Technical Report

Understanding Petoi Bittle, its Control and New Skill Development.

Name: Ravi Chaudhary (10947986)

Supervisor: Dr Khaled Al Khudir

Word count: 6000

JANUARY 31, 2023

COVENTRY UNIVERSITY

Contents

List of Tables:

List of Figures:

I. Introduction:

Bittle, a quadruped robot dog is developed by Petoi which is extensively programmable and can be controlled or operated by the IR remote. It is an open source, servo-based dog that was created for research purposes in robotics, STEM, coding, and AI learning. The founder and CEO of Petoi is Rong Zhong Li.

II. Bittle Components and functions:

Bittle is comprised of several components, and these are mainboard, servos, battery, IR sensor.

1. *Mainboard:* The mainboard is located and protected under the top cover which is removable. The current and updated version of the main board that Bittle holds is NyBoard V1_1. The NyBoard V1_1 is comprised of the infrared sensor, Buzzer, ATMega 328P, serial interface, MPU 6050, PWM controller, I2C switch, Arduino ICSP, Raspberry PI compatible interface, universal servo connector and groove interface(4x).

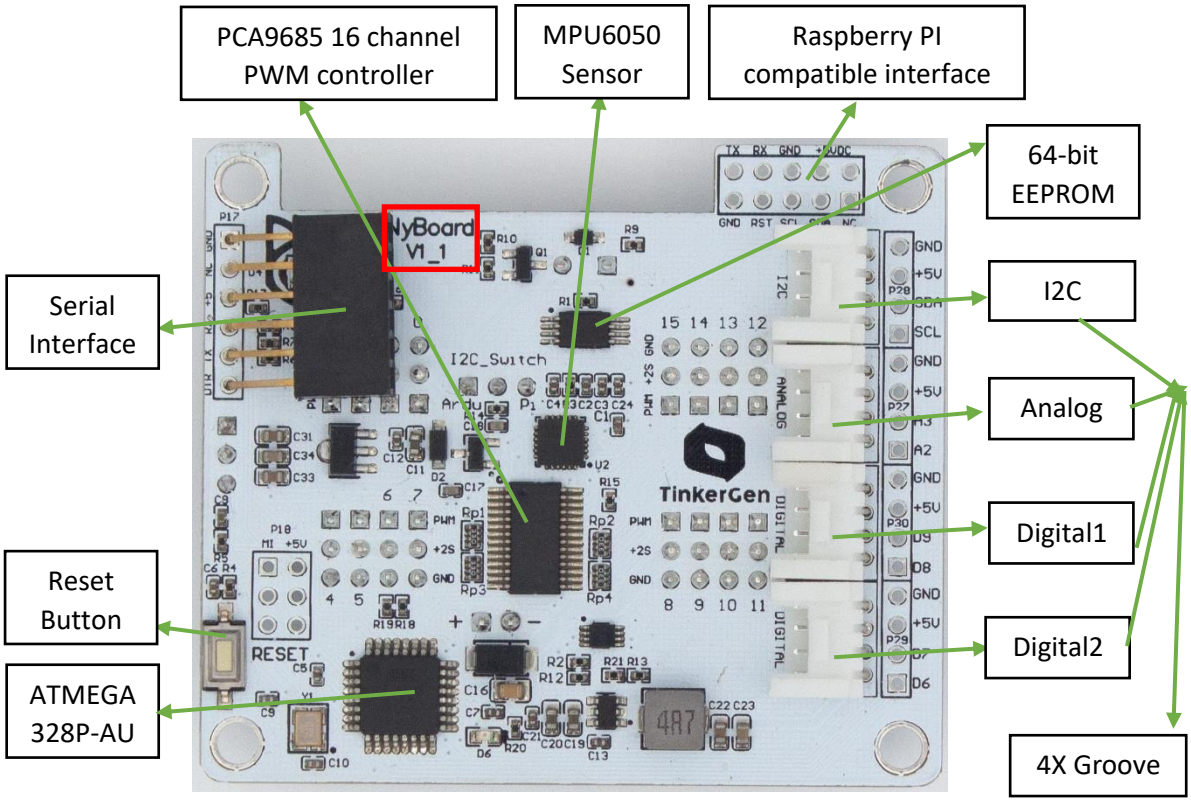


Figure 1: *NyBoard V1_1 front side.*

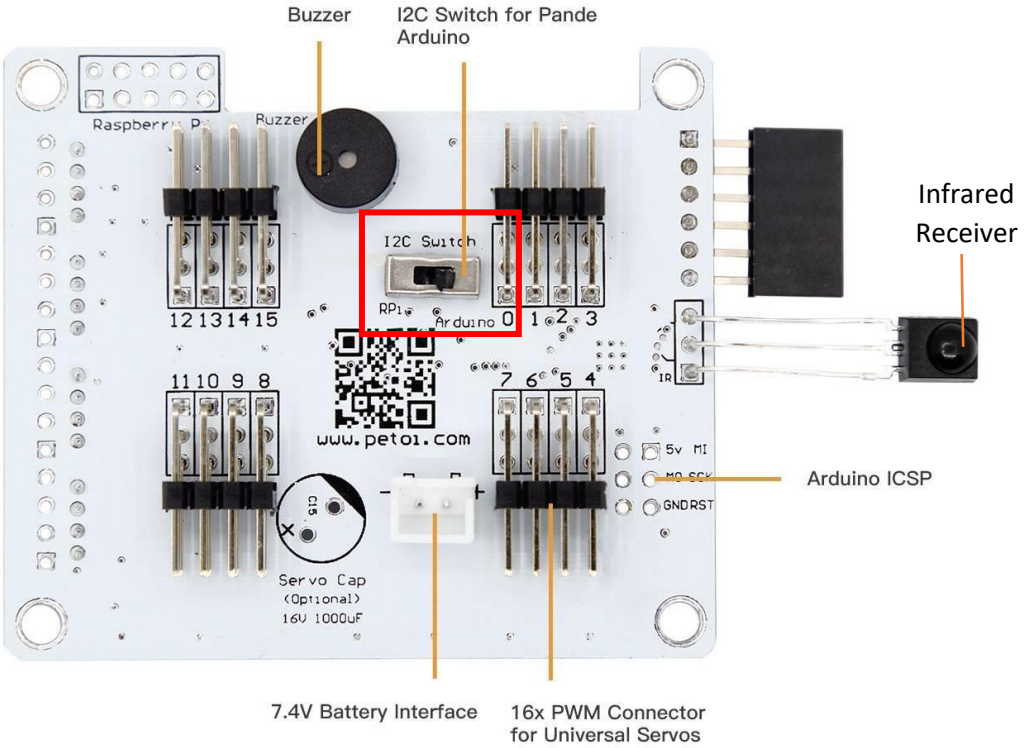


Figure 2: *NyBoard V1_1 back side.*

- **ATMEGA 328P-AU** is the main control chip of the Arduino UNO and Nano which has a maximum clock frequency of 16MHz, 32KB of programme Flash, and 2KB SRAM and 1KB on-chip EEPROM.
- The **16 channel PWM controller** based on PCA9685 chip, controls the servos on the joints and is located at the centre of the NyBoard (long structured).
- The **MPU6050 sensor**, has a 3-axis accelerometer and 3-axis gyroscope and functions to measure velocity, direction, acceleration, displacement, and other motion parameters, is also referred to as a 6-axis motion tracking device.
- The **buzzer** located below the mainboard beeps as the program is uploaded to the Arduino UNO, indication of the low battery or any electric current is passed through it.
- The **serial interface** is associated with the USB device, Bluetooth adapter, and Wi-Fi adapter for communication with other devices, such as uploading code or carrying out any programming.
- The **I2C switch** allows to switch the Arduino to RPI which is essential when the Raspberry PI is linked with the Arduino. Generally, Raspberry Pi has 3.3 V on UART interface while Arduino board have 5 volts in general. Therefore, the **PCA9306** is used convert the ATMega328P's I2C bus to 3.3V.
- The Arduino's firmware can be programmed using **ICSP (In-Circuit Serial Programming)** pins.
- The NyBoard V1_1 has four groove connectors; two of them are digital connectors, one is analog, and the fourth is an I2C Bus connector. The analog module includes the sound sensor, vision sensors, etc. and the signal voltage varies from zero to the board's operational voltage. Similarly, the digital module features a button, RGB LED, ultrasonic ranger, electromagnet, etc. and the signal is limited to values of 0 and 1. The I2C bus connector facilitates interaction with a wide variety of I2C devices.

- **Raspberry PI compatible interface** on Arduino allows a connection with the other boards such as Raspberry PI. Sometimes the memory in the ATMega (2KB) might not be enough for the stable operation of the movement algorithm and ROS node. Similarly, it doesn't have a wireless interface or image processing capability which might lead to connecting Raspberry PI with an Arduino board.

2. *Battery:* The Lithium-Ion battery is used in the Bittle. It is due to its lightweight, and sturdiness, and can be recharged several times. Moreover, it has a low self-discharge tendency and higher voltage capacity. The Lithium-Ion battery used in the Bittle has 7.4V which has an input voltage of 5V-1A. It has a rated charging capacity of 1000mAh (milliampere-hour).

