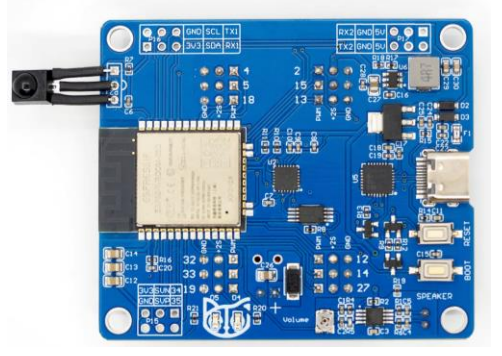
OpenCatEsp32
Code Walkthrough
for the
Bittle with BiBoard (ESP32 CPU)

v1.06

by este este
(https://github.com/este-este)

Source code from https://github.com/PetoiCamp/OpenCatEsp32
original walkthrough based on repo commit "fdae2e19" 2024-03-14
updates based on repo commit "1a008994" 2024-05-17



Disclaimer

This document is based on source code as it appeared in the specific GitHub repo commit(s) indicated on the title page. As such, it may not apply to future iterations of the code[1].

It was prepared for my own edification but is shared in the hopes that it may also benefit others who enjoy this Bittle robot.

I am not affiliated with Petoi LLC, but I am an enthusiastic fan and supporter of their work. All opinions, observations and assertions herein are my own. Any mistakes are also my own, for which I apologize in advance. Special thanks go to the Petoi team for insightful software discussions along this "path" to understanding!

Best regards,

este este

1. At minimum, the line numbers shown here may be different! 😊

Outline

- Background Information
- My Setup
- My Observations
- Code Walkthrough Plan
- Code Walkthrough
  - "Sketch" Process Map
  - OpenCatEsp32.ino (Part 1)
  - OpenCat.h
  - OpenCatEsp32.ino (Part 2)
  - io.h
  - reaction.h (Part 1)
  - taskQueue.h
  - moduleManager.h
  - reaction.h (Part 2)
  - skill.h

Background Information: Directives

- C++ compiler:
  - The compiler produces a binary form of the source code that can be understood by the CPU.
- C++ preprocessor:
  - The preprocessor acts before the compiler to transform the source code into a form that the compiler can process.
  - The preprocessor looks for ***directives*** which are statements, starting with "#", that do the following:
    - An inclusion directive (#include) references a C++ header (.h) file and causes the preprocessor to replace that directive with the contents of that header file.
    - A macro directive (#define) names a code block and causes the preprocessor to replace that name with the contents of that macro wherever the macro is used.
    - A conditional directive (e.g. #if, #if defined, #ifdef) allows a code block, including subordinate directives, to be processed & compiled only under specific conditions.
      - Any code or directive that is always processed is considered "unconditional".

See https://gcc.gnu.org/onlinedocs/cpp/ for more info about C++ preprocessors.

Background Information: Directives (cont.)

- Inclusion directive usage is simple.
  - They introduce code held elsewhere in a header file.
  - Example: "#include "src/OpenCat.h"
- Macro directive usage is more complex.
  - Can indicate the presence of an attribute.
    - Example: "#define BITTLE"
    - This type is used by conditional directives.

  OR
  - Can define a constant (could be a simple type or could be code).
    - Example: "#define WALKING_DOF 8"
    - This type is used in logic, calculations, and information display.
- Conditional directive usage is the most complex.
  - They sort of "write" code, based on macro directives.
    - Example: #if defined BITTLE
      < Bittle specific code that only complies if the BITTLE macro is defined >
      #endif
    - This type makes the code super flexible but, inherently, harder to read and understand!

Conditional directives are a type of "guard" clause. They enable or disable specific code compilation based on certain conditions.

Background Information: Skills

- Skills: A Posture, Gait or Behavior that the robot can perform.
  - Posture: a stationary robot pose where each servo is in a specific position (angle) that comprises the pose.
  - Gait: a locomotion where the robot moves through a sequence of poses.
  - Behavior: an action where the robot moves through a sequence of poses.
- Frame: A collection the joint angles for a specific pose.
  - Gaits and Behaviors use a set of Frames to compose robot motion, like video consists of a series of images.
    - Gaits loop through all Frames continuously.
    - Behaviors loop only through a subset of Frames and only for a specified number of times.
      - Behaviors therefore require more info than Postures or Gaits.
- Skills are represented in code[1] as integer (int8_t) arrays[2]
  - The array data is divided into Skill Info and Frame Info.

1. See https://docs.petoi.com/applications/skill-creation#understand-the-code.
2. Found in an Instinct header file (e.g. InstinctBittleESP.h)

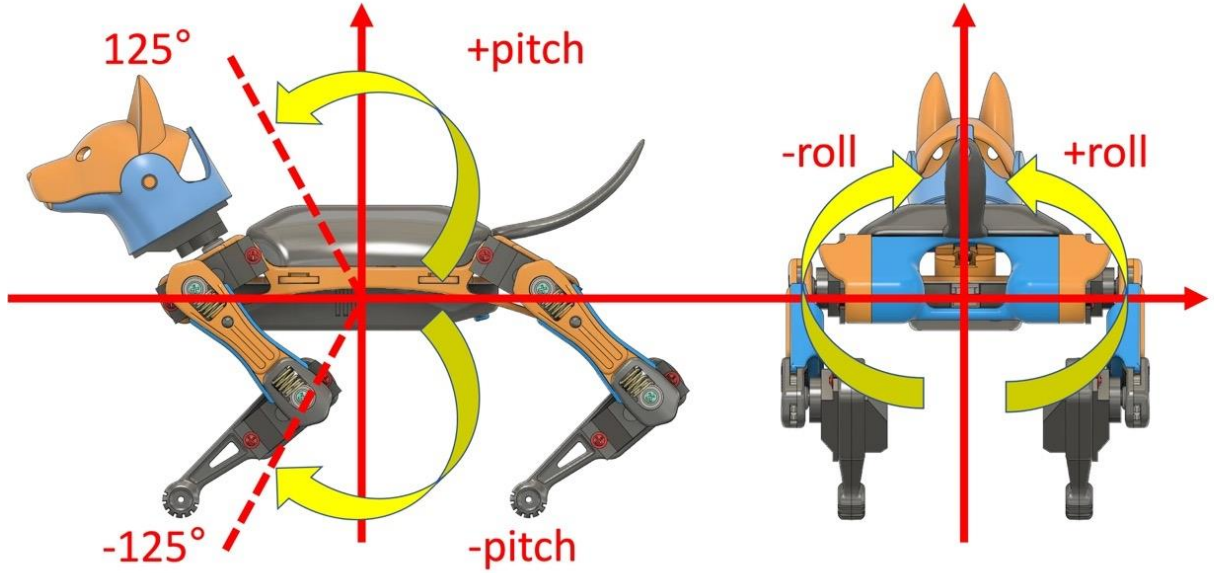
Background Information: Skill Array Data Structure

- Skill Info Elements
  - All Skills have Total # of Frames, Expected Body **Roll** & **Pitch** and Angle Ratio as the first 4 elements (in red below)
  - Behaviors have 3 extra Skill Info elements AND their first element has a minus sign (-)
    - Behavior Skill Info = **Start** Frame # & **End** Frame # plus # of Loops (in blue below)
- Frame Info Elements
  - Postures and Behaviors have 16 Indexed Joint Angle elements (in green below)
  - Behaviors have 4 extra Frame Info elements
    - Behavior Frame Info = Speed Factor, Delay Time, Trigger Axis, and Trigger Angle (in purple below)
  - Gaits use only the upper 8 Indexed Joint Angle elements (in yellow below)

	Skill Info							Frame Info																			
	Total # of Frames	Expected Body Orientation		Angle Ratio	Behavior Skill Info			Indexed Joint Angles																Behavior Frame Info			
		Roll	Pitch		Loop Start Frame #	Loop End Frame #	# Of Loops	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Speed Factor	Delay Time	Trigger Axis	Trigger Angle
Posture "sit" Array Data Structure ➔	1	0	-30	1				0	0	-45	0	-5	-5	20	20	45	45	105	105	45	45	-45	-45				
Gait "crF" Array Data Structure ➔	67	0	2	1												42	73	83	75	-43	-42	-49	-41				
Behavior "pu" Array Data Structure ➔	-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

Background Information: Skill Array Data Structure (cont.)

