

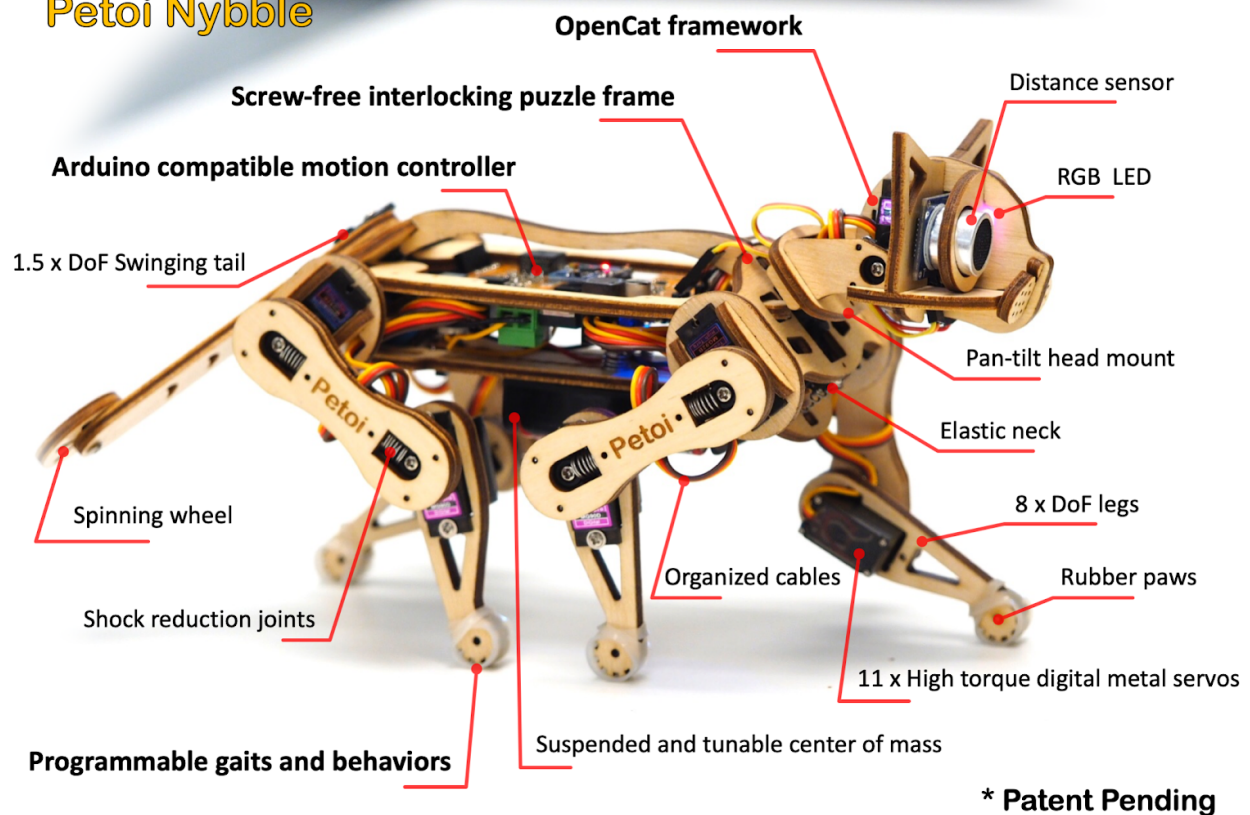
Assembling Instructions for Nybble

Rongzhong Li

Jan. 25, 2019

To keep this instruction simple to use, I'm focusing on the assembly rather than in-depth explanation. If you have specific questions on "why" rather than "how", please post on our forum at <https://www.petoi.com/forum>.

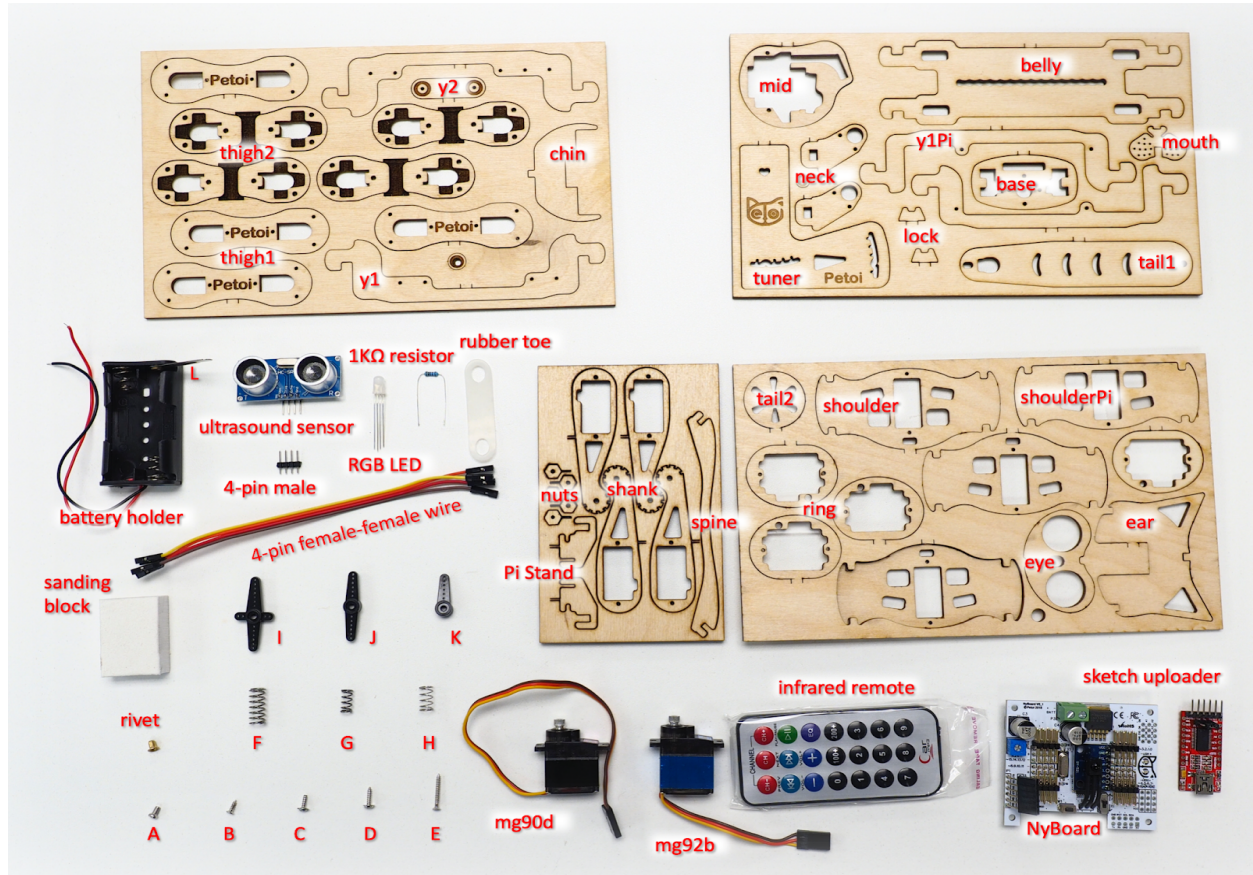
Petoi Nybble



*** Patent Pending**

The crowdfunding campaign is active on Indiegogo: igg.me/at/nybble

1. Open box. Get familiar with the components.



There's a larger picture in the Appendix.

This instruction will keep consistent with the current naming method.

1.1. Cut body pieces off the baseboard.

There might be some tar residue on the wooden pieces from laser cutting. Use a wet soft tissue to clean up the board.

The functional pieces are attached to the baseboard by lightly cut tabs. Though you could pop those pieces out by hand, it's highly recommended that you use a knife to cut on the back side of the tabs to avoid potential damage to the middle layer, where the fiber direction is perpendicular to the surface fiber.

After taking out all the pieces from the baseboard, you are encouraged to bend and break the remaining structures on the baseboard, to understand the mechanical

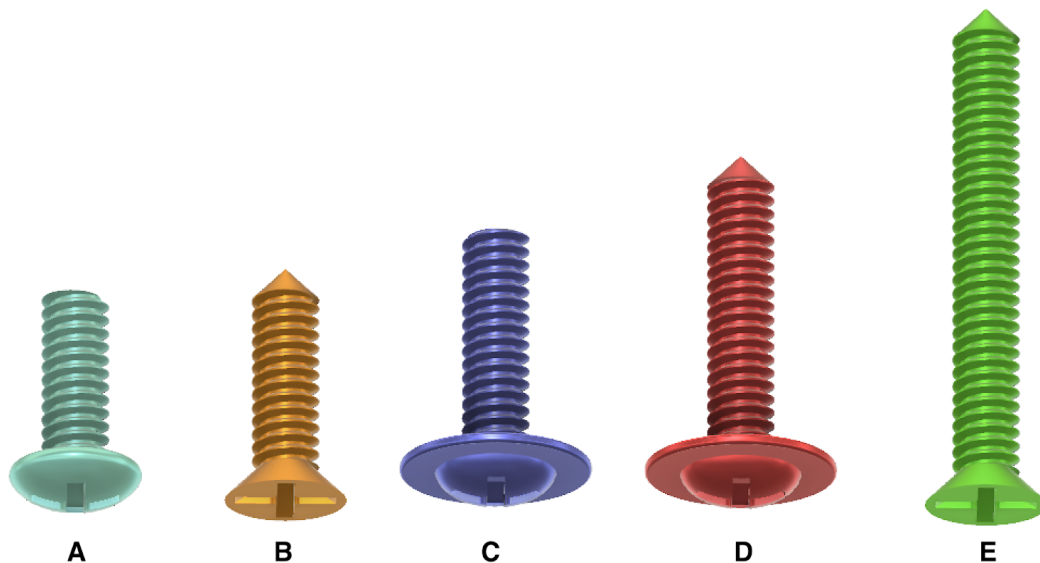
properties of plywood, such as anisotropic strength, elasticity, etc. That will give you confidence in later handling.

1.2. Remove pointy fibers.

Use the sanding foam to clean up any thorn on the pieces. Don't sand too much or it may affect the tightness between joints.

1.3. Screws

There are five different screws used in the kit. I'm coloring them differently to better indicate their locations. Not all screws are required to assemble Nybble. Some are spare for replacement.



- A is for attaching servo arms. D (sharp tip) is for attaching servos to the frame. A and D come in each servo's own accessory bags.
- B is for attaching servo arms/circuit boards to the frame. In later versions it may be replaced by C to avoid confusion.
- C (flat tip) is for binding the thighs.
- E (always the longest) is for attaching the battery holder.