3. *Servos:* The Bittle is powered by altogether 9 servos. Each leg has 2 servos: four on the shoulders and four on the knees and the remaining one on the head. Bittle can attain a variety of postures due to servo rotation and the flexibility of the shoulder and knee joints. The parameters in the servos are optimized where the speed of the servos is tuned with less torque unlike, the traditional servos whose torque is larger but are slower. The servos can perform more complex motion such as the rotation angle is very high.

4. *IR sensor:* The HX18388 IrDA infrared sensor is used in the Bittle's mainboard which is embedded on the back of the NyBoard and serves as a receiver for the remote controller. The remote's infrared LED emits a certain sequence of infrared pulses which is decoded into a number once it reaches the receiver and the specified action is determined. There are a few limitations to using the IR sensor as the IR remote uses LED light which requires line-of-sight for signal transmission to the receiver. It has a limited range of merely 10 meters.

III. Methods of control:

Before connecting a Bittle with a device, let's have a look at the different methods to control a Bittle.

- ***Use of Petoi Desktop App***

1. Arduino IDE
2. Python
3. IR remote controller and,
4. Petoi mobile app.

Among these, the Petoi mobile app and IR remote controller are only intended to control the Bittle that has already been programmed, whilst the Arduino IDE and Python allow the low-level programming of the Bittle to generate new skills. Similarly, the Petoi Desktop app creates an animation of each posture to generate a behaviour, or a gait and it also enables the export of data structure of the skill for usage in Arduino IDE and Python programming.

IV. Modes of Connection:

A Bittle can be connected to a computer or other devices via one of three connection types. These include Bluetooth adapter, Wi-Fi adapter, and USB adapter.

1. *USB adapter:* Bittle uses CH340C USB bus conversion chip which enables the conversion of serial signals to USB and USB to serial interface. The USB adapter used in the Bittle has 3 main interfaces; Micro USB interface, NyBoard download interface and Communication module debugging

interface. The **NyBoard download interface** is used to connect to the NyBoard, transfer programme firmware to the Bittle, and establish a serial port connection with the PC. The **communication module debugging interface** allows the connection of Bluetooth or wi-fi adapters within it. Moreover, it updates the function of modules and debug the bug parameters. The **Micro USB interface** supports the connection between PC and the NyBoard with the type-D USB cable.

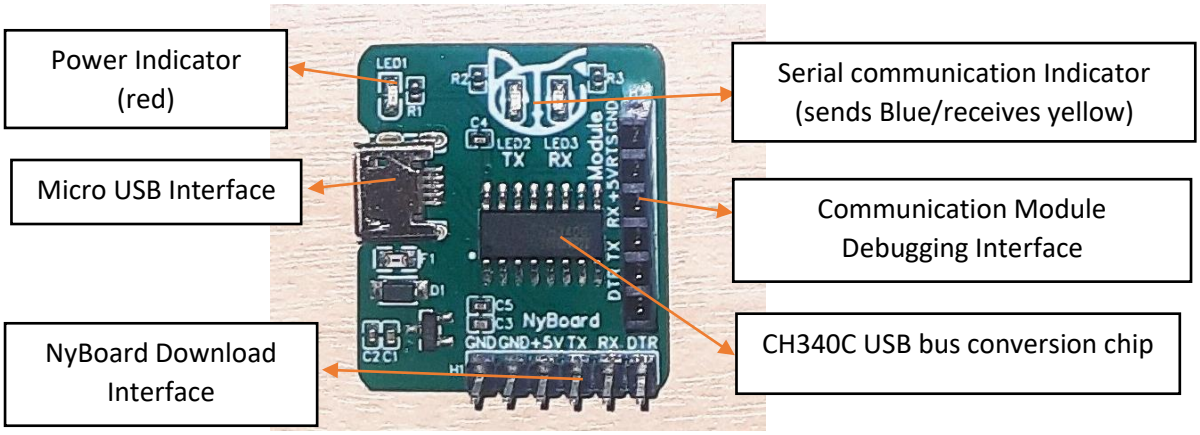


Figure 3: *USB Adapter(CH340C).*

2. ***Bluetooth adapter:*** One of the easiest and most convenient methods to programme and manage the Bittle is through the Bluetooth module connection since, unlike USB, it gives the Bittle a set range to test skills freely. One of the important things to notice is LED light indication; the flickering LED light indicates that the device is waiting to pair with the Bluetooth. Once it is paired, the LED light gets stable for a sec and again starts blinking. The absolute stable LED light implies the connected state of Bluetooth and the device, which usually happens during sketch uploading or any communication.

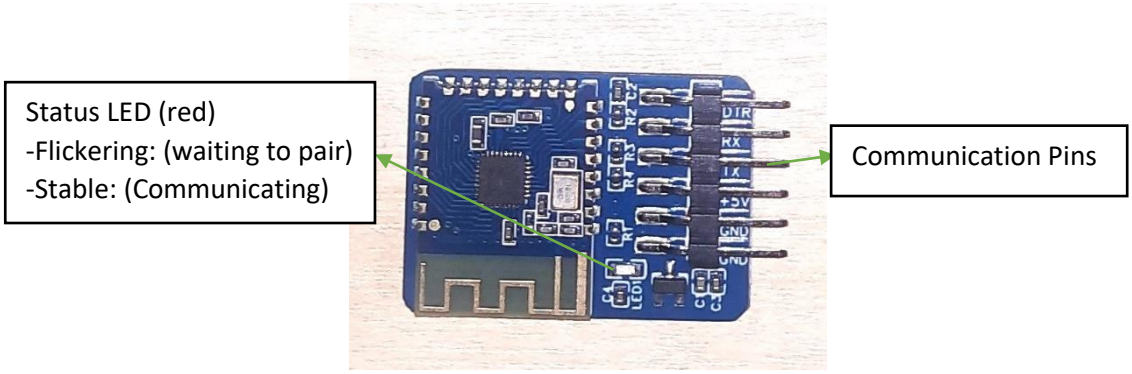


Figure 4: *Bluetooth Adapter.*

Note 1:

Firstly, the PC Bluetooth only pairs the device but doesn't connect. The connection is built after choosing the correct port while using the software's. While programming with any of the software's, it is important to select the correct serial port for establishing a connection.

3. ***Wi-fi Adapter:*** Since this report focuses more on Bluetooth and USB connection, please visit (Wi-Fi ESP8266) for more details about Wi-fi adapter.

V. Upload Sketch to NyBoard using Arduino IDE:

Sketch is a term that Arduino uses to describe a programme which is uploaded and executed to an Arduino board. There are following steps to upload a sketch to Arduino IDE software:

a. At first pair the device or PC either with Bluetooth or USB module. Although the procedures are the almost same for both USB and Bluetooth modules, the Bluetooth module will be focused in this discussion.
b. Check whether the device is connected to the Bluetooth or not. To do so, open device manager on PC and unhide ports (COM &LPT) by clicking its arrows. If it's paired, the COM ports with random number will appear.
c.

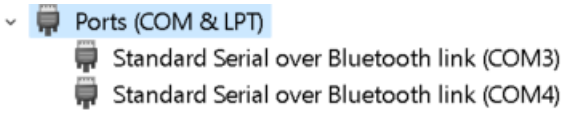


Figure 5: *Serial Ports check through device manager.*

d. Download the Arduino IDE software and install it.
e. Now, download the OpenCat repository from GitHub. Extract the downloaded file and rename the **OpenCat-main** folder as **OpenCat**, it is because the **OpenCat.ino** file within it needs to be inside the OpenCat folder to run.
f. Open the **OpenCat.ino** file in the Arduino IDE.
g. Go to the Tools menu bar and then Boards and select **Arduino Uno**.
h. Again, go to the Tools menu bar and then port and select the serial port which is appropriate.
i. **Uncomment the 40th line of the sketch** which allow the main sketch to be uploaded.
j. Upload the sketch either by going to sketch menu bar and upload, or just by clicking the right-headed arrow below the menu bar.
k. It will take some time to upload the sketch.

The downloaded OpenCat repository from GitHub contains all the pre-set skills in the Bittle. The following programmed skills codes can be found inside the folder; **OpenCat→src→InstinctBittle**. Similarly, the program to control the Bittle with the specific remote buttons can be found inside the folder; **OpenCat→src→infrared**. Uploading the mentioned OpenCat.ino file won't show any changes in the Bittle skill. It is because these are the pre-set skills used by the Bittle.

Now the general question that comes up in the mind is how to create the new skills and its procedure to upload as well as make the Bittle function according to the remote button of own's choice. This has been discussed further.