- Expected Body **Roll** & **Pitch**
  - Used by the Gyro to adjust leg servos to keep the body tilt near the expected values.

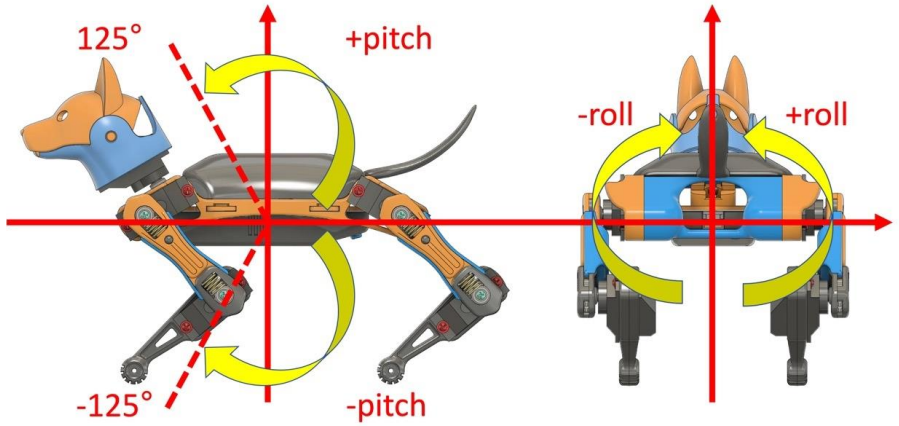


- Angle Ratio
  - A multiplier for the joint servo angles
    - Necessary because Frame joint servo angles must be int values in the range -125 to 125

	Skill Info							Frame Info																			
	Total # of Frames	Expected Body Orientation		Angle Ratio	Behavior Skill Info			Indexed Joint Angles																Behavior Frame Info			
		Roll	Pitch		Loop Start Frame #	Loop End Frame #	# Of Loops	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Speed Factor	Delay Time	Trigger Axis	Trigger Angle
Posture "sit" Array Data Structure ➔	1	0	-30	1				0	0	-45	0	-5	-5	20	20	45	45	105	105	45	45	-45	-45				
Gait "crF" Array Data Structure ➔	67	0	2	1												42	73	83	75	-43	-42	-49	-41				
Behavior "pu" Array Data Structure ➔	-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

Background Information: Skill Array Data Structure (cont.)

- Speed Factor (in deg per step; default = 4)
  - How fast the servos will move (slow to fast = 1 to 125).
- Delay Time (in 50 millisecond increments; default = 0)
  - How long to wait before the next frame (none to long = 0 to 125).
- Trigger Axis
  - Sets body rotation direction when to trigger the next frame
    - 0 = no trigger axis
    - 1 = positive pitch, -1 = negative pitch
    - 2 = positive roll, -2 = negative roll
- Trigger Angle
  - Angle that must be achieved to trigger the next frame (Range = -125 to 125)



	Skill Info							Frame Info																			
	Total # of Frames	Expected Body Orientation		Angle Ratio	Behavior Skill Info			Indexed Joint Angles																Behavior Frame Info			
		Roll	Pitch		Loop Start Frame #	Loop End Frame #	# Of Loops	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Speed Factor	Delay Time	Trigger Axis	Trigger Angle
Posture "sit" Array Data Structure ➔	1	0	-30	1				0	0	-45	0	-5	-5	20	20	45	45	105	105	45	45	-45	-45				
Gait "crF" Array Data Structure ➔	67	0	2	1												42	73	83	75	-43	-42	-49	-41				
Behavior "pu" Array Data Structure ➔	-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

Background Information: Skill Code Examples

- Posture "sit"
- Gait "crF"

For these Gait and Behavior examples, only first 2 and last 2 Frames are shown.

- Behavior "pu"

```
1194  const int8_t sit[] PROGMEM = {
1195  1, 0, -30, 1,
1196  0, 0, -45, 0, -5, -5, 20, 20, 45, 45, 105, 105, 45, 45, -45, -45,};
```

```
141   const int8_t crF[] PROGMEM = {
142   67, 0, 2, 1,
143   42, 73, 83, 75, -43, -42, -49, -41,
144   40, 74, 80, 75, -43, -41, -50, -41,
                    :
208   48, 71, 88, 73, -45, -42, -48, -42,
209   46, 73, 84, 74, -45, -42, -49, -41,
210   };
```

```
1498  const int8_t pu[] PROGMEM = {
1499  -10, 0, 0, 1,
1500  7, 8, 3,
1501  0, 0, 0, 0, 0, 0, 0, 0, 30, 30, 30, 30, 30, 30, 30, 30, 8, 0, 0, 0,
1502  15, 0, 0, 0, 0, 0, 0, 0, 30, 35, 40, 21, 50, 15, 15, 41, 12, 0, 0, 0,
                                                    :
1509  30, 0, 0, 0, 0, 0, 0, 0, 70, 70, 85, 85, -50, -50, 60, 60, 16, 0, 0, 0,
1510  0, 0, 0, 0, 0, 0, 0, 0, 30, 30, 30, 30, 30, 30, 30, 30, 8, 0, 0, 0,
1511  };
```

	Skill Info							Frame Info																			
	Total # of Frames	Expected Body Orientation		Angle Ratio	Behavior Skill Info			Indexed Joint Angles																Behavior Frame Info			
		Roll	Pitch		Loop Start Frame #	Loop End Frame #	# Of Loops	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Speed Factor	Delay Time	Trigger Axis	Trigger Angle
Posture "sit" Array Data Structure ➔	1	0	-30	1				0	0	-45	0	-5	-5	20	20	45	45	105	105	45	45	-45	-45				
Gait "crF" Array Data Structure ➔	67	0	2	1												42	73	83	75	-43	-42	-49	-41				
Behavior "pu" Array Data Structure ➔	-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

Background Information: Skills[1] in InstinctBittleESP.h

- 62 Skills currently (11 Postures, 19 Gaits, 32 Behaviors)
  - Mirrorable skills must end in 'L' (two skills with one definition - saves memory!)