1.4. Springs

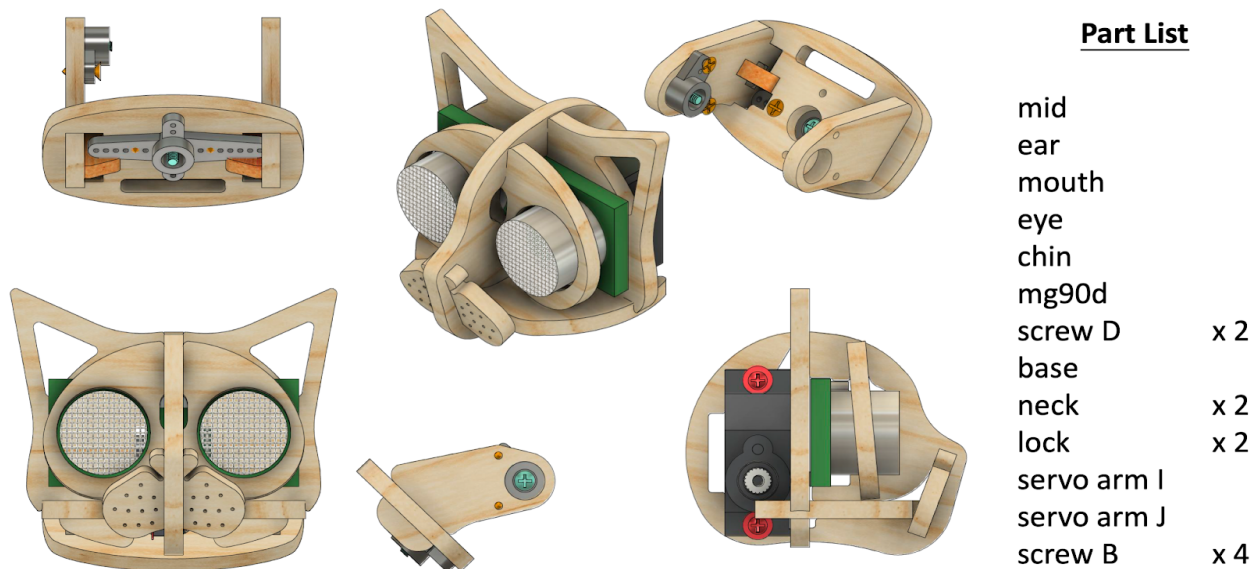
There are three different springs: F, G, H.

- The big spring F is used for elastic connection in the thigh. There's one spare unit;
- The hard short spring G is for the neck. It could be replaced by spring F;
- The soft short spring H is for attaching the battery holder.

2. Assemble the frame

2.1. Head and neck

2.1.1. Part list

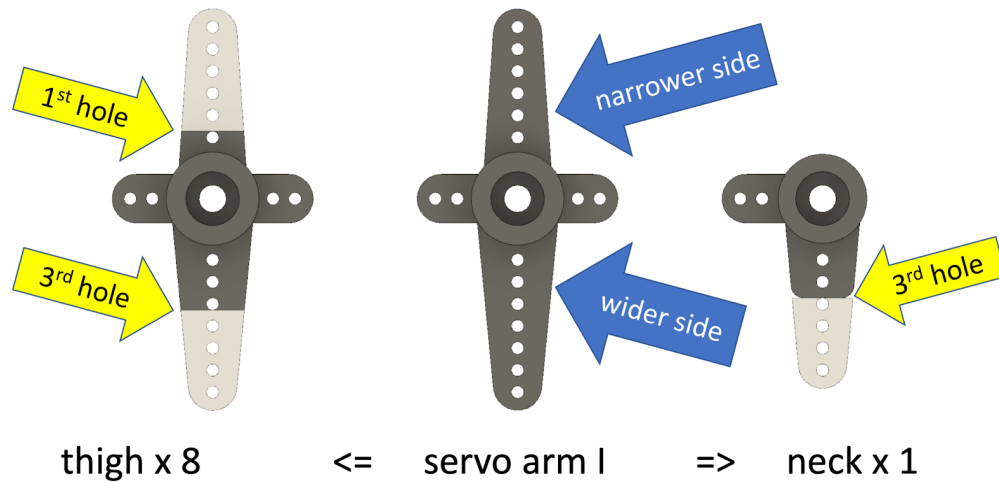


2.1.2. Solder on the optional LED to ultrasound sensor.

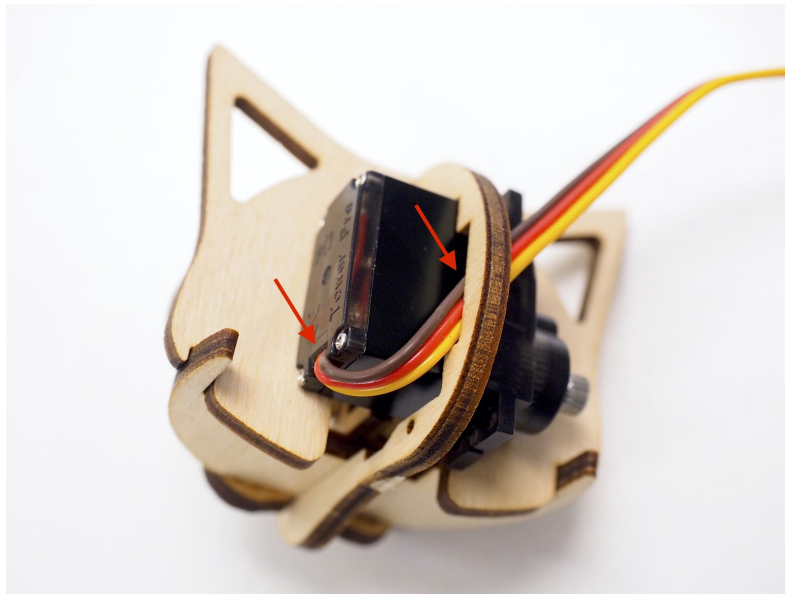
The obstacle avoidance algorithm using the ultrasound sensor has not yet been integrated in the code. The optional RGB LED can be soldered to the four pins of the ultrasound sensor ([instructions](#)) to indicate its working status, or can be programmed as decorative lights.

2.1.3. Trim the servo arms for attaching servos.

Pay attention to the width difference between servo arm I's two long sides, as well as the trimming location (using screw holes as references).



- 2.1.4. Assemble the head group as shown in the [head animation](#). Note that the base should only be partially assembled before the later calibration. Otherwise it will be difficult to insert the servo between neck pieces. Also notice how the servo wire is organized in the head.

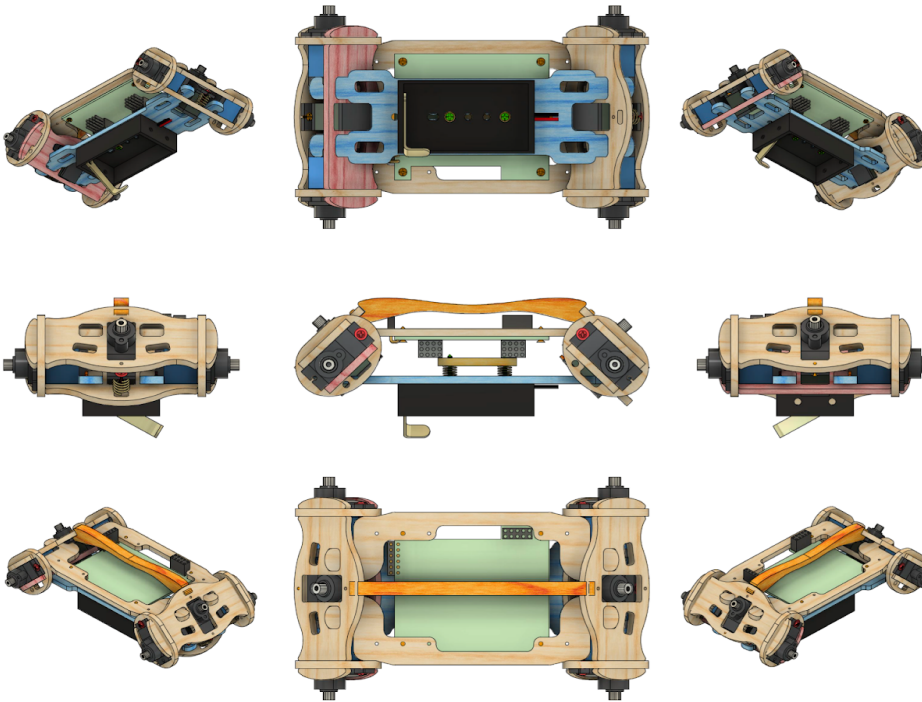


- 2.1.5. DO NOT connect the head with neck yet, because the tilt servo on the head has to be calibrated.

2.2. Body

2.2.1. Part list

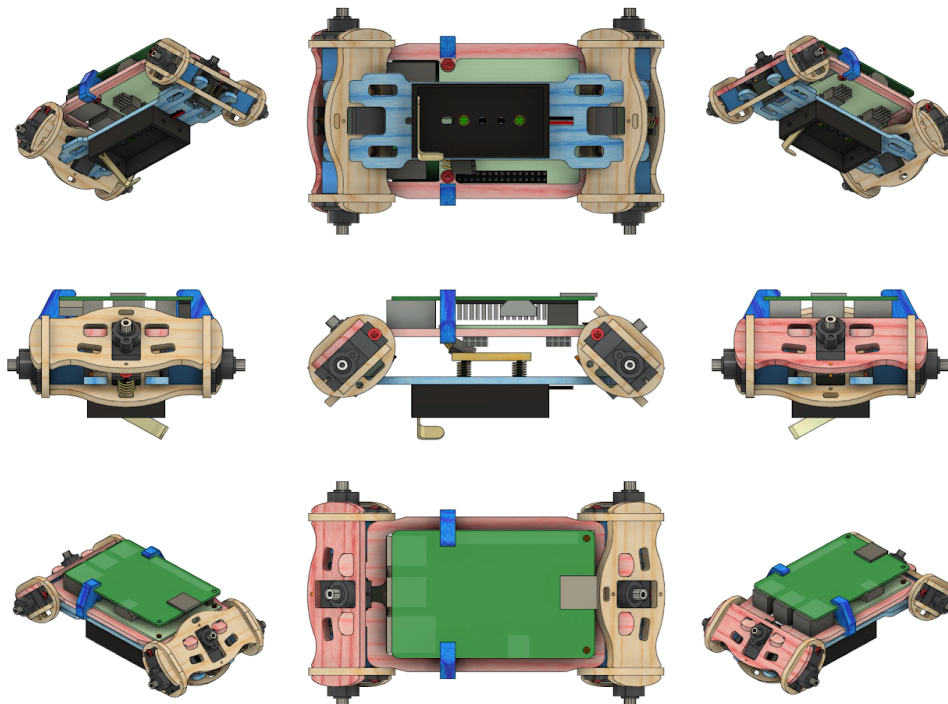
- NyBoard only



Part List

belly	
ring	x 4
mg92b	x 4
mg90d	x 2
shoulder	x 3
shoulderPi	
y1	x 2
y2	
battery holder	
rivet	x 2
spring G	
spring H	x 2
screw D	x 4
screw E	x 2

- NyBoard with Raspberry Pi: use y1Pi to replace y1, and add Pi Stand.