Note 2:

1) To go through the boot up stage, make sure the I2C switch is in the Arduino side rather than RPI side as marked in red in ***figure 2****.*

VI. Possible connection issues *(Read if applicable):*

a. For window users, sometimes the latest version of Arduino IDE (win 10 and above) isn't compatible for window 11 installed computers due to which connection problem arises while uploading the sketch in Arduino IDE.
b. The serial port chosen mightn't be the correct one. The message window will display an error indicating "Access is denied" mentioning serial Port as shown in fig.

```
An error occurred while uploading the sketch
avrdude: ser_open(): can't open device "\\.\COM3": Access is denied.
```

Figure 6: *Expected error due to point no. a and b.*

c. The physical connection between the serial interface and the communication pins might be wrong.
d. There may no driver for the uploader installed on the computer.
e. The uploader device may be faulty.

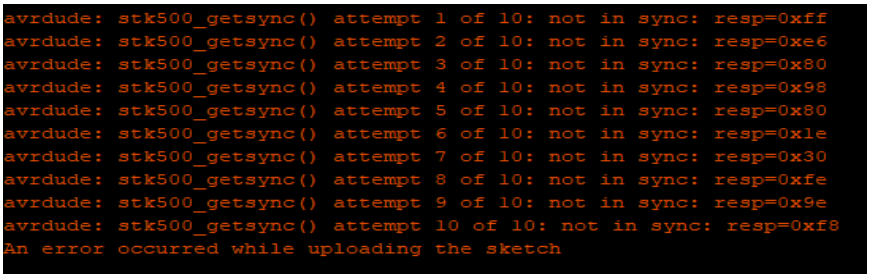


Figure 7: *Expected error due to point no. c and d.*

VII. Possible connection Solutions *(Read if applicable):*

1. If the connection problem occurs with the newer version of the Arduino IDE, try downloading the older one, Arduino IDE (win 7).
2. Try installing driver CH340.
3. Select the appropriate serial port and Arduino board **(Arduino UNO)**.
4. Check the connection pins if they are correctly connected with the serial interface.
5. If USB adapter is used, then the USB cable included in the kit is recommended to use.
6. Make sure no I2C device is connected with I2C interface, neither the digital pins 0 nor 1.

VIII. Ways to identify the correct serial port:

Technique 1:

1. In Arduino IDE software, go to **tools→serial ports** and list out all the ports.
2. Disconnect the USB or Bluetooth, then check which port is missing from the list.
3. After reconnecting the USB or Bluetooth, whichever port is appeared is the correct port.

Technique 2:

1. Go to **tools→serial ports** and try each port. Upload the sketch and whichever port doesn't show any error and successfully uploads the sketch will be the correct port. Basically, it's a trial-and-error method.
2. Same trial and error method can be applied In Petoi Desktop App and Code Craft to identify the correct port.

IX. Understanding the memory where the Skills are stored:

Bittle Skills have been classified as instincts and Newbility. The Instincts are the fixed skills whereas Newbility are the newly created skills that needs several tests. The **on-chip EEPROM (1KB)** of **ATMega 328P-AU** is exclusively used to store the addresses of both sorts of skills as a lookup table, however the data are stored to other memories.

1. *8KB Onboard I2C EEPROM:* The Instincts are stored in Onboard I2C EEPROM which is of size 8KB but as mentioned above the addresses are stored in 1KB on-chip EEPROM during the run time of OpenCat.ino. **OpenCat.ino** uploads the multiple Instincts at once that are written in I2C EEPROM.

2. *PROGMEM (32KB of programme Flash):* The Newbility skills are stored in the Flash memory of size 32KB in the PROGMEM format. It means, in order to save the Newbility to a Flash memory, the PROGMEM should be included while writing a program. It is also recommended to upload the Newbility to the Flash memory due to it higher storage capacity. Uploading the skills to EEPROM may cause instability due to low memory. The Newbility is uploaded as a component of the sketch of Arduino. The addresses of these skills are also stored in on-chip EEPROM and the value changes with the change in number of skills (entire Instinct and Newbility).

X. Creating new skills:

The skills are saved in the format (*.h) file, usually InstinctBittle.h if it's a Bittle and for Nibble InstinctNibble.h. Let's first understand the meaning of written program and skills.

***Table 1**: Example of written program in InstinctBittle.h file.*

```
// for instance, a short version of InstinctBittle.h

#define BITTLE
#define NUM_SKILLS 4
#define I2C_EEPROM

const int8_t balance[] PROGMEM = {
1, 0, 0, 1,
   0  0  0  0  0  0  0  0 30  30  30, 30  30  30  30  30};
const int8_t zero[] PROGMEM = {
1, 0, 0, 1,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,};

const int8_t frf[] PROGMEM = {
 -6,   0,   0,   2,
   0,   0,   0,
  42,   0,   0,   0,   0,   0,   0,   0,  15,  15,  15,  15,  15,  15,  15,  15,   8,  0,  0,  0,
  42,   0,   0,   0,   0,   0,   0,   0,  24,  24,  15,  15,  61,  67,  15,  15,  48,  0,  0,  0,
  42,   0,   0,   0,   0,   0,   0,   0, -59, -59,  15,  15,  41,  41,  15,  15,   0,  1,  0,  0,
  10,  20,   0,   0,   2,   2,   1,   1,   7,   7,   3,   3,  -3,  - 3,   2,   2,  48,  4,  0,  0,
 -15, -40, -23,   0,  -2,  -2,   1,   1,  35,  35,  35,  35, -28,  -28,  -28, -28,   8,  2,  0,  0,
   0,   0,   0,   0,   0,   0,   0,   0,  15,  15,  15,  15,  15,  15,  15,  15,   4,  2,  0,  0,
};

const int8_t run[] PROGMEM = {
28,   0,   0,   1,
  0,   0,  68,  68,  44,  44,  40,  37,
  7,   7,  64,  64,  51,  51,  36,  33,
 20,  20,  60,  60,  26,  26,  32,  29,
 29,  29,  63,  63,  16,  16,  21,  21,
 41,  41,  66,  66,   6,   6,  -7,  -7,
```

```
 …..
 18,  18,   71,    71,   15,  20, -16,  -16,
 26,  26,   75,    75,   23,  28,    1,    1,
 31,  31,   81,    81,   28,  33,    8,    8,
 37,  37,   84,   84,    37,  42,  21,   21,
  3,  43,   74,   74,   13,   18,   8,    8,
 47,  47,   70,   70,    7,   12,   4,    4,
};

#if !defined(MAIN_SKETCH) || !defined(I2C_EEPROM)

const char* skillNameWithType[]={"balanceI","zeroN","frfI","runI",};

const int8_t* progmemPointer[] = {balance, zero, frf, run,};

#else

const int8_t* progmemPointer[] = {zero,};

#endif
```

The aforementioned example demonstrates the way to write the program for Bittle, and any newly created skills might be added with minor adjustments.

Defined Constants:

Here, #define BITTLE defines the program is written for the Bittle. If it has been done for Nibble then it would be #define NIBBLE.

#define NUM_SKILLS 4 defines the total number of skills used to write the program is 4. The number of skills should correspond the number in the list const char* skillNameWithType[].

#define I2C_EEPROM states I2C EEPROM is enabled to store instincts.

Note 3:

If the custom circuit board is used that doesn't contain I2C EEPROM then this line could be commented out using (//) the double slash symbol. It makes the program save in the Flash as PROGMEM but it's obvious it will occupy the space for instincts too.

Skill Array Data Structure:

Table 2: *Skill scripts and data structure.*

```
const int8_t balance[] PROGMEM = {
1, 0, 0, 1,
   0,   0,   0,   0,   0,   0,   0,   0,  30,  30,  30,  30,  30,  30,  30,  30,};
const int8_t run[] PROGMEM = {
28,     0,    -2,     1,
  0,    0,   68,    68,   44,   44,   40,  37,
```

```
  7,   7,  64,   64,  51,  51,  36,  33,
 20,  20,  60,   60,  26,  26,  32,  29,
 29,  29,  63,   63,  16,  16,  21,  21,
 41,  41,  66,   66,   6,   6,  -7,  -7,
 …..
 18,  18,  71,   71,  15,  20, -16, -16,
 26,  26,  75,   75,  23,  28,   1,   1,
 31,  31,  81,   81,  28,  33,   8,   8,
 37,  37,  84,   84,  37,  42,  21,  21,
  3,  43,  74,   74,  13,  18,   8,   8,
 47,  47,  70,   70,   7,  12,   4,   4,
};
const int8_t frf[] PROGMEM = {
 -6,   0,   0,   2,
  0,   0,   0,
 42,   0,   0,   0,   0,   0,   0,   0,  15,  15,  15,  15,  15,  15,  15,  15,   8,   0,   0,   0,
 42,   0,   0,   0,   0,   0,   0,   0,  24,  24,  15,  15,  61,  67,  15,  15,  48,   0,   0,   0,
 42,   0,   0,   0,   0,   0,   0,   0, -59, -59,  15,  15,  41,  41,  15,  15,   0,   1,   0,   0,
 10,  20,   0,   0,   2,   2,   1,   1,   7,   7,   3,   3,  -3,  - 3,   2,   2,  48,   4,   0,   0,
-15, -40, -23,   0,  -2,  -2,   1,   1,  35,  35,  35,  35, -28, -28, -28, -28,   8,   2,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,  15,  15,  15,  15,  15,  15,  15,  15,   4,   2,   0,   0,
};
```

There are three different examples of skills shown above. The 1st one is the **static posture**, 2nd defines a **gait,** and the 3rd is a **behaviour**. The sequence of frames defines either it's a gait or behaviour. A **frame** can simply be defined as the number of rows Index joint angles. For instance, the index joint angles in the 3rd skill in ***table no. 2*** consists of 6 rows, which means it has 6 frames.

