. Body (pre-assembled in our package)

Materials:

- Assembled neck x 1
- Chasis x 1
- Front and back plate x 2
- Side shoulder x 2

Recognize the front and back side of the chassis. The location of the two large holes along the center track makes a difference. In our standard configuration, we define the holes to be shifted towards the tail. You can also find a small mark "A1" in the front of the chassis.

Push the back tip of the assembled neck into the slot of the chassis. Then press down the front hook until you hear a snap sound.
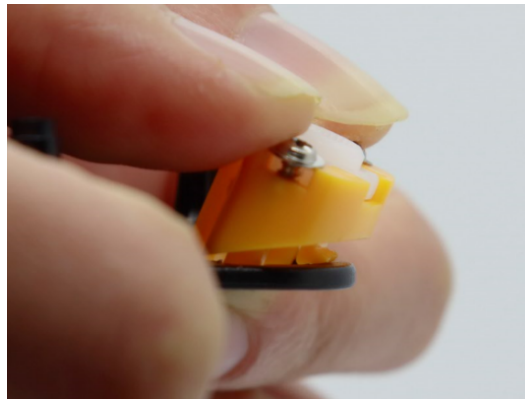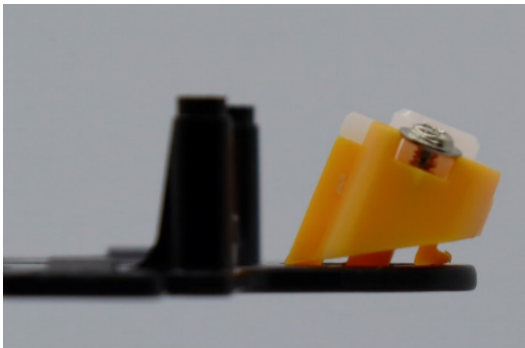
> (i) I designed the neck as a weak chain to disengage during a collision. Otherwise, the shock will be passed on to the servo or body frame and cause unrepairable damage.

Insert the chassis into one of the side shoulders.

Insert the front or back plate into the side shoulder. There are three tenons on each side of the plate. Insert the two front tenons as one group for better alignment.

Insert both the front and back plates.

Plug the other side shoulder to complete the body. Again pay attention to the alignment of the tenons of the front and back plates.
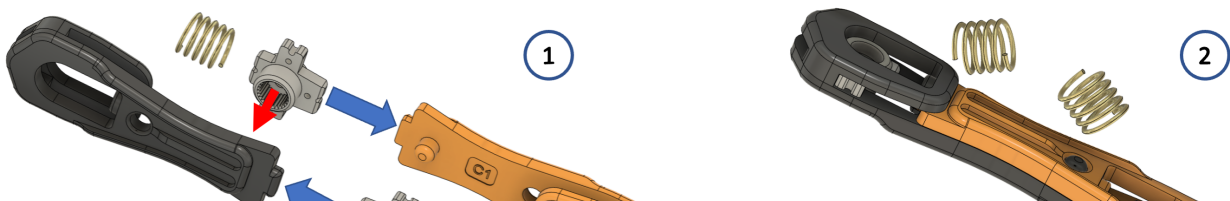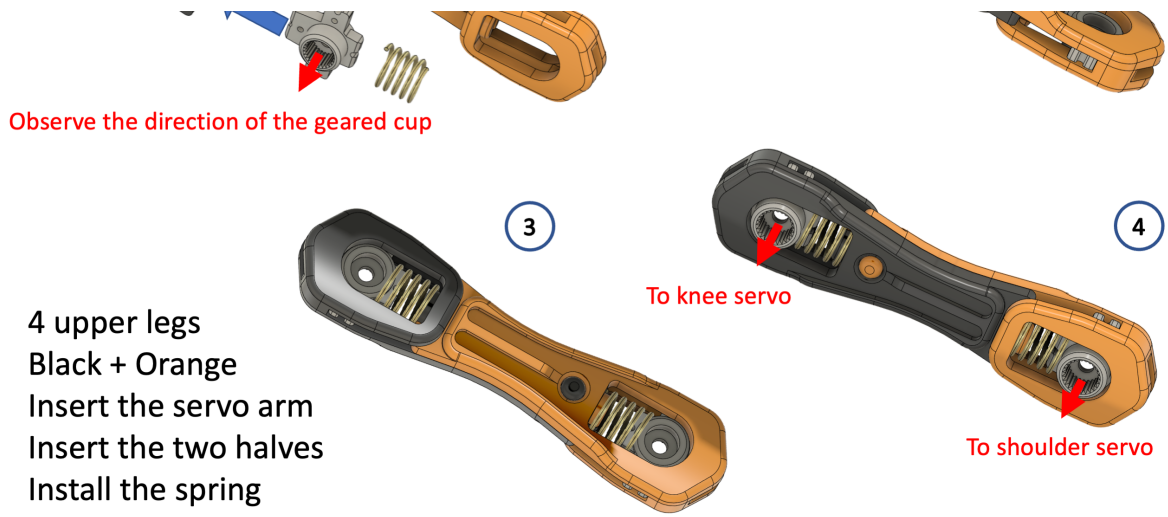
Finished:

3.2.3. Upper leg

It's a little hard to explain well with words. Please refer to the assembling instruction video at 3:23. It requires more skill than force. There's a forum post discussing installing springs with various tools.

Observe the direction of the geared cup

- • 4 upper legs
- • Black + Orange
- • Insert the servo arm
- • Insert the two halves
- • Install the spring

③

To knee servo

④

To shoulder servo

3.2.4. Lower leg

Materials:

- • P1S servo with long cable x 4
- • Lower leg piece x4
- • M2x8 self-tapping screw x 8

Assemble the right leg. Insert the servo into the window. Watch the tutorial video to see the correct direction to fold the wire. The notch on the internal edge is designed to let the wire go through. Use two M2x8 screws to fix the servo onto the lower leg.

The left leg is opposite to the right leg. However, the front and back legs are identical. So you should make two pairs of legs with opposite configuration.

3.2.5. Head

Materials:

- P1S servo with short cable x 1
- Chin x 1
- Skull x 1
- M2x8 self-tapping screw x 2

Put the servo in the chin as shown in the figure. Pay attention to the direction of the servo's wire. After that, install two M2x8 self-tapping screws.

Insert the skull into the chin so that it can rotate and bite on small gadgets. I recommend you apply some lubricant to the contact points between the skull and chin.

3.2.6. Shoulder servo

Materials:

- P1S servo with short cable x 4
- M2x8 self-tapping screws x 8
- Head (assembled)
- Body (assembled)
- Lower leg (assembled) x 4

Put the head, body, servos, and lower legs like the figure below. Insert the short servo wires through the servo slots on the side shoulders. Pay attention to the direction of the servos carefully and place them in the correct configuration.

Side view:

Top view:
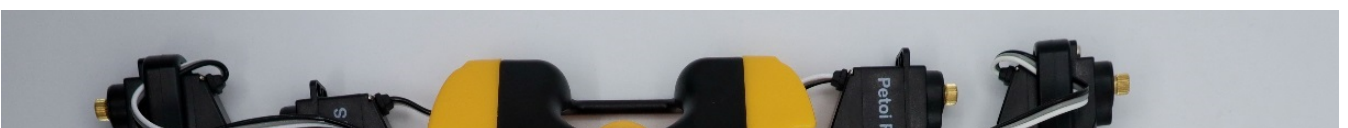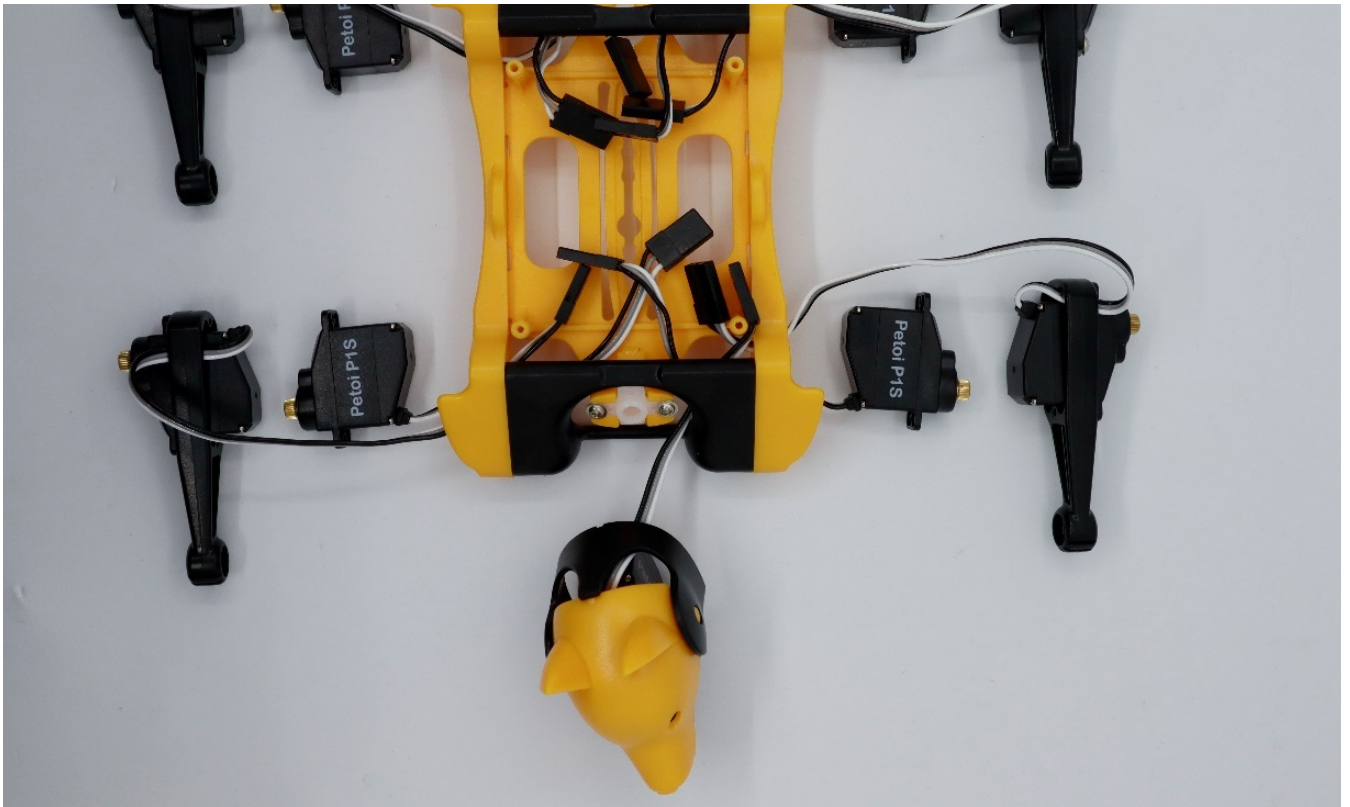
You also need to insert the wire of the head servo into the body.

Put the short wire servos into the side shoulders after confirming all the components' directions. Pay attention to the directions of the shoulder servos' output shaft. The long wires of the lower leg servos should be inserted into the opening between the shoulder servo and the shoulder window. Use two M2x8 self-tapping screws to fix each should servo.

4 🖳 Board Configuration

"Who loves fried chips?" 🍟
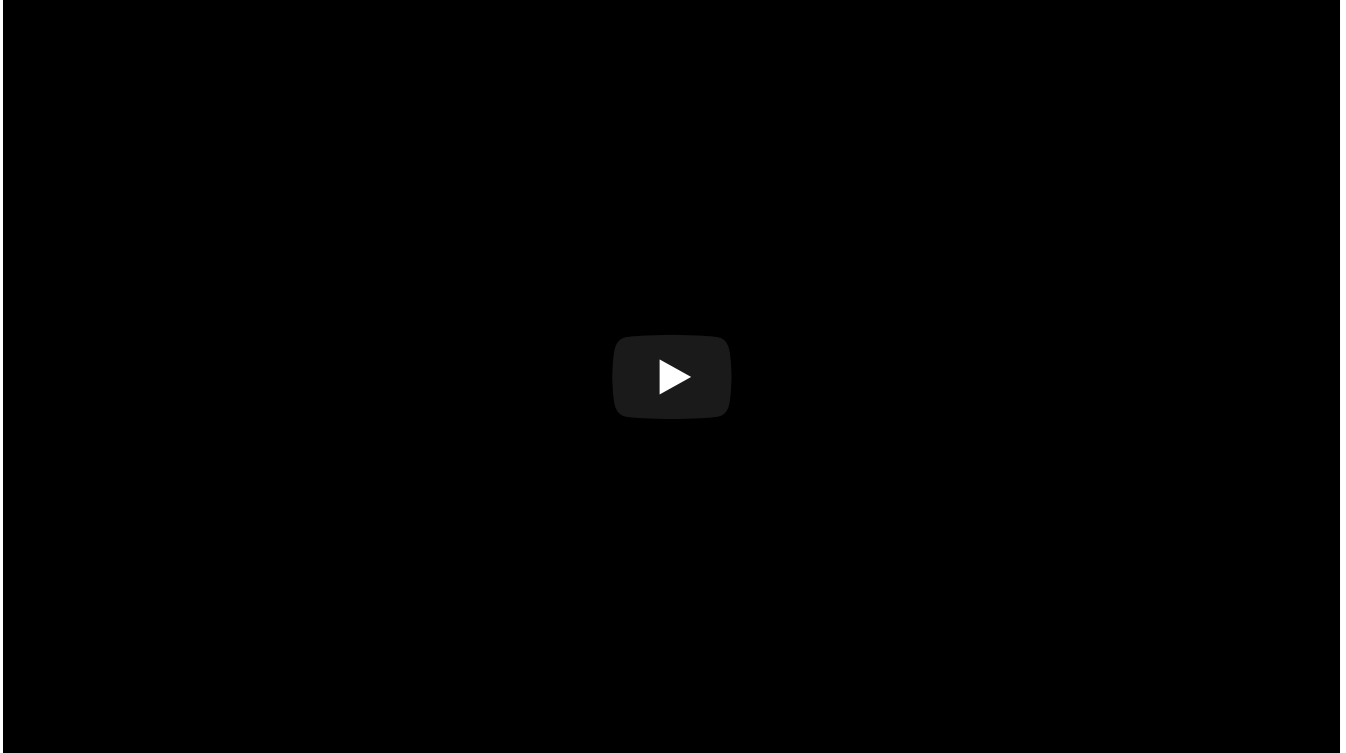
Quick Software Setup Guide

4.1. NyBoard

4.1.1. Read the user manual

Find the version info on NyBoard. Read the user manual for NyBoard V1_0 or NyBoard V1_1 (a light revision).
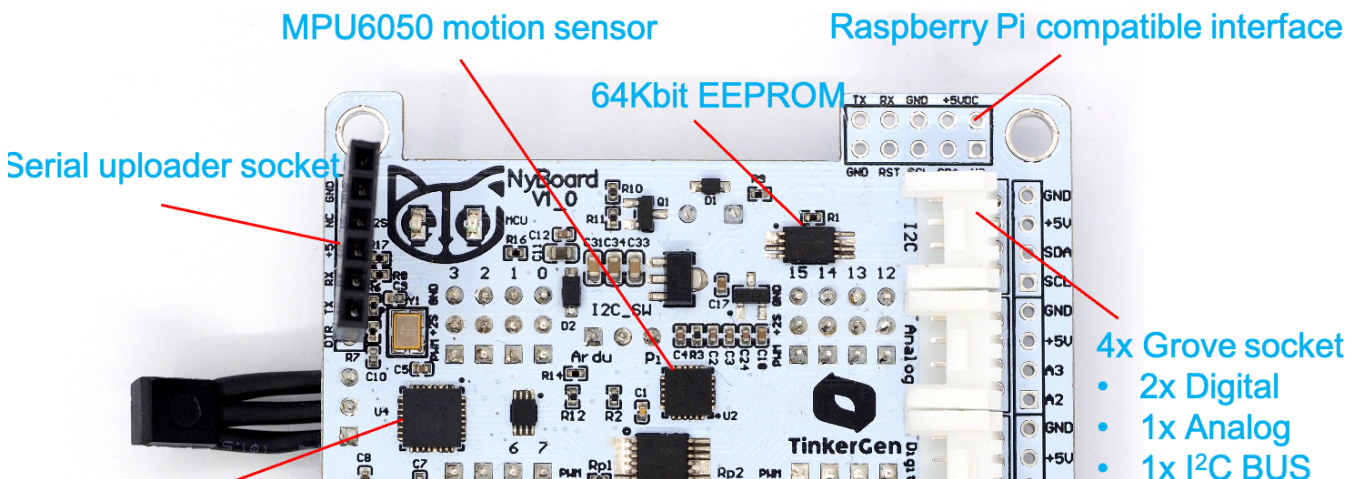
NyBoard V1_0 Top side

NyBoard V1_0 Bottom side

⚠ Wrong operations may damage your NyBoard!

4.1.2. Dial the slide switch to Arduino.

The slide switch changes the master of I2C devices (gyro/accelerometer, servo driver, external EEPROM). On default "Arduino", NyBoard uses the onboard ATmega328P as the master chip; On "Pi", NyBoard uses external chips connected through the I2C ports (SDA, SCL) as the master chip.

ⓘ Sometimes if you cannot go through the bootup stage, maybe you have accidentally dialed the switch to "Pi".

NyBoard powers the metal-geared servos directly with the 7.2V battery. It's 8.4V when fully charged. You CANNOT use NyBoard to drive your own plastic geared servos directly.

4.2. Downloads and installations

> ⓘ You will need the newest Arduino IDE to set up the environment. Older versions tend to compile larger hex files that may exceed the memory limit.

If you have previously added other libraries and see an error message "XXX library is already installed", I would recommend you delete them first (instruction: https://stackoverflow.com/questions/16752806/how-do-i-remove-a-library-from-the-arduino-environment). Due to different configurations of your Arduino IDE installation, if you see any error messages regarding missing libraries during later compiling, just google and install them to your IDE.

> ⚠ If you downloaded the newest OpenCat code from GitHub after Jan 3, 2022, you can skip all the following libraries.
>
> **4.2.1. Install through the library manager**
>
> Go to the library manager of Arduino IDE (instruction: https://www.arduino.cc/en/Guide/Libraries), search and install
>
> - **Adafruit PWM Servo Driver**
> - **QList (optional)**
> - **IRremote**
>
> The IRremote library was updated recently. And for some reason, they even changed the encoding of the buttons. You **MUST** roll back the library's version to **2.6.1** in the library manager.
>
> To save programming space, you **MUST** comment out the unused decoder in IRremote.h. It will save about 10% flash!
>
> Find **Documents/Arduino/libraries/IRremote/src/IRremote.h** and set unused decoders to 0. That is, only **DECODE_NEC** and **DECODE_HASH** are set to 1, and others are set to 0.
>
> ```
>  1 #define DECODE_RC5          0
>  2 #define SEND_RC5            0
>  3
>  4 #define DECODE_RC6          0
>  5 #define SEND_RC6            0
>  6
>  7 #define DECODE_NEC          1
>  8 #define SEND_NEC            0
>  9
> 10 #define DECODE_SONY         0
> 11 #define SEND_SONY           0
> 12
> ```

```
13 ...
14 set zeros all the way down the list
15 ...
16
17 #define DECODE_HASH          1 // special decoder for all protocols
```

4.2.2 Install by adding .ZIP library

Go to jrowberg/i2cdevlib: I2C device library collection for AVR/Arduino or other C++-based MCUs, download the zip file, and unzip. You can also git clone the whole repository.
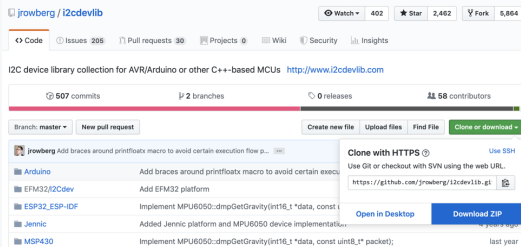
Write a caption

Use **Add .ZIP Library** to find **Arduino/MPU6050/** and **Arduino/I2Cdev/**. Click on the folders and **add them one by one**. They don't have to be **.ZIP** files.
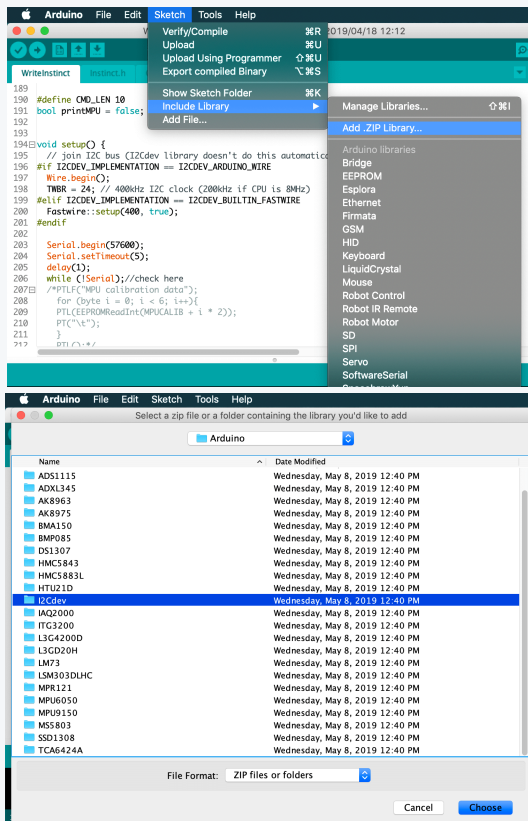
Write a caption

Write a caption

4.2.3. Add NyBoard support to Arduino IDE

With **NyBoard V1_***, you can simply choose **Arduino Uno**.

```
1   byte angleDataRatio;        //c
2   byte loopCycle[3];          //t
3   char* dutyAngles;           //t
4   Motion() {
5     period = 0;
6     expectedRollPitch[0] = 0;
7     expectedRollPitch[1] = 0;
8     dutyAngles = NULL;
9   }
0
1   int lookupAddressByName(char* skillName) {
2     int skillAddressShift = 0;
3     for (byte s = 0; s < NUM_SKILLS; s++) {//save skill info t
4       byte nameLen = EEPROM.read(SKILLS + skillAddressShift++)
5       char* readName = new char[nameLen + 1];
6       for (byte l = 0; l < nameLen; l++) {
7         readName[l] = EEPROM.read(SKILLS + skillAddressShift++
8       }
9       readName[nameLen] = '\0';
```

Board: "Arduino Uno"   ▶   Boards Manager...
Port: "/dev/cu.BittleSPP-877502-Port" ▶   Arduino AVR Boards ▶   Arduino Yún
Get Board Info   NyBoard-avr   ▶   ✓ Arduino Uno
                                                                   Arduino Duemilanove or Diecin
Programmer: "USBtinyISP"   ▶   Arduino Nano
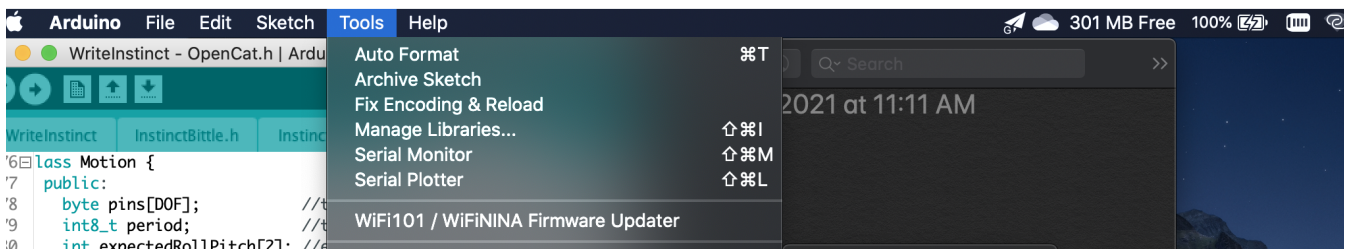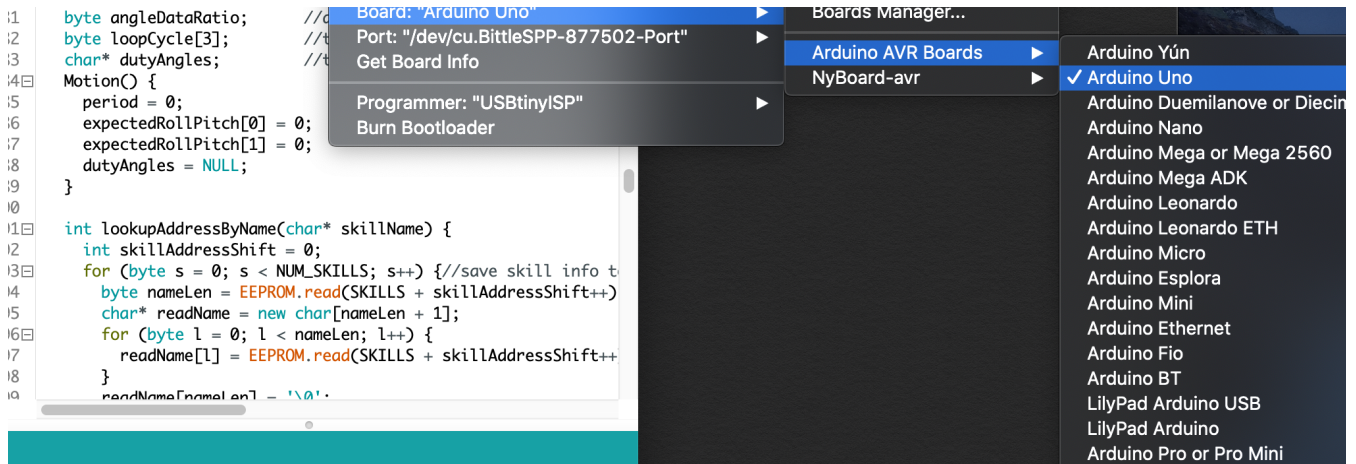Burn Bootloader   Arduino Mega or Mega 2560
                                                                   Arduino Mega ADK
                                                                   Arduino Leonardo
                                                                   Arduino Leonardo ETH
                                                                   Arduino Micro
                                                                   Arduino Esplora
                                                                   Arduino Mini
                                                                   Arduino Ethernet
                                                                   Arduino Fio
                                                                   Arduino BT
                                                                   LilyPad Arduino USB
                                                                   LilyPad Arduino
                                                                   Arduino Pro or Pro Mini

⚠ Only if the bootloader of NyBoard collapsed, which is very unlikely to happen

4.2.4. Burn the bootloader (no need for normal use)

- What is a bootloader?