Part List

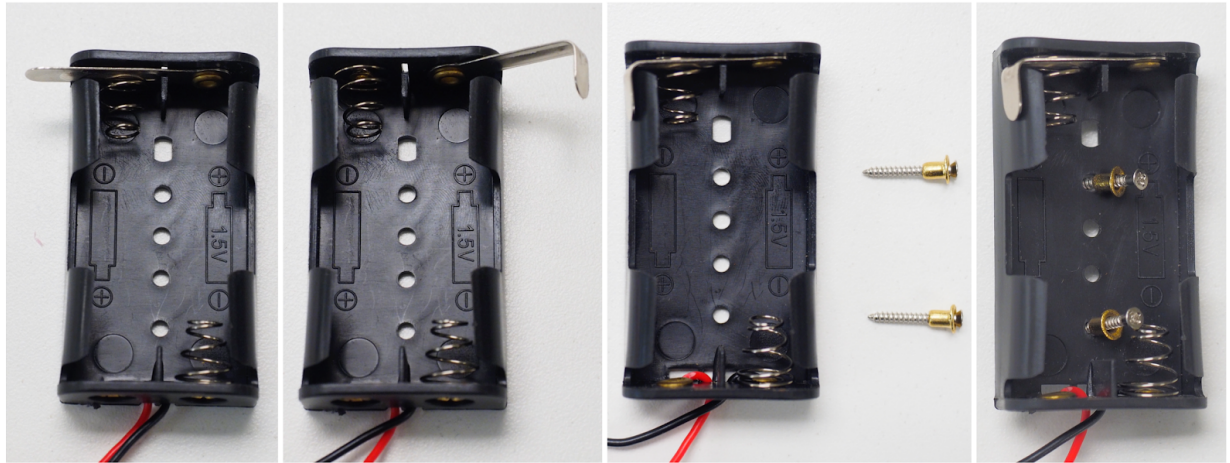
belly	
ring	x 4
mg92b	x 4
mg90d	x 2
shoulder	x 3
shoulderPi	
y1Pi	x 2
Pi Stand	x 2
y2	
battery holder	
rivet	x 2
spring G	
spring H	x 2
screw D	x 4
screw E	x 2

- Other controllers

I also included 5 x 1"/4 nuts for mounting other circuit boards.

2.2.2. Install the adjustable battery holder to belly

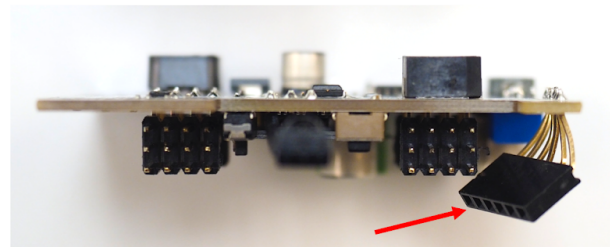
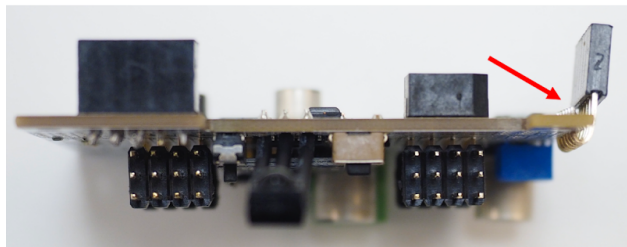
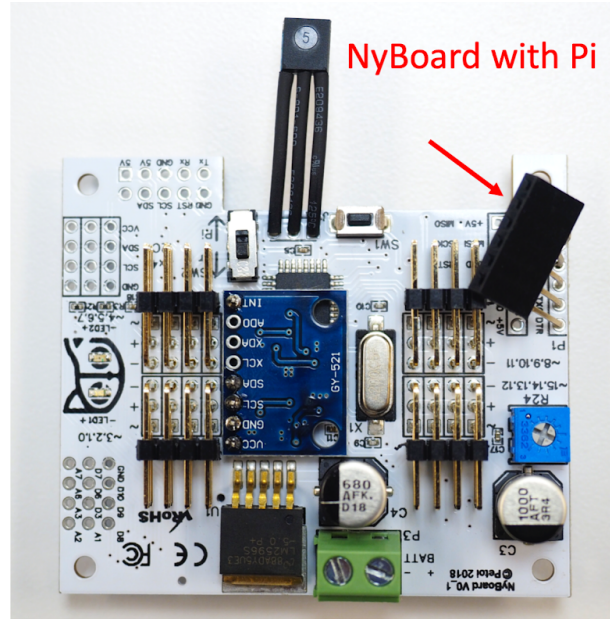
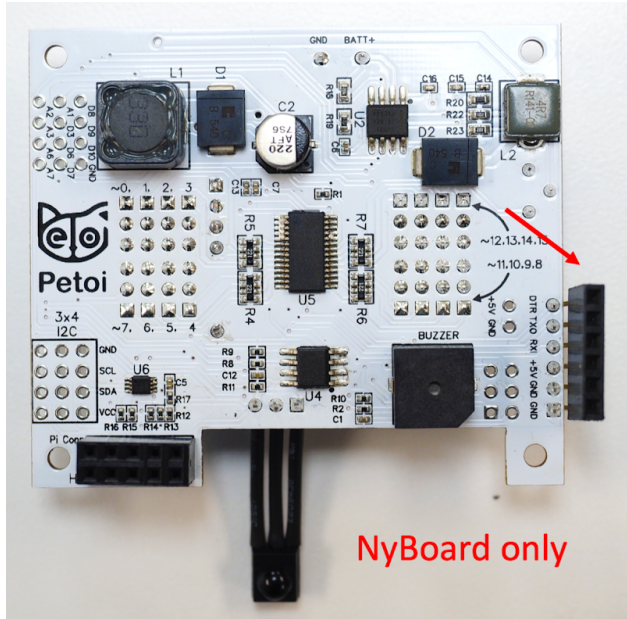
Bend the hinge L of battery holder to 90 degree, close to the wall. It functions as a switch. Insert the long screw E through the rivet so that you can better handle the rivet. Insert and push the rivet into the hole on the bottom of the battery holder. Pay attention to the holes' locations.



The spring attached structure of the battery holder is used for shifting the center of mass when fine tuning gaits.

2.2.3. Assemble the body group as shown in the [body animation](#).

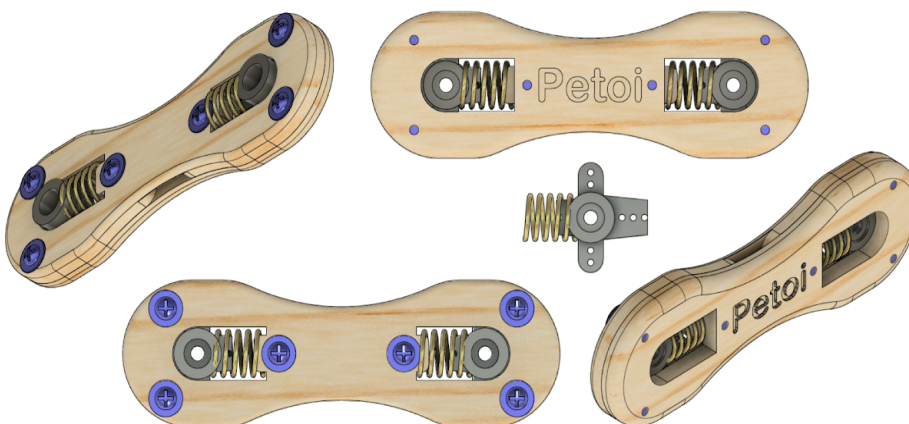
Pay attention to the long pins of infrared receiver and FTDI port. They are designed to be bent to favorable directions. Don't bend the pins too often or it will lead to metal fatigue.



2.2.4. Observe the [adjusted configuration](#) if you want to mount a Raspberry Pi.

2.3. Thigh

2.3.1. Part list



Part List

thigh1	x 4
thigh2	x 4
servo arm I	x 8
spring F	x 8
screw C	x 24

2.3.2. Trim the servo arms for attaching servos.

The location has been shown in the [Head and Neck](#) section. The trimmed narrower servo arm is designed to be inserted into spring F.

2.3.3. Assemble the thigh as shown in the [thigh animation](#).

Before closing thigh1 and thigh2, put the wire of the shank through the slot in the middle of the thigh. Think about symmetry of the four legs.

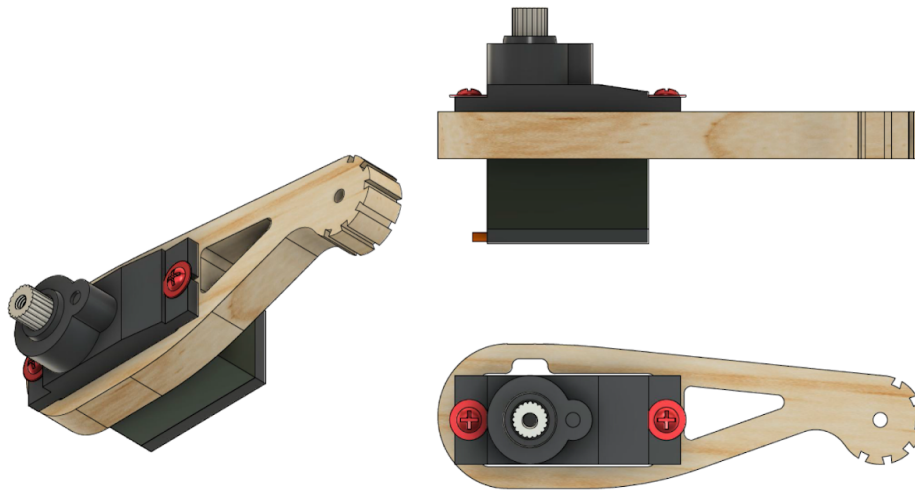


The servo arm should be able to slide in the track on thigh2 with subtle friction when thigh1 and thigh2 are screwed together. You can tune the tightness of screw C to achieve proper friction. If you need more control on the tightness:

- Scratch the track using a flat screw driver to reduce friction.
- Apply a little paper glue and let it dry to increase friction.

2.4. Shank

2.4.1. Part list



Part List

shank	x 4
rubber toe	x 4
mg90d	x 4
screw D	x 8

2.4.2. Attach the rubber to the tip of the shank.

The serrated structure on the tip of shank is already good for walking. The rubber toe is optional to increase friction and soften each step.



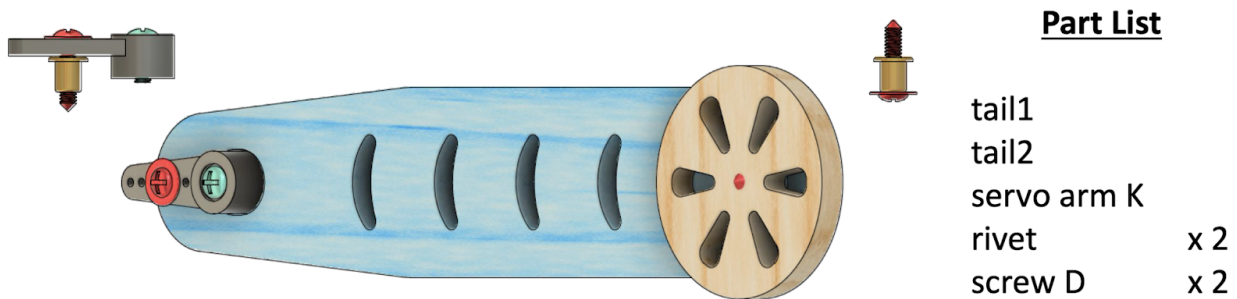
2.4.3. Insert the servo into the slot on the shank.

Pay attention to the direction that the wire is twisted. The small dent on the long edge is designed to let wire go through. Think about symmetry of the four legs.