The 1st skill (static posture) and the 2nd skill (gait) shown in **table no. 2** are formatted as following:

	Total No. of frames	Body Orientation		Angle ratio	Indexed Joint Angles															
		Roll	Pitch		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Balance	1	0	0	1	0	0	0	0	0	0	0	0	30	30	30	30	30	30	30	30
run	28	0	-2	1									0	0	68	68	44	44	40	37
													7	7	64	64	51	51	36	33
													…	…	…	…	…	…	…	…
													47	47	70	70	7	12	4	4

Figure 8:*Format showing the composition of "behaviour" and "gait" frames.*

Here, "Balance" is the static posture. It is because it consists only one frame of 16 joint angles. The "run" is the gait. Gait consists multiple frames with only 8 indexed joint angles in each frame (or 12, based on DOF of walking) which forms a loop or repetitive motion. The body orientation creates a certain angle while performing the skills. If the body tilts beyond the anticipated angles, certain adjustments will be computed by the balancing algorithm. The default angle ratio is 1and is suitable for angles between -127 to 128. Setting up the angle ratio to 2 will allow storing the larger angles than

-127 to 128 with the division of 2. The indexed joint angles determine how the shoulders, knees, and head (overall Bittle) function while performing the skill.

Note 4:

The theory to create the Bittle skill is discussed. The practical example to create the skills has been further explained in the next topic.

***Table 3:** Data structure of Behaviour.*

```
//just an example

const int8_t frf[] PROGMEM = {
 -6,   0,   0,   2,
   3,   4,   2,
  42,   0,   0,   0,   0,   0,   0,   0,  15,  15,  15,  15,  15,  15,  15,  15,    8,   0,   0,   0,
  42,   0,   0,   0,   0,   0,   0,   0,  24,  24,  15,  15,  61,  67,  15,  15,   48,   0,   0,   0,
  42,   0,   0,   0,   0,   0,   0,   0, -59, -59,  15,  15,  41,  41,  15,  15,    0,   1,   0,   0,
  10,  20,   0,   0,   2,   2,   1,   1,   7,   7,   3,   3,  -3,  - 3,   2,   2,   48,   4,   0,   0,
 -15, -40, -23,   0,  -2,  -2,   1,   1,  35,  35,  35,  35, -28,  -28, -28, -28,    8,   2,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,  15,  15,  15,  15,  15,  15,  15,  15,    4,   2,   0,   0,
};
```

This is the example of Behaviour, and the data structure looks a bit different than the gait. The "frf" is the symbol for front flip. The numbers in the green row and the grey rows have the same meaning as in the gait. Unlike gait, behaviour consists of 16 joint angles as can be seen in the grey rows. The negative sign (-6) in the number of frames, denotes a behaviour. The row in a sky-blue colour describes the repetition of the frames. For instance, (3, 4, 2) suggests a Bittle to repeat the fourth and fifth frames twice before the 6th frame. Despite the fact that frames 3 and 4 are indicated, the repeat will occur in frames 4th and 5th because the frame count begins from 0. While the gait loops over until it is stopped, the behaviour executes only once.

The yellow rows indicate speed, delay, trigger axis and trigger angle in each of the frames respectively.

a. The speed has the range from 1(slow) to 127(fast). The default speed is 4. The maximum speed is when it's set to 0 where the servo rotates at a speed of about 0.07sec/60 degrees.
b. The 2nd number is the delay and can be set from 1 to 127. The default delay is 0. The 1 in the example indicates delay of 50ms, 2 indicates 100ms, 3 indicated 200ms, 4 indicates 300ms and it increases such a way.
c. The 3rd number represents the trigger axis. The prior set delay will be disregarded if the trigger axis is more than 0. The trigger of the following frame is proportional to the corresponding axis body angle, 1 will be for pitch axis and for the roll axis, it's 2. The direction will be determined from sign and the sign will be determined comparing the size of current angle with the trigger angle.
d. The fourth element represents the trigger angle which can be from -128 to 127 degrees.

Statements and suffix:

The meaning of the statements is described in the table using the comments (//).

***Table 4:** Statements and suffix with detailed description.*

```
const char* skillNameWithType[]={"balanceI","zeroN","frfI","runI",};
#if !defined(MAIN_SKETCH) || !defined(I2C_EEPROM)

//use this if it is not the Main Sketch (OpenCat.ino) to save the data or there is no External EEPROM.

const int8_t* progmemPointer[] = {balance, zero, frf, run,};

#else
const int8_t* progmemPointer[] = {zero,};

//else, use this if the main sketch (OpenCat.ino) is used and the NyBoard has an I2C EEPROM, Pointer is required to the nobilities because the instincts are saved to External EEPROM, whereas the nobilities on progmem are stored to new addresses.

#endif

//For SRAM to function effectively, ensure sur to allow adequate memory. For safety, a single skill should not be more than 400 bytes.
```

The suffix "I" can be used if some new skill is added. For example, jump is defined as a new skill then "I" will be added in the skill name string as "jumpI". The suffix "I" and "N" prompts for the skill data storage location and their address assigning time.

XI. Practical ways to create new skills.

There are several ways to create new skills. These are as follows;

- Petoi Desktop App
- Code Craft
- Arduino IDE
- Python

1. Petoi Desktop App: In this report, the Petoi Desktop App will be used to create a new skill and the generated skill will be added to the InstinctBittle.h file which will further uploaded to the Arduino IDE.

   Petoi Desktop App is a graphical user interface which enables firmware configuration, calibration and create customised skills. It has the 3 main features; **Firmware uploader**, **Joint Calibrator** and **Skill Composer**.

1.0.1. Firmware uploader:

To upload the firmware is the first step. Follow the steps to upload the firmware and to create the new skills.

a. At first, connect the Arduino with the device either with USB or Bluetooth module.
b. Open the Petoi Desktop App. The interface as shown in ***figure 9*** will pop-up.

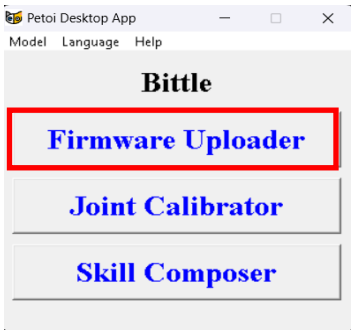


Figure 9: *Firmware uploader interface.*

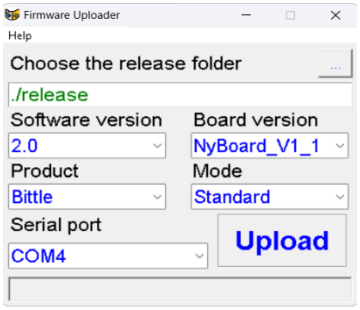


Figure 10: *Interface pop-up after firmware uploader.*

c. **Click on Firmware Uploader.** Another interface as shown in ***figure 10*** will appear. Select the software version as 2.0 which is default. The Board version can be found in NY Board's top surface, select accordingly. Choose the correct serial port.
d. Clicking upload will start uploading parameters and the board execute the configuration program as long as all the selected options including the serial port is correct.
e. Options for resetting the joints and calibration will appear. Do not need to redo the calibration if it's already calibrated whereas joints can be reset.
f. The window will appear showing "Parameter Initialized" and confirming the next step to upload the main function code, uploading begins. If the firmware will be successfully uploaded, the status bar shows completion of the Firmware Upload.
g. Now the Skill Composer is ready to use, clicking on the it will open the Skill Composer Interface which allows to customize the motion of Bittle or create new skills.

Notes 5:

1) Make sure that no I2C devices are connected to the I2C interface at the time of firmware upload in order to avoid firmware upload failure.

2) It is must to upload the Firmware before composing a skill, otherwise the Skill Composer and Joint Calibrator won't function.

1.0.2. Skill Composer:

Figure 11: *State Dials section of skill composer interface.*

At the state dials, The Bittle will automatically connect with the specified serial Port. If not, then it shows "listening" in place of the "connected" button. Clicking the "listening" will disconnect all the ports and enables connect button. Turning on the servo won't let the servos move manually by hand during skill composition which is crucial for promptly determining the robot's centre of mass for balance. Turning on Gyro helps Bittle to balance back once it rolls upside down and sense the body angle and motions.

Figure 12: *Preset Postures section of skill composer.*

These are the pre-set postures which are used initially while creating a sequence of movement. Usually, the balance is recommended to use initially. As the Postures are changed, the sliders in the joint calibrator also change which gives the idea to use the sliders in the joint calibrator.