Every NyBoard has to go through functionality checks before shipping, so they should already have a compatible bootloader installed. However, in rare cases, the bootloader may collapse then you won't be able to upload sketches through Arduino IDE.

Well, it's not always the bootloader if you cannot upload your sketch:

- Sometimes your USB board will detect a large current draw from a device and deactivate the whole USB service. You will need to restart your USB service, or even reboot your computers;

- You need to install the driver for the FTDI USB 2.0 to the UART uploader;

- You haven't selected the correct port;

- Bad contacts;

- Bad luck. Tomorrow is another day!

If you really decide to re-burn the bootloader:

- With **NyBoard V1_***, you can simply choose **Arduino Uno** under the **Tool** menu of Arduino IDE.

- Select your ISP (In-System Programmer). The above screenshot shows two popular programmers: the highlighted **USBtinyISP** is a cheap bootloader you can buy, while the checked **Arduino as ISP** can let you use a regular Arduino as ISP!

- Connect the programmer with the SPI port on NyBoard. Notice the direction when connecting. Make sure they have good contact.

- Burn bootloader. If it's your first time doing so, wait patiently until you see several percent bars reach 100% and no more messages pop up for one minute.

4.2.5. Connect the uploader (sometimes referred to as the programmer)
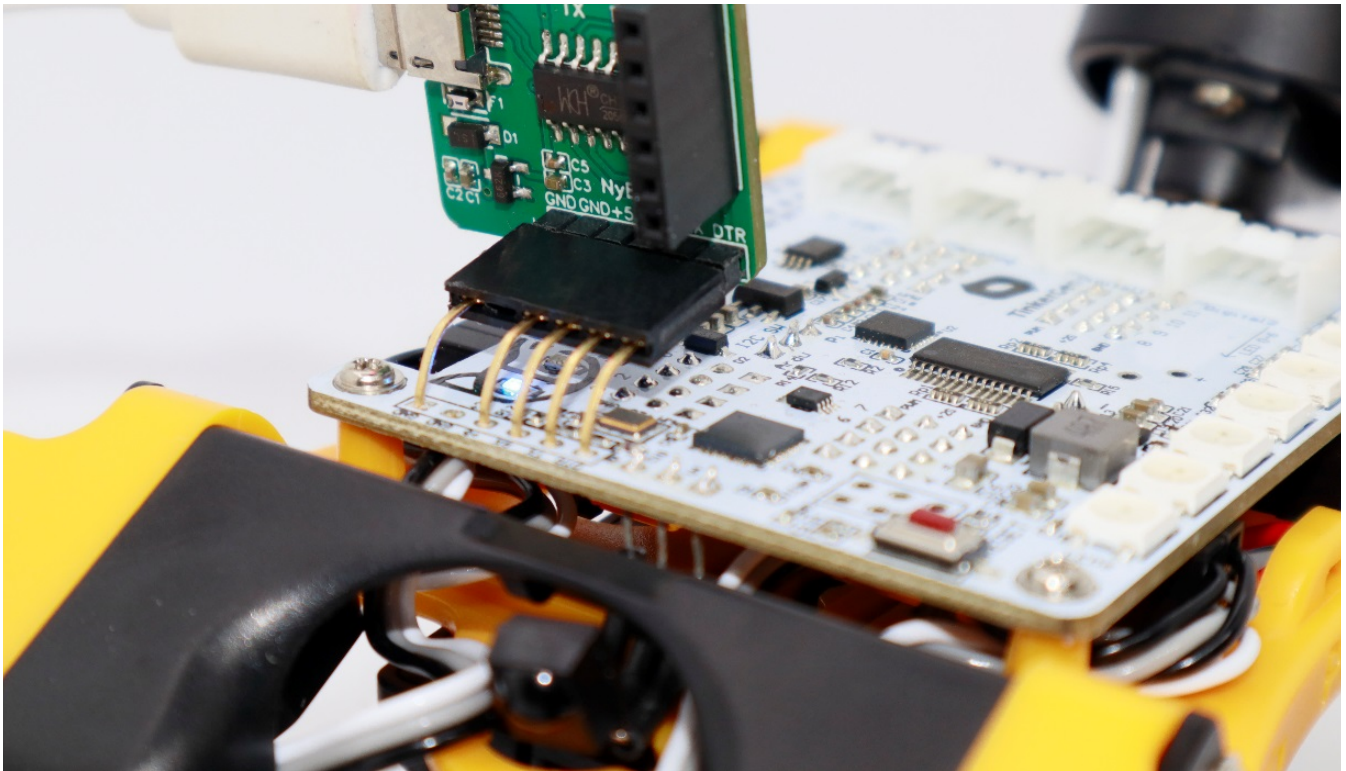
Connect your computer with the uploader through USB to micro-USB cable. The uploader has three LEDs, power, Tx, and Rx. Right after the connection, the Tx and Rx should blink for one second indicating initial communication, then dim. Only the power LED should keep lighting up. You can find a new port under **Tool->Port** as "/dev/cu.usbserial-xxxxxxxx" (Mac) or "COM#" (Windows).

For Linux, once the uploader is connected to your computer, you will see a "**ttyUSB#**" in the serial port list. But you may still get a serial port error when uploading. You will need to give the serial port permission. Please go to this link and follow the instructions: https://playground.arduino.cc/Linux/All/#Permission
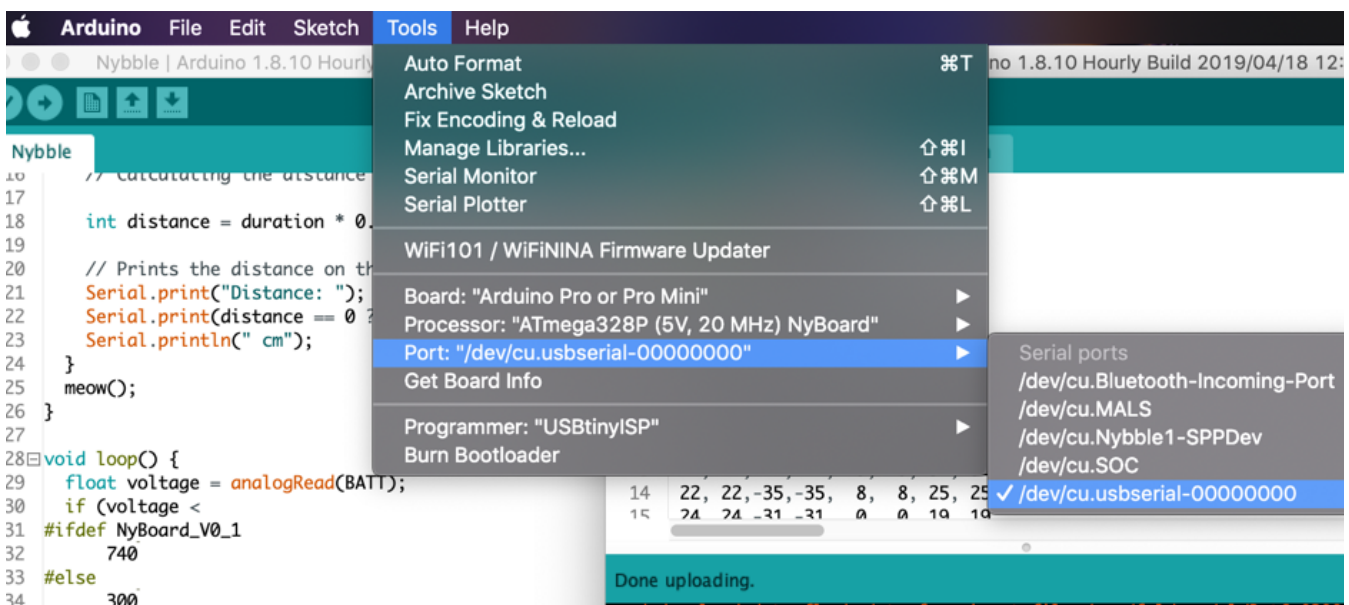
If Tx and Rx keep lighting up, there's something wrong with the USB communication. You won't see the new port. It's usually caused by overcurrent protection by your computer, if you're not connecting NyBoard with an external power supply and the servos move all at once.

> If you cannot find the serial port after connecting to your computer, you may need to install the driver for the CH340 chip.
>   - Mac
>   - Windows

4.2.6. Connect Bluetooth uploader (optional)

It's possible to program and communicate with Bittle wirelessly. You can even control Bittle with a smartphone APP or web API from there!

We include our official Bluetooth dongle in the standard Bittle kit. It can be used for wirelessly uploading sketches and communicating with the robot. **The baud rate is set to 115200.**

You need to connect it to your computer like what you do with a Bluetooth AirPod. The default passcode is 0000 or 1234. Then you can select it under **Tools->Port** of Arduino IDE, and use it in the same way as the wired uploader.
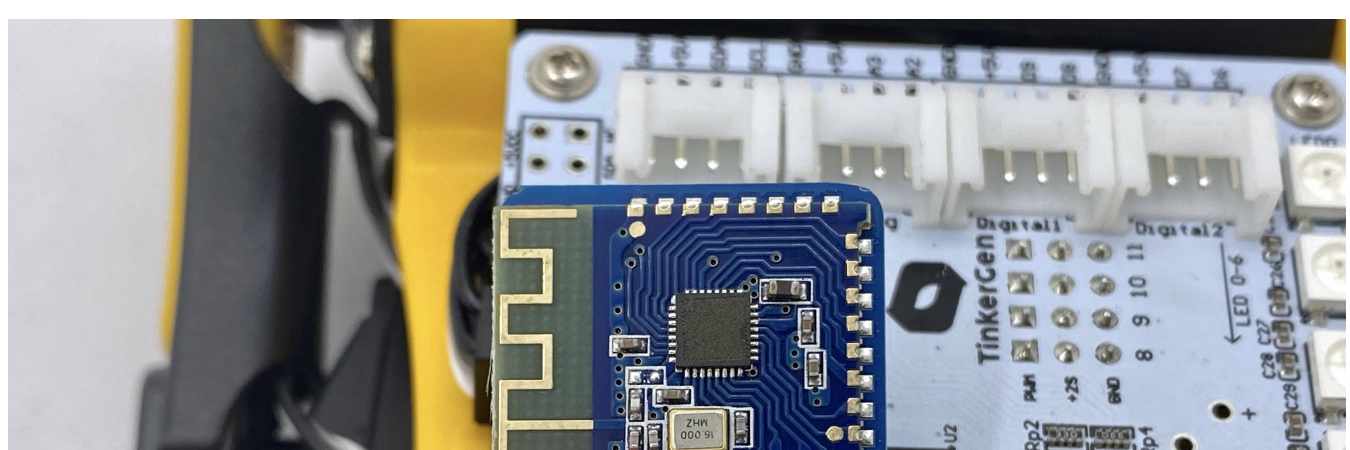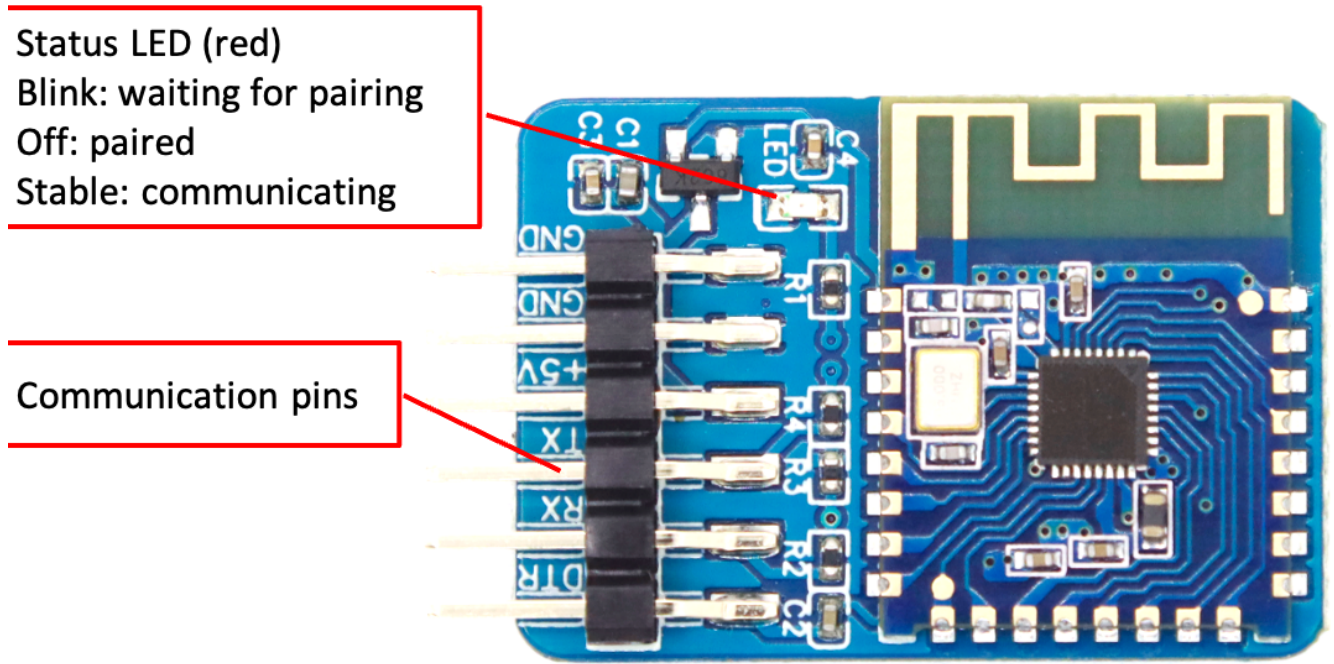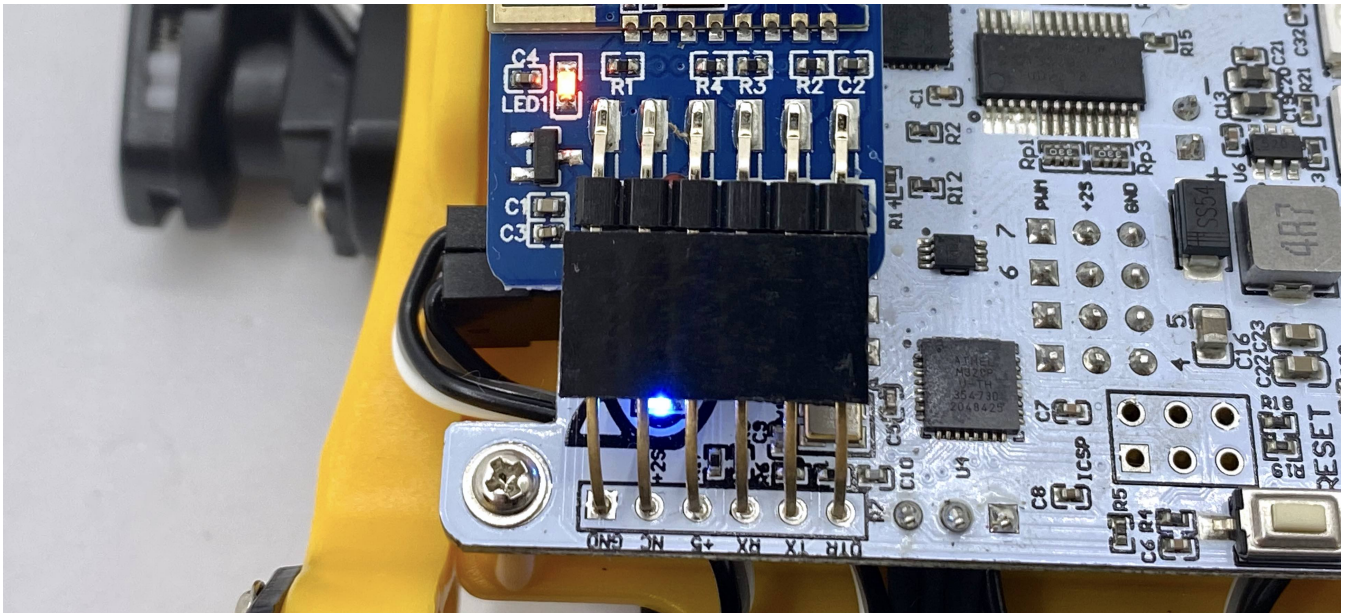
> ⚠ On Mac, the Bluetooth may lose connection after several uploads. In that case, delete the connection and reconnect to resume the functionality.

> ⚠ The Bluetooth dongle is not included in the kit sold by Seeed Studio or its partners. Please write to support@petoi.com for more information.
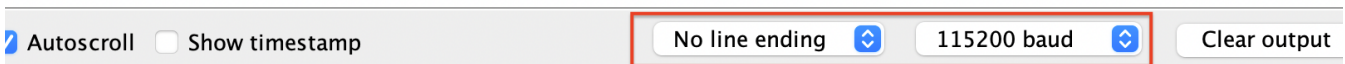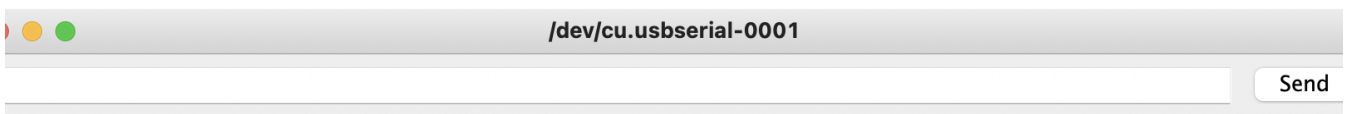
4.2.7. Download the OpenCat package

> ⓘ We keep updating the codes as an open-source project. You can star and follow our GitHub repository to get the newest features and bug fixes. You can also share your codes with worldwide OpenCat users.

- Download a fresh OpenCat repository from GitHub: https://github.com/PetoiCamp/OpenCat. It's better if you utilize GitHub's version control feature. Otherwise, make sure you download the **WHOLE OpenCat FOLDER** every time. All the codes have to be the same version to work together.
- If you download the Zip file of the codes, you will get an **OpenCat-main** folder after unzipping. You need to rename it to **OpenCat** before opening the OpenCat.ino.

> ⚠ No matter where you save the folder, the file structure should be:

- There are several **testX.ino** codes in **ModuleTests** folder. You can upload them to test certain modules separately. Open any **testX.ino** sketch with prefix "test". (I recommend using **testBuzzer.ino** as your first test sketch)

- Open up the serial monitor and set up the baud rate. With NyBoard V1_*, choose the board as **Arduino Uno** and **later set the baud rate** to 115200 in both the code and the serial monitor.

- Compile the code. There should be no error messages. Upload the sketch to your board and you should see Tx and Rx LEDs blink rapidly. Once they stop blinking, messages should appear on the serial monitor.

- Make sure you set "**No line ending**" to before entering your commands. Otherwise, the invisible '\n' or '\r' characters will confuse the parsing functions.

/dev/cu.usbserial-0001			
			Send

Autoscroll ☐ Show timestamp　　　　No line ending ⇅　　115200 baud ⇅　　Clear output

For Linux machines, you may see the error message like "permission denied". You will need to add execution privilege to the avr-gcc to compile the Arduino sketch: `sudo chmod +x filePathToTheBinFolder/bin/avr-gcc`

Furthermore, you need to add execution permission to **all files** within /bin, so the command would be : `sudo chmod -R +x /filePathToTheBinFolder/bin`

4.3. Arduino IDE as an interface

With the USB/Bluetooth uploader connecting NyBoard and Arduino IDE, you have the ultimate interface to communicate with NyBoard and change every byte on it.

I have defined a set of serial communication protocol for NyBoard:

OpenCat
Communication Protocol and Parsing

Interface	Token	Encoding	Parameters	Format	Bytes	Function
	'h'			char	1	print **h**elp information

							Type	Length	Description
RasPi Serial Port	Arduino Serial Monitor	Ascii	'c'	idx*,angle**		'\n'	string	strlen + 2	**c**alibrate servo$_{idx}$ by angle
			'm'	idx*,angle**		'\n'	string	strlen + 2	**m**ove servo$_{idx}$ to angle
			'j'				char	1	show all 16 **j**oint angles
			'd'				char	1	shut **d**own servos
			'p'				char	1	**p**ause motion
			'a'				char	1	**a**bandon calibration
			's'				char	1	**s**ave calibration
			'k'	abbreviation		'\n'	string	strlen + 2	load s**k**ill
			'w'	command		'\n'	string	strlen + 2	some future command **w**ords
			'r'				char	1	**r**eset board
		Binary	'i'	idx$_1$ a$_1$ ... idx$_N$ a$_N$	'~'		string	strlen + 2	list of **i**ndexed rotation angles
			'l'	a$_1$ a$_2$ ... a$_{DoF}$	'~'		string	DoF + 2	**l**ist of all DoF rotation angles

index range: 0 ~ (DoF - 1)

* angle range: -90 ~ 90. fits in the range of signed char (-128 ~ 127). Also depends on the servos' parameters

All the token starts with a single Ascii encoded character to specify its parsing format. They are case-sensitive and usually in lower case.

> (i) Some commands, like the **c** and **m** commands can be combined.
>
> For example:
>
> Successive "m8 40", "m8 -35", "m 0 50" can be written as "m8 40 8 -35 0 50". You can combine up to four commands of the same type. To be exact, the length of the string should be smaller than 30 characters. You can change the limit in the code but there might be a systematic constraint for the serial buffer.

> (!) Some tokens haven't been implemented, such as 'h'.

4.4. Raspberry Pi serial port as an interface

> (!) Only when using Pi as a master controller. Bittle doesn't need a Pi to move.

You can solder a 2x5 socket on NyBoard to plug in a Raspberry Pi. Pi 3A+ is the best fit for Bittle's dimension.
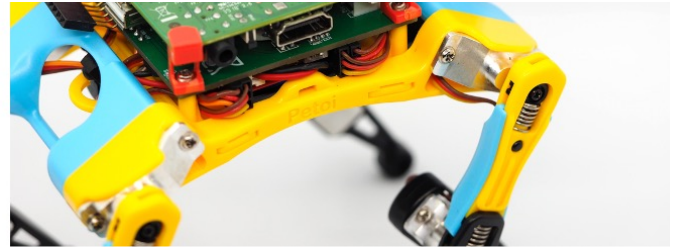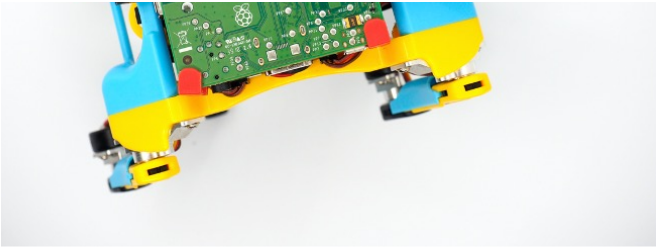
> ⚠ After you solder on the socket, you won't be able to install the back cover of Bittle.
>
> You need to unplug the 6-pin programmer for the NyBoard before mounting the Pi to the board.
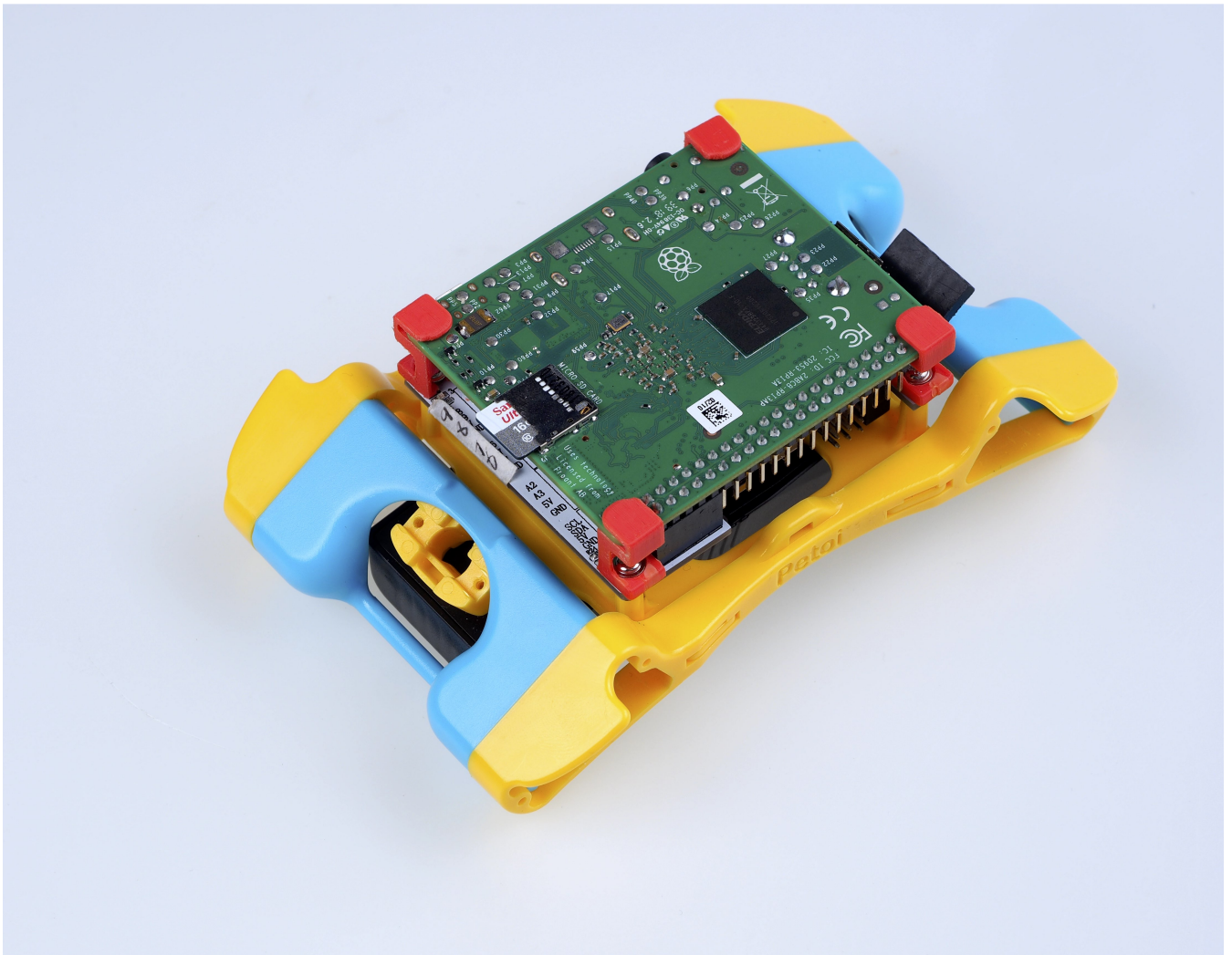
The red Pi standoff can be 3D printed.
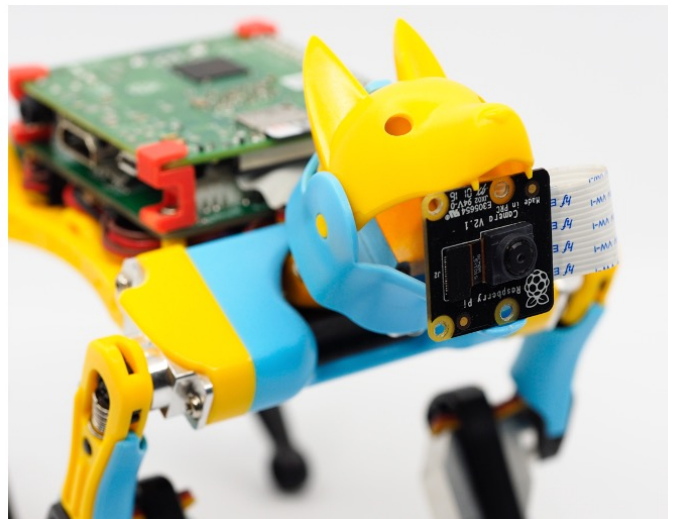
As shown in the serial protocol, the arguments of tokens supported by Arduino IDE's serial monitor are all encoded as Ascii char strings for human readability. While a master computer (e.g. RasPi) supports extra commands, mostly encoded as binary strings for efficient encoding. For example, when encoding angle 65 degrees:

- Ascii: takes 2 bytes to store Ascii characters '6' and '5'
- Binary: takes 1 byte to store value 65, corresponding to Ascii character 'A'

> ℹ What about value -113? It takes four bytes as an Ascii string but still takes only one byte in binary encoding, though the content will no longer be printable as a character.