2.4.4. Assemble the shank as shown in the [shank animation](#). Don't install the servo screw A yet.

2.5. Tail

2.5.1. Part list



2.5.2. Assemble the tail as shown in the [tail animation](#).

The screw D is installed in the third hole counted from the center of servo arm K. Pay attention to the order that every pieces are stacked. The wheel (tail2) should be able to rotate with little friction, and the whole tail should be able to tilt by a small degree.

2.5.3. DO NOT connect the tail to body yet.

2.6. DO NOT screw neck and legs to the body's servos yet

3. Configure Arduino IDE and NyBoard

3.1. NyBoard

3.1.1. Read the user manual for [NyBoard V0](#).

3.1.2. Dial the potentiometer clockwise to start from the lowest voltage.

Higher voltage will increase the servos' torque, making Nybble move faster. The downside is it will increase current draw, reduce battery life, affect the stability of circuit, and increase the wearing of the servos.

3.1.3. Dial the I²C switch (SW2) to Ar.

The I²C switch changes the master of I²C devices (gyro/accelerometer, servo driver, external EEPROM). On default "Ar", NyBoard uses the on-board ATmega328P

as the master chip; On "Pi", NyBoard uses external chips connected through the I²C ports (SDA, SCL) as the master chip.

3.2. Downloads and installations

Note: If you have previously added other libraries and see 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 it and install to your IDE.

3.2.1. Install through 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**, **IRremote** and **QList**.

3.2.2. Install by adding .ZIP library

Go to <https://github.com/jrowberg/i2cdevlib>, download the zip file and unzip. Find MPU6050/ and I2Cdev/ in the Arduino folder. Add their .ZIP library one by one.

3.2.3. Create and add NyBoard

- Locate the files

Mac location:

/Users/UserName/Library/Arduino15/packages/arduino/hardware/
avr/version#/
Or:

/Applications/Arduino.app/Contents/Java/hardware/arduino/avr

To access, right click on Arduino.app and choose Show Package Contents

Windows location:

C:\Program Files(x86)\Arduino\hardware\arduino\avr\

You should find a **boards.txt** is already there.

Linux (Fedora):

boards.txt is symlinked under /etc

- Make a copy of **boards.txt** in case you want to roll back.

- Create new **boards.txt**.

You can download my [boards.txt](#) file, or:

Edit your **boards.txt** with admin privilege. Find the section of

```
pro.name=Arduino Pro or Pro Mini
```

and insert the

```
## Arduino Pro or Pro Mini (5V, 20 MHz) w/ ATmega328P
```

code block. Save and quit your editor.

```
#####
```

```
pro.name=Arduino Pro or Pro Mini
```

```
pro.upload.tool=avrdude
```

```
pro.upload.protocol=arduino
```

```
pro.bootloader.tool=avrdude
```

```
pro.bootloader.unlock_bits=0x3F
```

```
pro.bootloader.lock_bits=0x0F
```

```
pro.build.board=AVR_PRO
```

```
pro.build.core=arduino
```

```
pro.build.variant=eightanaloginputs
```

```
## Arduino Pro or Pro Mini (5V, 20 MHz) w/ ATmega328P
```

```
## -----
```

```
pro.menu.cpu.20MHzatmega328=ATmega328P (5V, 20 MHz) NyBoard
```

```
pro.menu.cpu.20MHzatmega328.upload.maximum_size=30720
```

```
pro.menu.cpu.20MHzatmega328.upload.maximum_data_size=2048
```

```
pro.menu.cpu.20MHzatmega328.upload.speed=57600
```

```
pro.menu.cpu.20MHzatmega328.bootloader.low_fuses=0xFF
```

```
pro.menu.cpu.20MHzatmega328.bootloader.high_fuses=0xDA
```

```
pro.menu.cpu.20MHzatmega328.bootloader.extended_fuses=0xFD
```

```
pro.menu.cpu.20MHzatmega328.bootloader.file=atmega/ATmega328_20MHz.hex
```

```
pro.menu.cpu.20MHzatmega328.build.mcu=atmega328p
```

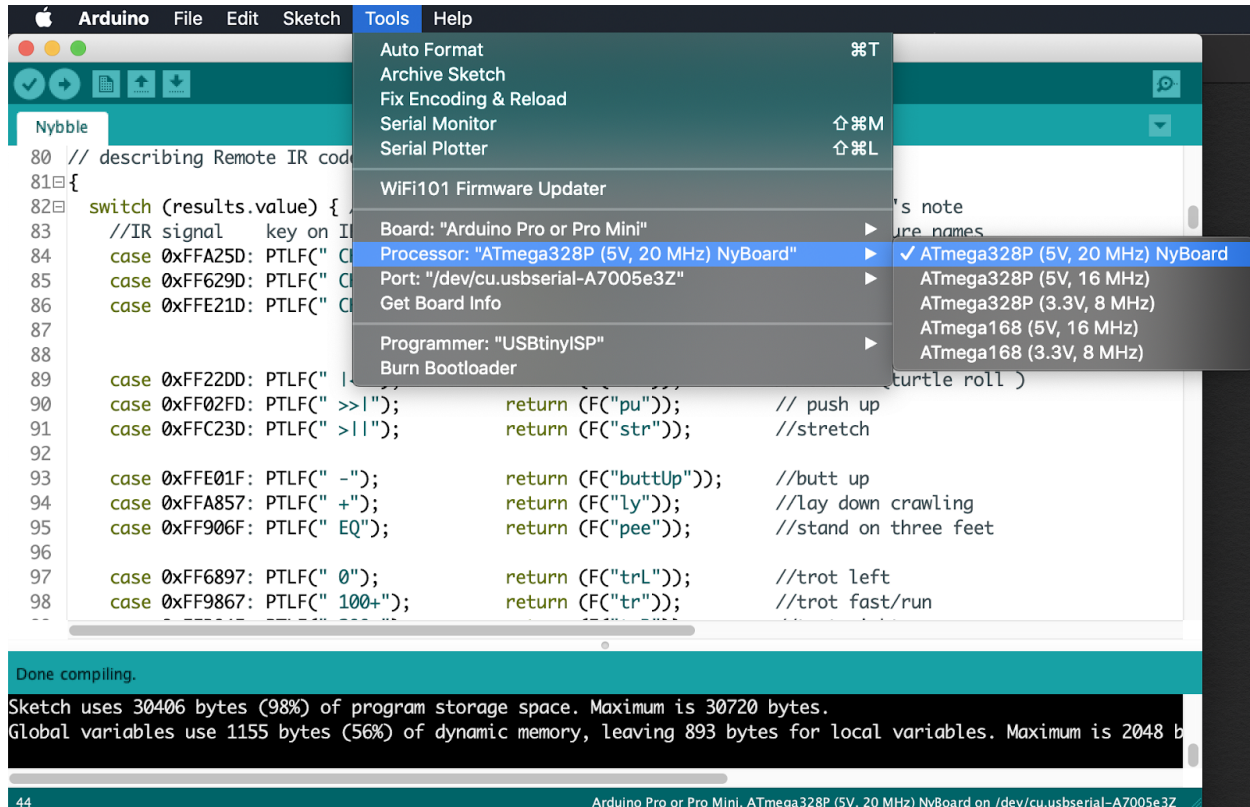
```
pro.menu.cpu.20MHzatmega328.build.f_cpu=2000000L
```

```
## Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328P
```



```
## -----  
...
```

- Download [ATmega328_20MHz.hex](#) and put it in your Arduino folder ./bootloaders/atmega/. You should see other bootloaders with .hex suffix in the same folder.
- Restart your Arduino IDE. In Tools->Boards, select Arduino Pro or Pro Mini. You should find ATmega328P (5V, 20 MHz) in Processor menu.



- Note: If you cannot find the board, your Arduino IDE may be using the boards.txt in another path. Search boards.txt in all the folders on your computer. Find out the right file that's in effect.

3.2.4. Burn the bootloader (only if the bootloader of NyBoard collapsed)

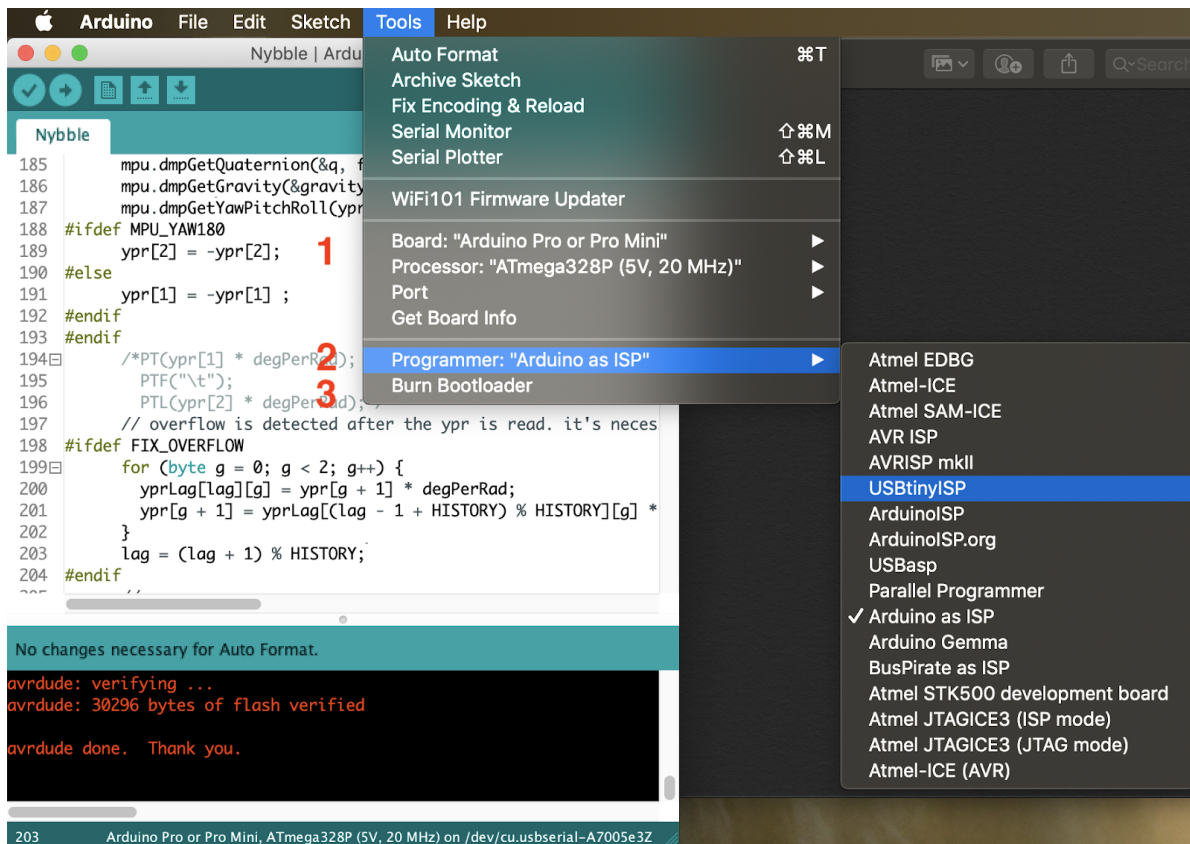
- [What is a bootloader?](#)

Every NyBoard has to go through functionality checks before shipping, so they should already have compatible bootloader installed. However, in rare cases, the bootloader may collapse then you won't be able to upload sketch 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 UART uploader;
- You haven't selected the correct port;
- Bad contacts;
- Bad luck. Tomorrow is another day!