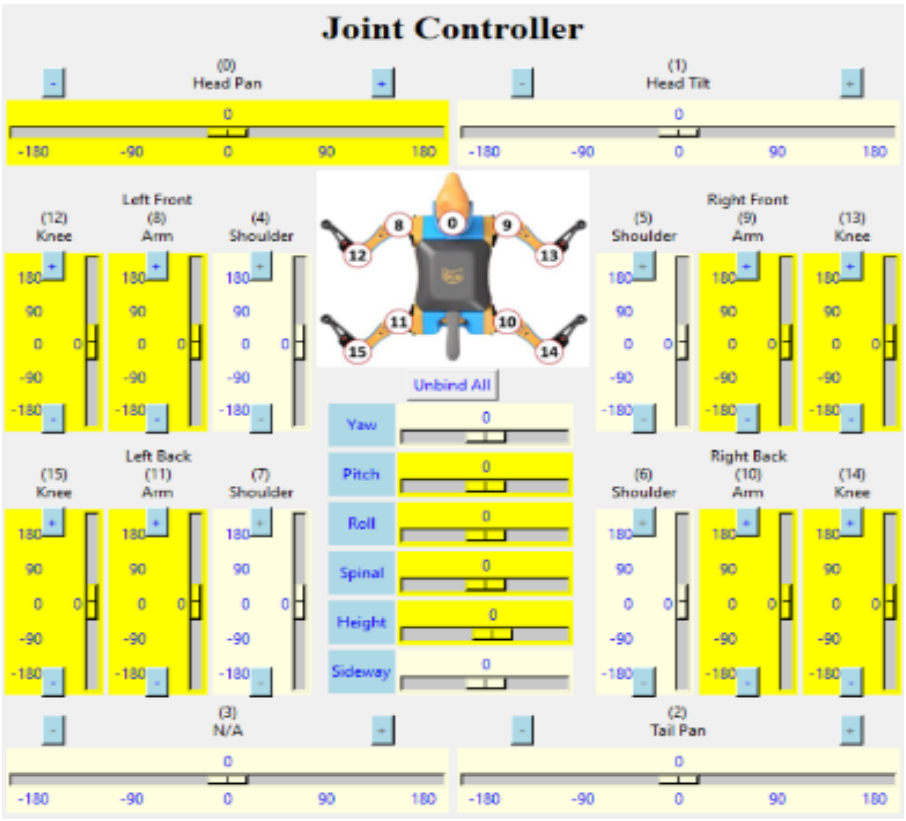


Figure 13: *Joint Controller section of skill composer.*

As shown in ***figure 13***, the knees, shoulders and the head consist separate sliders. By sliding the sliders up and down will make the Bittle joints move accordingly. Sliding the sliders down will move the knees and soldier's servo in an anti-clockwise direction whereas sliding it up will move them to clockwise direction. Similarly, the head pan will allow the head to rotate clockwise and anti-clockwise. Selecting the multiple ''+'' buttons will allow all the selected joints move with the same increment and similarly, decrement for ''–'' button. The "Unbind All" option will detach all the joints instantaneously. Using the larger angle than -125 to 125 isn't recommended which will increase the response time. The other four options roll, pitch, spinal and height will globally orient and translate the posture. The pitch and roll will adjust the pitch angle and roll angle respectively. Spinal makes the Bittle movement in the direction of spinal and the height will adjust the height of Bittle's body.

Figure 14: *Skill Editor section of skill composer.*

Only one move can be created at a time using a joint controller. Therefore, that one move can be stored clicking the **set "=" button**. The sequence of movements needs to be created and stored in the frames one after another to make Bittle perform a complete skill. The play button will play all the moves together in a sequence. If the one specific move needs to be edited or played, for instance, the movement stored at 1 can be played and edited clicking set button "=" beside 1. The edited move needs to be again stored clicking "=" in the same frame.

- The red back arrow button (del) will terminate or delete the corresponding frame or move. Similarly, the green "v" (add) button allows to insert a new frame and to create a new move in between the already existed frames.
- The names of the different animals at each of the frames is the note to remind movement created, The names can be edited to make them more descriptive of what the frame holds.
- The default delay is 0. The delay can be prolonged by a factor of 50ms, 100ms, 200ms, 300ms and continues till 6000ms. The given delay period causes the Bittle to delay on the selected frame.
- The trigger and angle are inter-related. Clicking either upper or lower arrow in the trigger will activate the roll and Pitch angle (pitch, -pitch, roll, -roll) and those angles can be set in the angle section.

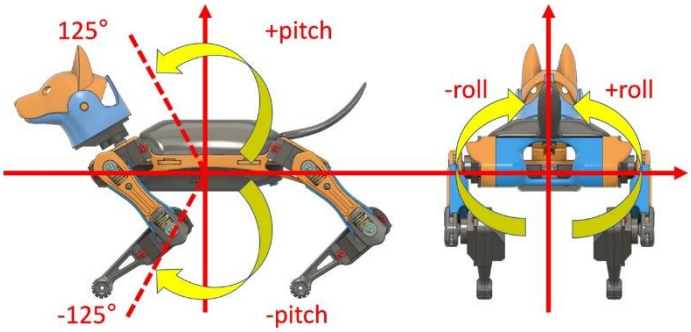


Figure 15: *Figure showing Roll and Pitch angle.*

- The “step” is the speed which determines the speed of the certain frame. The default speed is 4, the lowest is 2 and the highest is 48, although the maximum is 0 which is called “max”. Controlling the speed between each frame is necessary for some of the skills. For example, for a jumping Bittle, the speed should be instantly increased after a certain point.
- The frames can be looped over for certain times entering the number in the repeat text box located beside set. Similarly, as an example, if 1st to 5th frames need to be looped over then 1st and 5th frame can be selected as shown in ***figure 14*** and set the number in the repeat text box.
- The “>|<” button allows to mirror the posture of the active frame and the “mirror all” button will exchange the moves in left and right side of the Bittle’s joint.
- The export option allows to export or save the created frame list as a skill which will be further utilised in Arduino IDE. The example of the exported skill (behaviour) looks like as shown in ***figure 16.***

```
{
-5, 0, 0, 1,
 0, 0, 0,
    0,   0,   0,   0,   0,   0,   0,   0,  54,  54,  54,  54, -18, -18, -18, -18,    8, 2, 0, 0,
   80,   0,   0,   0,   0,   0,   0,   0,  10,  10, -44, -44, 100, 100,  50,  50,   48, 0, 0, 0,
   80,   0,   0,   0,   0,   0,   0,   0, -16, -16,  52,  52, -56, -56, 125, 125,    0, 1, 0, 0,
   80,   0,   0,   0,   0,   0,   0,   0,  54,  54,  82,  82, -44, -44, -50, -50,   48, 4, 0, 0,
    0,   0,   0,   0,   0,   0,   0,   0,  30,  30,  30,  30,  30,  30,  30,  30,    8, 2, 0, 0,
};
```

Figure 16: *Example of Behaviour created using Petoi Desktop App.*

- Clicking on the behaviour button will change the behaviour to gait. Therefore, choose the option correctly either behaviour or gait is required to create.

Note 6:

The speed of the servos in gait is much higher in comparison to behaviour. The extremely high speed of the servos may damage the servo. Therefore, while creating the gait, it is recommended to add more frames (depending on skills) with very small change in the postures in each frame.

XII. Adding the created skill in the InstinctBittle.h file:

The created skill will be added to the **InstinctBittle.h** file and the program inside the **infrared** file will be edited to make the Bittle perform the newly added skill with the preferred remote button.

For Instance, the created skill using the Petoi Desktop App is;

Table 5: *Customized behaviour using Petoi Desktop App.*

```
{
-5,  0,  0,  1,
 0,  4,  3,
-30, -80, -45,  0,  -3, -3,  3,  3, 70,  70,  70, 70, -55, -55, -55, -55,   8,  0,  0,  0,
-30, -80, -46,  0,  -4, -4,  2,  2, 36,  36,   3,  3, -50, -46,  12,  12,  48,  2,  0,  0,
-30, -80, -46,  0,  -4, -4,  2,  2, 24,  24,  20, 20,   9,  13,  69,  69,   0,  0,  0,  0,
 0,   0,   0,   0,   0,  0,  0,  0, 22,  22,  46, 46,  -5,  -5, -19, -19,  48,  0,  0,  0,
 0,   0,   0,   0,   0,  0,  0,  0, 30,  30,  30, 30,  30,  30,  30,  30,   4,  0,  0,  0,
};
```

- Copy the skill and Open **InstinctBittle.h** file under **OpenCat→src→InstinctBittle.** The very similar program as described in the **creating new skills** section of this report can be seen. The program looks like as in ***Table 6***.