• Postures

Skill	Description
balance	stand up neutral
buttUp	butt up
calib	calibration pose
dropped	dropped by back legs
lifted	lifted by neck
lnd	landing pose
rest	rest
sit	sit
str	stretch
up	stand up neutral (= balance)
zero	set all joints to 0 degrees

• Gaits

Skill	Description
bdF	bound Forward
bk	backward
bkL	backward Left
crF	crawl Forward
crL	crawl Left
gpF	gap Forward
gpL	gap Left
hlw	halloween gait
jpF	jump Forward

Skill	Description
lftF	low foot Forward
lftL	low foot Left
phF	push Forward
phL	push Left
trF	trot Forward
trL	trot Left
vtF	step at origin
vtL	spin Left
wkF	walk Forward
wkL	walk Left

Gaits in red highlight are defined for leftward locomotion. They can be mirrored to rightward locomotion by changing the skill last character 'L' to 'R'.

Changing the skill last character 'L' to 'X' will cause random selection of the corresponding leftward or rightward locomotion.

• Behaviors

Skill	Description
ang	angry
bf	backflip
bx	boxing
chr	cheers
ck	check
cmh	come here
dg	dig
ff	front flip
fiv	high five
gdb	good boy
hds	handstand
hg	hug
hi	hi
hsk	handshake
hu	hands up
jmp	jump

Skill	Description
kc	kick
lpov	leap over
mw	moon walk
nd	nod
pd	play dead
pee	pee
pu	push ups
pu1	push ups with one hand
rc	recover (returns to standing)
rl	roll
scrh	scratch
snf	sniff
tbl	be a table
ts	test
wh	wave head
zz	all joints to 0 degrees

1. See this link about tokens and skills in https://docs.petoi.com/apis/serial-protocol

My Setup

- I use Visual Studio 2019 with Visual Micro (https://www.visualmicro.com/) as my IDE (Integrated Development Environment).
  - This is just my preference. The code is designed for the Arduino IDE and runs just fine there.
- All code is displayed as screenshots from Visual Studio 2019.
  - The VS2019 code formatting (at about 110% magnification) and line numbers facilitate the walkthrough[1].

1. Though not so useful when updates to preceding code *changes* those line numbers… 😢

My Observations

- The source code's purpose is to...
  - provide mechanisms for the robot to perceive itself and its surroundings
    - Inward perception is achieved via...
      - inputs from the IMU (Inertial Measurement Unit) module to "know" its body orientation,
        and/or
        inputs via servo feedback (future servos may have that capability) to "know" its joint positions.
    - Outward perception is achieved via...
      - inputs from sensors and from communication channels,
        - that arise from *passive* or *active* interactions[1].

  AND THEN

  - respond to those perceptions.
    - Responses are outputs involving changes in movement, sound and light.

1. *Passive* vs. *active* refers to robot interactions with stationary vs moving/animate objects in the environment. The term is from the robot's perspective.

My Observations (cont.)

- Commands connect perceptions to responses.
  - For communication channel[1] inputs, the perceptions "are" the commands
    - Used for machine-robot[2] interaction.
  - For all other inputs, the perceptions are interpreted into commands.
    - Used for *passive* and *active* sensor interactions.
- A command[3] uses…
  - single character tokens that categorizes and initiates the desired command

  plus
  - token parameters that provide information the command requires.
    - *Parameters for capital letter tokens immediately follow their token (no space separator).*
    - *Parameters for most lower-case tokens, except 'k', can have a space separator following their token.*

1. USB & Bluetooth serial communication.
2. IMO, use of communication channel inputs is always machine-robot interaction where "machine" is e.g. your laptop or cell phone. True human-robot interaction is via sensors (e.g. the touch sensor).
3. The source code stores the token parameters in the "newCmd" variable. It can hold the skill name (for token 'k') and skill data (initially, in serial buffer format, then reformatted to skill array format to save memory).

My Observations (cont.)

- The source code has important functions to support its' purpose.
  - readEnvironment()
    - Currently used for inward perception.
      - E.g. robot senses it has flipped over.
    - Could be expanded for *passive* or *active* outward perceptions.
      - E.g. robot senses an object (stationary or moving/animate) that it is about to walk into.
  - readSignal()
    - Currently used for *active* outward perception.
      - E.g. robot receives a command on a communication channel.
      - E.g. robot receives touch, voice or IR sensor data from a moving/animate object (such as a person tapping a touch sensor, speaking to a voice sensor or pressing buttons on an IR remote).
  - reaction()
    - Provides a response to perceptions from the above "read" functions.
      - If the perception is via a communication channel, the code processes the command that was input.
      - If the perception is via sensors, the input(s) must be interpreted into command(s) before processing.

My Observations (cont.)

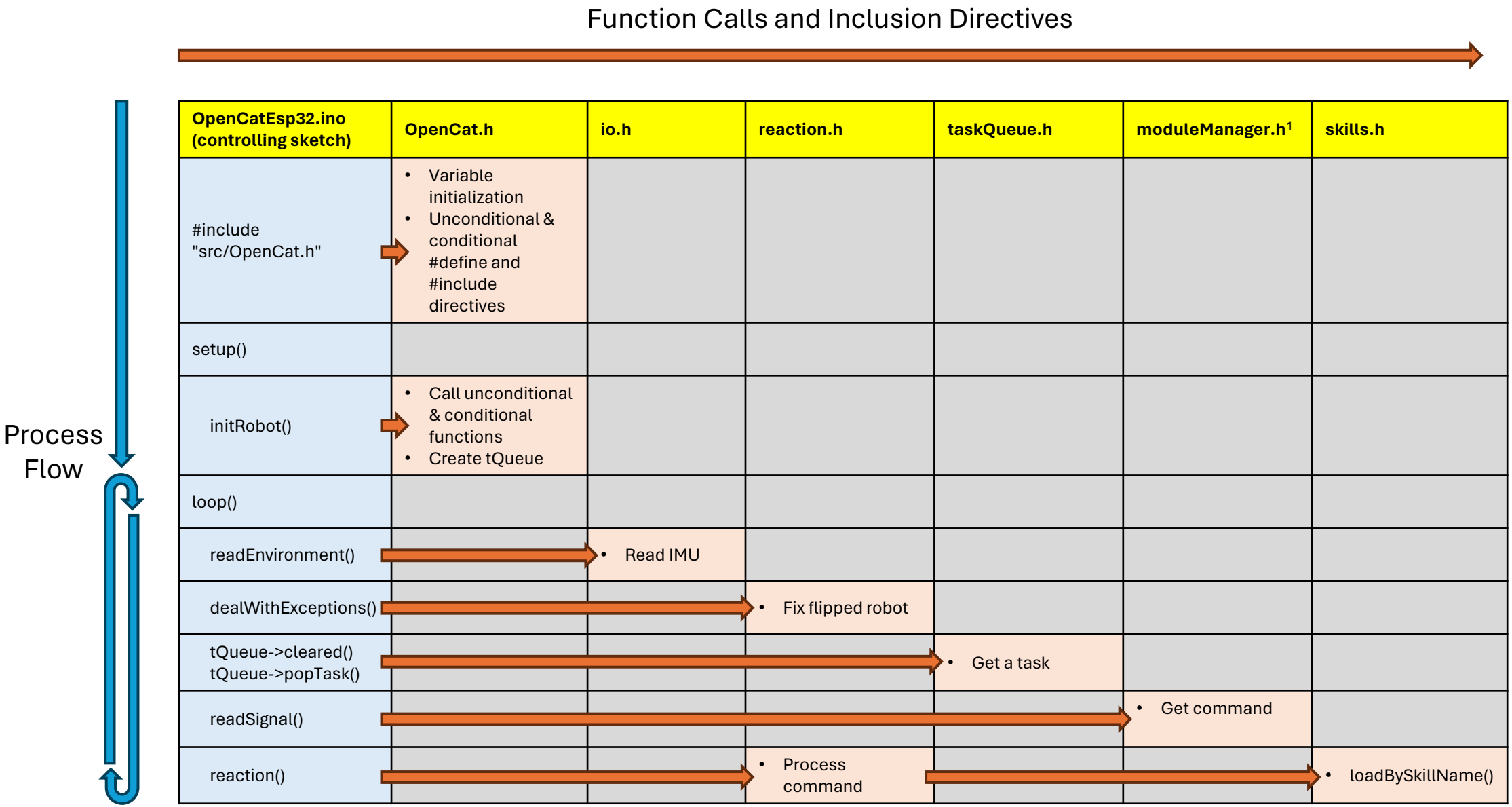
- In this source code, the C++ preprocessor conditional directives, when nested, are unfortunately not indented.
  - This lack of indentation makes following the logic of such nested blocks more difficult.
    - Note: The "fold" capability of VS2019 helps somewhat to determine the start and end of nested blocks, so this walkthrough will attempt to cover such nested blocks in segments.
  - There are historical reasons for this lack of indentation (see, for example, https://stackoverflow.com/questions/789073/indenting-defines) but those no longer apply to current C++ preprocessors.
    - I know of no way to automatically indent such nested blocks so if some energetic person wants to do that formatting, it would be greatly appreciated! 😊