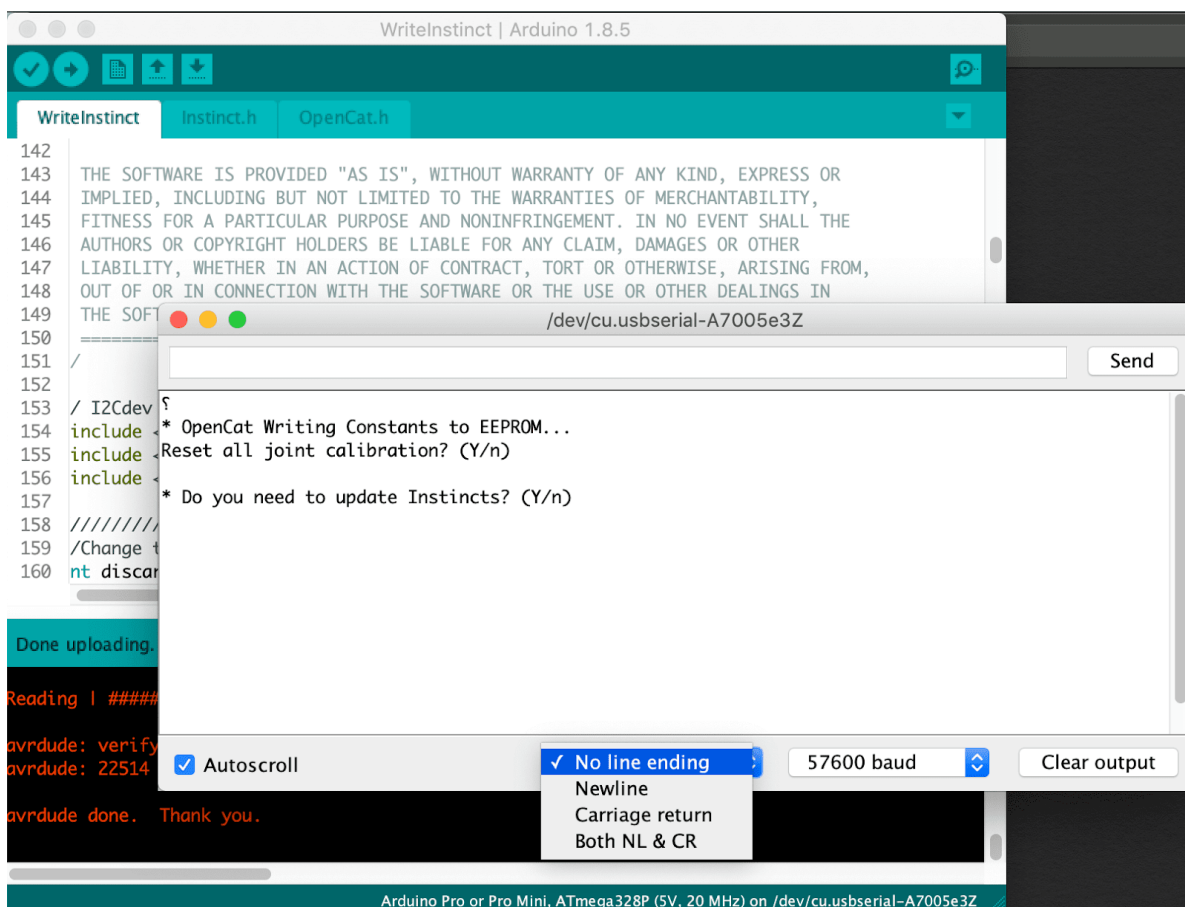
If you really decide to re-burn the bootloader:



- Select the **ATmega328P (5V, 20 MHz)** board under the **Tool** tab 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](#)!
- 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.

3.2.5. Download OpenCat package

- Download a fresh OpenCat repository from GitHub. It's better if you utilize Git's version control feature. Otherwise make sure you download the **WHOLE FOLDER** every time. All the codes should be the same version to work together.
- Open any testX.ino sketch with test prefix. (I recommend using **testBuzzer.ino** as your first test sketch)
- Choose board as **Arduino Pro or Pro Mini** and compile. There should be no error messages. Upload the sketch to your board and you should see messages in the serial monitor. Make sure that your baudrate setting (57600) and board frequency(16MHz or 20MHz) matches with the configuration.
- If there're input prompts, make sure you set "No line ending". Otherwise the invisible '\n' or '\r' characters will confuse the parsing functions.



3.3. Arduino IDE as interface

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

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

OpenCat Communication Protocol and Parsing													
Interface		Token	Encoding	Parameters					Format	Bytes	Function		
RasPi Serial Port	Arduino Serial Monitor	'h'	Ascii						char	1	print h elp information		
		'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					
		'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 k ill					
		'w'		command	"\n"	string	strlen + 2	some future command w ords					
		'r'							char	1	r eset board		
		Binary	'i'	len	idx ₁	a ₁	...	idx _N	a _N	string	len + 2	list of i ndexed rotation angles	
	'l'		a ₁	a ₂	...		a _{DoF}	string	DoF + 1	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.

Note: Some tokens haven't been implemented, such as 'h'. Token 'i' and 'l' still have some bugs.

3.4. Raspberry Pi serial port as interface (only if you are going to use Pi as a master controller)

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.

3.4.1. Config Raspberry Pi serial port

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

Under **Interface** option, find **Serial**. Disabled the serial login shell and enable the serial interface.

[A good tutorial on Instructable](#)

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.

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

If you want to run it as 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:

```
#!/usr/bin/python
```

3.4.3. Use **ardSerial.py** as the commander of Nybble

You need to UNPLUG the FTDI converter if you want to control Nybble with Pi's serial port.

Typing `./ardSerial.py <args>` is almost equivalent to typing `<args>` in Arduino's serial monitor.

For example, `./ardSerial.py kcr` means "perform skill **crawl**".

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

Note: Reduced motion capability when connected to Pi!

With the additional current draw by Pi, Nybble will be less capable for intense movements, such as trot (the token is "**ktr**"). The system is currently powered by two 14500 batteries in serial. You may come up with better powering solutions, such as using high drain 7.4 Lipo batteries, or 2S-18650. There're a bunch of considerations to collaborate software and hardware for a balanced performance. With Nybble's tiny body, it's better to serve as a platform for initiate the communication framework and behavior tree rather than a racing beast.

3.5. Battery

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

When powering the NyBoard with only USB FTDI, there's obviously charging and uncharging in the servo's capacitor and cause the yellow LED to pulse. However the

USB's current is not sufficient to keep the servos working. The servo circuit has to be powered by external batteries to work properly.

3.5.1. Voltage

NyBoard requires 7.4~12V external power to drive the servos. That's usually two li-ion or li-poly batteries connected in serial. A single battery is 4.2V when fully charged and can work normally until voltage drops to 3.6V. That's about 7.2V with two batteries connected in serial. Before installation, dial the potentiometer on NyBoard clockwise to try minimum output first for best output stability. You can turn it up depending on your future need.

3.5.2. Dimensions

The included battery holder is sized for 14500 batteries, that's 14 mm in diameter, and 50 mm in length. 50 ± 1 mm should still fit in. They are the same size as AA batteries, but much more powerful. If you are in the US, we have tested with EBL 14500 li-ion batteries on Amazon.

You can also design other battery holders to carry larger batteries for better performance. That's especially necessary if you mount a Raspberry Pi and want Nybble run as fast.

3.5.3. Connection

Be careful with the polarity when connecting the power supply. **Reversed connection may damage NyBoard!** Make sure you can find the positive (+) and negative (-) sign on both the NyBoard's power terminal and your power supply.

Loosen the screws of the power block. Insert the wires of the battery holder then tighten the screws. When turn the switch on, both the blue LED (for chip) and the yellow LED (for servo) should lit up.

3.5.4. 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 Nybble running.

When the battery is low, the yellow LED will blink slowly. Although NyBoard can still drive one or two servos, it will be very unstable to drive multiple servos at once. **That will lead to repeatedly restarting the program, or awkward joint rotations.** In rare cases, it may even alter the bits in EEPROM. You will need to re-save the constants to restore their values.

3.5.5. Charging

You will need compatible smart chargers for the batteries. Keep batteries attended during charging.

3.5.6. After use

After playing, remember to remove the batteries from battery holder to avoid over discharging.

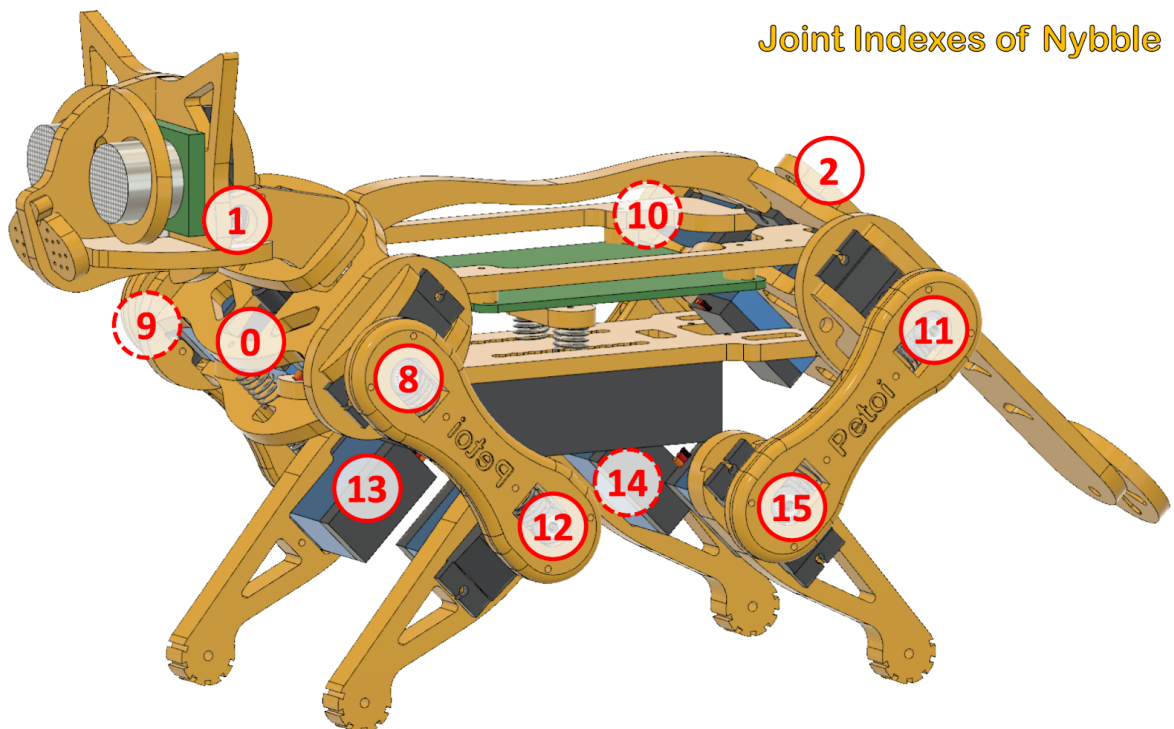
3.5.7. Signal Interference

It's ok to connect both FTDI and battery at the same time. You can type in serial commands while the battery is connected. I do notice that the USB serial port could be disabled randomly. I think that's due to the sudden current draw by servos. It will trigger the computer's over current protection and disable the USB port. In that case, you can change the USB port you're connecting to, reset the USB bus, or restart the computer. So actually it's better to power the board by battery before plug in the FTDI.