Obviously, binary encoding is much more efficient than the Ascii string. However, the message transferred will not be directly human-readable. In the OpenCat repository, I have put a simple Python script ardSerial.py that can handle the serial communication between NyBoard and Pi.

4.4.1. Config Raspberry Pi serial port

In Pi's terminal, type `sudo raspi-config`

Under the **Interface** option, find **Serial**. Disabled the serial login shell and enable the serial interface to use the primary UART:

1. Run **raspi-config** with **sudo** privilege: `sudo raspi-config`.
2. Find Interface Options -> Serial Port.
3. At the option `Would you like a login shell to be accessible over serial?` select 'No'.
4. At the option `Would you like the serial port hardware to be enabled?` select 'Yes'.
5. Exit raspi-config and reboot for changes to take effect.

> ⚠ You also need to DISABLE the 1-wire interface of Pi to avoid repeating reset signals sent by Pi's GPIO 4.

> ℹ A good tutorial on the Pi Serial

If you plug Pi into NyBoard's 2x5 socket, their serial ports should be automatically connected at 3.3V.

Otherwise, pay attention to the Rx and Tx pins on your own AI chip and its voltage rating. The Rx on your chip should connect to the Tx of NyBoard, and Tx should connect to Rx.

> (i) Note: If you installed **Ubuntu OS** on Raspberry Pi, please config it as follows:
>
> - add `enable_uart=1` to `/boot/config.txt`
>
> - remove `console=serial0,115200` from `/boot/firmware/cmdline.txt` on Ubuntu and similar to `/boot/cmdline.txt` on Raspberry Pi OS
>
> - disable the serial console: `sudo systemctl stop serial-getty@ttyS0.service && sudo systemctl disable serial-getty@ttyS0.service`
>
> - make sure you have `pyserial` installed if you're using the python serial library, not python-serial from apt.
>
> - create the following udev file (I created `/etc/udev/rules.d/50-tty.rules`):
>
> ```
> 1 KERNEL=="ttyS0", SYMLINK+="serial0" GROUP="tty" MODE="0660"
> 2 KERNEL=="ttyAMA0", SYMLINK+="serial1" GROUP="tty" MODE="0660"
> ```
>
> - reload your udev rules: `sudo udevadm control --reload-rules && sudo udevadm trigger`
>
> - change the group of the new serial devices:
>
> ```
> 1 sudo chgrp -h tty /dev/serial0
> 2 sudo chgrp -h tty /dev/serial1
> ```
>
> - The devices are now under the tty group. Need to add the user to the tty group and dialout group:
>
> ```
> 1 sudo adduser $USER tty
> 2 sudo adduser $USER dialout
> ```
>
> - update the permissions for group read on the devices:
>
> ```
> 1 sudo chmod g+r /dev/ttyS0
> 2 sudo chmod g+r /dev/ttyAMA0
> ```
>
> - reboot
>
> Or just create a script that will do this automatically.

4.4.2. Change the permission of [ardSerial.py](#)

If you want to run it as a bash command, you need to make it executable:

```
chmod +x ardSerial.py
```

You may need to change the proper path of your Python binary on the first line:

```
#!/user/bin/python
```

4.4.3. Use ardSerial.py as the commander of Bittle

> ⚠ NyBoard has only one serial port. You need to UNPLUG the FTDI converter if you want to control Bittle with Pi's serial port.

Typing `./ardSerial.py <args>` is almost equivalent to typing <args> in Arduino's serial monitor. For example, `./ardSerial.py kcrF` means "perform s**k**ill **cr**awl **F**orward".

Both **ardSerial.py** and the parsing section in **OpenCat.ino** need more implementations to support all the serial commands in the protocol.

4.5. Battery

Though you can program NyBoard directly with the USB uploader, external power is required to drive the servos.

4.5.1. Voltage

NyBoard requires 7.4~8.4V external power to drive the servos. We include our customized Li-ion battery with built-in charging and protection circuit in the Bittle kit. Short press the battery's button will show its status. Blue light indicates it has power, and red means the power is low.

4.5.2. Connection and Power On

Be careful with the polarity when connecting the power supply. The terminal on NyBoard has an anti-reverse socket, so you won't be able to plug in the wrong direction.   See here for the detailed connection instruction.

Please long press the button of the battery for 3 seconds to power on the battery.

> ⚠ **Reversed connection may damage your NyBoard!**

4.5.3. Battery life varies according to usage

It can last hours if you're mainly coding and testing postures, or less than 30 mins if you keep Bittle running.

The battery light will turn red when the power is low. The power will be cut off automatically.

4.5.4. Charging

Use a **5V-1A** USB charger to charge the battery. We don't recommend using fast chargers. The battery will **NOT** supply power during charging. Keep the battery in your sight when charging.

4.5.5. After use

After playing, remember to turn off the battery. It's recommended to unplug the battery from the NyBoard's

5 ⬡ Connect Wires

"Everything is connected." ⬡

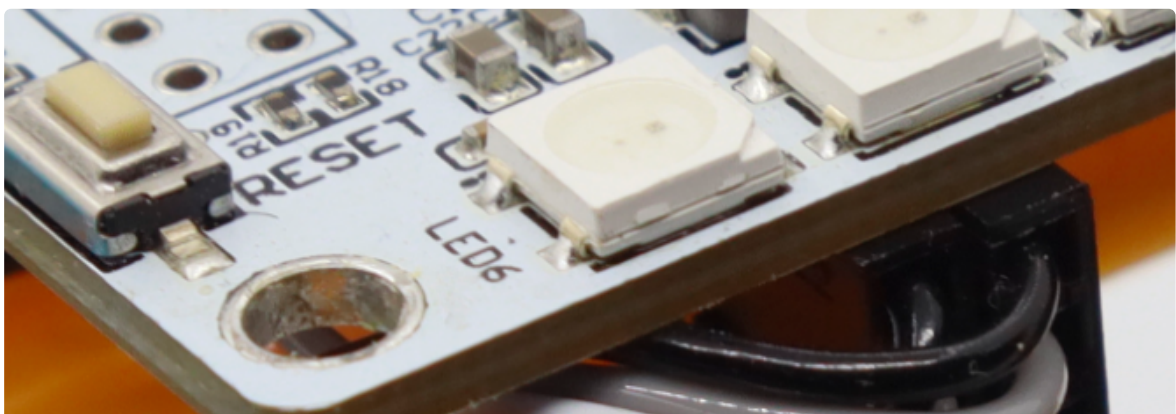## 5.1. Servo wiring

Connect the wires of servos to the NyBoard servo pins as below. They follow a circular symmetry. Notice the skipped pins between the head and leg servos. They are reserved for our future robots with more joints.
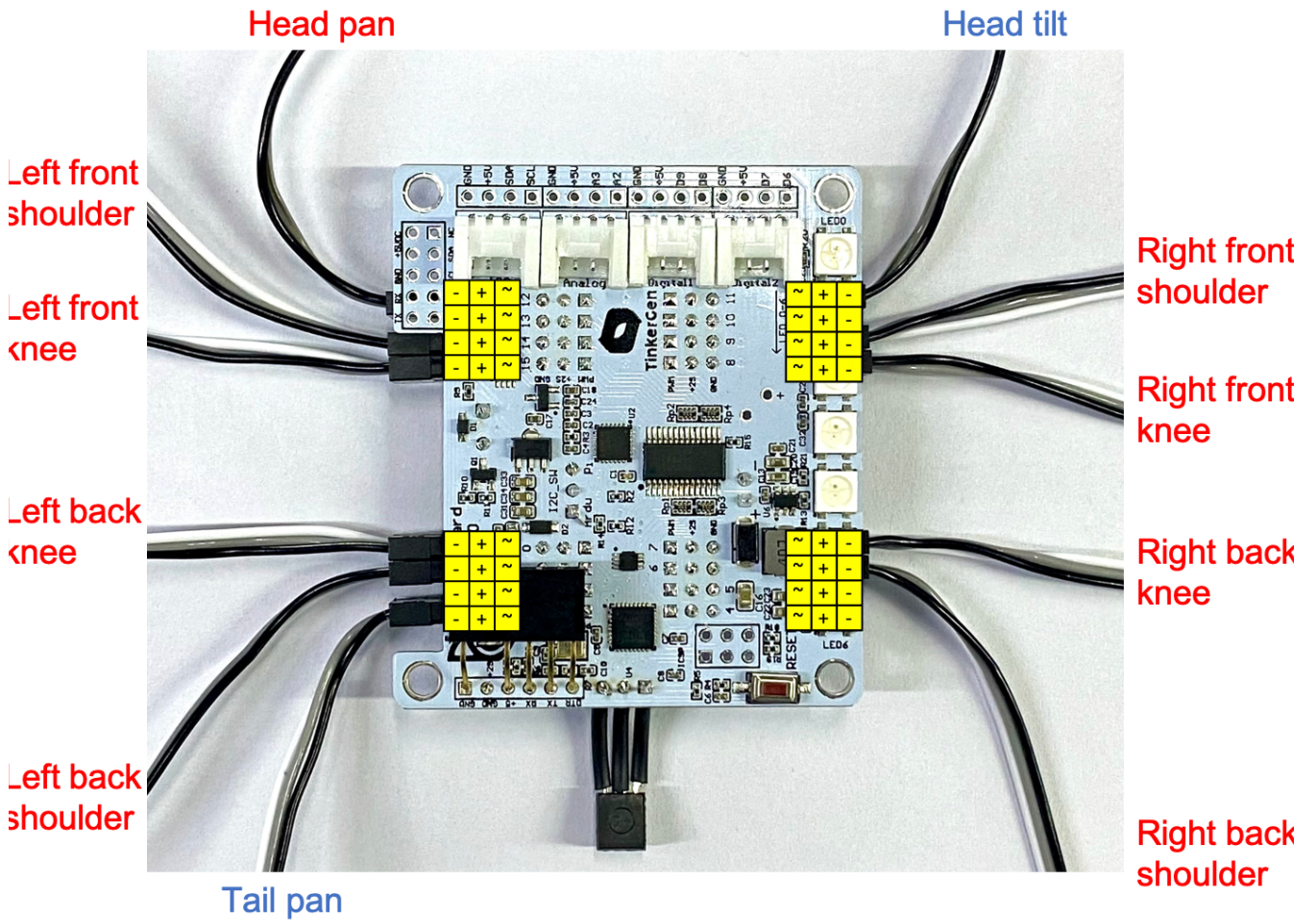
The black wire (GND wire) should be closer to the board.

Bottom view:
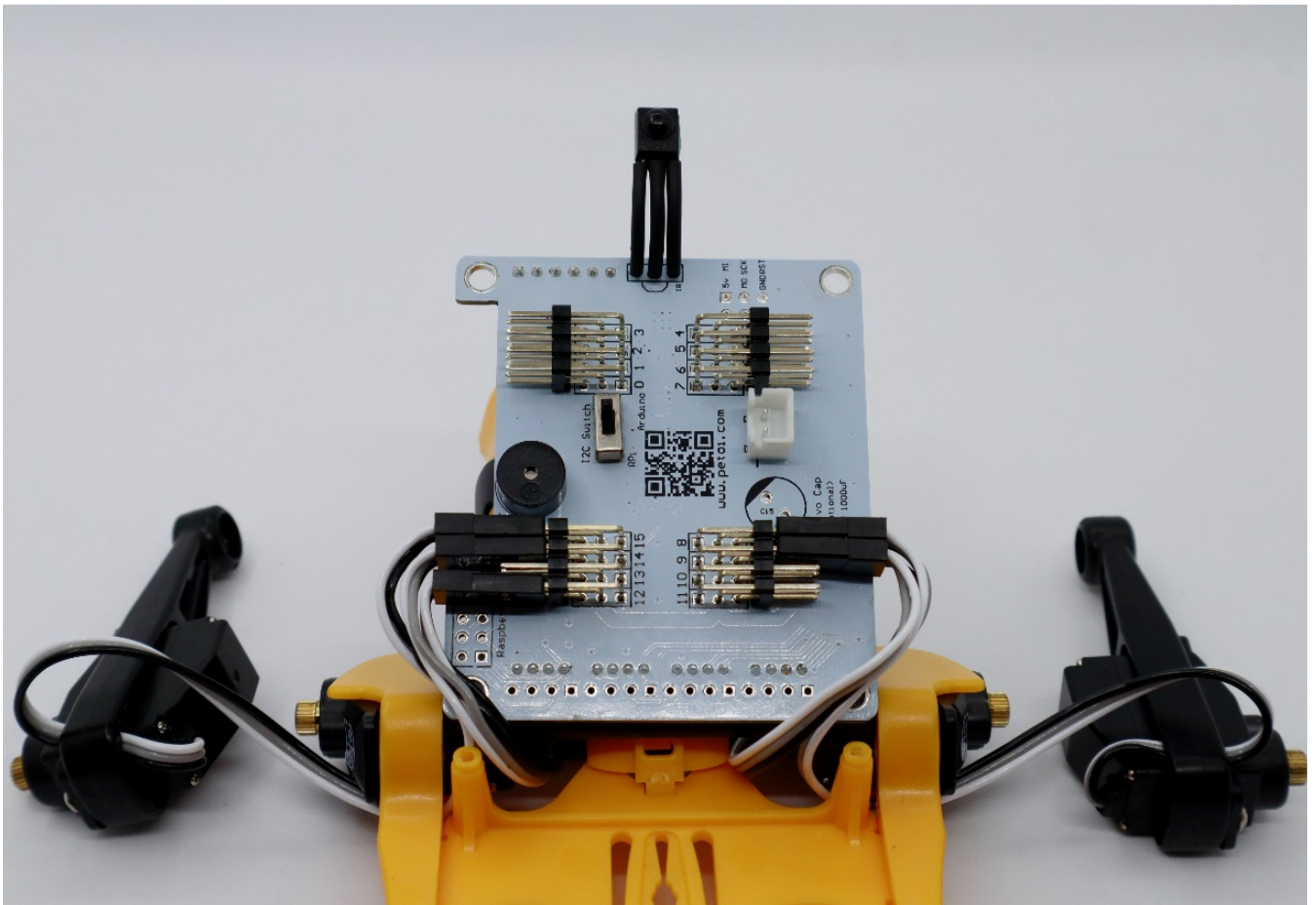
Insert the infrared receiver under the back plate so that the sensor can expose to the outside.
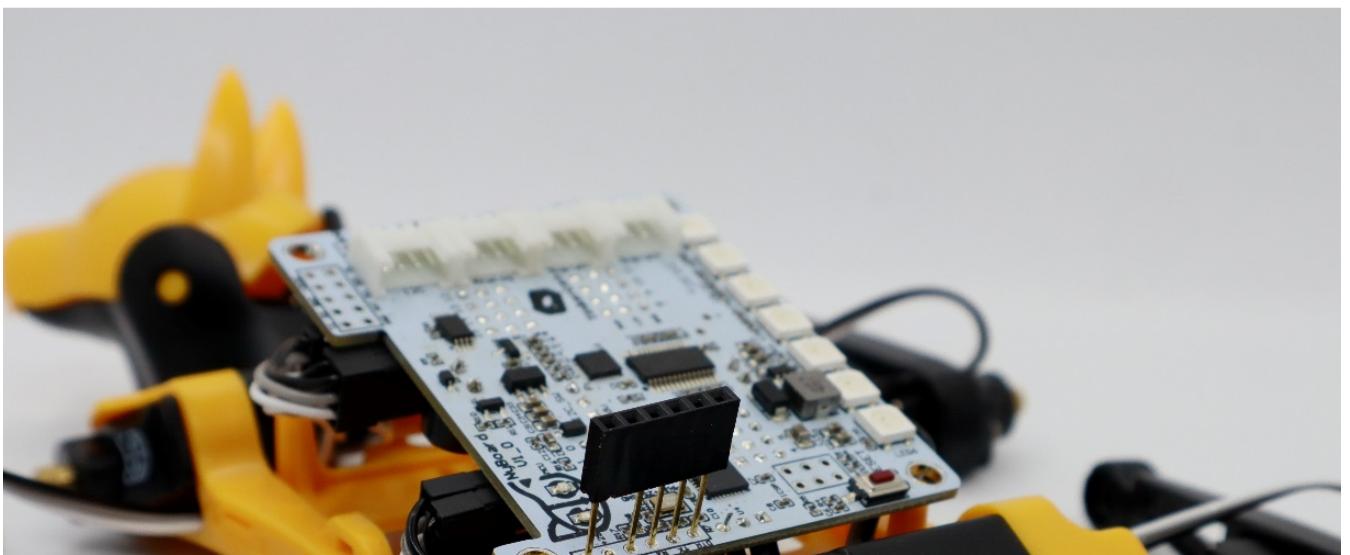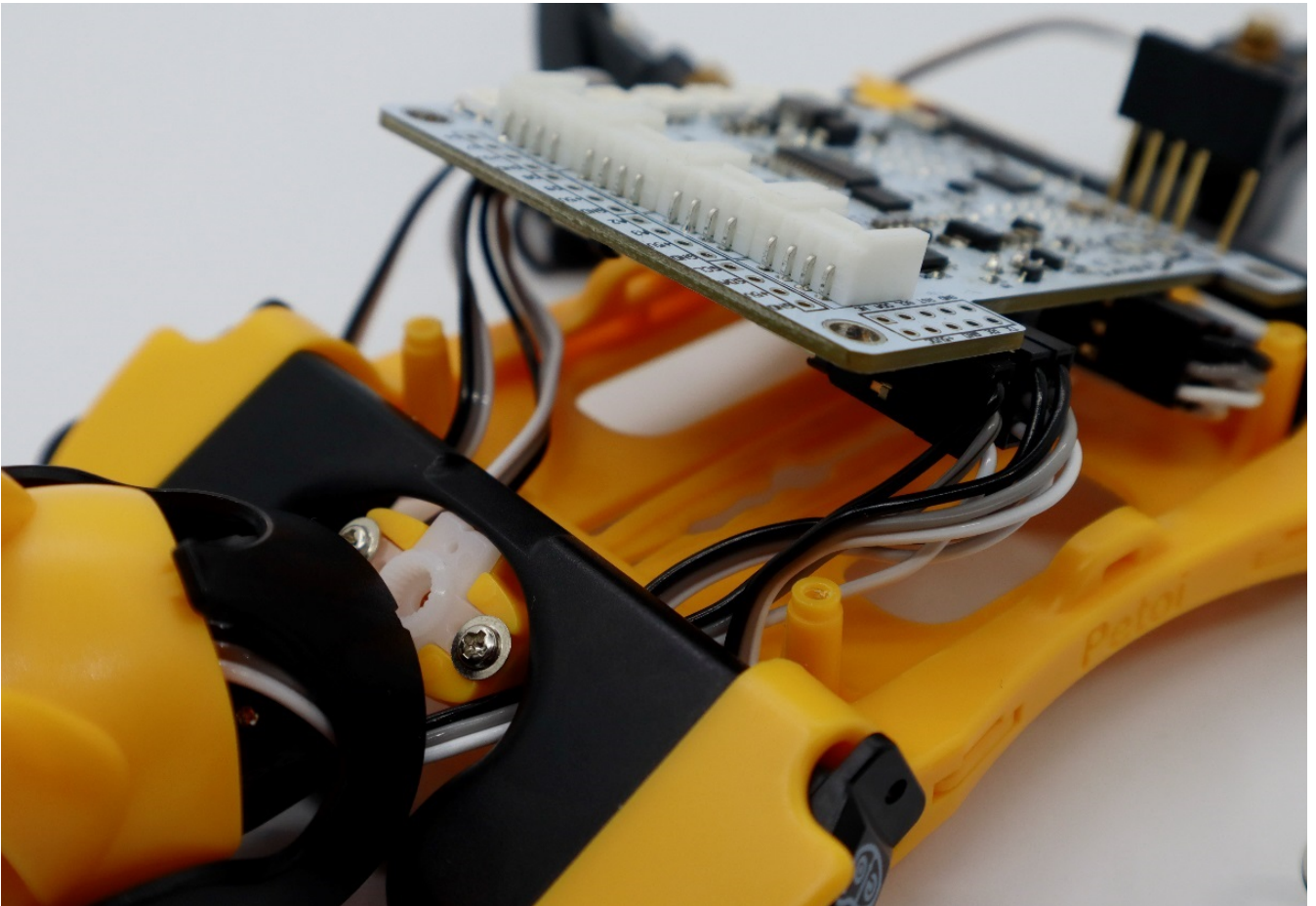
Make sure that the wires go into the body, bypass the columns, then plug into the servo pins. Arrange the wires so that they don't get in between the servo plugs and the body frame. NyBoard should be leveled on the four standoffs. Use four M2x8 self-tapping screws to fix the NyBoard.
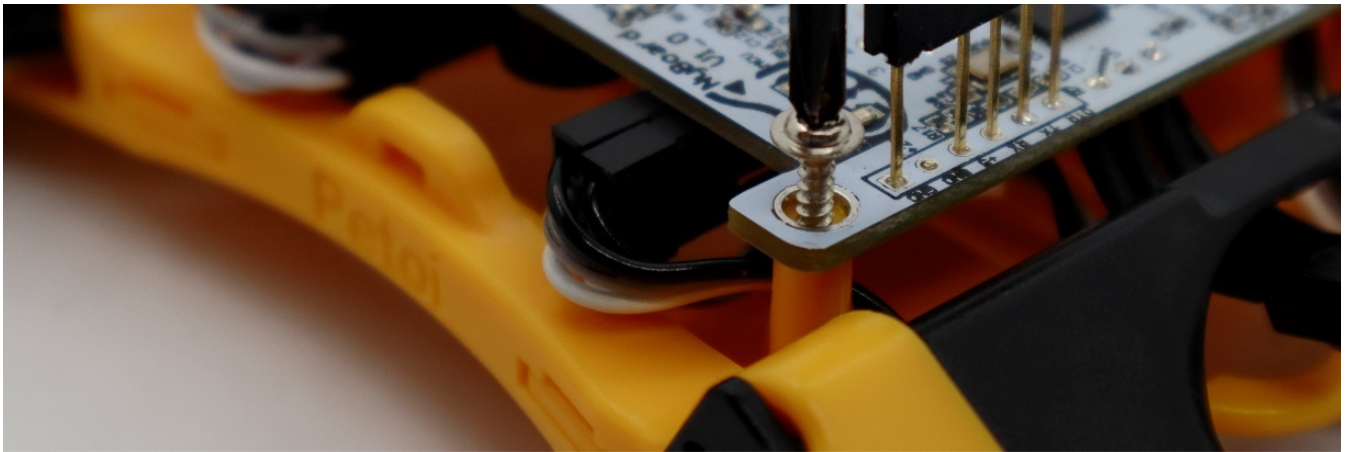
Completed:
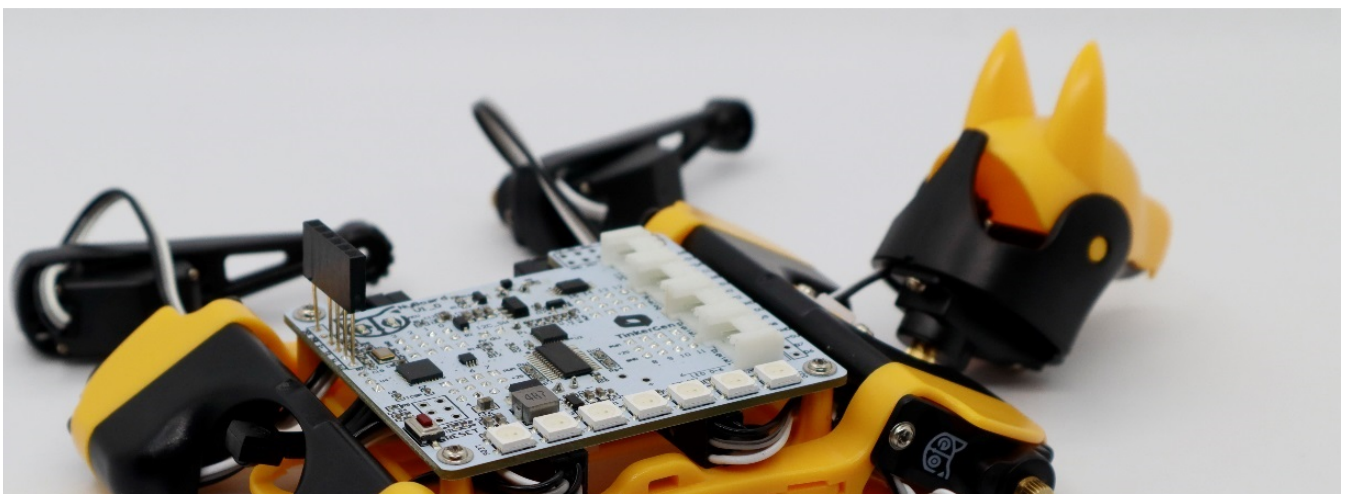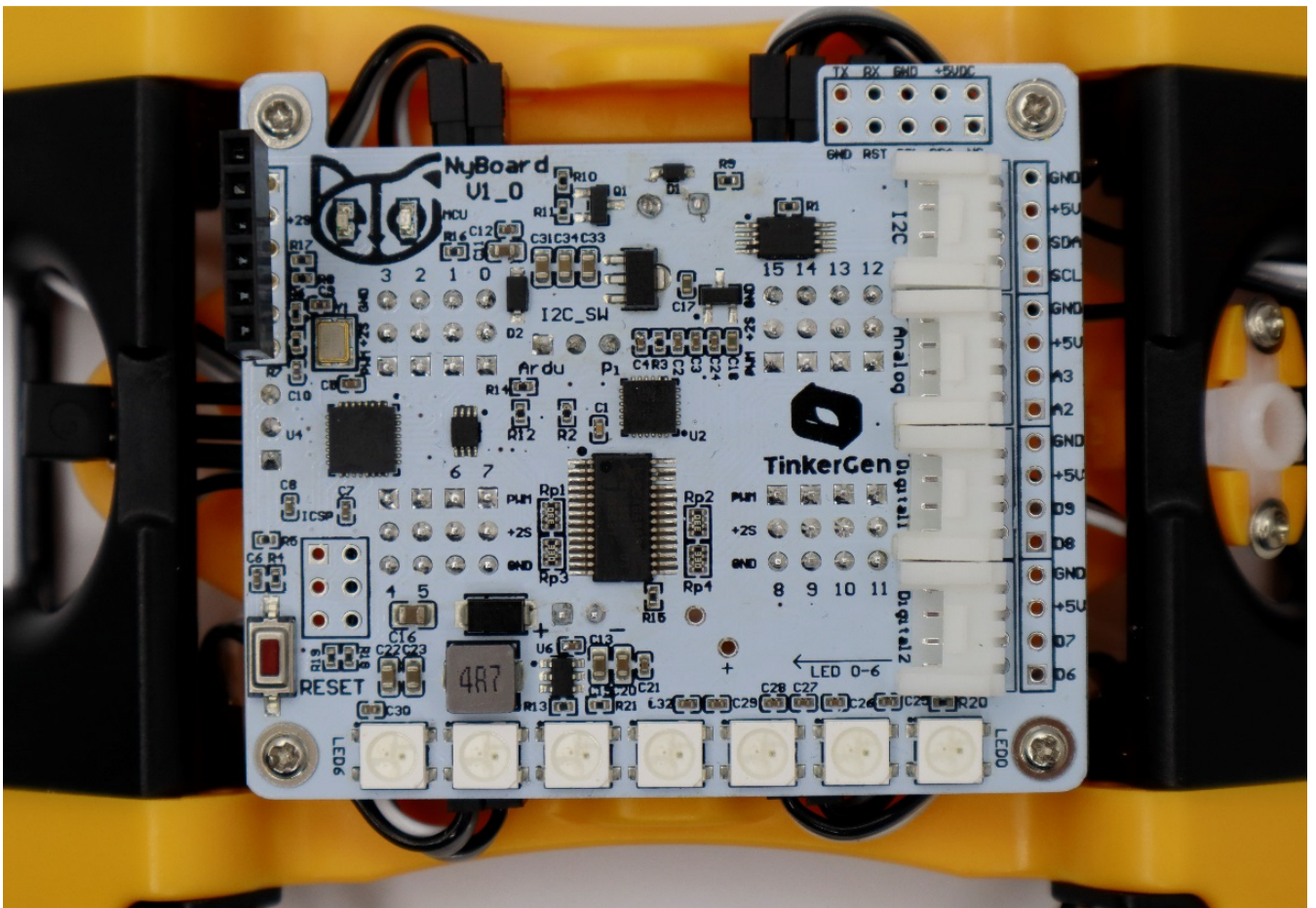
5.2. Battery