Code Walkthrough Plan

- In code walkthrough slides:
  - Notes will be given at the top of the slide and interspersed with the code lines that are below.
  - The code walkthrough will be kept at a higher level
    - Lower-level code and less important lines will sometimes be omitted for brevity.
- The focus will be on the Bittle robot model using the BiBoard (ESP32 CPU).
- We will take a top-down, "sketch process", approach
  - This means we will the follow the thread of the code controlled by the sketch "OpenCat32.ino", from file to file and function to function as needed.
- A visual high-level "Sketch Process Map" will be used.

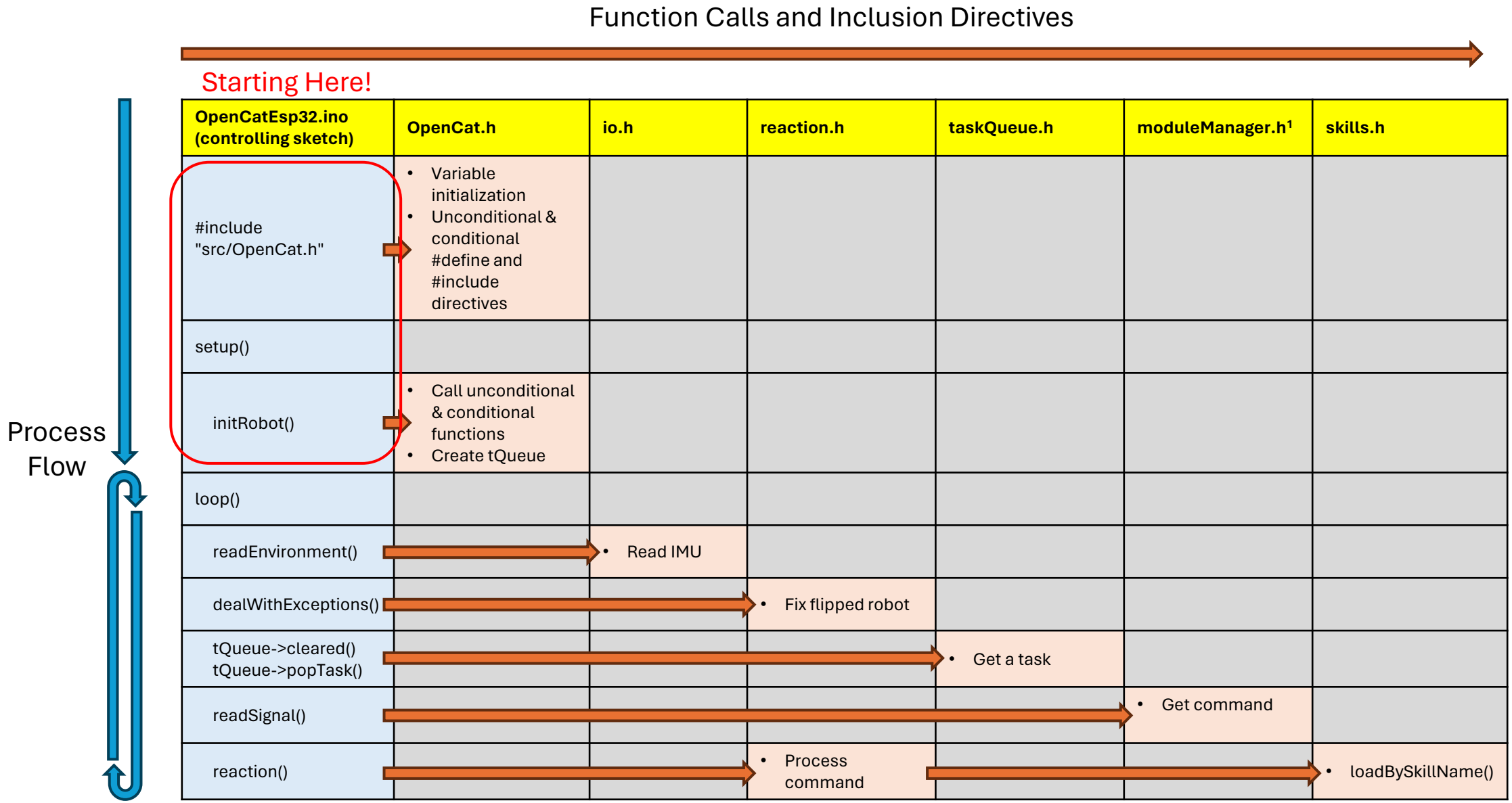
Code Walkthrough Begins!