Table 6: *Similar example of the program as in InstinctBittle.h file.*

```
#define BITTLE
//number of skills: 6

const int8_t balance[] PROGMEM = {
1, 0, 0, 1,
   0,  0,  0,  0,  0,  0,  0,  0, 30, 30, 30, 30, 30, 30, 30, 30,};
const int8_t zero[] PROGMEM = {
1, 0, 0, 1,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,};
const int8_t rest[] PROGMEM = {
1, 0, 0, 1,
 -30, -80, -45,  0, -3, -3,  3,  3, 75, 75, 75, 75, -55, -55, -55, -55,};
const int8_t crF[] PROGMEM = {
 34, 0, 2, 1,
  42,  73,  83,  75, -43, -42, -49, -41,
  37,  75,  78,  77, -41, -41, -50, -41,
  36,  78,  73,  80, -38, -41, -50, -41,
  37,  81,  68,  82, -36, -41, -50, -40,
  41,  83,  62,  85, -36, -40, -48, -40,
  42,  88,  57,  85, -36, -40, -47, -39,
  45,  92,  51,  88, -37, -44, -44, -38,
  48,  91,  50,  90, -38, -45, -41, -37,
  51,  89,  55,  91, -39, -48, -40, -36,
  53,  84,  55,  93, -40, -49, -40, -35,
  56,  80,  58,  99, -40, -50, -41, -37,
  59,  75,  61, 101, -41, -50, -41, -40,
  62,  69,  64, 101, -41, -50, -41, -42,
  64,  64,  65,  99, -41, -49, -41, -44,
  67,  58,  68,  95, -42, -48, -42, -46,
  70,  53,  71,  92, -42, -47, -42, -47,
  73,  48,  74,  88, -42, -45, -41, -48,
  74,  42,  75,  83, -41, -43, -41, -49,
  77,  37,  78,  78, -41, -41, -41, -50,
  80,  36,  81,  73, -41, -38, -41, -50,
  82,  37,  83,  68, -40, -36, -40, -50,
  85,  41,  85,  62, -40, -36, -40, -48,
  89,  42,  87,  57, -41, -36, -39, -47,
  93,  45,  89,  51, -45, -37, -38, -44,
  92,  48,  91,  50, -47, -38, -37, -41,
  88,  51,  93,  55, -48, -39, -36, -40,
  83,  53,  96,  55, -49, -40, -35, -40,
  78,  56, 100,  58, -50, -40, -38, -41,
  73,  59, 102,  61, -50, -41, -42, -41,
  68,  62,  99,  64, -50, -41, -43, -41,
  62,  64,  96,  65, -48, -41, -45, -41,
```

```
  57,  67,  93,  68, -47, -42, -47, -42,
  51,  70,  89,  71, -46, -42, -48, -42,
  46,  73,  84,  74, -45, -42, -49, -41,
};
//const int8_t hlw[] PROGMEM = {
//14, 0, 0, 1,
  //10,  29,  51,  46,  33,  30,   4,  15,
  //10,  32,  43,  48,  40,  29,   3,  16,
  //15,  35,  34,  49,  35,  30,  13,  18,
  //19,  44,  36,  52,  33,  17,  14,  19,
  //22,  38,  39,  53,  31,  16,  14,  22,
  //25,  30,  42,  61,  30,  17,  14,  11,
  //28,  21,  44,  56,  30,  21,  15,   6,
  //31,   8,  47,  49,  29,  40,  16,   3,
  //33,  13,  48,  41,  30,  37,  17,   3,
  //38,  16,  51,  33,  28,  34,  18,  17,
  //42,  20,  52,  37,  16,  32,  20,  14,
  //35,  23,  59,  40,  16,  31,  16,  14,
  //26,  26,  59,  42,  19,  30,   8,  14,
  //15,  28,  52,  45,  27,  30,   5,  15,
//};
const int8_t jump[] PROGMEM = {
-5,   0,   0,   1,
 0,   4,   3,
-30,  -80,  -45,   0,   -3,  -3,   3,   3,  70,   70,   70,  70,  -55,  -55,  -55,  -55,    8,   0,    0,    0,
-30,  -80,  -46,   0,   -4,  -4,   2,   2,  36,   36,    3,   3,  -50,  -46,   12,   12,   48,   2,    0,    0,
-30,  -80,  -46,   0,   -4,  -4,   2,   2,  24,   24,   20,  20,    9,   13,   69,   69,    0,   0,    0,    0,
  0,    0,    0,   0,    0,   0,   0,   0,  22,   22,   46,  46,   -5,    -5,  -19,  -19,   48,   0,    0,    0,
  0,    0,    0,   0,    0,   0,   0,   0,  30,   30,   30,  30,   30,    30,   30,   30,    4,   0,    0,    0,
};
const int8_t pu[] PROGMEM = {
-10, 0, 0, 1,
 7, 8, 3,
   0,   0,   0,   0,   0,   0,   0,   0,  30,  30,   30,   30,   30,  30,  30,  30,    8, 0, 0, 0,
  15,   0,   0,   0,   0,   0,   0,   0,  30,  35,   40,   21,   50,  15,  15,  41,   12, 0, 0, 0,
  15,   0,   0,   0,   0,   0,   0,   0,  30,  35,   40,   30,   50,  15,  15,  14,   16, 0, 0, 0,
  30,   0,   0,   0,   0,   0,   0,   0,  27,  35,   40,   60,   50,  15,  20,  45,   16, 0, 0, 0,
  15,   0,   0,   0,   0,   0,   0,   0,  42,  35,   40,   60,   25,  20,  20,  60,   12, 0, 0, 0,
   0,   0,   0,   0,   0,   0,   0,   0,  48,  45,   75,   60,   20,  37,  20,  60,   12, 0, 0, 0,
 -15,   0,   0,   0,   0,   0,   0,   0,  60,  60,   70,   70,   15,  15,  60,  60,   16, 0, 0, 0,
   0,   0,   0,   0,   0,   0,   0,   0,  30,  30,  110,  110,   60,  60,  60,  60,    8, 1, 0, 0,
  30,   0,   0,   0,   0,   0,   0,   0,  75,  70,   85,   85,  -50, -50,  60,  60,   16, 0, 0, 0,
  s 0,   0,   0,   0,   0,   0,   0,   0,  30,  30,   30,   30,   30,  30,  30,  30,   8, 0, 0, 0,
};

#if !defined(MAIN_SKETCH) || !defined(I2C_EEPROM)

const char* skillNameWithType[]={"balanceI","zeroN","restI", "crFI","jumpI","puI",};

const int8_t* progmemPointer[] = {balance, zero, rest, crF, hlw, pu,};
```

```
#else

const int8_t* progmemPointer[] = {zero,};

#endif
```

The above skills in ***table 6*** are taken from InstinctBittle.h file (only few skills to demonstrate example) and these are the Instincts stored in I2C_EEPROM. To add the new skill, it is recommended to comment the instincts, more like replacing with the new skill to save the memory space. Else adding too many new skills might cause instability problem in Bittle due to less memory. Follow the bullet points and example in ***table 6*** together for better understanding.

- Let's add the example skill created using Petoi Desktop App i.e., **jump**. Copy the newly created skill and paste below one of the Instincts. For instance, in this example, the "**hlw",** is commented out first and "**jump**" is added just below it. Comment each line of the frames as shown in green colour.
- Make sure to rename the skill name in the statement. For above example: const int8_t jump[] PROGMEM. Any name can be given to the skill but the same given name is required to be used in the program for remote control. Symbols are frequently used instead of names, and the proper way of creating a symbol for naming can be followed in ***Appendix 2***.
- As "hlw" is commented out and the new skill "jump" is added, the suffix "hlwI" should be replaced by "jumpI". Similarly, the suffix "hlw" in the progmemPointer should be replaced by "jump".
- Now copy the program and paste it in the **void loop()** section of the Arduino IDE software. It is because it allows to run the code repeatedly whereas pasting the program in the **void setup()** will only run the code once. Example of void setup() is the Bittle will not turn on more than once. Follow as shown in ***Appendix 1***.
- Don't forget to select the Board and connection Port from tool menu bar as discussed in **Upload Sketch to NyBoard using Arduino IDE** section in this report.
- Compile the program clicking the tick button which will check the program and shows the result in the command window. If there is no error found, upload the program using right headed arrow in the menu bar.
- The skill will be uploaded. After successful upload, the Bittle will stop responding.