4. Connect servos

4.1. Joint map

Nybble's servos are connected to NyBoard's PWM pins symmetrically and resembles the nerves along spinal cord. Though Nybble doesn't have shoulder roll DoF, those indexes(4~7) are reserved for the full OpenCat framework.



Joint Indexes of Nybble

Use h for head, t for tail, r for shoulder roll joint, s for shoulder pitch joint, k for knee joint, F for front, H for hind, L for left, R for right, the full joints map of OpenCat is:

Joint Map of OpenCat

-- Top view on the back



left-hand		Joint	Joint Index	PWM Pin	PWM Pin	Joint Index	Joint	right-hand	
		hPan	0	7	0	1	hTilt		
		rFL	4	6	1	5	rFR		
		sFL	8	5	2	9	sFR		
		kFL	12	4	3	13	kFR		
		kHL	15	11	12	14	kHR		
		sHL	11	10	13	10	sHR		
		rHL	7	9	14	6	rHR		
		tPan	2	8	15	3	N/A		

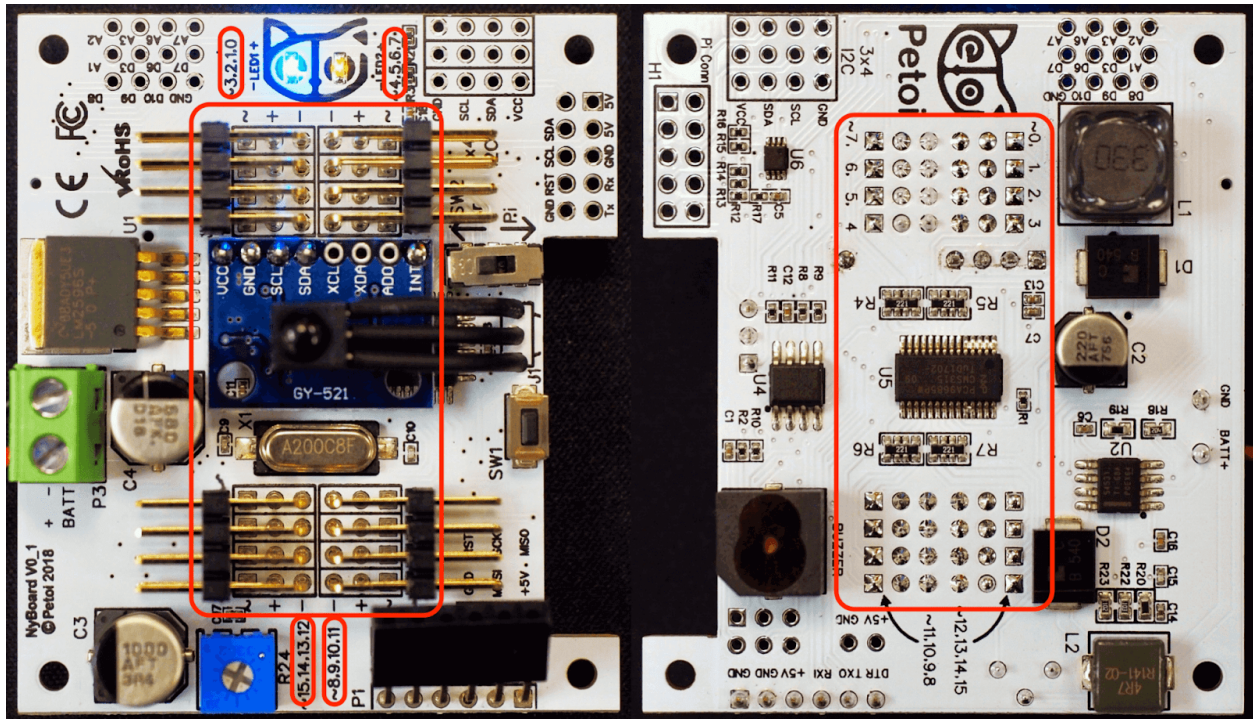
Abbreviation	Joint
h	head
t	tail
r	shoulder roll
s	shoulder pitch
k	knee

Abbreviation	Direction
F	Front
H	Hind
L	Left
R	Right

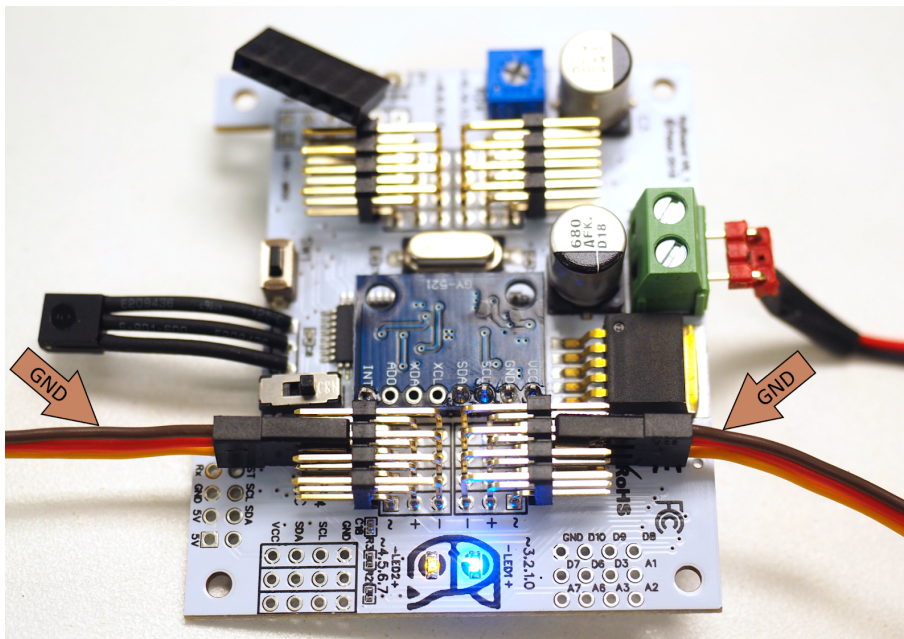
Encoding Array	Joint Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Virtual
	PWM Pin	7	0	8	15	6	1	14	9	5	2	13	10	4	3	12	11	Hardware

4.2. Plug in the servos

Observe the indexing pattern to connect servos with correct PWM pins. Be careful with the wires' direction. The brown wire of servo is GND, while the GND on NyBoard are along the centerline.



A quick check is that all the brown wires should be on top of the other two wires.



5. Calibration

Calibration is vital for Nybble to work properly.

In previous sections, we have prepared those body parts but haven't screw them onto servos. If we don't calibrate the servos before attaching them, they may rotate to any direction, get stuck, and cause damage to either the servos or body parts.

The calibration has four 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 offsets in software.

5.1. Write constants

5.1.1. There are three types of constants to be saved to NyBoard:

- 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;
- Calibration related parameters, like MPU6050 offsets and joint corrections. They are measured in realtime and are saved in on-board EEPROM. They only need to be measured once;
- Skill related data, like postures, gaits, and pre-programmed behaviors. They are mostly defined in **Instinct.h**. You can add more customized skills too.

5.1.2. Upload and run **WriteInstinct.ino**.

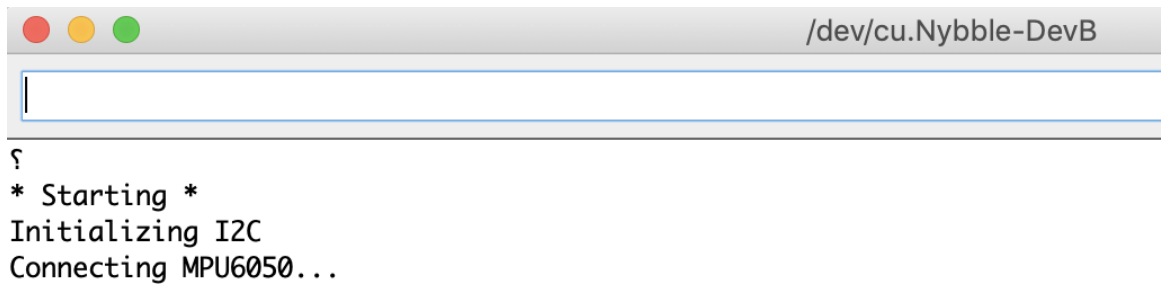
The role for **WriteInstinct.ino** is to write constants to either onboard or I²C EEPROM, and save calibration values. It will be overwritten by the main sketch **Nybble.ino** afterward.

After finish uploading **WriteInstinct.ino**, open the serial monitor. 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 fresh start, type 'Y' to the question. The 'Y' is CASE SENSITIVE!
- 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 in the 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 hang at the connection stage. You can close the serial monitor and reopen it, or press the reset button on NyBoard, to restart the program.



```
?
* Starting *
Initializing I2C
Connecting MPU6050...
```

5.2. Enter calibration mode

The calibration state is defined as the middle point of servo's reachable range. Calibration for servos can be done in either **WriteInstinct.ino** or **Nybble.ino**. I recommend you do it with **WriteInstinct.ino** in case there's something wrong with the constants.