Insert the battery's plug into NyBoard's anti-reverse power socket.
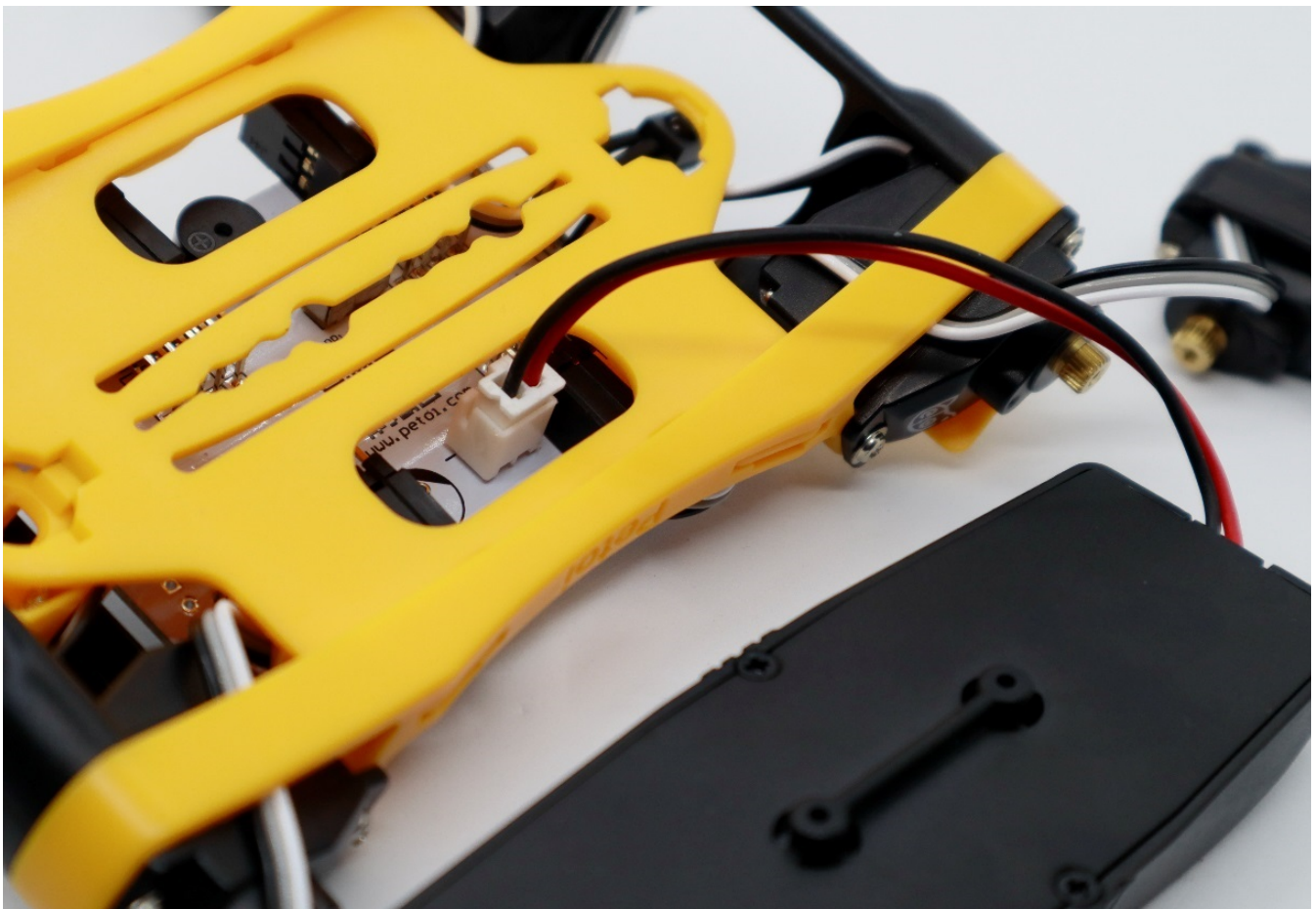
Hold the button of the battery for 3 seconds to power on the battery. If the LED on the battery lights red, recharge the battery until it turns green. The blue and orange "eyes" of the Petoi logo on NyBoard should light on.
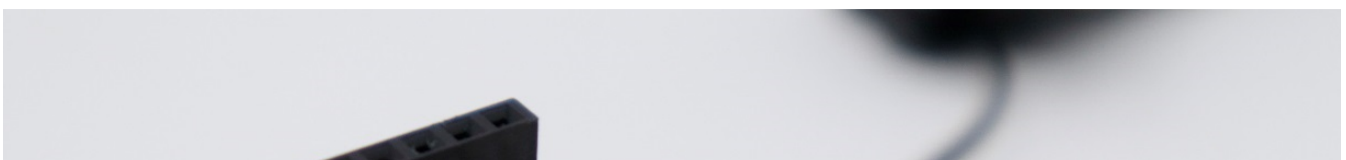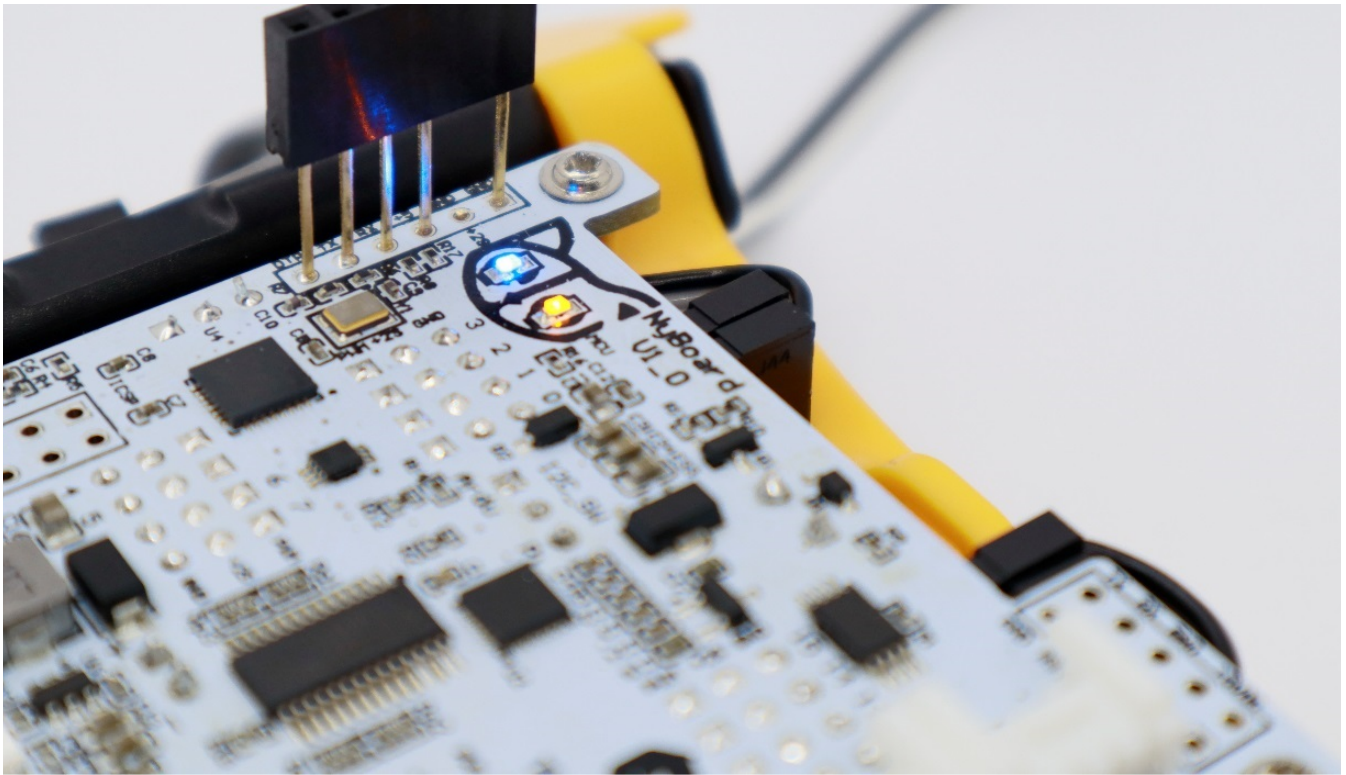
6 Calibration and Final Assembly

"A miss is as good as a mile."

> (i) Calibration is vital for Bittle to work properly.

In the previous sections, we have prepared Bittle's body parts, but haven't screwed them onto servos because we still need to calibrate body parts to be attached to servos at the right angles.

If we don't calibrate the servos before attaching them, they may rotate to any direction, get stuck, cause damage to either the servos or body parts and cause robot limbs not to function properly.

The calibration has the following steps:

1. Write constants to the board

2. Power on the circuit, let servos rotate freely to zero angle/calibration state

3. Attach body parts to the servos

4. Fine-tune the body part placements and store the calibration results on board.

Please make sure you turn on the battery first.

The logic behind calibration can be found on the OpenCat forum. The principles are the same for Nybble and Bittle.

6.1 Write constants to NyBoard

6.1.1. Three types of constants to be saved to NyBoard:

1. Assembly-related definitions, like joint mapping, rotation direction, sensor pins. They are pretty fixed and are mostly defined in **OpenCat.h**. They are even kept consistent with my future robots;

2. Calibration-related parameters, like MPU6050 offsets and joint corrections. They are measured in realtime and are saved in the onboard EEPROM. They only need to be set up once;

3. Skill-related data, like postures, gaits, and behaviors. They are mostly defined in **InstinctBittle.h**. You can add more customized skills too.

6.1.2. Upload and run WriteInstinct.ino.

The role for **WriteInstinct.ino** is to write constants to either onboard or I2C EEPROM, and save calibration values. It will be overwritten by the main sketch **OpenCat.ino** afterward.

> ⚠ You need to change the ***** on line `#define NyBoard_V*_*` in **OpenCat.h** to match your NyBoard's version. The version number is printed beside the logo on the NyBoard.

> ⚠ Before uploading the sketch, make sure to close all opened windows of the serial monitor. Otherwise, the serial port may be occupied and the sketch will not be uploaded.

> ⚠ You also need to dial the slide switch on NyBoard to **Arduino** rather than Pi!

After you upload **WriteInstinct.ino** via Arduino IDE, open the serial monitor.

> ⚠ Make sure to set the serial monitor as **115200 baud rate and no line ending**.

● ● ●   /dev/cu.usbserial-0001

                                                                  Send

No line ending   115200 baud   Clear output
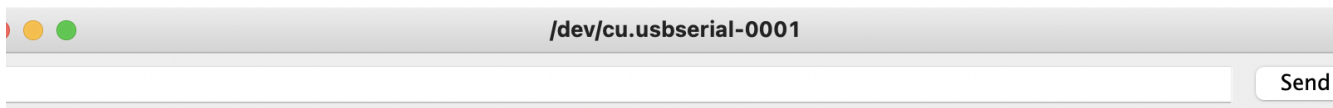
You will see several questions:

```
Reset all joint calibration? (Y/n)
```

If you have never calibrated the joints, or if you want to recalibrate the servos with a fresh start(zero state), type 'Y' to the question.

```
Do you need to update Instincts? (Y/n)
```

If you have modified the Instinct.h in any way, you should type 'Y'. Though it's not always necessary once you have a deeper understanding of memory management.

```
Calibrate MPU? (Y/n)
```

If you have never calibrated the MPU6050, i.e. the gyro/accelerometer sensor, type 'Y'.

Sometimes the program could halt at the connection stage. You can close the serial monitor and reopen it, or press the reset button on NyBoard, to restart the program.

/dev/cu.Nybble-DevB
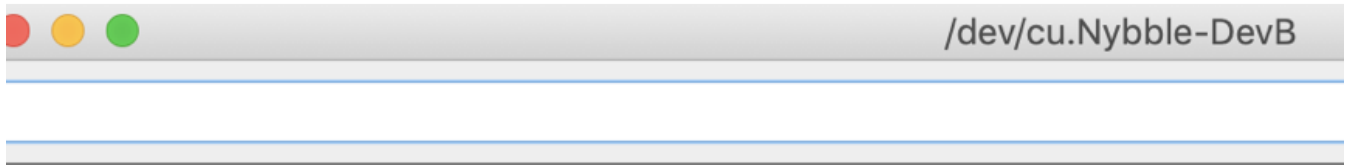
```
 Starting *
nitializing I2C
onnecting MPU6050...
```

6.1.3. Upload OpenCat.ino

You also need to upload **OpenCat.ino** to save the last constants and activate the demo functionalities.

6.2 Initial calibration

6.2.1 Enter calibration mode via remote controller

You MUST plug in all the servos and battery for proper calibration.

Before we attach the legs, the servos' output shafts should be at the zero state. You can read this forum post to understand the logic behind the calibration process.

6.2.2 Head and limb calibration

Take one assembled upper leg and keep Bittle in the calibration mode by pressing the third row, third column (3, 3) button on the IR remote controller.

Attach the yellow side of the upper leg to the shoulder servo. Try to find the direction closest to the vertical angle. Don't rotate the servo during installation. Install the lower leg at a perpendicular angle to the upper leg. Then repeat to install all other limbs.
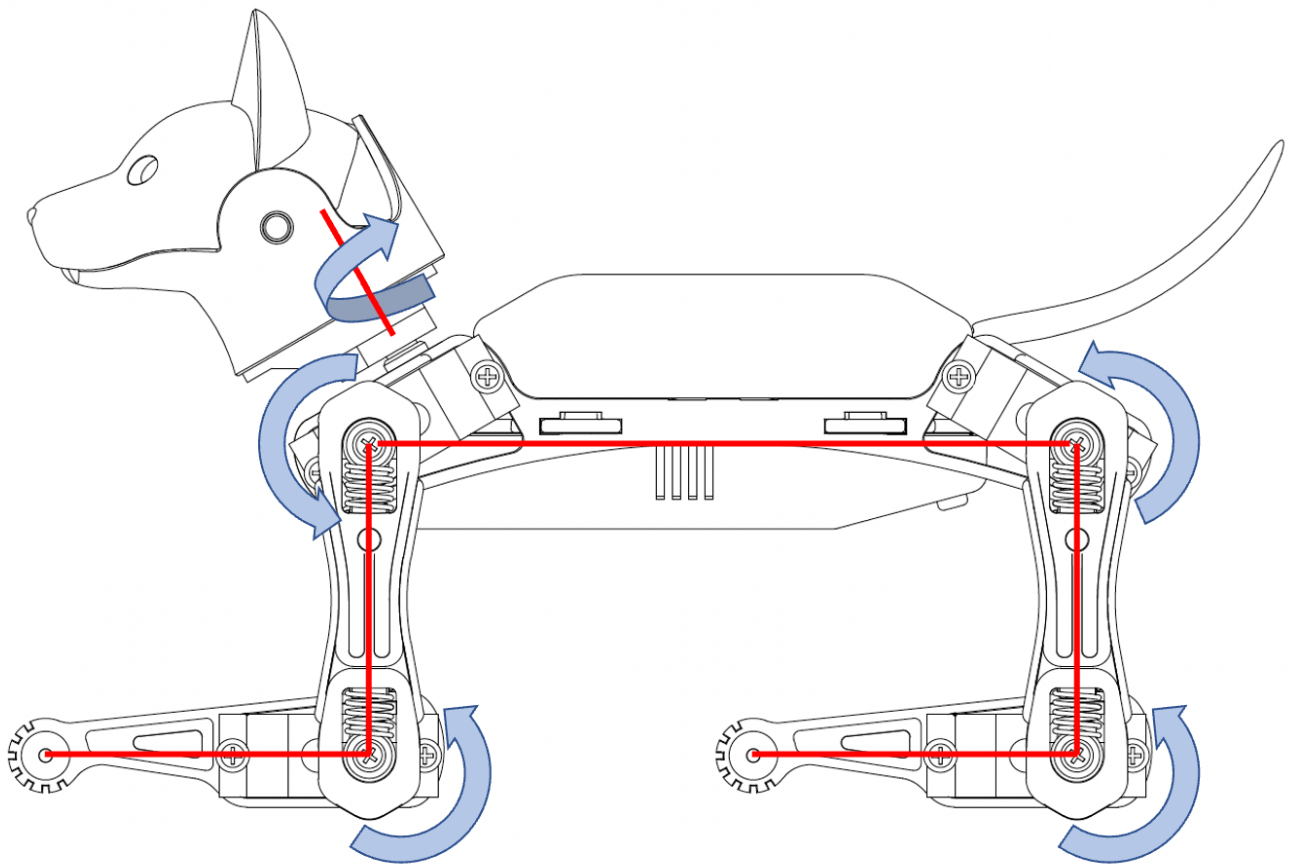
A visualization of the perfect zero(calibrated) state when you enter the calibration mode

You can use the included "L" shape tuner as a reference(see next section) for keeping things in line.

Next, insert the output shaft of the head servo into the servo arm of the neck. Try your best to attach the head to point straight ahead.

6.2.3 Testing and validations

Hold the body of Bittle in the air. Press the "1" button on the IR remote controller to make Bittle perform a few "stretches".   Observe if all the limbs are moving fine.

If the movements look intuitive ， Place Bittle on a flat floor. Press a few action buttons on the IR remote controller to make Bittle move around and check if its moves look fine.

Then press the "EQ" button to get back to the calibration mode.   Observe if its upper legs and lower legs are perpendicular to each other and whether the lower legs are parallel to the ground.

If the legs/head are at the right positions, use the flat end screw to fix the limbs and head onto servos.

Otherwise, take off limbs to do a few rounds around of calibrations.

If you can't achieve good results, it's time to do the precise calibration.

6.3 Precise calibration

6.3.1 Understand the zero state and the coordinate system

The zero state is defined as the middle point of the servo's reachable range.

In calibration mode with all servos rotated to their zero states(zero angles), rotating the limbs counter-clockwise from their zero states will be positive (same as in polar coordinates). The only exception is the tilt angle for the head. It's more natural to say head up, while it's the result of rotating clockwise.

A visualization of the perfect zero(calibrated) state when you enter the calibration mode

If we take a closer look at the servo shaft, we can see it has a certain number of teeth. That's for attaching the servo arms, and to avoid sliding in the rotational direction. In our servo sample, the gears are dividing 360 degrees to 25 sectors, each taking **14.4** degrees(offset of -7.2~7.2 degrees). That means we cannot always get exact perpendicular installation.

6.3.2 Understand the serial outputs via software-based calibration

Software-based calibration for servos can be done in either **WriteInstinct.ino** or **OpenCat.ino**. I recommend you do it with **WriteInstinct.ino** in case there's something wrong with the constants.

Upload either file to Bittle via Arduino IDE. Then in the serial monitor, type 'c' to enter calibration mode.

The servos should rotate one by one with unnoticeable time intervals then stop. Depending on their initial shaft direction, some may travel larger angles until stopping at the middle point. There will be noise coming from the gear system of the servos. You will see a calibration table like the following:

1,	2,	3,	4,	5,	6,	7,	8,	9,	10,	11,	12,	13,	14,	15
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0														

The first row is the joint indexes, the second row is their calibration offsets:

| Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Offset	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Initial values are "-1" or "0", and should be changed by later calibration.

6.3.3 Use 'L' shaped joint tuner

When watching something, the observation will change from different perspectives. That's why when measuring length, we always want to read directly above a referencing ruler.