Now, lets add the new skill to make it perform with one of the buttons of the IR remote. To make it easy, replace the IR remote button of commented skill with the new skill. For that, go to **OpenCat→src→infrared.** The program seems like in the ***Table 7.***

Table 7: *Program written for IR remote control.*

```
#include "ir/IRremote.h"

//#include <IRremote.h>
//The included library is identical to the IRremote library by shirriff, version 2.6.1
//Source: https://github.com/Arduino-IRremote/Arduino-IRremote
//Here, we include the decoding functions in our folder only to make it more convenient for newbie users
```

```
//All rights belong to the original author, and we follow the MIT license.
//You no longer need to modify ~/Documents/Arduino/libraries/src/IRremote/IRremote.h as mentioned in our old manual.

IRrecv irrecv(IR_PIN);
decode_results results;
//abbreviation //gait/posture/function names
#define K00 "d"       //rest and shutdown all servos
#define K01 "F"       //forward
#define K02 "g"       //turn off gyro feedback to boost speed

#define K10 "L"       //left
#define K11 "balance" //neutral stand up posture
#define K12 "R"       //right

#define K20 "p"       //pause motion and shut off all servos
#define K21 "B"       //backward
#define K22 "c"       //calibration mode with IMU turned off

#define K30 "vt"      //stepping
#define K31 "cr"      //crawl
#define K32 "wk"      //walk

#define K40 "tr"      //trot
#define K41 "sit"     //sit
#define K42 "str"     //stretch

//#define K50 "hlw"      //greeting
#define K50 "jump"      //jump
#define K51 "pu"      //push up
#define K52 "pee"     //standng with three legs

#ifdef NYBBLE
#define K60 "stand" //"lu"      //look up
#define K61 "buttUp"    //butt up
#elif defined BITTLE
#define K60 "ck"      //check around
#define K61 "jy"      //joy
#else
#define K60 "fd"      //fold
#define K61 "rt"      //
#endif

//#define K62 "z"  //turn on/off the random behaviors
#define K62 "zero" //call your customized Newbility saved to PROGMEM

//#define K62 "T"    //call the last skill data received from the serial port

#define SHORT_ENCODING // activating this line will use a shorter encoding of the HEX values
```

```
String translateIR() // takes action based on IR code received
// describing Remote IR codes.
{
……..
…….. continued.
```

- As highlighted in yellow, the "hlw" is commented out and the same button "K50" is copied and replaced by the "jump" as highlighted in grey. This makes the Bittle jump with the same button of IR remote which it used to do greeting before. Make sure the name of the skill is same as uploaded in InstinctBittle.h. In this example the name is "jump".
- Save the file and done. This file doesn't need to be uploaded.

Notes 7:

*1) Make sure both the file InstinctBittle and infrared are inside the src folder and OpenCat.ino should be inside the OpenCat folder. Therefore **OpenCat-main** folder should be renamed by **OpenCat**.*
2) Also make sure, the name of file containing program of the skills i.e., InstinctBittle shouldn't be changed, otherwise there might cause an error while uploading OpenCat.ino.

Now,

- Upload the **OpenCat.ino** file in Arduino IDE. Once it's uploaded, the Bittle will start responding and it follows the IR remote as programmed.
- Make sure to uncomment the 40th line of the sketch i.e., (#define MAIN_SKETCH) before uploading.

Notes 8:

*1) If the Bittle doesn't perform the skill as programmed after uploading the main sketch, then it might require the **joints reset**.*
2) To reset joints, comment the 40th line of the sketch i.e., (//#define MAIN_SKETCH) and upload it. After successful upload, open the serial monitor (lens like logo on the right top of menu bar). It will automatically ask for joint reset. Type "y" and enter. Now the joints are reset.
3) It will further ask for calibration. If the joints are already calibrated, then type "n" and enter. This won't reset calibration.
4) Close the serial monitor and uncomment line 40. Upload the sketch. Now the Bittle should perform accordingly with the IR remote as programmed.

XIII. Calibration:

Calibration is the configuration of servos to ensure precision, consistency, and standardization in measurements. Calibration is must for the stability before any skills are produced or the Bittle perform any skills. There are different ways to calibrate the servos in the Bittle which are as follows:

1. **Petoi Desktop App:** After uploading the firmware, go to Joint Calibrator which opens the Joint Calibrator Interface. Calibrate the joints with the help of "L" shaped tuner included in the box. Save the calibrated joints. *(Tutorial on how to upload firmware and how to use Joint Calibrator Interface can be found in topic: **Practical ways to create new skills**).*

2. **Code Craft:** Select Bittle in the Code Craft. First select and connect the correct Serial Port clicking on the connect button. After the serial port gets connected, go to "Add Extension" located on the

bottom left. Now add “Calibrate Servos”. The calibration Interface will open and calibrate the servos using the “L” shaped tuner included in the box. Don’t forget to save after calibration is done.

3. **Arduino IDE:** Upload the OpenCat.ino file in Arduino IDE. Make sure the 40th line of the sketch *(//#define MAIN_SKETCH)* is commented out. After successful upload, click on the serial monitor. First, it will ask for the joint reset. Type “y” and enter to reset joints. Next it will ask for calibration. Type “y” and enter. Now, type and enter "c" to access the calibration mode.

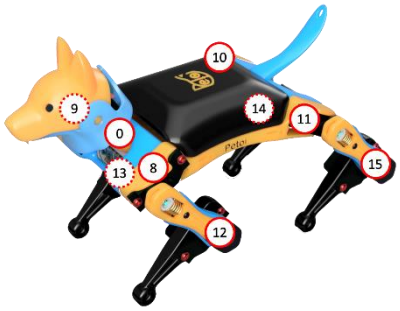


Figure 17: *Bittle joints with number specified for calibration.* ***Figure 18:*** *way to use "L" shaped tuner.*

To move the servo joints to its calibration position, follow the ***figure 17*** and the example below.

- Type C12 4 to move 12th servo by an offset of 4 degree.
- Type C0 -7 to move the 0th servo by an offset of -7 degree.

Use the “L” shaped tuner and select offsets according to it for perfect calibration.

Follow the video for better understanding.

Note 9:

The unassembled Bittle requires to be calibrated before its components (heads, and legs) are installed. Calibrate the servos in the similar way using serial monitor, reset the joints and enter “y” for calibration and done. This will bring the servos at zero state. Now, install its parts close to the calibration state. Again, repeat all the process in the topic calibration no.3 (Arduino IDE).

XIV. Control the Bittle using Serial Monitor in Arduino IDE:

To control the Bittle using Arduino IDE, upload the OpenCat.ino file with 40th line uncommented (#define MAIN_SKETCH). After the successful upload, open the serial monitor. Now, for instance, typing and entering “**ksit**” will make the Bittle sit. It means to make the Bittle perform a certain skill, “**k**” should be added before the skill name. Here, the thing that needs to be considered is the names of the skills given while writing the program which will be provided in the ***Appendix 2***. The other skills which are in the program but aren’t added in the IR remote could also be accessed through serial monitor. These are also called the hidden skills.

It also allows to tune the new skills by sending posture frames using **m, i, l**, command without the upload of new sketch. The skill created can be saved to InstinctBittle.h and can be uploaded as an Instinct or newbility.

XV. Other method to control Bittle:

This report doesn't include simulation of Bittle but here is little guidance for it.

ISSAC SIM: ISSAC Sim is the Nvidia software which provides the platform for robot simulation. Creating a complex Bittle skill requires lots of trial and error which may damage the servos in it. Therefore, simulation of Bittle is one of a good option to test and create the Bittle skills virtually. For the simulation of Bittle, download the Unified Robot Description Format URDF file and follow the video for more information.

Note: *The requirements to use the ISSAC SIM are minimum 32GB RAM, 50GB SSD, any RTX GPU, Intel core i7(7th generation) AMD 4 Ryzen 5.*

References:

Bittle Course. (n.d.). Retrieved from TinkerGen Help Center: https://www.yuque.com/tinkergen-help-en/bittle_course