You MUST plug in all the servos and battery for proper calibration. Then in the serial monitor, type 'c' to enter calibration mode. The servos should rotate, make noise, then stop. You will see the calibration table:

```
c
0, 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
c0 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	15
Offset	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

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

Note: The servos are using potentiometer for feedback loop in position control. When holding at static position, they tend to vibrate around target angle. This Parkinson's will develop after a period of use. It won't affect during continuous motion.

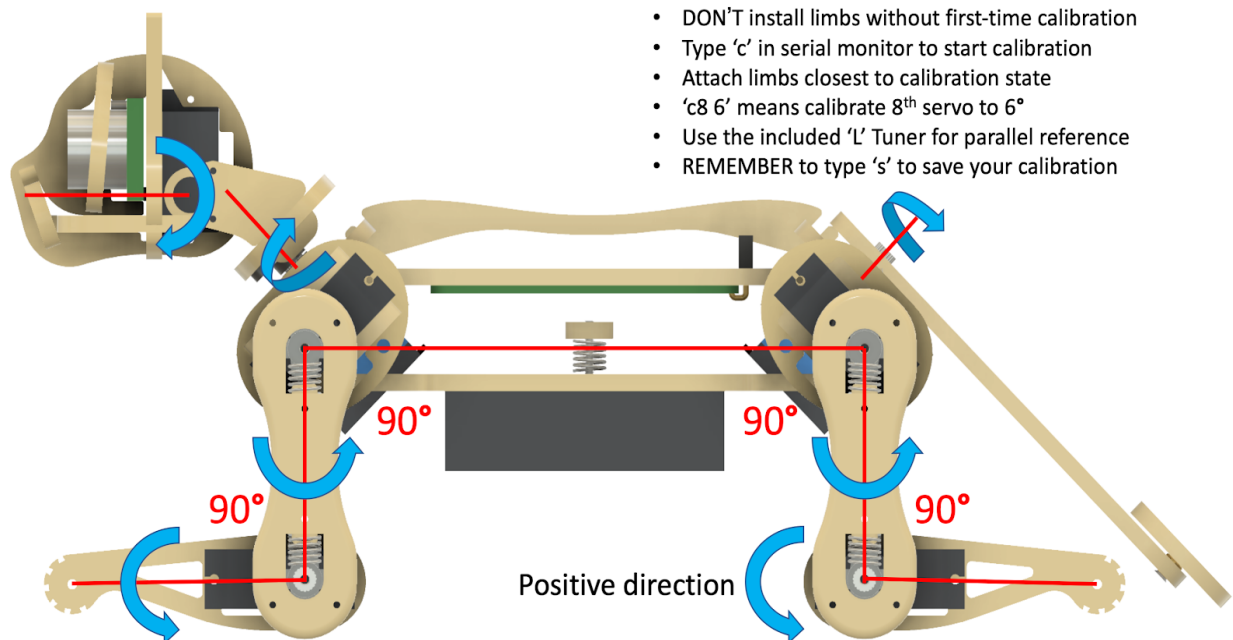
5.3. Attach head, tail, and legs.

5.3.1. Coordinate system

With all servos rotated to their zero angle, now attached the head, tail, and legs prepared in previous section to the body. They are generally perpendicular to their linked body frames. Avoid rotating the servo shaft during the operation.

Rotating the limbs counter-clockwisely from their zero state will be positive (same as in polar coordinates). The only exception is the tilt angle for head. It's more natural to say head up, while it's the result from rotating clockwise.

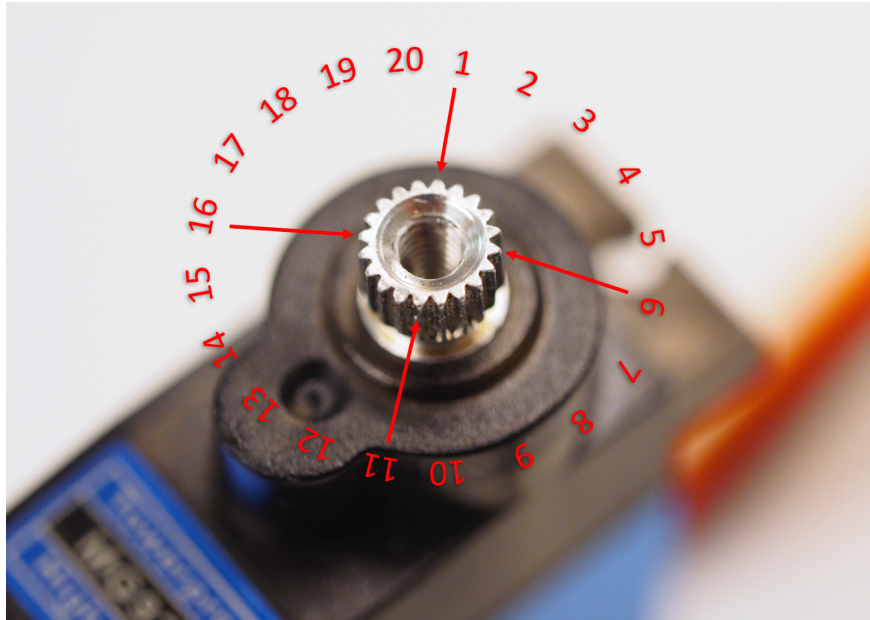
Zero Angle/Calibration State



5.3.2. Understand the angle divisions

If we take a close look at the servo shaft, we can see it has a certain number of gears. That's for attaching the servo arms, and avoid sliding in the rotational direction. In our servo sample, the gears are dividing 360 degree to 20 sectors, each taking **18** degree. That means we cannot always get exact perpendicular

installation. But try to get them as close as possible to their zero states. Use screw A to fix the limbs onto servos.



5.4. Find and save calibration offsets

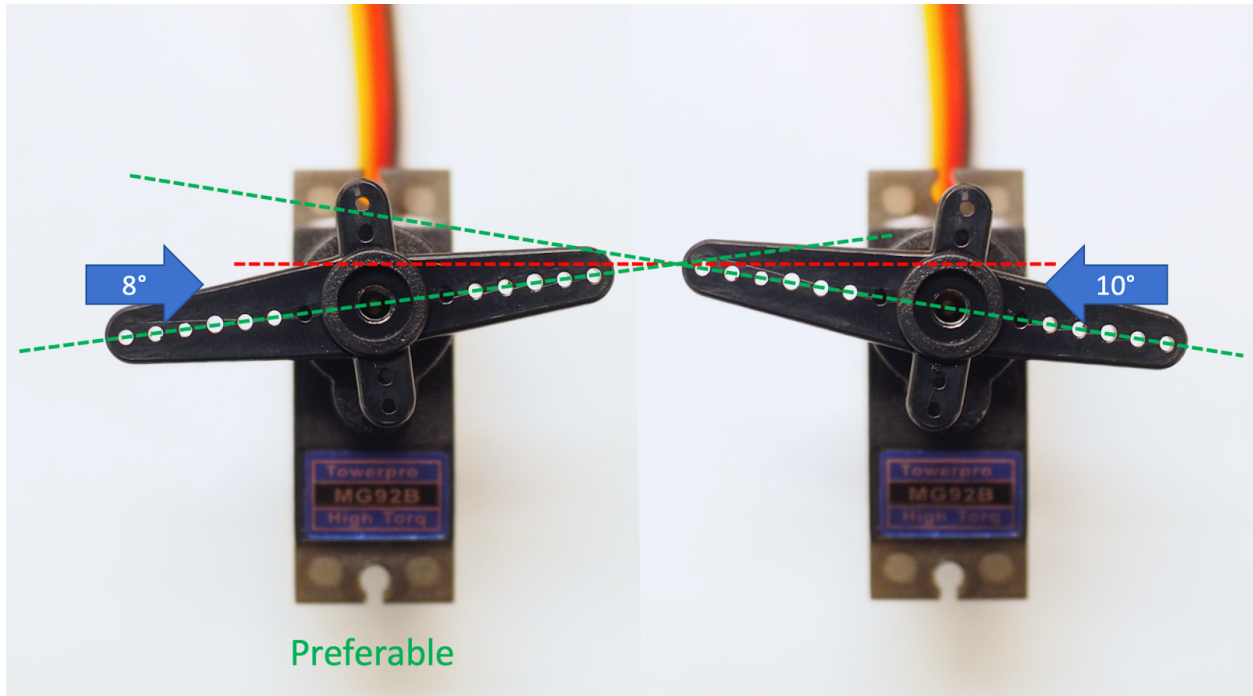
5.4.1. Fine tune the calibration on software side

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

For example, `c8 6` means giving the 8th servo an offset of 6 degree. Find the best offset that can bring the limb to zero state.

Note that 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 decreased reachable range of the servo on either side. Take off the limb and rotate it by one gear. It will result in an opposite but smaller offset.

For example, if you have to use -13 as the calibration value, take the limb off, rotate by one gear then attach back. The new calibration value should be around 5, i.e., they sum up to 18. Avoid rotating the servo shaft during this adjustment.



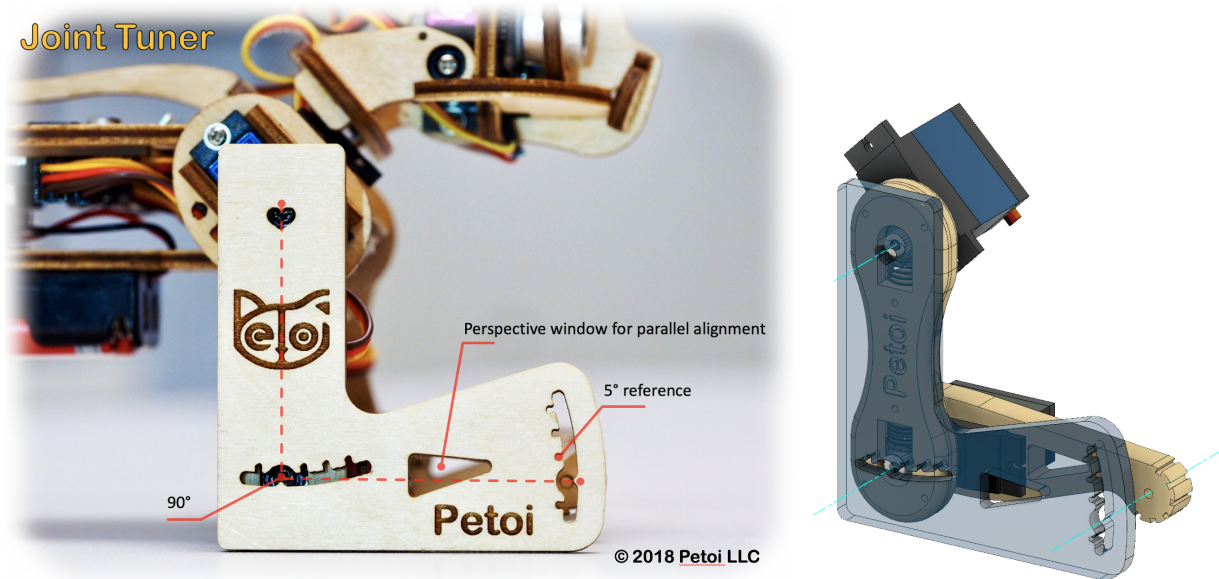
After calibration, remember to type '**s**' to save the offsets. Otherwise they will be forgotten when exiting the calibration state. You can even save every time you're done with one servo.

5.4.2. 'L' shaped joint tuner

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

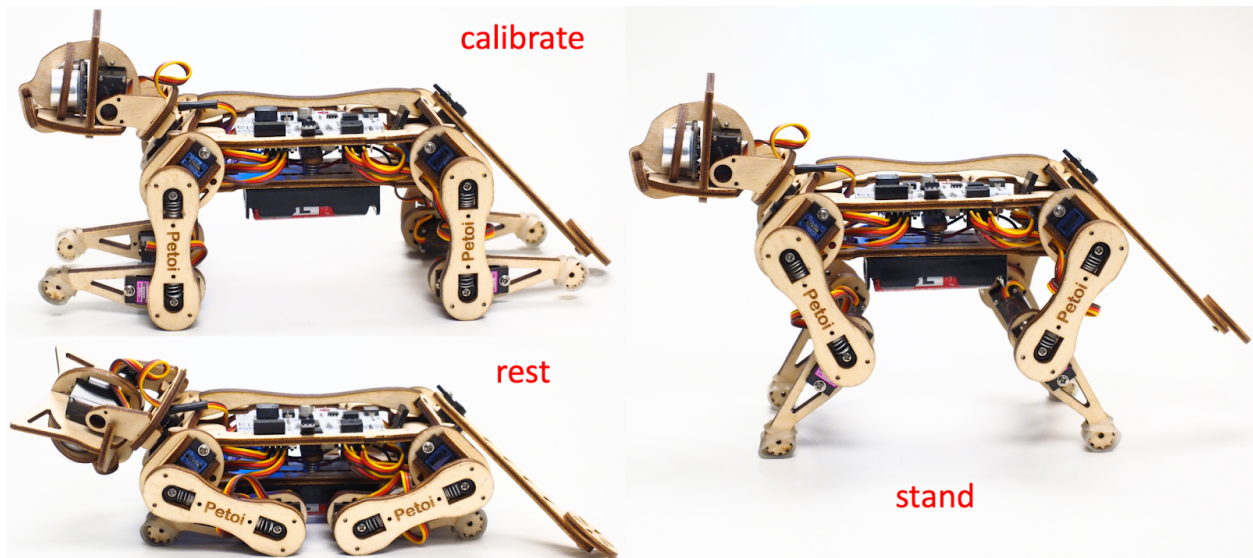
It's especially important that you keep parallel perspective when calibrating Nybble. Use the 'L' shaped joint tuner as a parallel reference to avoid reading errors. Align the tips on the tuner with the center of the screws in shoulder and knee joints, and the little hole on the tip of the foot. Look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

Joint Tuner



5.4.3. Validation

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



5.4.4. Center of mass

Try to understand how Nybble keeps balance even during walking. If you are adding new components to Nybble, 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.

6. Play with Nybble (default usage)

6.1. Control with Arduino IDE

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

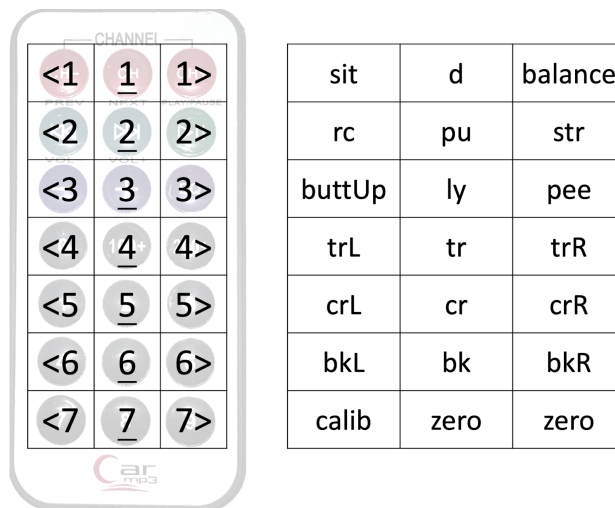
- "ksit"
- "m0 30"
- "m0 -30"
- "kbalance"
- "ktr"
- "ktrL"
- "d"

6.2. Control with Infrared remote

6.2.1. Key map

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

Check the detailed key definitions in function **String translateIR(){...}** in **Nybble.ino**, and they are open to your customization. I'm using abbreviations for key definitions to reduce SRAM usage.



6.2.2. Check out the following featured motions

- Button 1 shuts down the servos and send Nybble to sleep. It's always safe to click it if Nybble is doing something **AWKWARD**. I'm serious. There's still some mechanism in the system I don't fully understand.

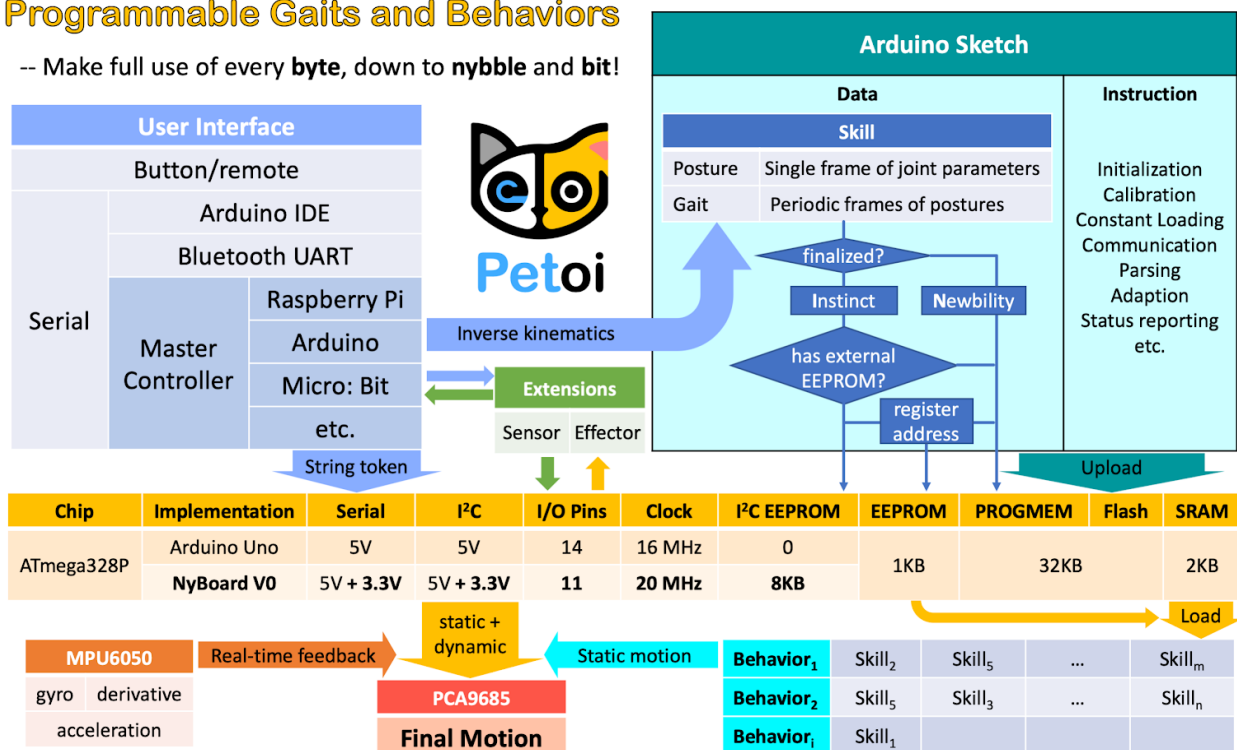
- Button **1>** is the neutral standing posture. You can push Nybble from side, or make it stand up will hind legs and tail. You can test its balancing ability on a fluctuating board. Actually balancing is activated in most postures and gaits.
- With Nybble flat in your hand, click all the buttons on the IR remote to see what they do. Then put Nybble on a wide flat table and try those buttons again.
- You can pull the battery pack down and slide along the longer direction of the body. That will tune the center of mass, which is important for walking performance.
- When Nybble is walking, you can let it climb up/down a small slope (<10 degrees)
- Whatever Nybble is doing, you can lift it vertically, and it will stop moving, just like a cat scruffed on the neck.
- Don't keep Nybble walking for too long. That will overheat the electronics and reduce the servos' life span. The servos are designed to be driven by internal gears. Avoid rotating the servos too fast from outside.
- Be tolerant as if you are playing with a real kitten. (^=●●●=^)

7. Teach Nybble new skills (advanced)

7.1. Understand skills in **Instinct.h**.

Programmable Gaits and Behaviors

-- Make full use of every **byte**, down to **nybble** and **bit**!



One frame of joint angles defines a static posture, while a series of frames defines a periodic motion, usually a gait.

EEPROM has limited (1,000,000) write cycles. So I want to minimize 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. Multiple Instincts are linearly written to the I²C EEPROM only once with WriteInstinct.ino. Their addresses are calculated 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 a 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 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.

7.2. Example **Instinct.h**