"Sketch" Process Map



1. sense.h was replaced by moduleManager.h on 2024-04-19

"Sketch" Process Map



1. sense.h was replaced by moduleManager.h on 2024-04-19

OpenCatEsp32.ino: Model, Board & Software "Defines"[1]

- The enabled #define macro directives (in purple colored font below)…
  - set the model [BITTLE], board [BiBoard version] and software options.
- *Important note: If you have the BiBoard v0_1, disable "#define BiBoard_V0_2" and enable "#define BiBoard_V0_1".*

```
1   //·modify·the·model·and·board·definitions
2   //************************
3   #define·BITTLE··//·Petoi·9·DOF·robot·dog:·1·on·head·+·8·on·leg
4   //·#define·NYBBLE·//·Petoi·11·DOF·robot·cat:·2·on·head·+·1·on·tail·+·8·on·leg
5   //·#define·CUB
6
7    //·#define·BiBoard_V0_1··//ESP32·Board·with·12·channels·of·built-in·PWM·for·joints
8    #define·BiBoard_V0_2
9   //·#define·BiBoard_V1_0
10  //·#define·BiBoard2··//ESP32·Board·with·16·channels·of·PCA9685·PWM·for·joints
11  //************************
12
13  //·Send·'!'·token·to·reset·the·birthmark·in·the·EEPROM·so·that·the·robot·will·restart·to·reset
14  //·#define·AUTO_INIT··//activate·it·to·automatically·reset·joint·and·imu·calibration·without·prompts
15
16  //·you·can·also·activate·the·following·modes·(they·will·diable·the·gyro·to·save·programming·space)
17  //·allowed·combinations:·RANDOM_MIND·+·ULTRASONIC,·RANDOM_MIND,·ULTRASONIC,·VOICE,·CAMERA
18   #define·VOICE·····················//·Petoi·Grove·voice·module
19   #define·ULTRASONIC···············//·for·Petoi·RGB·ultrasonic·distance·sensor
20   #define·PIR·····················//·for·PIR·(Passive·Infrared)·sensor
21   #define·DOUBLE_TOUCH·············//·for·double·touch·sensor
22   #define·DOUBLE_LIGHT·············//·for·double·light·sensor
23   #define·DOUBLE_INFRARED_DISTANCE··//·for·double·distance·sensor
24   #define·GESTURE··················//·for·Gesture·module
25  #define·CAMERA····················//·for·Mu·Vision·camera
```

These are now enabled by default since access is now controlled in moduleManager.h, via the 'X' token at runtime, rather than via these macro directives at compile time.

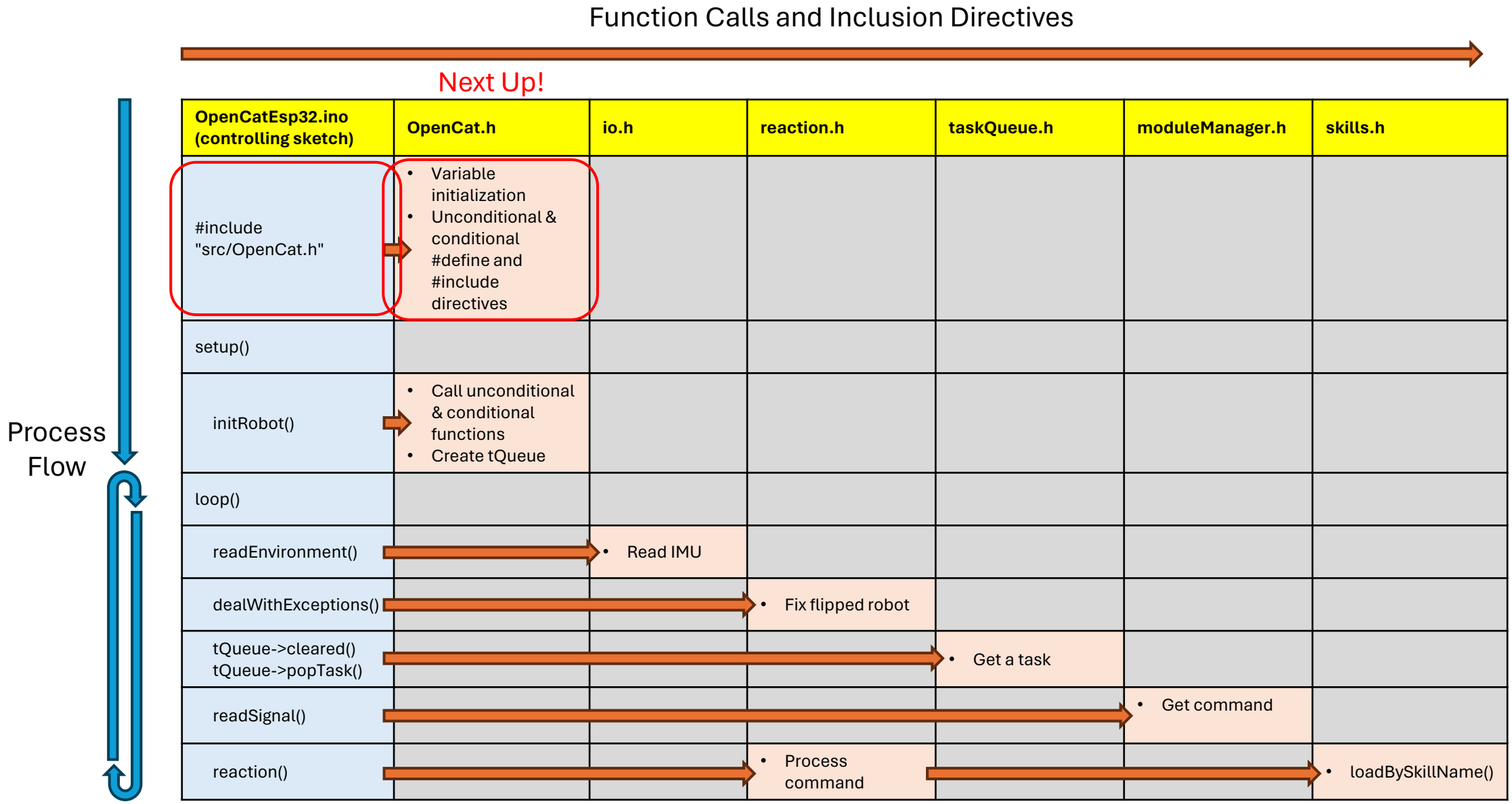
1. updated based on repo commit "1a008994" 2024-05-17

OpenCatEsp32.ino: Master include & setup() function

- The #include directive takes in the OpenCat.h header file.
  - This file is the master header file.
  - It has many conditional directives which control the defining of many macro directives and can also trigger many inclusion directives.
  - We will begin looking at the OpenCat.h header file after this slide.
- The setup() function…
  - is the Arduino function that is run once, to set up the software.
  - initializes the serial port and then clears the serial buffer.
  - calls the initRobot(), which we will see in the OpenCat.h header file.

```
35    #include "src/OpenCat.h"
36
37    void setup() {
38      // put your setup code here, to run once:
39      Serial.begin(115200);  // USB serial
40      Serial.setTimeout(SERIAL_TIMEOUT);
41      //  Serial1.begin(115200); //second serial port
42      while (Serial.available() && Serial.read())
43        ;  // empty buffer
44      initRobot();
45    }
```

"Sketch" Process Map



OpenCat.h: General Observations

- It is unique among OpenCat specific header files!
  - It is the first header file that is included.
  - As such, it provides the most initializations[1]
    - The largest number of macro directives in a single file (mostly in lines 65 thru 275).
    - The majority of global variable definitions (mostly in lines 278 thru 326).
    - The largest number of conditional directives (mostly in lines 328 thru 505).
- It includes initRobot() function which does the robot initialization upon start up.
  - **If you need to add custom initializations, this function is a place to do it!** 😊