It's especially important that you keep a parallel perspective when calibrating Bittle. Use the 'L' shaped joint tuner as a parallel reference to avoid reading errors. Align the tips on the tuner with the center of the screws in the shoulder and knee joints, and the little hole on the tip of the foot. Look along the co-axis of the centers. For each leg, calibrate shoulder servos (index 8~11) first, then the knee servos(index 12~15). When calibrating the knee, use the matching triangle windows on both the tuner and shank to ensure parallel alignment.
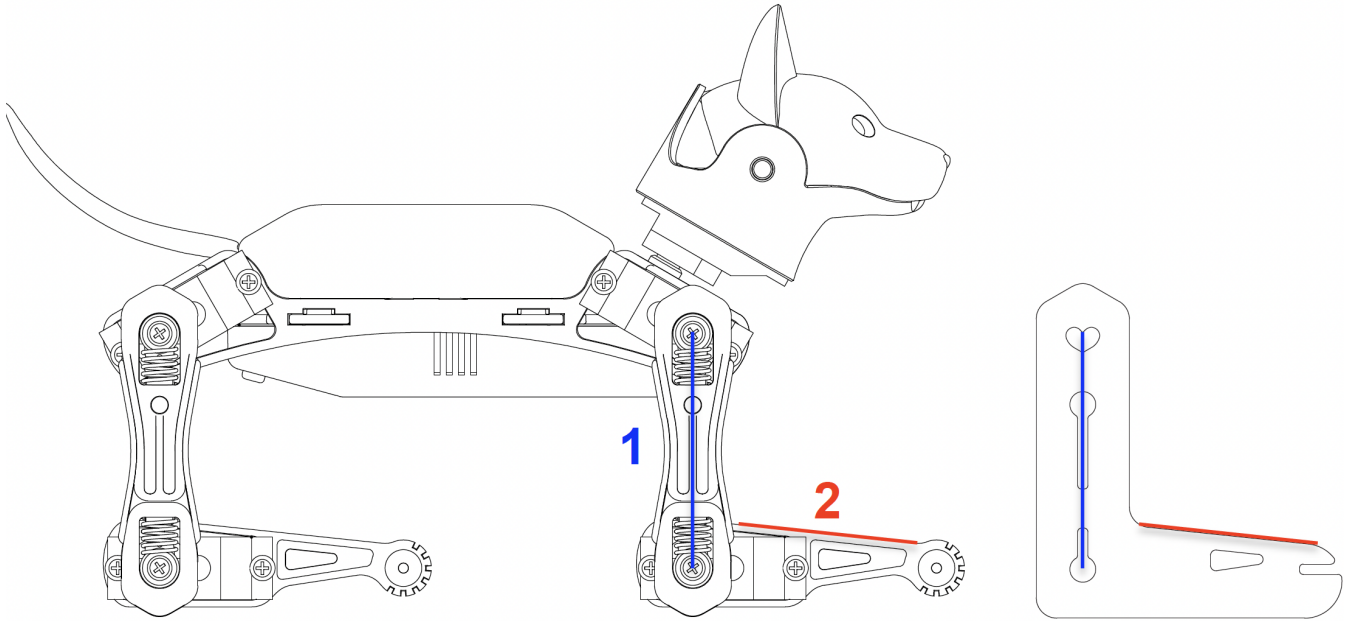
Align the upper leg first
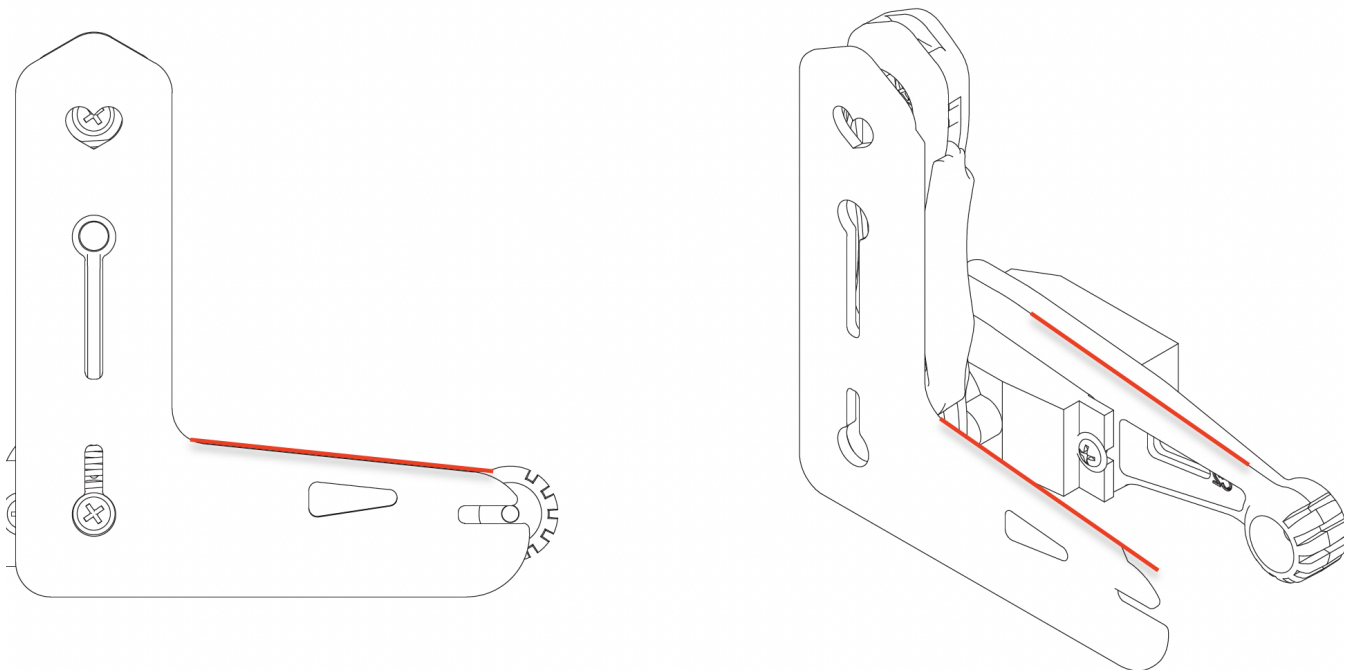
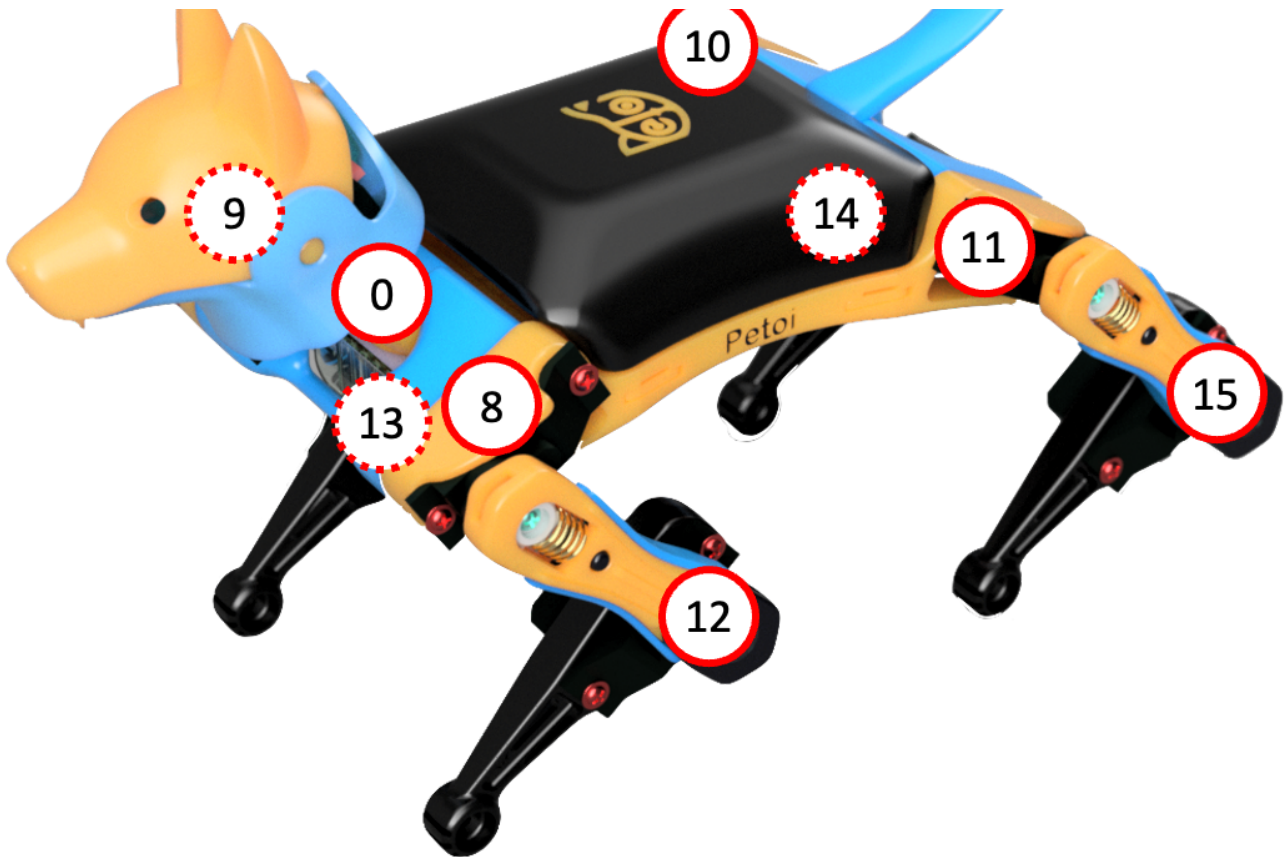Pay attention to the reference edges for the lower leg

6.3.4 Fine-tune the calibration on the software side

The command for fine-tuning calibration (refer to the serial communication protocol for NyBoard) is formatted as `cIndex Offset`. Notice that there's a space between cIndex and Offset.

Joint Indexes

For example,

- `c8 6` means giving the 8th servo an offset of 6 degrees.
- `c0 -4` means giving the 0th servo(the head) an offset of -4 degrees.

> (i) If you find the absolute value of offset is larger than 9, that means you are not attaching the limb closest to its zero state. That will result in a decreased reachable range of the servo on either side. Take off the limb and rotate it by one tooth. It will result in an opposite but smaller offset.

For example, if you have to use -9 as the calibration value, take the limb off, rotate by one tooth then attach back. The new calibration value should be around 5, i.e., they sum up to 14. Avoid rotating the servo shaft during this adjustment.

Find the best offset that can bring the limb to the zero state. It's a process of trial and error.

After calibration, **remember to type 's' to save the offsets**. Otherwise, they will be forgotten when exiting the calibration state. You can even save every time after you're done with one servo.

6.3.5 Testing and validation

After calibration, type 'd' or 'kbalance' to validate the calibration. It will result in Bittle symmetrically moving its limbs between rest and stand state.

Please refer to the previous testing & validation section for the process.

You may need to do a few rounds of calibrations to achieve optimal states.

6.3.6 Center of mass

Try to understand how Bittle keeps balance even during walking. If you are adding new components to Bittle, try your best to distribute its weight symmetrically about the spine. You may also need to slide the battery holder back and forth to find the best spot for balancing. Because the battery is heavier in the front, you can also insert it in a reversed direction to shift the center of mass more towards the back.

You may need to recalibrate if there's a change to the center of mass.

6.4 Finish assembling

6.4.1 Head

To lock in the head, use an M2x4 flat head screw to lock the servo shaft and the servo arm from the bottom side of the chassis.

6.4.2 Limbs(upper and lower legs)

Use the M2x4 screws to lock all the limbs.

6.4.3 Wire shield

There're two snaps on both sides of the wire shield. Their relative height is different.

Flatting the wire of the lower leg to remove any coils. Then snap the wire shield to the upper leg.

The edge of the shield should be parallel to the surface of the upper leg.

You may need to recalibrate if more or less weight is put on Bittle or the center of mass has been changed.

Below shows the wrong installation configuration.

6.4.4 Body cover and tail

The 6-pin female socket is used for connecting the programmer and communication dongles. Carefully bend it forward in a smooth arc if you are not going to attach a Raspberry Pi.
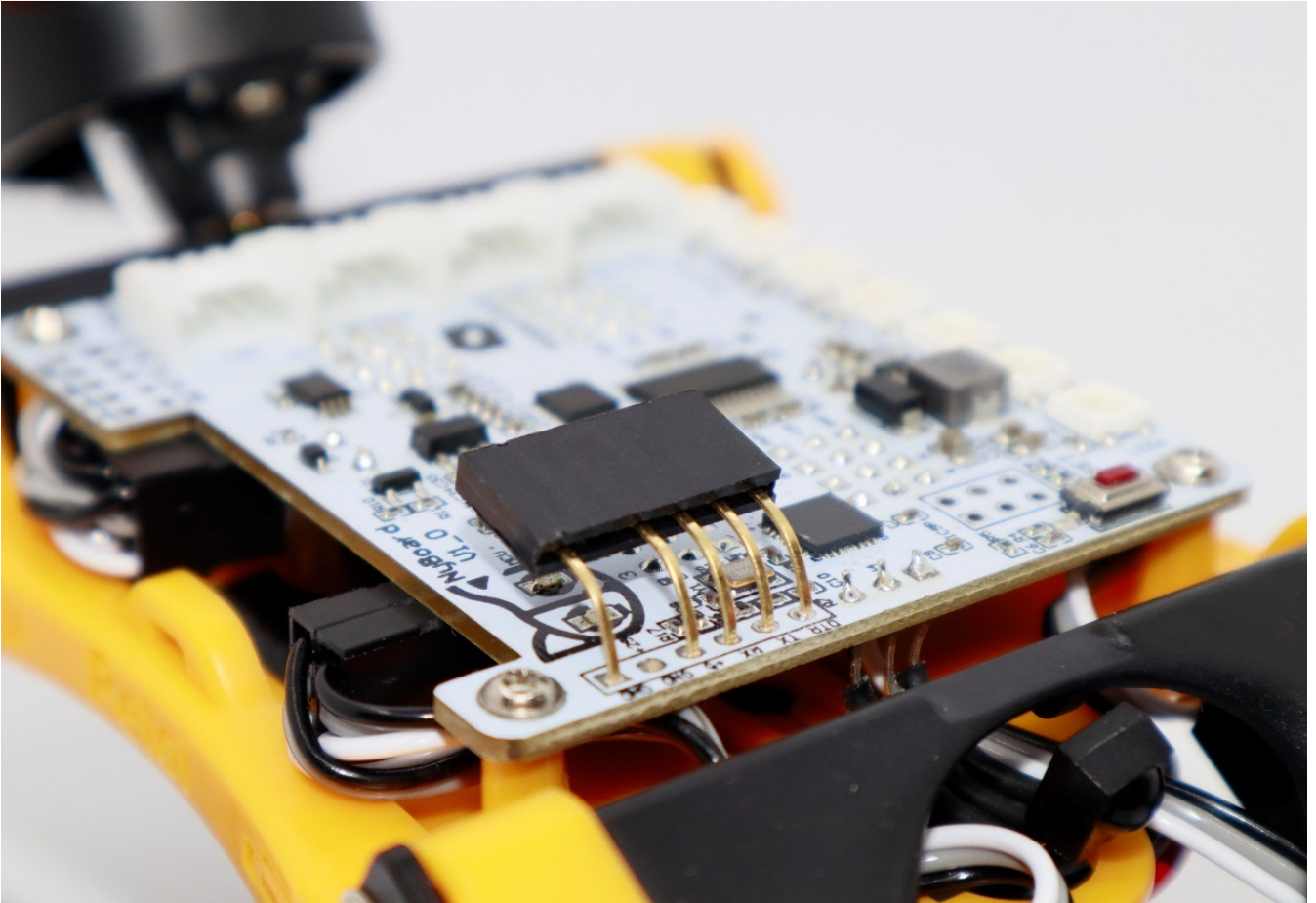
Attach the tail to the pin on the cover.

Snap the cover from one side of the body. Organize the wires so that they don't get stuck between the cover and the body. Then push the cover down to the other side of the body. You should hear a clear snap sound.

7 ⬛ Play with Bittle

"You can't direct the wind, but you can adjust your sails." ⬛

7.1. Control with Arduino IDE

Try the following serial commands in the serial monitor:

- "**k**sit"

- "**m**0 30"

- "**m**0 -30"

- "**k**balance"

- "**k**wkF"

- "**k**trL"

- "**d**"

> ⚠ The quotation mark just indicates that they are character strings. Don't type quotation marks in the serial monitor.

Some more available commands:

	Type	Command	Note	备注
Control	Token	d	rest and shutdown all servos	休息并关闭所有舵机
		g	turn on/off gyro to boost speed	开关陀螺仪加速运动
		p	pause motion and shut off all servos	暂停动作并关闭所有舵机
		c	enter calibration mode	进入校准模式
		m	move a joint to certain angle	把某关节转动到某角度
Skill	Gait	bk	back	后退
		bkL	backLeft	后退 左
		bkR	backRight	后退 右
		vt	stepping on the same spot	原地踏步
		crF	crawl	爬 低重心对角步态
		crL	crawl left	爬 左
		crR	craw right	爬 右
		wkF	walk	走 三脚着地的步态
		wkL	walk left	走 左
		wkR	walk right	走 右
		trF	trot	小跑 对角步态
		trL	trot left	小跑 左
		trR	trot right	小跑 右
		bdF	bound (not recommanded)	兔子跳（不推荐）
	Posture	balance		正常站立演示自平衡性
		buttUp	buttom up	撅屁股
		calib	calibration	校准姿态
		rest		休息
		sit		坐
		sleep		睡
		str	stretch	伸懒腰
		zero		零姿势 给用户自己设计动作的模板
	Behavior	ck	check around	左右看
		hi	hi sequence	打招呼
		pee	pee sequence	撒尿
		pu	push up sequence	俯卧撑
		rc	recovering sequence	四脚朝天后恢复站立(自动激活)
		pd	play dead	主动翻身倒地装死
		bf	back flip	后空翻（隐藏）

ou need to add a 'k' before skills. For example, kbk, kbalance, kck...

e Gaits can be the result of combined command: gait+direction

		Gait	Direction	Serial Command
		cr	F	kcrF
		wk	L	kwkL
		tr	R	ktrR

7.2. Control with Infrared remote

7.2.1. Keymap

Only the position of the buttons matters, though those symbols can help you remember the functionalities. I'm going to define position-related symbols to refer to those keys.

I'm using abbreviations for key definitions to reduce SRAM usage. Due to the limited keys of a physical remote, I always change the definitions for fun.

> ⚠ The following map is just an illustration. Check **OpenCat.h** for the actual key definitions in effect. They are also open to your customization.

rest	F ⬆	gyro
L ⬅	balance	R ➡
pause and shutdown	B ⬇	calibrate
stepping	crawl	walk
trot	sit	stretch
greeting	push up	pee
check	play dead	zero

- **Rest** puts the robot down and shuts down the servos
- Pressing **F/L/R** will make the robot to move forward/left/right
- **B** will make the robot move backward
- **Calibrate** puts the robot into calibration posture and turns off the gyro
- **Stepping** lets the robot step at the original spot
- **Crawl/walk/trot** are gaits that can be switched and combined with the direction buttons
- Buttons after **trot** are preset postures or other skills
- **Gyro** will turn on/off the gyro for self-balancing. Turning off the gyro can accelerate and stabilize the slower gaits. But it's NOT recommended for faster gaits such as trot.

We also made a customized remote panel for future batches. Previous users can download the design file and print it on A4 paper.

> 📎 newPanel.pdf 2MB
> PDF
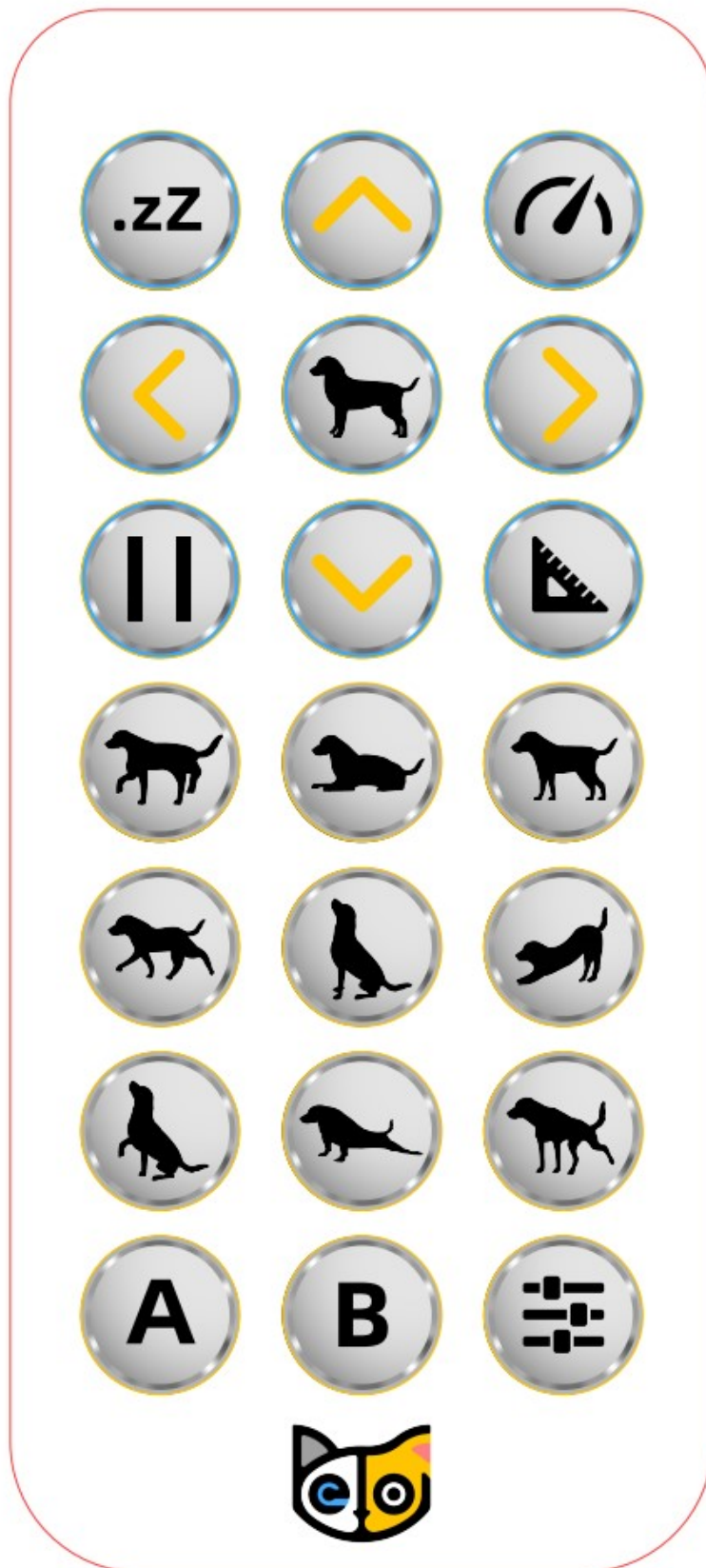
7.2.2. Check out the following featured motions

- **Rest** puts the robot down and shuts down the servos. It's always safe to click it if Bittle is doing

something **AWKWARD**.

- **Balance** is the neutral standing posture. You can push Bittle from the side, or make it stand up will hind legs and tail. You can test its balancing ability on a fluctuating board. Actually balancing is activated in most postures and gaits.

- **Shut down servos** will pause the current motion and turn off the power of the servos so that they will not hold the position.

- Pressing **F/L/R** will make the robot move forward/left/right

- **B** will make the robot move backward

- **Calibrate** puts the robot into calibration posture and turns off the gyro

- **Stepping** lets the robot step at the original spot

- **Crawl/walk/trot/run** are the gaits that can be switched and combined with the direction buttons

- **Gyro** will turn on/off the gyro for self-balancing. Turning off the gyro can accelerate and stabilize the slower gaits. But it's NOT recommended for faster gaits such as run and bound.

- Buttons after **trot** are preset postures or other skills

- Different surfaces have different friction and will affect walking performance. The current trained parameters work better without the silicone toe covers. They are provided for your experiments. The carpet will be too bushy for Bittle's short legs. It may only crawl (command **kcrF**) over this kind of tough terrain.

- You can pull the battery pack down and slide along the longer direction of the belly. That will tune the center of mass, which is very important for walking performance. Otherwise, it may keep falling down.

- When Bittle is walking, you can let it climb up/down a small slope (<10 degrees)

- Whatever Bittle is doing, you can lift it vertically, and it will stop moving, just like a dog scruffed on the neck.

ⓘ
- If Bittle keeps beeping after you connect the USB uploader, with numbers printed in the serial monitor, it's the low voltage alarm being triggered. You need to power NyBoard with the battery to pass the threshold.

- The servos are designed to be driven by internal gears. Avoid rotating the servos too fast from outside.

- **Don't keep Bittle running for too long.** It will overheat the electronics and reduce the servos' life span.

- If you feel something is wrong with Bittle, press the reset button on NyBoard to restart the program.

- Bittle has acrophobia! If you lift it and rotate it over a certain degree, its current movement will be interrupted. Don't flip Bittle over to scare it!

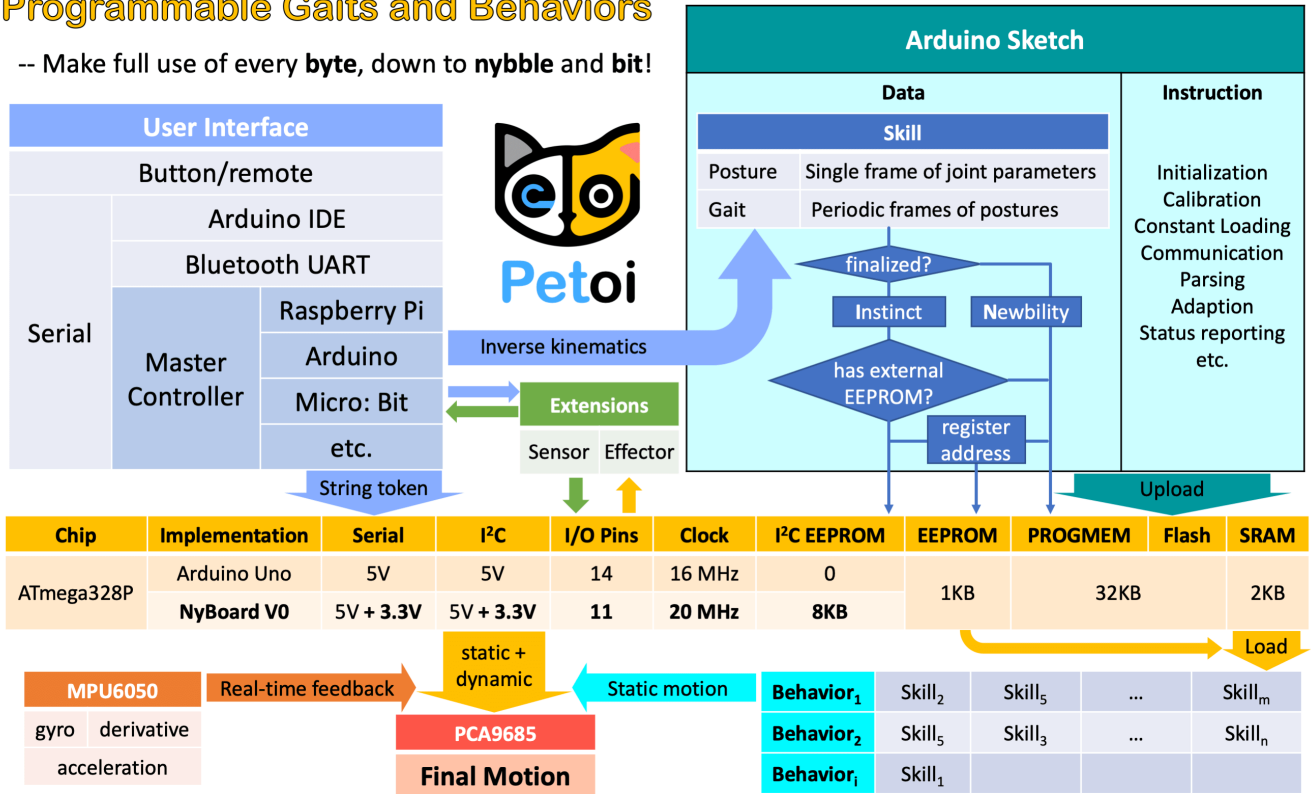- Be kind as if you were playing with a real dog. (^=◕ᴥ◕=^)

8 🐱 Teach Bittle New Skills

"Give a cat a fish and you feed it for a day. Teach a cat to fish and you feed it for a lifetime." 🐱

8.1. Understand skills in InstinctBittle.h.

EEPROM has limited (1,000,000) write cycles. So I want to minimize the write operations on it.

There are two kinds of skills: **Instincts** and **Newbility**. The addresses of both are written to the onboard EEPROM(1KB) as a lookup table, but the actual data is stored at different memory locations:

- I2C EEPROM (8KB) stores **Instincts**.

The Instincts are already fine-tuned/fixed skills. You can compare them to "muscle memory". Multiple Instincts are linearly written to the I2C EEPROM only once with **WriteInstinct.ino**. Their addresses are generated and saved to the lookup table in onboard EEPROM during the runtime of **WriteInstinct.ino**.

- PROGMEM (sharing the 32KB flash with the sketch) stores **Newbility**.