```
#define WalkingDOF 8
#define NUM_SKILLS 6
#define I2C_EEPROM

const char cr[] PROGMEM = {
  26, 0, -5,
  35, 37, -46, -53, -23, -32, -3, 12,
  40, 28, -42, -59, -24, -28, -4, 12,
  ...
  33, 39, -47, -51, -22, -32, -3, 11,
};
const char stair[] PROGMEM = {
  54, 0, 30,
  44, 90, -39, -38, 10, -32, -10, 32,
  45, 90, -32, -46, 16, -38, -16, 38,
  ...
  43, 90, -44, -32, 6, -26, -6, 26,
};
const char pu1[] PROGMEM = {
```



```

1, 0, 0,
0,-30, 0, 0, 0, 0, 0, 0, 20, 20, 60, 60, 60, 60,-55,-55,};
const char pu2[] PROGMEM = {
1, 0, 0,
0, 10, 0, 0, 0, 0, 0, 0, 0, 60, 60, 40, 40,-45,-45,-55,-55,};
const char rest[] PROGMEM = {
1, 0, 0,
-30,-80,-45, 0, -3, -3, 3, 3, 60, 60,-60,-60,-45,-45, 45, 45,};
const char zero[] PROGMEM = {
1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,};

#if !defined(MAIN_SKETCH) || !defined(I2C_EEPROM)
    const char* skillNameWithType[] =
        {"crI", "stairN", "pu1I", "pu2I", "restI", "zeroN",};
    const char* progmemPointer[] =
        {cr, stair, pu1, pu2, rest, zero, };
#else
    const char* progmemPointer[] = {stair, zero};
#endif

```

7.2.1. Defined constants

```
#define WalkingDOF 8
```

Means the number of DoF for walking is 8 on Nybble.

```
#define NUM_SKILLS 6
```

Means the total number of skills is 6. It should be the same as the number of items in 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. Both skills will be saved to the flash as PROGMEM.

7.2.2. Data structure of skill array

Observe the following two skills:

```

const char rest[] PROGMEM = {
1, 0, 0,
-30,-80,-45, 0, -3, -3, 3, 3, 60, 60,-60,-60,-45,-45, 45, 45,};

const char cr[] PROGMEM = {

```

```

26, 0, -5,
35, 37, -46, -53, -23, -32, -3, 12,
40, 28, -42, -59, -24, -28, -4, 12,
...
33, 39, -47, -51, -22, -32, -3, 11,
};

```

They are formatted as:

	Total # of Frames	Expected Body Orientation		Indexed Joint Angles															
		Roll	Pitch	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Posture	1																		
Gait	>1											#define WalkingDOF 8							
							#define WalkingDOF 12												

7.2.3. Suffix for indicating Instinct and Newbility

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

```

const char* skillNameWithType[] =
    {"crI", "stairN", "pu1I", "pu2I", "restI", "zeroN"};
const char* progmemPointer[] =
    {cr, stair, pu1, pu2, 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 main sketch **Nybble.ino**, and you are using NyBoard that has an I²C EERPOM, the program will only need the pointer to Newbility list `const char* progmemPointer[] = {stair, zero};` to extract the full knowledge of defined skills.

7.3. Define new skills

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

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

```

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

```

Because it's declared as a Newbility and doesn't require write to I²C EEPROM, you can simply upload **Nybble.ino** everytime you fine-tune the array (without uploading **WriteInstinct.ino**), and trigger the new posture by pressing Z on the IR remote, or type **kzero** in the serial monitor.

You can rename this skill, but remember to update the keymap of IR remote. You can also write short programs to perform multiple skills sequentially, like the push up behavior in Nybble.ino. By integrating sensory data, you can even define behaviors that's triggered by certain interactions!

8. Understand parameters in **OpenCat.h** (research)

9. Mess up with the code and hardware.

To be written by **YOU!**

Share your knowledge and creativity with the community at

<https://www.petoi.com/forum>.

Nybble Olympics

- Speed: Faster
- Skill: Smarter
- Cuteness: Cuter



Appendix I: Parts