1. line numbers here were updated based on repo commit "1a008994" 2024-05-17

OpenCat.h: BiBoard Documentation Comment

- This multiline comment documents aspects of the BiBoard, including the layout & connectivity.

```
/* BiBoard
   PWM:
                        |------------------------------
                        |    PWM[0]          PWM[6]    |
                        |    PWM[1]          PWM[7]    |
                        |    PWM[2]          PWM[8]    |
                        |-----------------------------|
                        |           |                 |
                        |   ESP32   |  IMU        USB-C|~~~~Tail~~~~
                        |           |                 |
                        |-----------------------------|
                        |    PWM[3]          PWM[9]    |
                        |    PWM[4]          PWM[10]   |
                        |    PWM[5]          PWM[11]   |
                        |------------------------------|

   Pin Name    | ESP32 Pin   | Arduino Pin Name    | Alternative Function
   PWM[0]          GPIO4              4                GPIO / Ain / Touch
   PWM[1]          GPIO5              5                GPIO / VSPI SS
   PWM[2]          GPIO18             18               GPIO / VSPI SCK
   ------------------------------------------------------------------------------
   PWM[3]          GPIO32             32               GPIO / Ain / Touch
   PWM[4]          GPIO33             33               GPIO / Ain / Touch
   PWM[5]          GPIO19             19               GPIO / VSPI MISO
   ------------------------------------------------------------------------------
   PWM[6]          GPIO2              2                boot pin, DO NOT PUT HIGH WHEN BOOT!
   PWM[7]          GPIO15             15               GPIO / HSPI SS / Ain Touch
   PWM[8]          GPIO13             13               built-in LED / GPIO / HSPI MOSI / Ain / Touch
   ------------------------------------------------------------------------------
   PWM[9]          GPIO12             12               GPIO / HSPI MISO / Ain / Touch
   PWM[10]         GPIO14             14               GPIO / HSPI SCK / Ain / Touch
   PWM[11]         GPIO27             27               GPIO / Ain / Touch

   I2C:

   Pin Name    | ESP32 Pin   | Arduino Pin Name    | Alternative Function
   I2C-SCL         GPIO22             22               Fixed - ICM20600 - Pulled
   I2C-SDA         GPIO21             21               Fixed = ICM20600 - Pulled

   System default, nothing to declaration!

   Other Peripherals:

   Pin Name    | ESP32 Pin   | Arduino Pin Name    | Alternative Function
   IR_Remote       GPIO23             23               Fixed - VS1838B IR
   DAC_Out         GPIO25             25               Fixed - PAM8302
   IMU_Int         GPIO26             26               Fixed - MPU6050 Interrupt

   System default, nothing to declare!
*/
```

```
/* BiBoard2
   IMU_Int     27
   BUZZER      14
   VOLTAGE     4
   RGB LED     15
   GREEN-LED   5
*/

/*  Total DOF            Walking DOF
                         Nybble    Bittle    Cub
   BiBoard  (12)  skip 0~4  skip 0~4    12
   BiBoard2 (16)  skip 0~8  skip 0~8  skip0~4
*/
```

OpenCat.h: Boards, Version, Birthmark, Toggles[1]

- Sets serial port parameters and defines some board parameters.
- Sets a version date.
- Sets the "BIRTHMARK" character (prevents automatic resetting).
- Enables Bluetooth and Gyro, i.e. the IMU (Inertial Measurement Unit) module.
- Set the servo frequency for PWM control

```
65    #define SERIAL_TIMEOUT 10  // 5 may cut off the message
66    #define SERIAL_TIMEOUT_LONG 150
67    #ifdef BiBoard_V0_1
68    #define BOARD "B01"
69    #elif defined BiBoard_V0_2
70    #define BOARD "B02"
71    #else
72    #define BOARD "B"
73    #endif
74    #define DATE "240511"  // YYMMDD
75    String SoftwareVersion = "";
76
77    #define BIRTHMARK 'x'  // Send '!' token to reset the birthmark in the EEPROM so that the robot will know to restart and reset
78
79    #define BT_BLE     // toggle Bluetooth Low Energy (BLE)
80    #define BT_SSP     // toggle Bluetooth Secure Simple Pairing (BT_SSP)
81    #define GYRO_PIN  // toggle the Inertia Measurement Unit (IMU), i.e. the gyroscope
82    #define SERVO_FREQ 240
```

1. updated based on repo commit "1a008994" 2024-05-17

OpenCat.h: Pins[1]

- Based on the BiBoard version, these defines…
  - Set the pin numbers, including those for servo pulse width modulation (PWM) control
    - Note the constant array "PWM_pin" holds those pin numbers for the servo PWM control

```
84  #if defined BiBoard_V0_1 || defined BiBoard_V0_2
85  #define ESP_PWM
86  #define PWM_NUM 12
87  #define INTERRUPT_PIN 26  // use pin 2 on Arduino Uno & most boards
88  #define BUZZER 25
89  #define IR_PIN 23
90  #define ANALOG1 34
91  #define ANALOG2 35
92  #define ANALOG3 36
93  #define ANALOG4 39
94  #define UART_RX2 16
95  #define UART_TX2 17
96
97  // L:Left-R:Right-F:Front-B:Back---LF, RF, RB, LB
98  const uint8_t PWM_pin[PWM_NUM] = {
99    19, 4, 2, 27,   // head or shoulder roll
100   33, 5, 15, 14,  // shoulder pitch
101   32, 18, 13, 12  // knee
102 };
```

1. updated based on repo commit "1a008994" 2024-05-17

OpenCat.h: rate, DOFs, ServoModel_t enum

- Sets the sample rate used in doubleLight.h and doubleInfraredDistance.h
- MAX_READING and BASE_RANGE
  - For analog (inc. PWM) IO with 12-bit (4096 steps) and 10-bit (1024 steps) resolutions, respectively.
  - "rate" proves a scaling factor between these two different analog resolutions.
- Sets various maximum degrees of freedom (DOF = maximum possible servos)
- Based on the model (NYBBLE vs BITTLE vs CUB)
  - Sets number of walking servos (WALKING_DOF) and the number of servos in a "gait" array (GAIT_ARRAY_DOF)
    - Note: GAIT_ARRAY_DOF is not currently used.
- Creates an enumeration for the servo models

```
129     #define MAX_READING 4096.0
130     #define BASE_RANGE 1024.0
131     double rate = 1.0 * MAX_READING / BASE_RANGE;
132
133     #define DOF 16
134   #if defined NYBBLE || defined BITTLE
135     #define WALKING_DOF 8
136     #define GAIT_ARRAY_DOF 8
137   #else  // CUB
138     #define WALKING_DOF 12
139     #define GAIT_ARRAY_DOF 8
140     #endif
141
142   enum ServoModel_t {
143     G41 = 0,
144     P1S,
145     P2K
146     };
```

OpenCat.h: Joints, servoModeList[]

- Conditionally sets joint names for current robot model then includes the appropriate instinct header file. Note: Bittle does not have NECK TILT or TAIL joints.
- Sets the servoModeList[] array with ServoModel_t enum values specified with the joint names.
- Sets bool variable "newBoard" to false so the new (uncalibrated) board setup (found in I2cEEPROM.h) will not run unless the BIRTHMARK has been cleared.
  - See code line "newBoard = newBoardQ(EEPROM_BIRTHMARK_ADDRESS);" in i2cEepromSetup()
  - i2cEepromSetup() is called in the initRobot() function (see later slides).

```
148   //·Tutorial:·https://bittle.petoi.com/11-tutorial-on-creating-new-skills
149   #ifdef·NYBBLE
150   #define·MODEL·"Nybble"
151   #define·HEAD
152   #define·TAIL
153   #define·X_LEG
154   #define·REGULAR·P1S··//G41
155   #define·KNEE·P1S·····//G41
156   #include·"InstinctNybbleESP.h"
157
158   #elif·defined·BITTLE
159   #define·MODEL·"Bittle"
160   #define·HEAD
161   #define·LL_LEG
162   #define·REGULAR·P1S
163   #define·KNEE·P1S
164   #include·"InstinctBittleESP.h"
165
```