A Newbility is any new experimental skill that requires a lot of tests. It's not written to the I2C nor onboard EEPROM, but the flash memory in the format of PROGMEM. It has to be uploaded as one part of the Arduino sketch. Its address is also assigned during the runtime of the code, though the value rarely changes if the total number of skills (including all Instincts and Newbilities) is unchanged.

8.2. Example InstinctBittle.h

```
1  //a short version of InstinctBittle.h as example
2
3  #define BITTLE
4  #define NUM_SKILLS 4
5  #define I2C_EEPROM
6
7  const char rest[] PROGMEM = {
8  1, 0, 0, 1,
9    -30, -80, -45,   0,  -3,  -3,   3,   3,  75,  75,  75,  75, -55, -55, -55, -55,};
10 const char zero[] PROGMEM = {
11 1, 0, 0, 1,
12    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,};
13
14 const char crF[] PROGMEM = {
15 36, 0, -3, 1,
16   61,  68,  54,  61, -26, -39, -13, -26,
17   66,  61,  58,  55, -26, -39, -13, -26,
18   ...
19   51,  81,  45,  72, -25, -37, -12, -25,
20   55,  76,  49,  68, -26, -38, -13, -26,
21   60,  70,  53,  62, -26, -39, -13, -26,
22 };
23
24 const char pu[] PROGMEM = {
25 -8, 0, -15, 1,
26  6, 7, 3,
27    0,   0,   0,   0,   0,   0,   0,   0,  30,  30,  30,  30,  30,  30,  30,  30,       5
28   15,   0,   0,   0,   0,   0,   0,   0,  30,  35,  40,  29,  50,  15,  15,  15,       5
29   30,   0,   0,   0,   0,   0,   0,   0,  27,  35,  40,  60,  50,  15,  20,  45,       5
30   15,   0,   0,   0,   0,   0,   0,   0,  45,  35,  40,  60,  25,  20,  20,  60,       5
31    0,   0,   0,   0,   0,   0,   0,   0,  50,  35,  75,  60,  20,  30,  20,  60,       6
32  -15,   0,   0,   0,   0,   0,   0,   0,  60,  60,  70,  70,  15,  15,  60,  60,       6
33    0,   0,   0,   0,   0,   0,   0,   0,  30,  30,  95,  95,  60,  60,  60,  60,       6
34   30,   0,   0,   0,   0,   0,   0,   0,  75,  70,  80,  80, -50, -50,  60,  60,       8
35 };
36
37 #if !defined(MAIN_SKETCH) || !defined(I2C_EEPROM)
38 const char* skillNameWithType[] =
39 {"crI", "puI", "restI", "zeroN",};
40 const char* progmemPointer[] =
41 {cr, pu, rest, zero, };
42 #else
43 const char* progmemPointer[] = {zero};
44 #endif
```

8.2.1. Defined constants

```
define WalkingDOF 8
```

defines the number of DoF (Degree of Freedom) for walking is 8 on Bittle.

```
define NUM_SKILLS 4
```

defines the total number of skills is 4. It should be the same as the number of items in the list `const char* skillNameWithType[]`.

```
define I2C_EEPROM
```

Means there's an I2C EEPROM on NyBoard to save Instincts.

> ⚠ If you are building your own circuit board that doesn't have it, comment out this line. Then both kinds of skills will be saved to the flash as PROGMEM. Obviously, it will reduce the available flash space for functional codes. If there were too many skills, it may even exceed the size limit for uploading the sketch.

8.2.2. Data structure of skill array

One frame of joint angles defines a static **posture**, while a series of frames defines sequential postures, such as a **gait** or a **behavior**. Observe the following two examples:

```
 1 const char rest[] PROGMEM = {
 2 1, 0, 0, 1,
 3   -30, -80, -45,   0,  -3,  -3,   3,   3,  75,  75,  75,  75, -55, -55, -55, -55,};
 4
 5 const char crF[] PROGMEM = {
 6 36, 0, -3, 1,
 7   61,  68,  54,  61, -26, -39, -13, -26,
 8   66,  61,  58,  55, -26, -39, -13, -26,
 9   ...
10   51,  81,  45,  72, -25, -37, -12, -25,
11   55,  76,  49,  68, -26, -38, -13, -26,
12   60,  70,  53,  62, -26, -39, -13, -26,
13 };
14
```

They are formatted as:

	Total # of Frames	Expected Body Orientation		Angle Ratio	Indexed Joint Angles															
		Roll	Pitch		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	1!
rest	1	0	0	1	-30	-80	-45	0	-3	-3	3	3	60	60	-60	-60	-45	-45	45	4!
crF	36	0	-3	1									61	68	54	61	-26	-39	-13	-2
													...	...	...	...	...	...	...	..
													60	70	53	62	-26	-39	-13	-2

rest is a static posture, it has only one frame of 16 joint angles. **crF** is the abbreviation for "crawl forward". It

has 36 frames of 8 (or 12, depending on the number of walking DOF) joint angles that form a repetitive gait. Expected body orientation defines the body angle when the robot is conducting the skill. If the body is tilted from the expected angles, the balancing algorithm will calculate some adjustments. Angle ratio is used when you want to store angles larger than the range of -128 to 127. Change the ratio to 2 so that you can save those large angles by dividing 2.

> ⓘ A posture has only one frame, and a gait has more than one frames and will be looped over.

The following example is a behavior:

```
 1 const char pu[] PROGMEM = {
 2 -8, 0, -15, 1,
 3  6, 7, 3,
 4     0,   0,   0,   0,   0,   0,   0,   0,  30,  30,  30,  30,  30,  30,  30,  30,        5
 5    15,   0,   0,   0,   0,   0,   0,   0,  30,  35,  40,  29,  50,  15,  15,  15,        5
 6    30,   0,   0,   0,   0,   0,   0,   0,  27,  35,  40,  60,  50,  15,  20,  45,        5
 7    15,   0,   0,   0,   0,   0,   0,   0,  45,  35,  40,  60,  25,  20,  20,  60,        5
 8     0,   0,   0,   0,   0,   0,   0,   0,  50,  35,  75,  60,  20,  30,  20,  60,        6
 9   -15,   0,   0,   0,   0,   0,   0,   0,  60,  60,  70,  70,  15,  15,  60,  60,        6
10     0,   0,   0,   0,   0,   0,   0,   0,  30,  30,  95,  95,  60,  60,  60,  60,        6
11    30,   0,   0,   0,   0,   0,   0,   0,  75,  70,  80,  80, -50, -50,  60,  60,        8
12 };
```

pu is short for "push up", and is a "behavior". Its data structure contains more information than posture and gait.

The first four elements are defined the same as before, except that the number of frames is saved as a negative value to indicate that it's a behavior. The next three elements define the repeating frames in the sequence: starting frame, ending frame, looping cycles. So the **6, 7, 3** in the example means the behavior should loop from the 7th to the 8th frame for 3 times (the index starts from 0). The whole behavior array will be executed only once, rather than looping over like the gait.

Each frame contains 16 joint angles, and the last 4 elements define the speed of the transition, and the delay condition after each transition:

1. The default speed factor is 4, it can be changed to an integer from 1 (slow) to 127 (fast). The unit is **degree per step**. If it's set to 0, the servo will rotate to the target angle by its maximal speed (about 0.07sec/60 degrees). It's not recommended to use a value larger than 10 unless you understand the risks.

2. The default delay is 0. It can be set from 0 to 127, the unit is **50 ms**.

3. The 3rd number is the trigger axis. If it's not 0, the previous delay time will be ignored. The trigger of the next frame will depend on the body angle on the corresponding axis. 1 for the pitch axis, and 2 for the roll axis. The sign of the number defines the direction of the threshold, i.e. if the current angle is smaller or larger than the trigger angle.

4. The 4th number is the trigger angle. It can be -128 to 127 degrees.

8.2.3. Suffix for indicating Instinct and Newbility

You must upload **WriteInstinct.ino** to have the skills written to EEPROM for the first time. The following information will be used:

```
1 const char* skillNameWithType[] =
2 {"crI", "puI", "restI", "zeroN",};
3 const char* progmemPointer[] =
4 {cr, pu, rest, zero, };
```

> ⓘ Notice the suffix **I** or **N** in the skill name strings. They tell the program where to store skill data and when to assign their addresses.

Later, if the uploaded sketch is the main sketch **OpenCat.ino**, and if you are using NyBoard that has an I2C EEPROM, the program will only need the pointer to the Newbility list

```
const char* progmemPointer[] = {zero};
```

to extract the full knowledge of pre-defined skills.

8.3. Define new skills and behaviors

8.3.1 Modify the existing skill template

There's already a skill called "zeroN" in InstinctBittle.h. It's a posture at the zero state waiting for your new definition.

You can first use the command `mIndex Offset` to move an individual joint to your target position, then replace the joint angles (bold fonts) in array at once:

```
1 const char zero[] PROGMEM = {
2 1, 0, 0, 1,
3   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,};
```

Because it's declared as a Newbility and doesn't require writing to I2C EEPROM, you can simply upload **OpenCat.ino** everytime you change the array (without uploading **WriteInstinct.ino**). You can trigger the new posture by pressing **7>** on the IR remote, or type `kzero` in the serial monitor.

You can rename this skill, but remember to update the keymap of the IR remote.

8.3.2. Add more skills in InstinctBittle.h

You can add more skills in InstinctBittle.h. Remember to increase the skill number at the beginning of the file and add the corresponding skill name and pointer in the skill list array.

A skill can be called from the serial monitor with the token 'k' command. For example, `ksit` will move Bittle to posture "sit".

You can also tune new skills by sending posture frames through the serial port, using the **m, i, l** tokens without uploading a new sketch. After fine-tuning the skill, you can save it in the instinctBittle.h and upload it to the board as a newbility or instinct.

> ⚠ Always check the actual code for the available skill names. We may alter the skill set as we iterate the software.

This git repo is a good starting point if you want to develop a customized gait. If you are going to do some inverse kinematics calculation, you may use the following key dimensions to build your model.
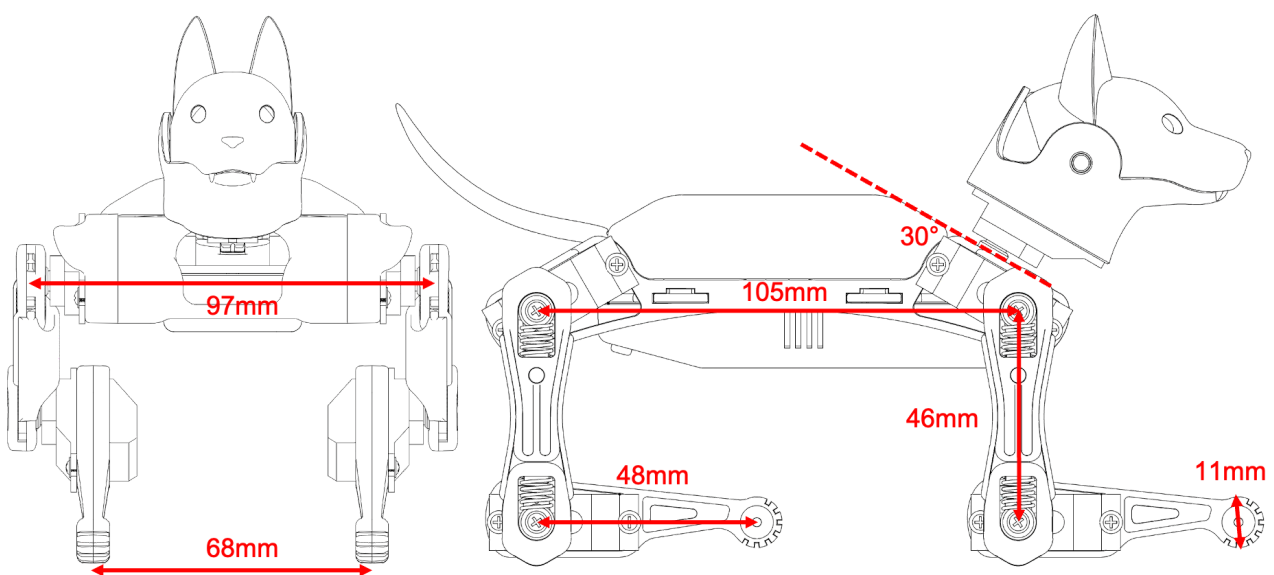
8.3.3. Automation

So far Bittle is controlled by the infrared remote. You make decisions for Bittle's behavior.

You can connect Bittle with your computer or smartphone, and let them send out instructions automatically. Bittle will try its best to follow those instructions.

By adding some sensors (like a touch sensor), or some communication modules (like a voice control module), you can bring new perception and decision abilities to Bittle. You can accumulate those automatic behaviors and eventually make Bittle live by itself!

Another direction is to set up a simulation for Bittle then test the model in reality. You may use this Unified Robot Description Format (URDF) file for Bittle to set up an NVIDIA Omniverse simulation.
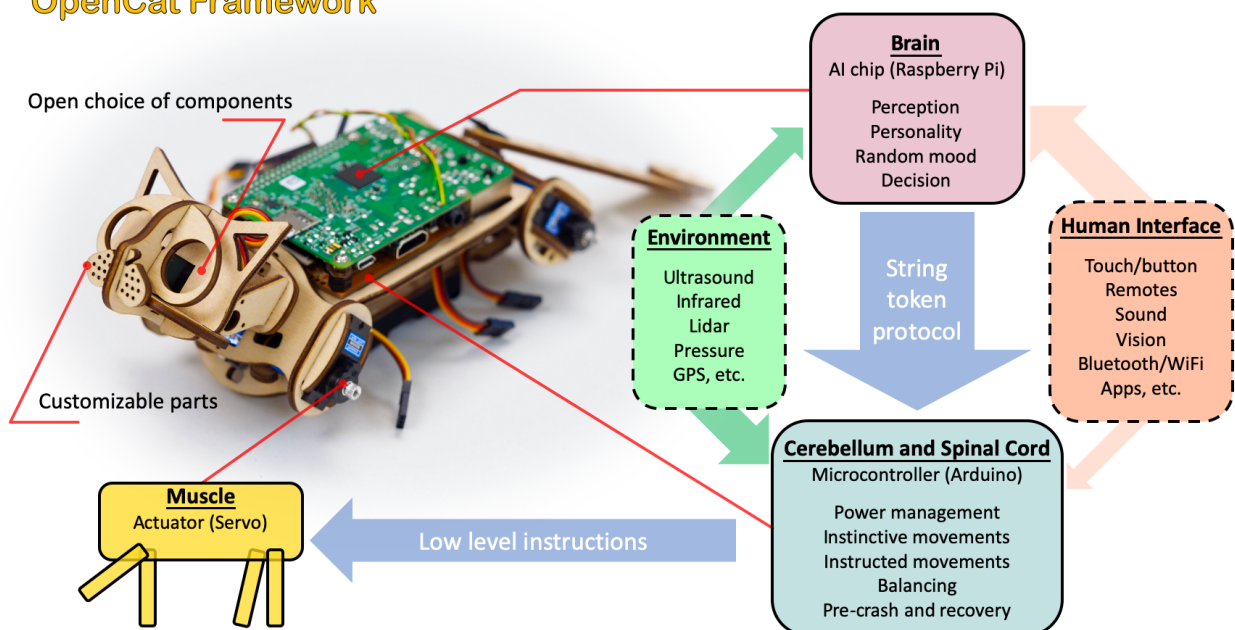
9 🐾 Understand OpenCat.h

"Let the cat out of the bag." 🤷‍♂️

🐾It will make another textbook.

The controlling framework behind Bittle is OpenCat, which I've been developing for a while. You can read more stories from my posts on Hackster.io.


It's too much work for now 🤷‍♂️ but you are welcome to discuss it with me on the forum or through email. For

example, there are code reviews on the forum. I will keep the code compatible with future OpenCat models, and even your own DIY robots! Hopefully, the documentation will be completed during the process.

10⃝ Extensible Modules

The head of Bittle is designed to be a clip to hold extensible modules. We compiled a sensor pack with some popular modules, but its contents may change in the future. You can also wire other add-ons thanks to the rich contents of the Arduino and Raspberry Pi community.
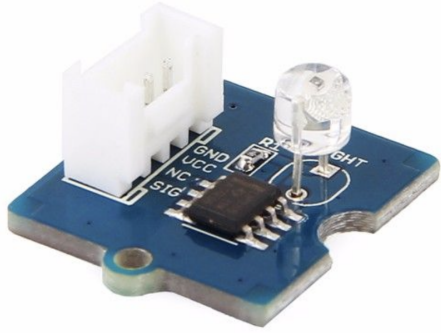
You can find the demo codes of these modules in our GitHub repository. They should be in the ModuleTests folder if you download the whole OpenCat repository.
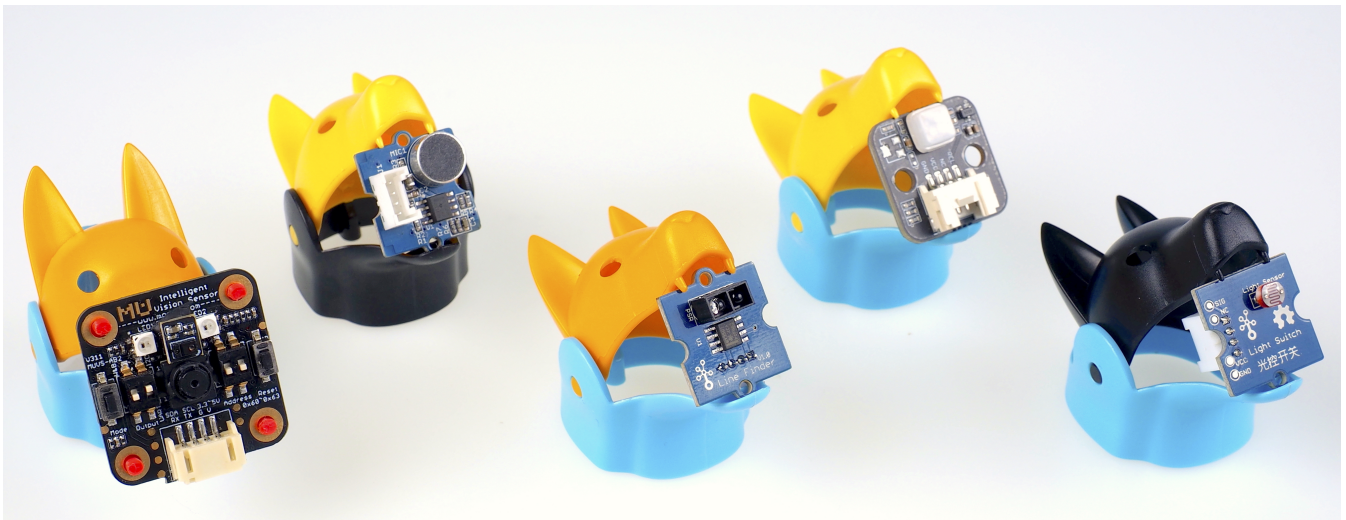
The loudness and light level modules can generate analog readings for the corresponding signals and should be connected to the analog Grove socket.
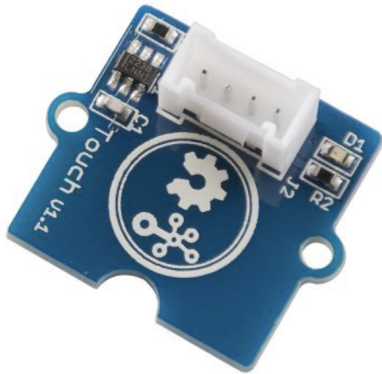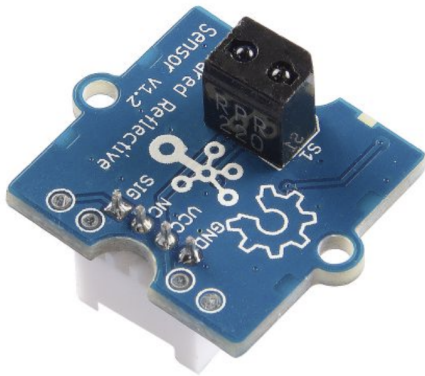

Grove - Sound Sensor/ Noise Detector
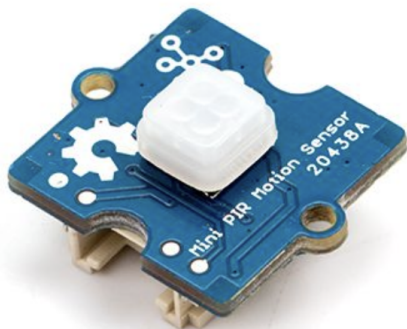
Grove - Light Sensor v1.2 - LS06-S phototransistor

The touch, reflection, PIR sensors can generate digital 1 or 0 as a switch signal. So they should be connected to the digital Grove socket. In the demo code, we use the fourth socket with D6 and D7.

Grove - Touch Sensor

Grove - Infrared Reflective Sensor v1.2

Grove - mini PIR motion sensor

The intelligent camera, gesture, and OLED module should be connected to the I2C Grove socket.

Grove - Gesture Sensor for Arduino (PAJ7620U2)

Grove - OLED Display 0.96" (SSD1315)
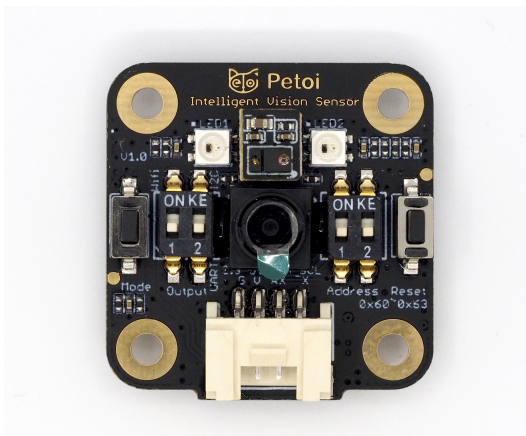
MU Vision Sensor 3

- documentation
- advanced user manual

11 Tutorial on Creating New Skills

Preparation

Get familiar with the standard process of assembly, uploading the standard program **Opencat.ino**, and calibration (refer to chapter 6 of Bittle User Manuals). Validate that the following functions work as expected.

- Press the button （**>>|**）which is in the 2nd row, the 2nd column on the IR remote. Later we will use (2, 2) as the index. You can also enter **kbalance** via serial port. Bittle should stand up and keep balance;

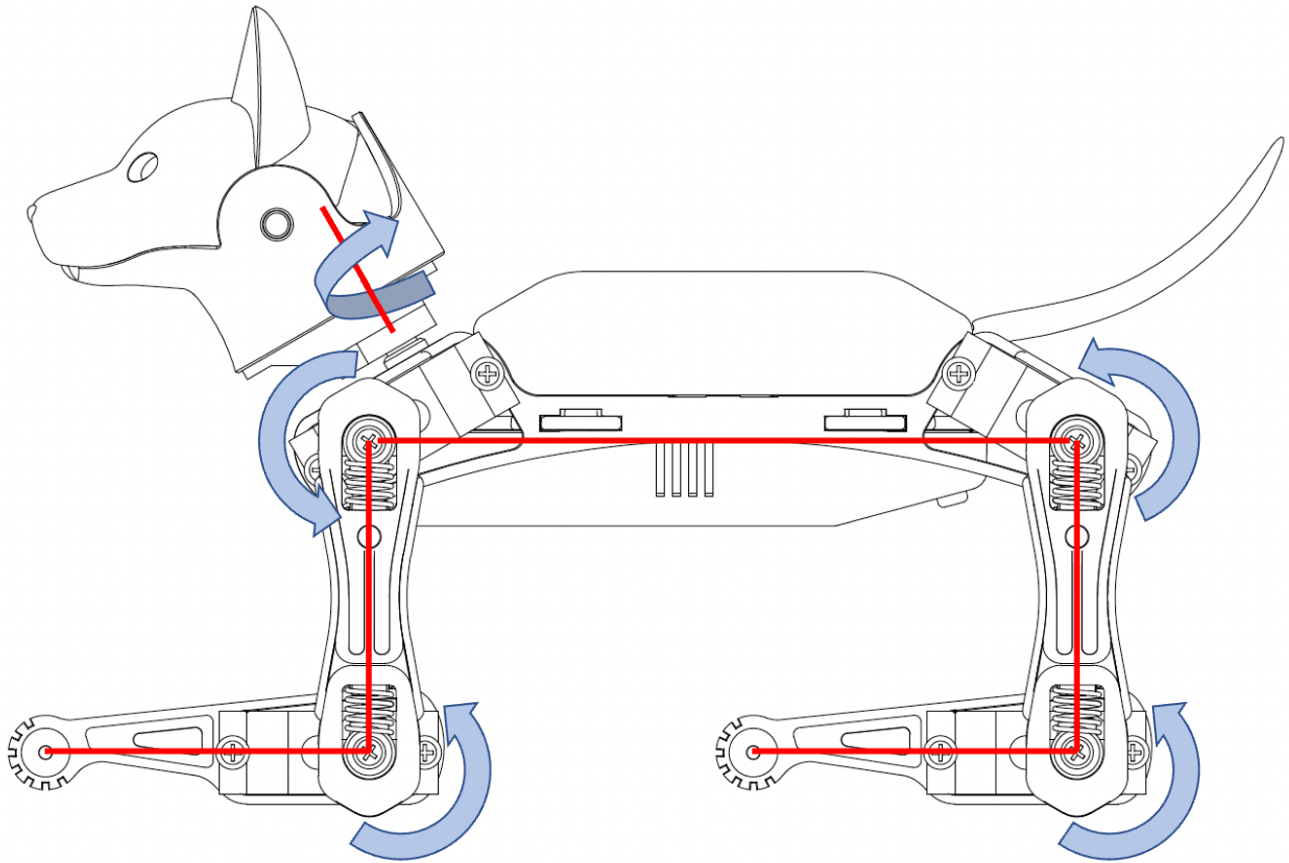- Press the button （**9**）which is in the 7th row, the 3rd column on the IR remote. Later we will use (7, 3) as the index. You can also enter **kzero** via serial port. Bittle should enter a posture similar to the calibration state, which is the "zero" skill in the program (as shown in the figure below).

A visualization of the perfect zero state

Open the folder **WriteInstinct/**, create a backup file of instinctBittle.h as instinctBittle.hBAK.