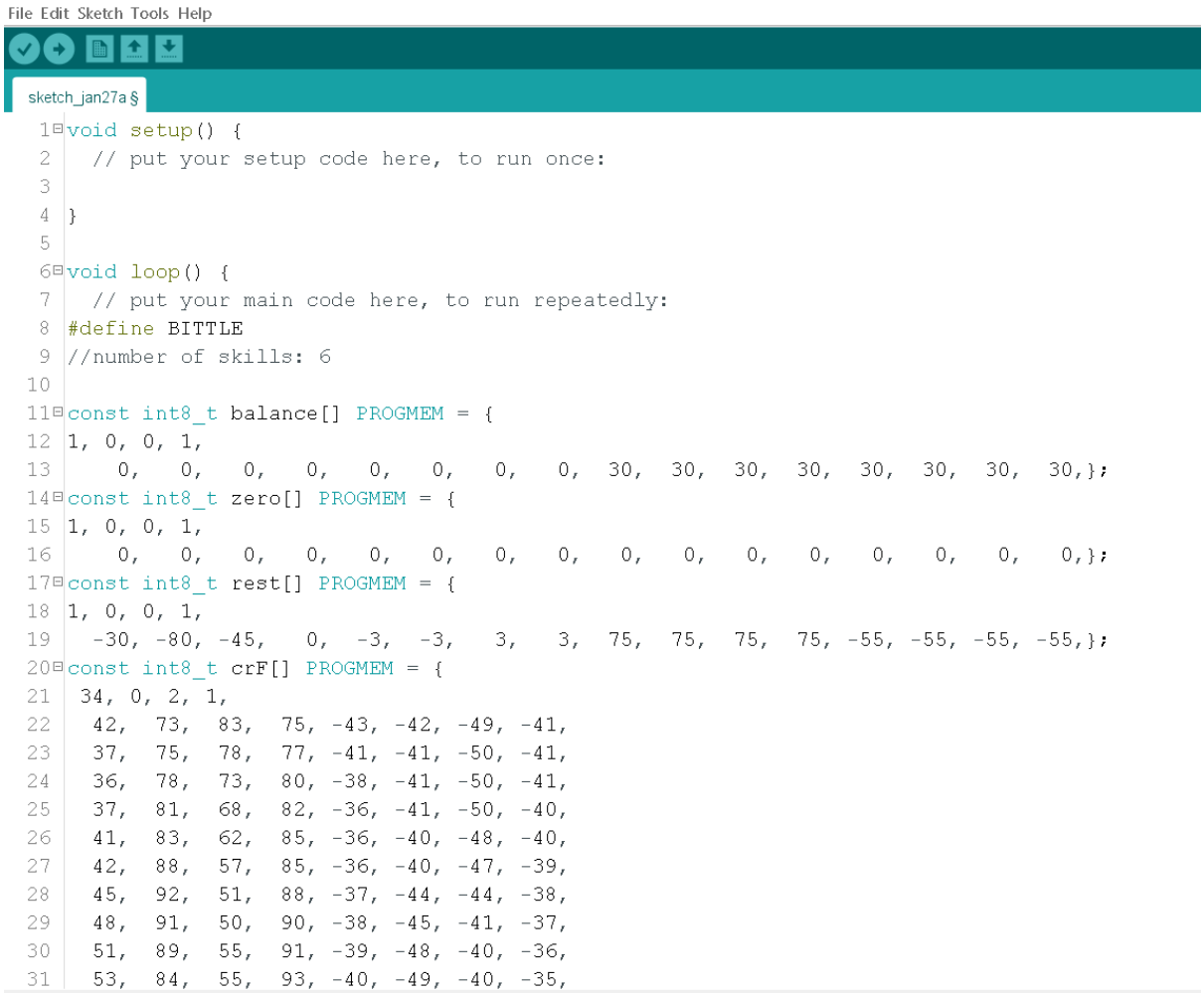
Petoi Bittle Manuals. (n.d.). Retrieved from Petoi: https://bittle.petoi.com/

Petoi Doc Center. (n.d.). Retrieved from Petoi: https://docs.petoi.com/

Teach Bittle new skills. (n.d.). Retrieved from Petoi: https://bittle.petoi.com/9-teach-bittle-new-skills

Appendix:

Appendix 1: The way to paste the program in InstinctBittle file in void loop() section of Arduino IDE.




```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:
#define BITTLE
//number of skills: 6

const int8_t balance[] PROGMEM = {
1, 0, 0, 1,
    0,   0,   0,   0,   0,   0,   0,   0,  30,  30,  30,  30,  30,  30,  30,  30,};
const int8_t zero[] PROGMEM = {
1, 0, 0, 1,
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,};
const int8_t rest[] PROGMEM = {
1, 0, 0, 1,
  -30, -80, -45,   0,  -3,  -3,   3,   3,  75,  75,  75,  75, -55, -55, -55, -55,};
const int8_t crF[] PROGMEM = {
 34, 0, 2, 1,
  42,  73,  83,  75, -43, -42, -49, -41,
  37,  75,  78,  77, -41, -41, -50, -41,
  36,  78,  73,  80, -38, -41, -50, -41,
  37,  81,  68,  82, -36, -41, -50, -40,
  41,  83,  62,  85, -36, -40, -48, -40,
  42,  88,  57,  85, -36, -40, -47, -39,
  45,  92,  51,  88, -37, -44, -44, -38,
  48,  91,  50,  90, -38, -45, -41, -37,
  51,  89,  55,  91, -39, -48, -40, -36,
  53,  84,  55,  93, -40, -49, -40, -35,
```

```
  56,  80,  58,  99, -40, -50, -41, -37,
  59,  75,  61, 101, -41, -50, -41, -40,
  62,  69,  64, 101, -41, -50, -41, -42,
  64,  64,  65,  99, -41, -49, -41, -44,
  67,  58,  68,  95, -42, -48, -42, -46,
  70,  53,  71,  92, -42, -47, -42, -47,
  73,  48,  74,  88, -42, -45, -41, -48,
  74,  42,  75,  83, -41, -43, -41, -49,
  77,  37,  78,  78, -41, -41, -41, -50,
  80,  36,  81,  73, -41, -38, -41, -50,
  82,  37,  83,  68, -40, -36, -40, -50,
  85,  41,  85,  62, -40, -36, -40, -48,
  89,  42,  87,  57, -41, -36, -39, -47,
  93,  45,  89,  51, -45, -37, -38, -44,
  92,  48,  91,  50, -47, -38, -37, -41,
  88,  51,  93,  55, -48, -39, -36, -40,
  83,  53,  96,  55, -49, -40, -35, -40,
  78,  56, 100,  58, -50, -40, -38, -41,
  73,  59, 102,  61, -50, -41, -42, -41,
  68,  62,  99,  64, -50, -41, -43, -41,
  62,  64,  96,  65, -48, -41, -45, -41,
  57,  67,  93,  68, -47, -42, -47, -42,
  51,  70,  89,  71, -46, -42, -48, -42,
  46,  73,  84,  74, -45, -42, -49, -41,
};
//const int8_t hlw[] PROGMEM = {
//14, 0, 0, 1,
  //10,  29,  51,  46,  33,  30,   4,  15,
  //10,  32,  43,  48,  40,  29,   3,  16,
  //15,  35,  34,  49,  35,  30,  13,  18,
  //19,  44,  36,  52,  33,  17,  14,  19,
  //22,  38,  39,  53,  31,  16,  14,  22,
  //25,  30,  42,  61,  30,  17,  14,  11,
  //28,  21,  44,  56,  30,  21,  15,   6,
  //31,   8,  47,  49,  29,  40,  16,   3,
  //33,  13,  48,  41,  30,  37,  17,   3,
  //38,  16,  51,  33,  28,  34,  18,  17,
  //42,  20,  52,  37,  16,  32,  20,  14,
  //35,  23,  59,  40,  16,  31,  16,  14,
  //26,  26,  59,  42,  19,  30,   8,  14,
  //15,  28,  52,  45,  27,  30,   5,  15,
//};
const int8_t jump[] PROGMEM = {
-5, 0, 0, 1,
 0, 4, 3,
-30, -80, -45, 0, -3, -3, 3, 3, 70, 70, 70, 70, -55, -55, -55, -55,  8,  0, 0, 0,
-30, -80, -46, 0, -4, -4, 2, 2, 36, 36,  3,  3, -50, -46,  12,  12,  48, 2, 0, 0,
-30, -80, -46, 0, -4, -4, 2, 2, 24, 24, 20, 20,   9,  13,  69,  69,  0,  0, 0, 0,
  0,   0,   0, 0,  0,  0, 0, 0, 22, 22, 46, 46,  -5,  -5, -19, -19,  48, 0, 0, 0,
  0,   0,   0, 0,  0,  0, 0, 0, 30, 30, 30, 30,  30,  30,  30,  30,   4, 0, 0, 0,
};
const int8_t pu[] PROGMEM = {
-10, 0, 0, 1,
 7, 8, 3,
    0, 0, 0, 0, 0, 0, 0, 0,  30,  30,  30,  30,  30,  30,  30,  30,  8, 0, 0, 0,
   15, 0, 0, 0, 0, 0, 0, 0,  30,  35,  40,  21,  50,  15,  15,  41, 12, 0, 0, 0,
   15, 0, 0, 0, 0, 0, 0, 0,  30,  35,  40,  30,  50,  15,  15,  14, 16, 0, 0, 0,
   30, 0, 0, 0, 0, 0, 0, 0,  27,  35,  40,  60,  50,  15,  20,  45, 16, 0, 0, 0,
   15, 0, 0, 0, 0, 0, 0, 0,  42,  35,  40,  60,  25,  20,  20,  60, 12, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0,  48,  45,  75,  60,  20,  37,  20,  60, 12, 0, 0, 0,
```

```
  -15,  0,  0,  0,  0,  0,  0,  0,  60,  60,  70,  70,  15,  15,  60,  60, 16, 0, 0, 0,
    0,  0,  0,  0,  0,  0,  0,  0,  30,  30, 110, 110,  60,  60,  60,  60,  8, 1, 0, 0,
   30,  0,  0,  0,  0,  0,  0,  0,  75,  70,  85,  85, -50, -50,  60,  60, 16, 0, 0, 0,
    0,  0,  0,  0,  0,  0,  0,  0,  30,  30,  30,  30,  30,  30,  30,  30,  8, 0, 0, 0,
};

#if !defined(MAIN_SKETCH) || !defined(I2C_EEPROM)

const char* skillNameWithType[]={"balanceI","zeroN","restI", "crFI","jumpI","puI",};

const int8_t* progmemPointer[] = {balance, zero, rest, crF, hlw, pu,};

#else

const int8_t* progmemPointer[] = {zero,};

#endif

}
```

Appendix 2: Picture showing the command name and the full name of the Instincts and the way to name or serial command the newbility.

	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）

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

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

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