Model dependent LEG joint style (obviously different when in calib pose)

The LEG joint is a shoulder/hip joint since it connects the body to the upper leg

```
165
166   #elif·defined·CUB
167   #define·MODEL·"DoF16"
168   #ifdef·BiBoard2
169   #define·HEAD
170   #define·TAIL
171   #endif
172   #define·LL_LEG
173   #define·REGULAR·P1S
174   #define·KNEE·P2K
175   #include·"InstinctCubESP.h"
176   //·#define·MPU_YAW180
177   #endif
178
179   ServoModel_t·servoModelList[]·=·{
180   ··REGULAR,·REGULAR,·REGULAR,·REGULAR,
181   ··REGULAR,·REGULAR,·REGULAR,·REGULAR,
182   ··REGULAR,·REGULAR,·REGULAR,·REGULAR,
183   ··KNEE,·KNEE,·KNEE,·KNEE
184   };
185
186   bool·newBoard·=·false;
```

calib pose:
Nybble vs. Bittle

OpenCat.h: math library, token list[1]

- Begin the defining of token names and values
  - The token values are what you can send, via communication channels, for initiating commands to the robot.
  - See https://docs.petoi.com/apis/serial-protocol for more information.
- Token "T_Skill" (value is 'k') requires Skill Names[2] found in Instinct<Model>ESP.h header files
  - E.g. for Model = Bittle, Skill Names are found in InstinctBittleESP.h

```
#include <math.h>
// token list
#define T_ABORT 'a'      //abort the calibration values
#define T_BEEP 'b'       //b note1 duration1 note2 duration2 ... e.g. b12 8 14 8 16 8 17 8 19 4 \
                         //bVolume will change the volume of the sound, in scale of 0~10. 0 will mute all sound effect. e.g. b3. \
                         //a single 'b' will toggle all sound on/off
#define T_BEEP_BIN 'B'   //B note1 duration1 note2 duration2 ... e.g. B12 8 14 8 16 8 17 8 19 4 \
                         //a single 'B' will toggle all sound on/off
#define T_CALIBRATE 'c'  //send the robot to calibration posture for attaching legs and fine-tuning the joint offsets. \
                         //c jointIndex1 offset1 jointIndex2 offset2 ... e.g. c0 7 1 -4 2 3 8 5
#define T_COLOR 'C'      //change the eye colors of the RGB ultrasonic sensor \
                         //a single 'C' will cancel the manual eye colors
#define T_REST 'd'

#define T_SERVO_FEEDBACK 'f'      //return the servo's position info if the chip supports feedback. \
                                  //e.g. f8 returns the 8th joint's position. A single 'f' returns all the joints' position
#define T_SERVO_FOLLOW 'F'        //make the other legs follow the moved legs
#define T_GYRO_FINENESS 'g'       //adjust the finess of gyroscope adjustment to accelerate motion
#define T_GYRO_BALANCE 'G'        //toggle on/off the gyro adjustment
#define T_INDEXED_SIMULTANEOUS_ASC 'i'  //i jointIndex1 jointAngle1 jointIndex2 jointAngle2 ... e.g. i0 70 8 -20 9 -20 \
                                        //a single 'i' will free the head joints if it were previously manually controlled.
#define T_INDEXED_SIMULTANEOUS_BIN 'I'  // I jointIndex1 jointAngle1 jointIndex2 jointAngle2 ... e.g. I0 70 8 -20 9 -20
#define T_JOINTS 'j'              // A single "j" returns all angles. "j Index" prints the joint's angle. e.g. "j 8" or "j11".
#define T_SKILL 'k'
```

1. updated based on repo commit "1a008994" 2024-05-17

2. There must be no space between the 'k' and the Skill Name (e.g. "ksit")

OpenCat.h: token list (cont.)[1]

- Complete the defining of token names and values.

```
#define T_SKILL_DATA 'K'
#define T_SLOPE 'l'                    // inverse the slope of the adjustment function
#define T_LISTED_BIN 'L'               // a list of the DOFx joint angles: angle0 angle1 angle2 ... angle15
#define T_INDEXED_SEQUENTIAL_ASC 'm'   // m jointIndex1 jointAngle1 jointIndex2 jointAngle2 ... e.g. m0 70 0 -70 8 -20 9 -20
#define T_INDEXED_SEQUENTIAL_BIN 'M'   // M jointIndex1 jointAngle1 jointIndex2 jointAngle2 ... e.g. M0 70 0 -70 8 -20 9 -20
#define T_NAME 'n'                     // customize the Bluetooth device's broadcast name. e.g. nMyDog will name the device as "MyDog" \
                                       // it takes effect the next time the board boosup. it won't interrupt the current connecton.
#define T_MELODY 'o'
#define T_PAUSE 'p'
#define T_TASK_QUEUE 'q'
#define T_SAVE 's'
#define T_TILT 't'
#define T_TEMP 'T'   // call the last skill data received from the serial port
#define T_MEOW 'u'
#define T_PRINT_GYRO 'v'               // print Gyro data once
#define T_VERBOSELY_PRINT_GYRO 'V'     // toggle verbosely print Gyro data
#define T_SERVO_MICROSECOND 'w'        // PWM width modulation
#define T_XLEG 'x'
#define T_RANDOM_MIND 'z'   // toggle random behaviors

#define T_READ 'R'          // read pin     R
#define T_WRITE 'W'         // write pin                    W
#define TYPE_ANALOG 'a'     //            Ra(analog read)   Wa(analog write)
#define TYPE_DIGITAL 'd'    //            Rd(digital read)  Wd(digital write)

#define T_RESET '!'
#define T_QUERY '?'
#define T_ACCELERATE '.'
#define T_DECELERATE ','

#define EXTENSION 'X'
#define EXTENSION_GROVE_SERIAL 'S'          // connect to Grove UART2
#define EXTENSION_VOICE 'A'                 // connect to Grove UART2 (on V0_*: a slide switch can choose the voice or the Grove), or UART1 (on V1). Hidden on board.
#define EXTENSION_DOUBLE_TOUCH 'T'          // connect to ANALOG1, ANALOG2
#define EXTENSION_DOUBLE_LIGHT 'L'          // connect to ANALOG1, ANALOG2
#define EXTENSION_DOUBLE_IR_DISTANCE 'D'    // connect to ANALOG3, ANALOG4
#define EXTENSION_PIR 'I'                   // connect to ANALOG3
#define EXTENSION_ULTRASONIC 'U'            // connect to Grove UART2
#define EXTENSION_GESTURE 'G'               // connect to Grove I2C
#define EXTENSION_CAMERA 'C'                // connect to Grove I2C
```

These were added to allow runtime control of the modules with moduleManager.h, via the 'X' token.