The coordinate system of Bittle is shown in the figure below.
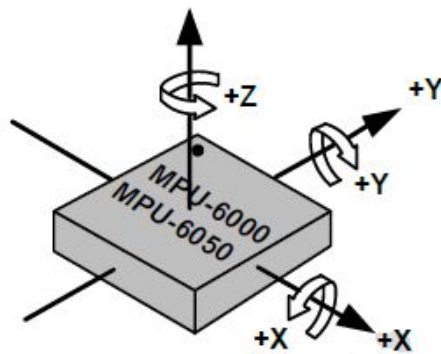
The coordinate system

Yaw: Rotate around the Z-axis.

Pitch: Rotate around the Y-axis (nose up/down).

Roll: Rotate around the X-axis (tilt left/right).

For the legs, on the left side, counterclockwise is positive, clockwise is negative. On the right side, the rotation directions are mirrored. The origin position and rotation direction of a single leg (upper/lower leg) around the joint are shown in the first figure.
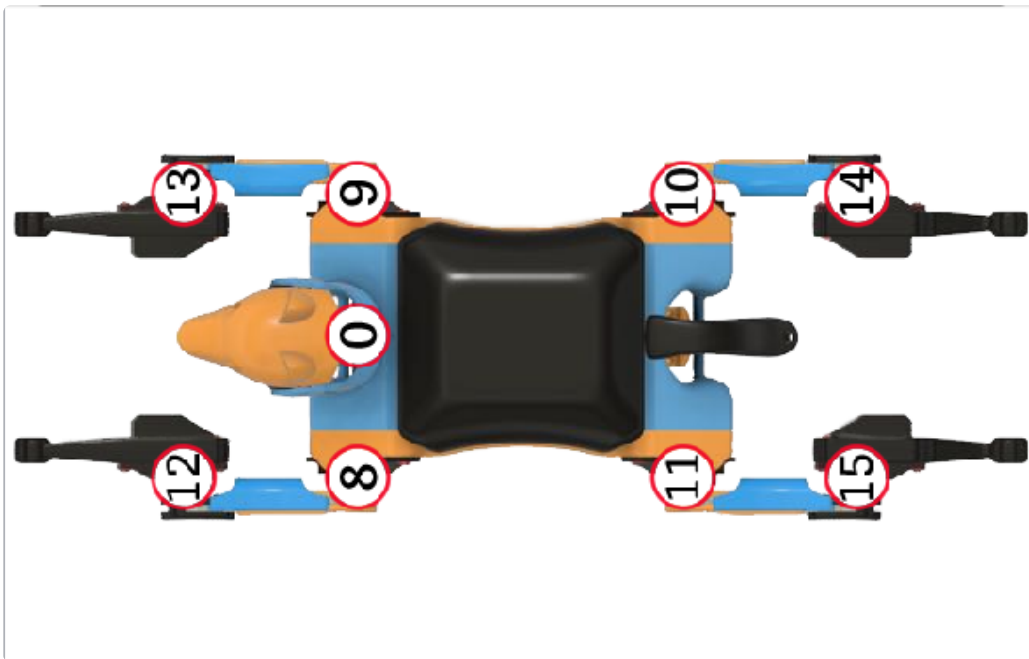
The indexing order of all the joints is shown in the figure below:

The indexing order of all the joints

The skill arrays are defined in WriteInstinct/instinctBittle.h, formated as the figure below. **Note** the index starts from 0.

	Total # of Frames	Expected Body Orientation		Angle Ratio	Indexed Joint Angles															
		Roll	Pitch		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	1!
rest	1	0	0	1	-30	-80	-45	0	-3	-3	3	3	60	60	-60	-60	-45	-45	45	4!
crF	36	0	-3	1									61	68	54	61	-26	-39	-13	-2
													...	...	...	...	...	...	...	..
													60	70	53	62	-26	-39	-13	-2

The format of skill arrays

Total # of Frames defines the number of frames of a certain skill.

For example, **rest** is a static posture, it has only one frame of 16 joint angles.

crF is the abbreviation for "crawl forward". It has 36 frames of 8 (or 12, depending on the number of walking DOF) joint angles that form a repetitive gait.

Expected body orientation defines the body angle when the robot is conducting the skill. If the body is tilted from the expected angles, the balancing algorithm will calculate some adjustments.

Angle ratio is used when you want to store angles larger than the range of -128 to 127. Change the ratio to 2 so that you can save those large angles by dividing 2.

Understand the format of a posture

Zero is a static posture, Find the **zero** array in instinctBittle.h :

```
1 const char zero[] PROGMEM = {
2 1, 0, 0, 1,
3    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,};
```

Change some of these values to :

```
1 const char zero[] PROGMEM = {
2 1, 0, 0, 1,
3 70, 0, 0, 0, 0, 0, 0, 0, -60, 0, 0, 0, 60, 0, 0, 0,};
```

```
1 const char zero[] PROGMEM = {
2 1, 0, 0, 1,
3 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,};
```

```
1 const char zero[] PROGMEM = {
2 1, 0, 0, 1,
3 70, 0, 0, 0, 0, 0, 0, 0, -60, 0, 0, 0, 60, 0, 0, 0,};
```

Save the change, and upload the program **OpenCat.ino** to Bittle (**Note**: there is no need to upload writeinsict.ino or opencat.h), after upload, press the button (**9**) which is in the 7th row, the 3rd column on the IR remote to trigger the modified **zero** skill. You can see the posture is changed.

Explanation why the new zero posture looks in that way. The coordinate system with the new angle data is shown in the figure below.
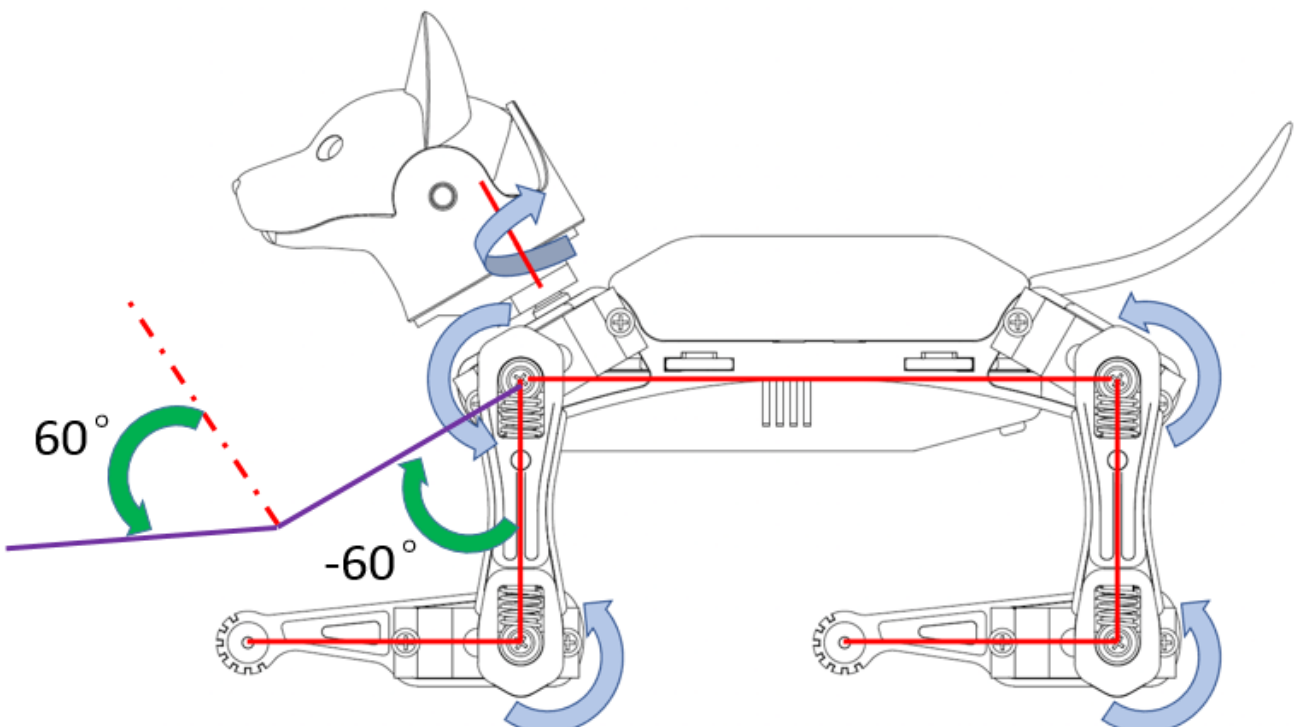
The first element (1) represents the total number of frames of the skill , 1 means it is a static posture.

The 4th element (1) represents the Angle ratio. It means all the following indexed joint angles are actual angles (because each of them multiply by 1).

From the 5th to the 20th elements represent 16 indexed joint angles.

For Bittle, the 5th element (joint index 0) means the servo on Bittle's neck rotates counterclockwise 70 (the unit is in degrees). Bittle's head turn to it's left side.

The 13th element (joint index 8) means Bittle's left upper leg rotates clockwise 60 (the unit is in degrees) around the joint.

The 17th element (joint index 12) means Bittle's left lower leg rotates counterclockwise 60 (the unit is in degrees) around the joint.

All the other indexed joint angles remain 0 (the unit is in degrees).

You can define a new posture and upload it to the robot. Call the new posture with the IR remote or the serial monitor.

Explain exceptions of expected body orientations

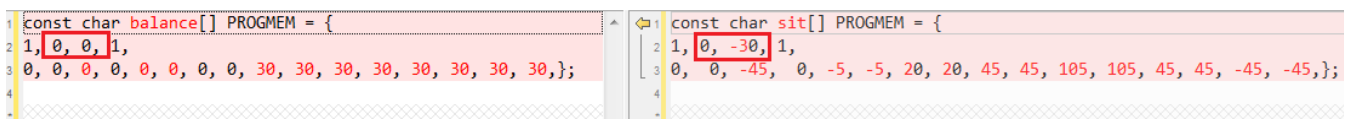Look at the example of:

```
1 const char balance[] PROGMEM = {
2 1, 0, 0, 1,
3    0,   0,   0,   0,   0,   0,   0,   0,  30,  30,  30,  30,  30,  30,  30,  30,};
```

and

```
1 const char sit[] PROGMEM = {
2 1, 0, -30, 1,
3    0,   0, -45,   0,  -5,  -5,  20,  20,  45,  45, 105, 105,  45,  45, -45, -45,};
```


With the gyro activated, rotate the robot and see its adjustments. Explain why the expected angle is needed in the definition.

The 2nd and 3rd elements represent Expected body orientation, corresponding to the roll angle and the pitch angle.

The unit is in degrees.

The sign of the number follows the right-handed spiral rule, look in the direction pointed by the axis arrow, clockwise is positive, counterclockwise is negative.

With the gyro activated, rotate Bittle, when the body is tilted from the expected angles, the balancing algorithm will calculate some adjustments to keep it in this posture.

Explain exceptions of large angles

Look at the example of

```
1 const char rc[] PROGMEM = {
2 -3, 0, 0, 2,
3  0, 0, 0,
4    0,   0,   0,   0,   0,   0,   0,   0, -88, -43,  67,  87,  42, -35,  42,  42,      15
5    0,   0,   0,   0,   0,   0,   0,   0, -83, -88,  87,  42,  42,  42,  42, -40,      15
6   -8, -20, -11,   0,  -1,  -1,   0,   0,  18,  18,  18,  18, -14, -14, -14, -14,      10
7 };
```

It's designed for large angles out of the range -128~127.

The 4th element represents the angle ratio. It means all the following all indexed joint angles real values is equal to each of them multiply by the value of this angle ratio.

Understand the format of a gait

A series of frames defines sequential postures, such as a gait. Find the **bk** array in instinctBittle.h :

```
1 const char bk[] PROGMEM = {
2   35, 0, 0, 1,
3   46,  54,  46,  54,  -5, -23,  -5, -23,
4   43,  58,  43,  58,  -5, -24,  -5, -24,
5   ......
6   52,  43,  52,  43,  -5, -21,  -5, -21,
7   50,  48,  50,  48,  -5, -22,  -5, -22,
8   47,  53,  47,  53,  -5, -23,  -5, -23,
9 };
```

bk is the abbreviation for "back".

The first four elements are defined the same as before, The first element (35) means it has 35 frames. Next are 35 frames of 8 indexed joint angles that form a repetitive gait.

Understand the format of a behavior

Modify the zero skill as:

```
1 const char zero[] PROGMEM = {
2 -1, 0, 0, 1,
3 0, 0, 0,
4 70, 0, 0, 0, 0, 0, 0, 0, -60, 0, 0, 0, 60, 0, 0, 0,4,0,0,0};
```

```
const char zero[] PROGMEM = {
-1, 0, 0, 1,
0, 0, 0,
70, 0, 0, 0, 0, 0, 0, 0, -60, 0, 0, 0, 60, 0, 0, 0,4,0,0,0};
```
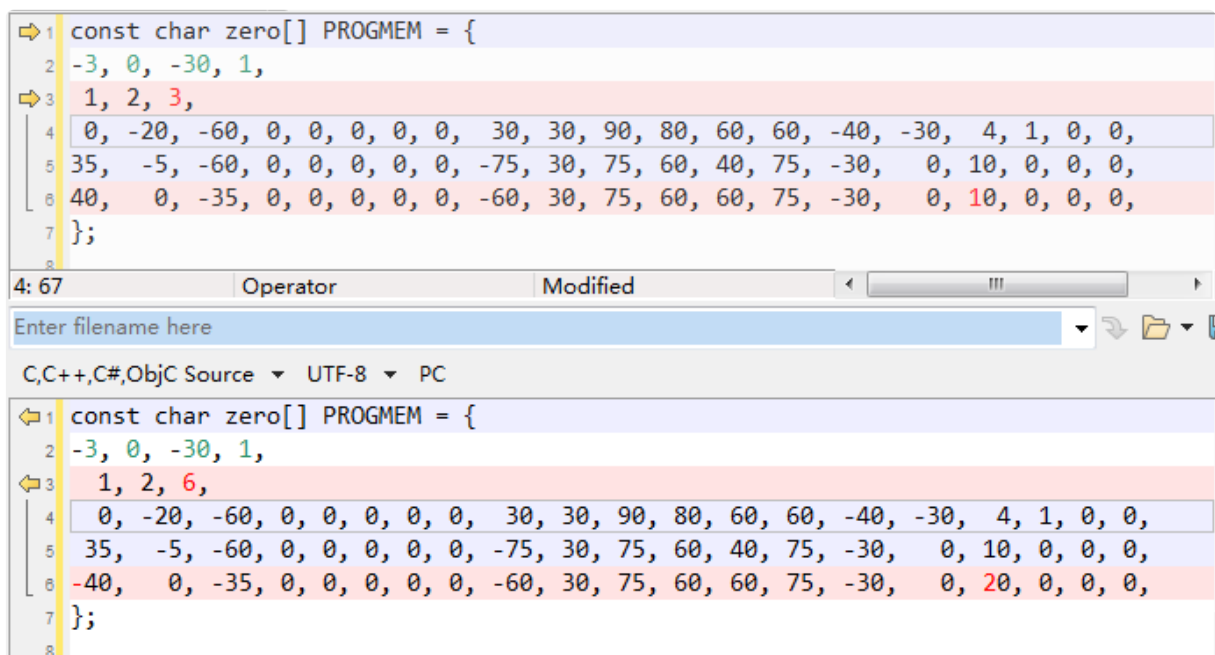
Upload the new zero skill and see the effect. It should be the same. (Later explanation a posture can be considered as a behavior or gait with one frame.)

Copy the content of **hi** array to zero:

```
1 const char zero[] PROGMEM = {
2 -3, 0, -30, 1,
3  1, 2, 3,
4  0, -20, -60,   0,   0,   0,   0,   0,  30,  30,  90,  80,  60,  60, -40, -30,  4, 1, 0, 0
5 35,  -5, -60,   0,   0,   0,   0,   0, -75,  30,  75,  60,  40,  75, -30,   0,  10, 0, 0, 
6 40,   0, -35,   0,   0,   0,   0,   0, -60,  30,  75,  60,  60,  75, -30,   0,  10, 0, 0, 
7 };
```

Change a few values:

```
1 const char zero[] PROGMEM = {
2 -3, 0, -30, 1,
3  1, 2, 6,
4  0, -20, -60, 0, 0, 0, 0, 0, 30, 30, 90, 80, 60, 60, -40, -30, 4, 1, 0, 0,
5 35, -5, -60, 0, 0, 0, 0, 0, -75, 30, 75, 60, 40, 75, -30, 0, 10, 0, 0, 0,
6 -40, 0, -35, 0, 0, 0, 0, 0, -60, 30, 75, 60, 60, 75, -30, 0, 20, 0, 0, 0,
7 };
```


Save and upload OpenCat.ino. Call the new zero skill to see the effect.

Explanation on the format of a behavior.

For example, the modified **hi** behavior:

The first four elements are defined the same as before, except that the number of frames is saved as a negative value (-3) to indicate that it's a behavior.

The 2nd element (0) means the Roll rotation body angle is 0 (The unit is in degrees). The 3rd element (-30) means the Pitch rotation body angle is -30 (The unit is in degrees). If the body is tilted from the expected angles, the balancing algorithm will calculate some adjustments. The 4th element (1) means all the following all indexed joint angles are real values.

The next three elements define the repeating frames in the sequence: starting frame (1), ending frame (2), looping cycles (6). So the **1, 2, 6** in the example means the behavior should loop from the 2nd to the 3rd frame for 6 times (the index starts from 0). The whole behavior array will be executed only once, rather than looping over like the gait.

For behavior, each frame contains 16 indexed joint angles, and the last 4 elements define the speed of the transition, and the delay condition after each transition:

1. The first number represents speed factor. The default speed factor is 4, it can be changed to an integer from 1 (slow) to 127 (fast). The unit is in **degrees per step**. If it's set to 0, the servo will rotate to the target angle by its maximal speed (about 0.07sec/60 degrees). It's not recommended to use a value larger than 10 unless you understand the risks. Here for this example, in the first frame, it is default value (4).
2. The 2nd number represents delay time. The default delay is 0. It can be set from 0 to 127, the unit is **50 ms**. Here for this example, in the first frame, it is 1.
3. The 3rd number represents the trigger axis. If it's not 0, the previous delay time will be ignored. The trigger of the next frame will depend on the body angle on the corresponding axis. 1 for the pitch axis, and 2 for the roll axis. The sign of the number defines the direction of the threshold, i.e. if the current angle is smaller or larger than the trigger angle. Here for this example, in the first frame, it is 0.
4. The 4th number represents the trigger angle. It can be -128 to 127 degrees. Here for this example, in the first frame, it is 0.

Understand the memory structure

Explanation the locations:

There are two kinds of skills: **Instincts** and **Newbility**. The addresses of both are written to the onboard EEPROM(1KB) as a lookup table, but the actual data is stored at different memory locations:

- I2C EEPROM (8KB) stores **Instincts**.

The Instincts are already fine-tuned/fixed skills. You can compare them to "muscle memory". Multiple Instincts are linearly written to the I2C EEPROM only once with **WriteInstinct.ino**. Their addresses are generated and saved to the lookup table in onboard EEPROM during the runtime of **WriteInstinct.ino**.

- Flash (sharing the 32KB flash with the program) stores **Newbility**.

A Newbility is any new experimental skill that requires a lot of tests. It's not written to the I2C nor onboard EEPROM, but the flash memory in the format of PROGMEM. It has to be uploaded as one part of the

Arduino sketch. Its address is also assigned during the runtime of the code, though the value rarely changes if the total number of skills (including all Instincts and Newbilities) is unchanged.

Explanation the implementation code :

```
1 #if !defined(MAIN_SKETCH) || !defined(I2C_EEPROM)
2   //if it's not the main sketch to save data or there's no external EEPROM,
3   //the list should always contain all information.
4   const char* skillNameWithType[]={"bdFI","bkI","bkLI","bkRI","crFI","crLI","crRI","trFI",
5   const char* progmemPointer[] = {bdF, bk, bkL, bkR, crF, crL, crR, trF, trL, trR, vt, wkF
6 #else   //only need to know the pointers to newbilities, because the intuitions have been
7         //while the newbilities on progmem are assigned to new addresses
8   const char* progmemPointer[] = {zero, };
9 #endif
```

The first section is active when uploading WriteInstinct.ino. It contains all the skills' data and pointers. The skill names contain a suffix, "N" or "I" to indicate whether it's a Newbility or Instinct. The Instinct will be saved to the external I2C EEPROM while the Newbility will be saved to the flash. The address of all the skills will be saved to the onboard EEPROM.

The second section is active when uploading OpenCat.ino. Because the Instincts are already saved in the external EEPROM, their data is omitted to save space. The Newbility will be saved to the flash with the upload so you can update them directly with OpenCat.ino. It's very useful if you are still tuning a new skill.

In the example code, only zero is defined as a Newbility so you don't need to re-upload WriteInstinct.ino for experiments.

Create a new behavior

What if we want to add more skills with customized names and button assignments?

	Type	Command	Note	备注
Control	Token	d	rest and shutdown all servos	休息并关闭所有舵机
		g	turn on/off gyro to boost speed	开关陀螺仪加速运动
		p	pause motion and shut off all servos	暂停动作并关闭所有舵机
		c	enter calibration mode	进入校准模式
		m	move a joint to certain angle	把某关节转动到某角度
Skill	Gait	bk	back	后退
		bkL	backLeft	后退 左
		bkR	backRight	后退 右
		vt	stepping on the same spot	原地踏步
		crF	crawl	爬 低重心对角步态
		crL	crawl left	爬 左
		crR	craw right	爬 右
		wkF	walk	走 三脚着地的步态
		wkL	walk left	走 左
		wkR	walk right	走 右
		trF	trot	小跑 对角步态
		trL	trot left	小跑 左
		trR	trot right	小跑 右
		bdF	bound (not recommanded)	兔子跳（不推荐）
		balance		正常站立演示自平衡性
		buttUp	buttom up	撅屁股

Posture	callb	callbration	校准姿态	
	rest		休息	
	sit		坐	
	sleep		睡	
	str	stretch	伸懒腰	
	zero		零姿势 给用户自己设计动作的模村	
Behavior	ck	check around	左右看	
	hi	hi sequence	打招呼	
	pee	pee sequence	撒尿	
	pu	push up sequence	俯卧撑	
	rc	recovering sequence	四脚朝天后恢复站立（自动激活）	
	pd	play dead	主动翻身倒地装死	
	bf	back flip	后空翻（隐藏）	

ou need to add a 'k' before skills. For example, kbk, kbalance, kck...
e Gaits can be the result of combined command: gait+direction

		Gait	Direction	Serial Command
		cr	F	kcrF
		wk	L	kwkL
		tr	R	ktrR

Serial commands
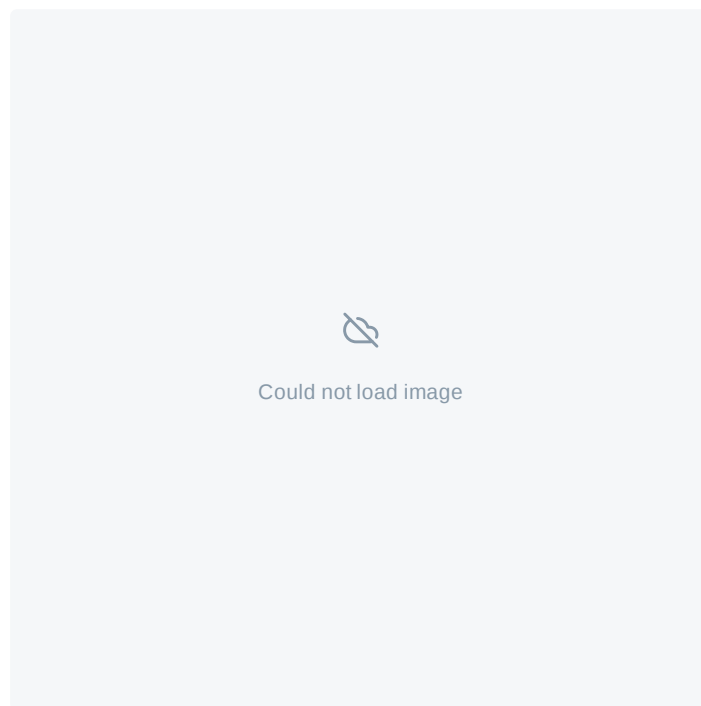
Keymap on the IR remote

If you want to add more skills, you can refer to the implementation of serial commands (the table above) and keymap (the figure above) in the program code.

Add an example of a **Newbility test** and assign it to a button on the IR remote. Upload the skill and call it with both IR remote and serial monitor.

```
1 #if !defined(MAIN_SKETCH) || !defined(I2C_EEPROM)
2 //if it's not the main sketch to save data or there's no external EEPROM,
3 //the list should always contain all information.
4 const char* skillNameWithType[]={"bdFI","bkI","bkLI","bkRI","crFI","crLI","crRI","trFI","t
```

```
  6 #elsechy//onlymneedmtovknowithe[pbifers,tokewbilities,cbecauserthecinfuitionsrhaverbeenFsav
  7 //while the newbilities on progmem are assigned to new addresses
  8 const char* progmemPointer[] = {zero, test};
  9 #endif
```

Modify a few fields of **test** to make it an **Instinct**. Call the behavior with the IR remote or serial monitor.

```
  1 #if !defined(MAIN_SKETCH) || !defined(I2C_EEPROM)
  2 //if it's not the main sketch to save data or there's no external EEPROM,
  3 //the list should always contain all information.
  4 const char* skillNameWithType[]={"bdFI","bkI","bkLI","bkRI","crFI","crLI","crRI","trFI","t
  5 const char* progmemPointer[] = {bdF, bk, bkL, bkR, crF, crL, crR, trF, trL, trR, vt, wkF, v
  6 #else //only need to know the pointers to newbilities, because the intuitions have been sav
  7 //while the newbilities on progmem are assigned to new addresses
  8 const char* progmemPointer[] = {zero, };
  9 #endif
```

```
#if !defined(MAIN_SKETCH) || !defined(I2C_EEPROM)          #if !defined(MAIN_SKETCH) || !defined(I2C_EEPROM)
//if it's not the main sketch to save data or there's no ext   //if it's not the main sketch to save data or there's no ext
#if !defined(MAIN_SKETCH) || !defined(I2C_EEPROM)          #if !defined(MAIN_SKETCH) || !defined(I2C_EEPROM)
//if it's not the main sketch to save data or there's no ext   //if it's not the main sketch to save data or there's no ext
ernal EEPROM,                                              ernal EEPROM,
//the list should always contain all information.          //the list should always contain all information.
const char* skillNameWithType[]={"bdFI","bkI","bkLI","bkRI",  const char* skillNameWithType[]={"bdFI","bkI","bkLI","bkRI",
"crFI","crLI","crRI","trFI","trLI","trRI","vtI","wkFI","wkLI   "crFI","crLI","crRI","trFI","trLI","trRI","vtI","wkFI","wkLI
","wkRI","balanceI","buttUpI","calibI","droppedI","liftedI",  ","wkRI","balanceI","buttUpI","calibI","droppedI","liftedI",
"restI","sitI","strI","zeroN","bfI","ckI","hiI","pdI","peeI"  "restI","sitI","strI","zeroN","bfI","ckI","hiI","pdI","peeI"
,"puI","rcI","stpI","testN"};                              ,"puI","rcI","stpI","testI"};
const char* progmemPointer[] = {bdF, bk, bkL, bkR, crF, crL,  const char* progmemPointer[] = {bdF, bk, bkL, bkR, crF, crL,
 crR, trF, trL, trR, vt, wkF, wkL, wkR, balance, buttUp, cal   crR, trF, trL, trR, vt, wkF, wkL, wkR, balance, buttUp, cal
ib, dropped, lifted, rest, sit, str, zero, bf, ck, hi, pd, p  ib, dropped, lifted, rest, sit, str, zero, bf, ck, hi, pd, p
ee, pu, rc, stp, test};                                    ee, pu, rc, stp, test};
#else //only need to know the pointers to newbilities, becau  #else //only need to know the pointers to newbilities, becau
se the intuitions have been saved onto external EEPROM,    se the intuitions have been saved onto external EEPROM,
//while the newbilities on progmem are assigned to new addre  //while the newbilities on progmem are assigned to new addre
sses                                                       sses
const char* progmemPointer[] = {zero, test};               const char* progmemPointer[] = {zero, };
#endif                                                     #endif
```

Remember to add 1 to the number of skills at the beginning of the header file.

Then you can call this test by entering ktest in the serial monitor. You can also call it from the IR remote if you replace a button definition in OpenCat.h.

Tune skills in realtime

You need to understand the above structure to store a skill on the robot. However, when tuning the skills, things can be easier. To do this, you can connect your computer with the robot through the USB or Bluetooth connector. Then you can send string commands that define the joint angles. The program on the NyBoard will listen and perform the instructions in real-time. In the SerialMaster folder, you can find the ArdSerial.py to play the role of Arduino IDE's serial monitor, but with the ability to write more logic and controls within the script.
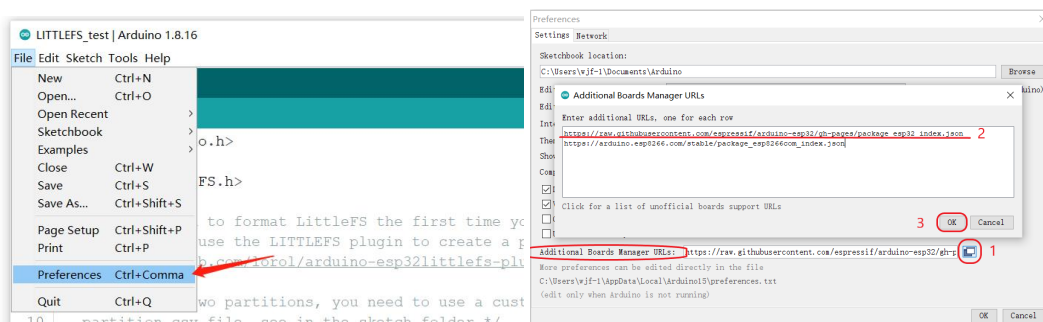
12. BiBoard Configuration

12.1. Read the user manual

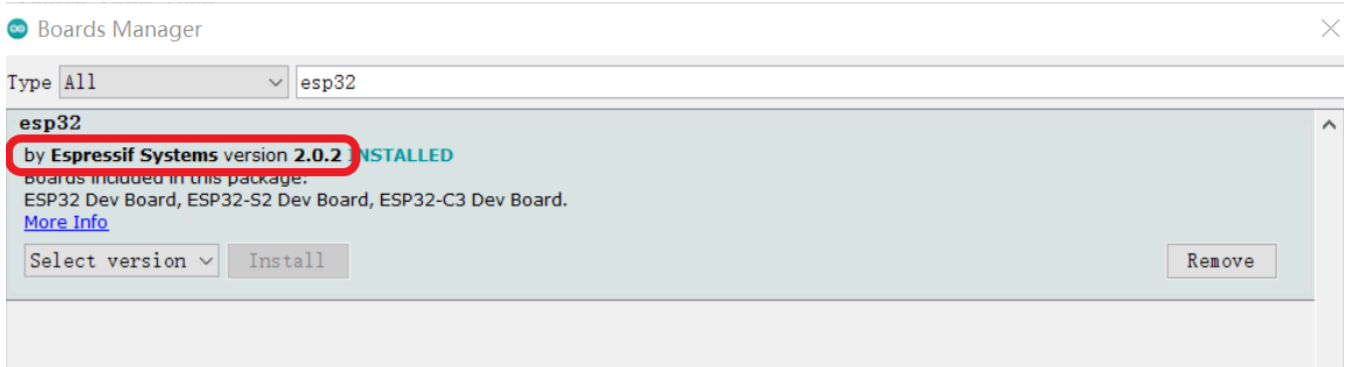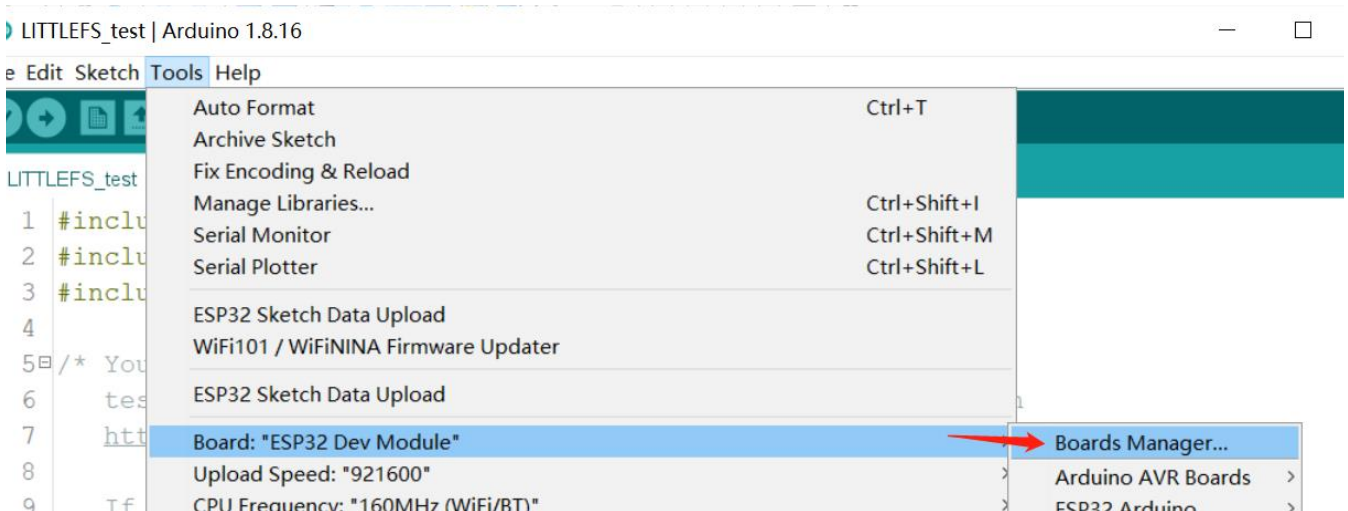Read the user manual for BiBoard Quick Start Guide.
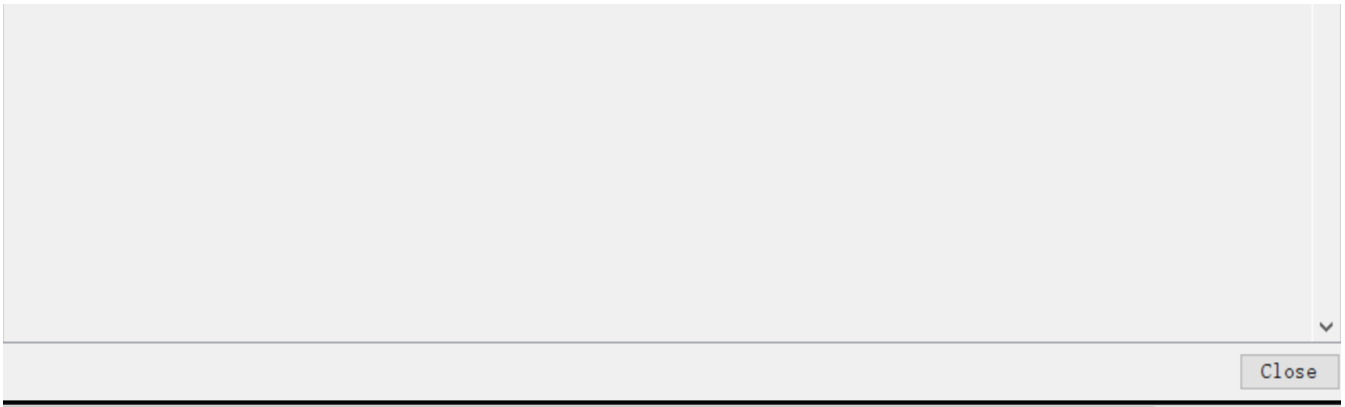
12.2. Set up BiBoard

12.2.1. Add URL in Arduino->Preferences->Additional Boards Manager URLs

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.js…

12.2.2. In Boards Manager, search ESP32 and install the 2.0.* version

Close

12.2.3. Modify code files

Download the BiBoard package.

- In file (for Windows):
  C:\Users{username}\AppData\Local\Arduino15\packages\esp32\hardware\esp32\2.0.2\tools\sdk\esp32\include\config**sdkconfig.h** append
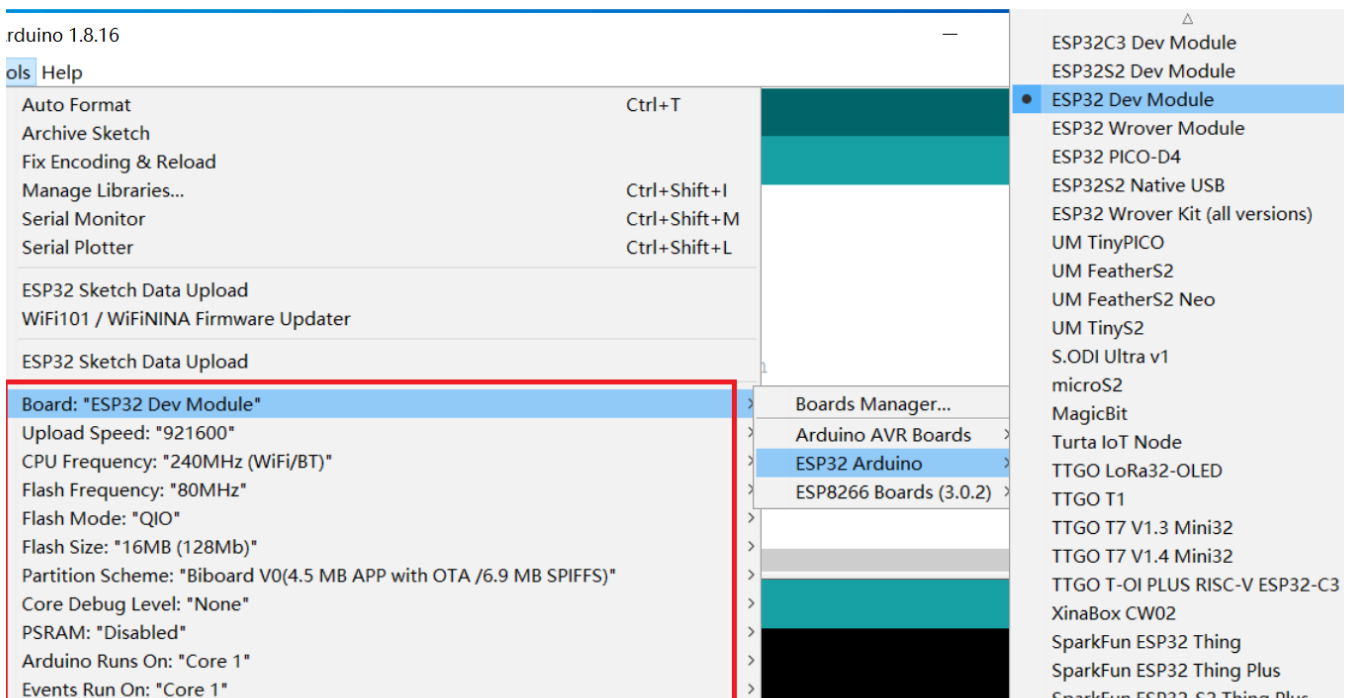
```
1 #define CONFIG_DISABLE_HAL_LOCKS 1
```
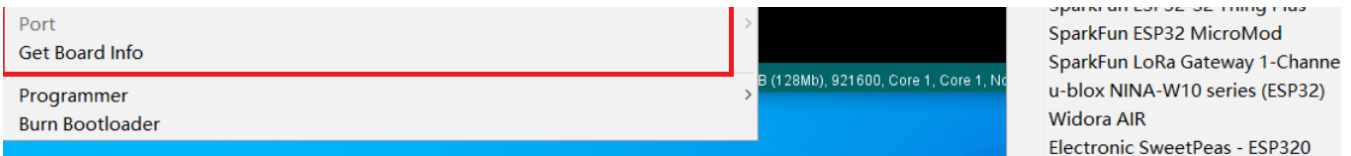
  to the end. or replace it with the file in this folder (BiBoard\ESP32config\sdkconfig.h).

- Replace (for Windows)
  C:\Users{username}\AppData\Local\Arduino15\packages\esp32\hardware\esp32\2.0.2\cores\esp32\esp32-hal-i2c-slave.c with the file in this folder (BiBoard\ESP32config\esp32-hal-i2c-slave.c).

12.2.4. Add hardware partition

Read the user manual for Add hardware partition configuration option in Arduino IDE

Port
Get Board Info

Programmer
Burn Bootloader

B (128Mb), 921600, Core 1, Core 1, No

SparkFun ESP32-S2 Thing Plus
SparkFun ESP32 MicroMod
SparkFun LoRa Gateway 1-Channe
u-blox NINA-W10 series (ESP32)
Widora AIR
Electronic SweetPeas - ESP320

### 12.2.5. Compile and upload the sketch

Modify the device type macro definition in **BiBoard.ino** according to the device type.

```
1  #define BITTLE    //Petoi 9 DOF robot dog: 1 on head + 8 on leg
2  //#define NYBBLE  //Petoi 11 DOF robot cat: 2 on head + 1 on tail + 8 on leg
3  //#define CUB
```

Modify the motherboard model macro definition in **BiBoard.ino** according to the motherboard model.

```
1  #define BiBoard     //ESP32 Board with 12 channels of built-in PWM for joints
2  //#define BiBoard2  //ESP32 Board with 16 channels of PCA9685 PWM for joints
```

After uploading the sketch (BiBoard\BiBoard.ino), you need to open the serial monitor and follow the prompts for the first time. Then you can calibrate the joints and imu. The joint indexes are configured in the same order as NyBoard, though the BiBoard only has 12 pins. You can find more features in the ChangeLog.md (BiBoard\ChangeLog.md).
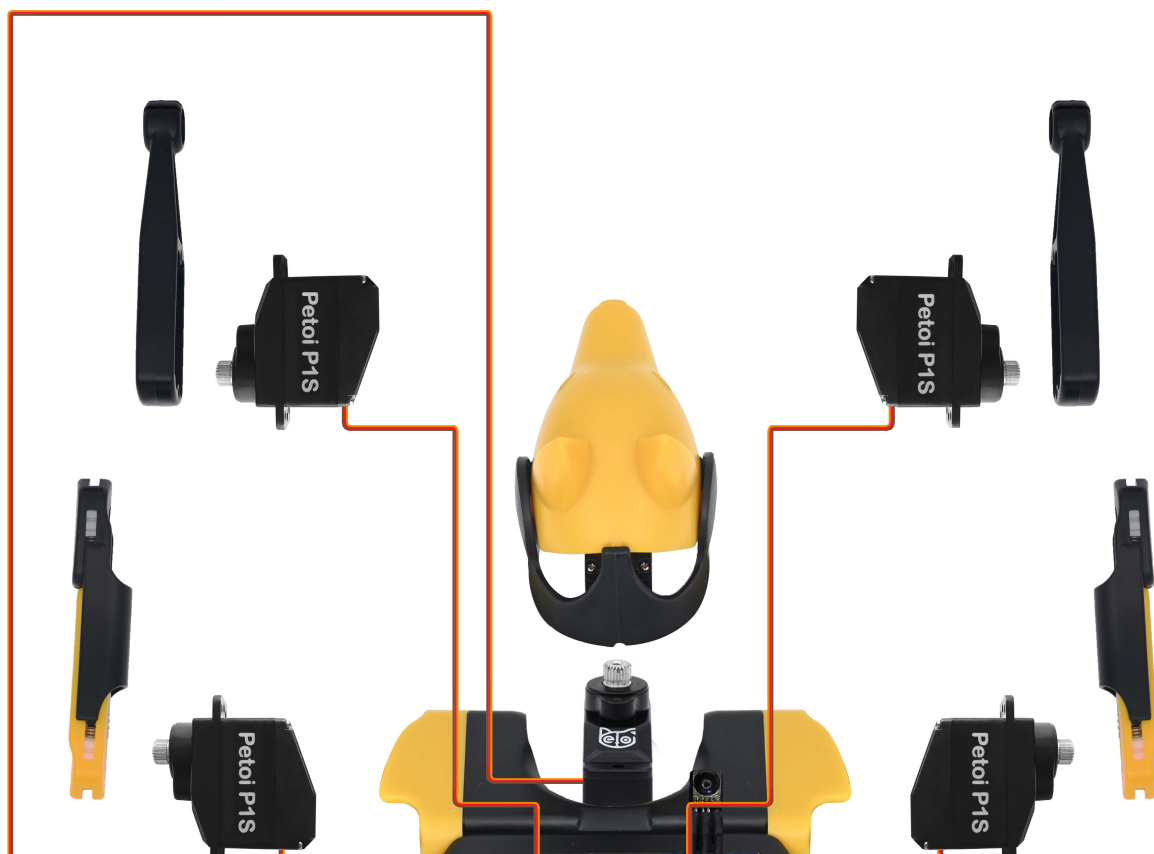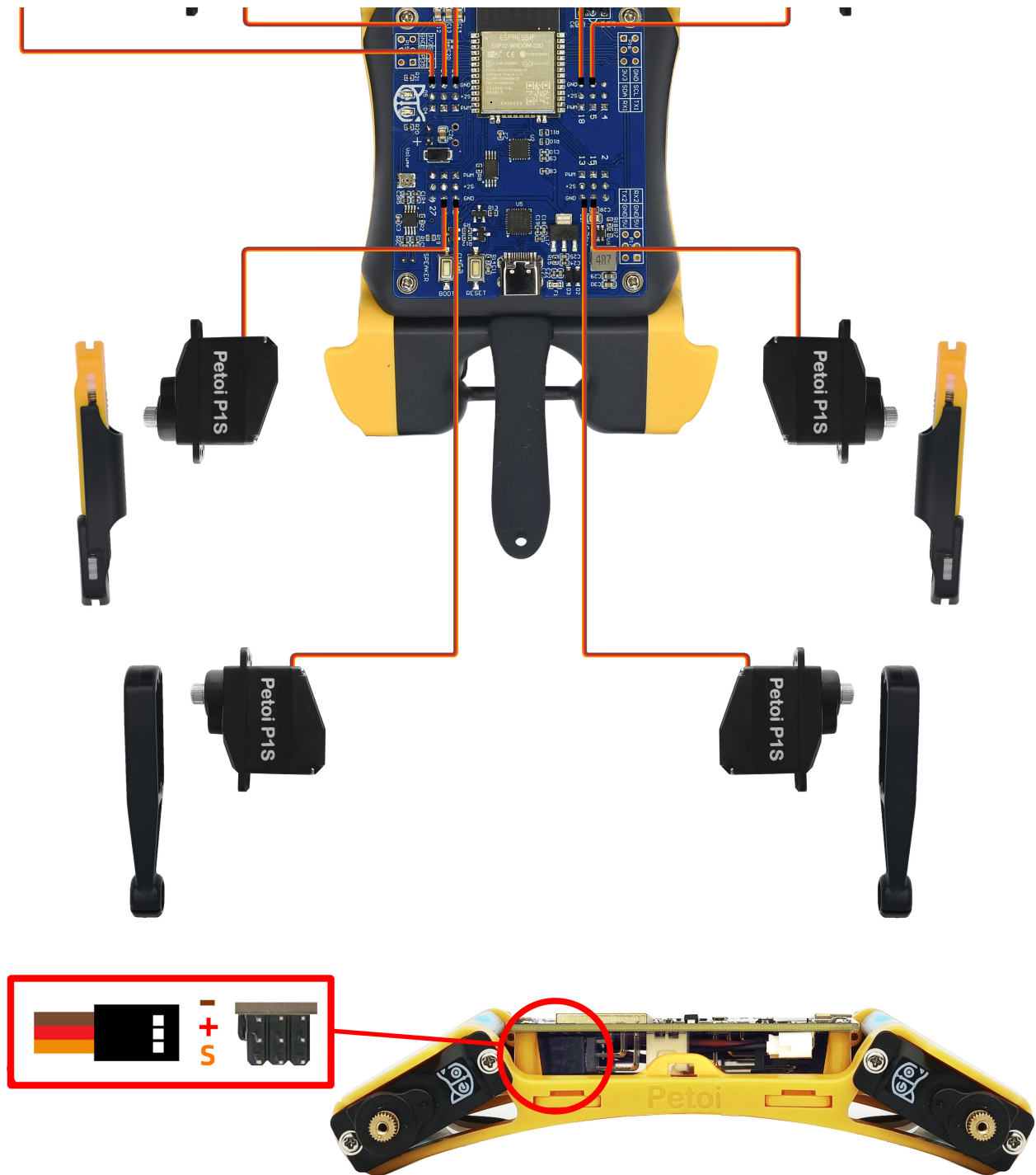
The BiBoard has built-in Bluetooth and you can connect it with the new Android app:

App for BiBoard (Android 4.4+).

12.2.6. Initialization

When the newly assembled Bittle is powered on, please connect the serial port to the computer, open the serial monitor window, and complete the program initialization process:

> ⚠ Make sure to set the serial monitor as **115200 baud rate and no line ending**.

Send

Autoscroll Show timestamp   No line ending ⬍   115200 baud ⬍   Clear output

You will see several questions:

```
1 Reset the joints' calibration offsets? (Y/n):
```

Type 'Y' to the question, which means to calibrate the servos with a fresh start (all offsets are set to zeros).

```
1 - Calibrate the Inertial Measurement Unit (IMU)? (Y/n):
```

Type 'Y' to the question, which means to calibrate the MPU6050, i.e. the gyro/accelerometer sensor.

> ⓘ Put the robot **FLAT** on the table and don't touch it during calibration.
>
> Sometimes the program could halt at the connection stage. You can close the serial monitor and reopen it, or press the reset button on BiBoard, to restart the program.

The details of serial port printing information are as follows :

```
 1 * Start *
 2 Scanning I2C network...
 3 - I2C device found at address 0x54  !
 4 - I2C device found at address 0x68  !
 5 - done
 6 Set up the new board...
 7 // The device name to use when connecting with bluetooth
 8 - Name the new robot as: BittleED
 9 Reset the joints' calibration offsets? (Y/n):
10 Y
11
12 Initializing MPU...
13 - Testing MPU connections...attempt 0
14 - MPU6050 connection successful
15 - Initializing DMP...
16 - Calibrate the Inertial Measurement Unit (IMU)? (Y/n):
```

```
17 Y
18
19 Put the robot FLAT on the table and don't touch it during calibration.
20 - Calibrating the Inertial Measurement Unit (IMU)...
21 >..........>..........
22 MPU offsets:
23 //          X Accel  Y Accel  Z Accel   X Gyro   Y Gyro   Z Gyro
24 //OFFSETS    3752,    -968,     942,     170,      76,       21
25 - Enabling DMP...
26 - Enabling interrupt detection (Arduino external interrupt 26)...
27 - DMP ready! Waiting for the first interrupt...
28 Bluetooth name: BittleED
29 Waiting for a client connection to notify...
30 Setup ESP32 PWM servo driver...
31 Ready!
32
```

All of Bittle's servos will rotate to the calibration states (zero angles) and stop moving. Then the serial monitor window will print out "Ready!"

Then you can attach body parts to the servos according to the calibrated position and use the 'c' commands to set the fine-calibration values.

Please refer to chapter 6 □ Calibration and Final Assembly for the complete calibration process.

> ⓘ If you need to recalibrate the servo offsets or recalibrate the IMU (MPU6050), you can modify the following configuration macro definitions in the **BiBoard.ino** then recompile and upload the sketch.
>
> ```
> 1 //change this mark to any other character if you want to reset
> 2 //the Bluetooth name and calibration values in the EEPROM
> 3 #define BIRTHMARK 'x'
> ```

🛠️ Supporting Application and Software

Programming Language Support

- Coding can be done in C-style language with the Arduino IDE.
- Python API for sending serial commands
- Bittle can be programmed with a Scratch-like web-based IDE Codecraft and curriculum

---

3rd-Party Code & Robots - Open-source Robot Controller App -

[The Code & Robots iOS app](#) can control a few open source Robots, include Bittle.

Please check out the following video for the app configuration to support controlling Bittle.

> 📎 code & robots configuration.mp4   4MB
> Binary

Code & Robots app configuration for supporting Bittle

3rd-party Open-source Projects

- [Inverse Kinematic Model OpenCat](#)
  - [a discussion thread](#)
- [OpenCatWeb](#) - a web interface to control Opencat-based robots
  - need to mount a Raspberry Pi
  - [a discussion thread](#) by the author leukipp

FAQ(Frequently Asked Questions)

Do you have any curriculum?

Our partner TinkerGen provides a [curriculum](#) which can be used with its a Scratch-like web-based IDE [Codecraft](#).

What do different sounds from the board mean?

How can I easily install the springs into the upper legs of Bittle?

Please check out [the forum post](#) discussing installing springs with varies tools.

Useful links