1. updated based on repo commit "1a008994" 2024-05-17

OpenCat.h: control & command variables[1]

- Other important variables are defined and, if appropriate, initialized.

```
277     //·bool·updated[10];
278     float·degPerRad·=·180·/·M_PI;
279     float·radPerDeg·=·M_PI·/·180;
280
281     //·control·related·variables
282     #define·IDLE_TIME·3000
283     long·idleTimer·=·0;
284     #define·CHECK_BATTERY_PERIOD·10000··//·every·10·seconds.·60·mins·->·3600·seconds
285     int·uptime·=·-1;
286     int·frame·=·0;
287     int·tStep·=·1;
288     long·loopTimer;
289     byte·fps·=·0;
290
291     char·token;
292     char·lastToken;
293     char·lowerToken;
294     #define·CMD_LEN·10
295     char·*lastCmd·=·new·char[CMD_LEN·+·1];··//·the·last·char·must·be·'\0'·for·safe·so·CMD_LEN+1·elements·are·required
296     int·cmdLen·=·0;
297     byte·newCmdIdx·=·0;
298     int8_t·periodGlobal·=·0;
299     #define·BUFF_LEN·2507··//·1524·=125*20+7=2507
300     char·*newCmd·=·new·char[BUFF_LEN·+·1];
301     int·spaceAfterStoringData·=·BUFF_LEN;
302     int·serialTimeout;
303     int·lastVoltage;
304     char·terminator;
305     //·int·serialTimeout;
306     long·lastSerialTime·=·0;
```

Define "token" variables.

Define "command" variables which hold the token parameters.

newCmdIdx:
This variable is used to set priority between command sources.

In principle, it could be used to set priority between different sensor data sources.

newCmd:
This variable does double duty.

- It can hold the token parameters as part of a command.
- It can hold skill array information, including the "duty angles" for skill frames. See the buildSkill() method of the Skill class in a later slide.

1. updated based on repo commit "1a008994" 2024-05-17

OpenCat.h: bool variables[1]

- bool variables used, variously, by reaction.h, io.h, skill.h, imu.h, moduleManager.h, infrared.h, motion.h, ultrasonic.h, sound.h, I2cEEPROM.h

```
308     bool interruptedDuringBehavior = false;
309     bool lowBatteryQ = false;
310     bool fineAdjust = true;
311     bool gyroBalanceQ = true;
312     bool printGyro = false;
313     bool autoSwitch = false;
314     bool walkingQ = false;
315     bool manualHeadQ = false;
316     bool nonHeadJointQ = false;
317     bool workingStiffness = true;
318     bool manualEyeColorQ = false;
319     // bool keepDirectionQ = true;
320     #define HEAD_GROUP_LEN 4  // used for controlling head pan, tilt, tail, and other joints independent from walking
321     int targetHead[HEAD_GROUP_LEN];
322
323     bool imuUpdated;
324     int exceptions = 0;
325     byte transformSpeed = 2;
326     float protectiveShift;  // reduce the wearing of the potentiometer
```

1. updated based on repo commit "1a008994" 2024-05-17

OpenCat.h: module & delay variables[1]

- module variables used by moduleManager.h
- delay variables used by reaction.h

```
int8_t moduleList[] = {
  EXTENSION_GROVE_SERIAL,
  EXTENSION_VOICE,
  EXTENSION_DOUBLE_TOUCH,
  EXTENSION_DOUBLE_LIGHT,
  EXTENSION_DOUBLE_IR_DISTANCE,
  EXTENSION_PIR,
  EXTENSION_ULTRASONIC,
  EXTENSION_GESTURE,
  EXTENSION_CAMERA,
};
String moduleNames[] = {"Grove_Serial", "Voice", "Double_Touch", "Double_Light ", "Double_Ir_Distance ", "Pir", "Ultrasonic", "Gesture", "Camera"};
bool moduleActivatedQ[] = {0, 1, 0, 0, 0, 0, 0, 0, 0};
bool initialBoot = true;
bool safeRest = true;
bool soundState;
byte buzzerVolume;
float amplifierFactor = 100.0;  //to fit the actual amplifier range of BiBoard

int delayLong = 20;
int delayMid = 8;
int delayException = 5;
int delayShort = 3;
int delayStep = 1;
int delayPrevious;
int runDelay = delayMid;
```

Used with the accelerate / decelerate tokens

1. updated based on repo commit "1a008994" 2024-05-17

OpenCat.h: servo related arrays

- Robot model dependent values (e.g. rotation directions and limits to servo angles) are defined here.
- middleShift[] is used to help define the "zero" position of each servo by "shifting" from the mathematical "middle" of the servo range by an amount specified in this integer array.

```
#ifdef NYBBLE
int8_t middleShift[] = { 0, 15, 0, 0,
                         -45, -45, -45, -45,
                         10, 10, -10, -10,
                         -30, -30, 30, 30 };
#elif defined BITTLE
int8_t middleShift[] = { 0, 15, 0, 0,
                         -45, -45, -45, -45,
                         55, 55, -55, -55,
                         -55, -55, -55, -55 };

#else // CUB
int8_t middleShift[] = { 0, 15, 0, 0,
                         -45, -45, -45, -45,
                         55, 55, -55, -55,
                         -45, -45, -45, -45 };
 #endif
```

```
 // #define INVERSE_SERVO_DIRECTION
#ifdef CUB
int8_t rotationDirection[] = { 1, -1, 1, 1,
                               1, -1, 1, -1,
                               1, -1, -1, 1,
                               1, -1, -1, 1 };
int angleLimit[][2] = {
  { -120, 120 },
  { -30, 80 },
  { -120, 120 },
  { -120, 120 },
  { -90, 60 },
  { -90, 60 },
  { -90, 90 },
  { -90, 90 },
  { -180, 120 },
  { -180, 120 },
  { -80, 200 },
  { -80, 200 },
  { -66, 100 },
  { -66, 100 },
  { -66, 100 },
  { -66, 100 },
};
#else
int8_t rotationDirection[] = { 1, -1, 1, 1,
                               1, -1, 1, -1,
                               1, -1, -1, 1,
                               -1, 1, 1, -1 };
```

```
#ifdef BITTLE
int angleLimit[][2] = {
  { -120, 120 },
  { -85, 85 },
  { -120, 120 },
  { -120, 120 },
  { -90, 60 },
  { -90, 60 },
  { -90, 90 },
  { -90, 90 },
  { -200, 80 },
  { -200, 80 },
  { -80, 200 },
  { -80, 200 },
  { -80, 200 },
  { -80, 200 },
  { -70, 200 },
  { -80, 200 },
};
#else
int angleLimit[][2] = {
  { -120, 120 },
  { -85, 85 },
  { -120, 120 },
  { -120, 120 },
  { -90, 60 },
  { -90, 60 },
  { -90, 90 },
  { -90, 90 },
  { -200, 80 },
  { -200, 80 },
  { -80, 200 },
  { -80, 200 },
  { -80, 80 },
  { -80, 80 },
  { -70, 80 },
  { -80, 80 },
};
#endif
 #endif
```

OpenCat.h: servo related arrays (cont.) & robot orientation variables

These arrays are used in many places!

```
#ifdef X_LEG
int currentAng[DOF] = {-30, -80, -45, 0,
                       0, 0, 0, 0,
                       75, 75, -75, -75,
                       -55, -55, 55, 55};
int previousAng[DOF] = {-30, -80, -45, 0,
                        0, 0, 0, 0,
                        75, 75, -75, -75,
                        -55, -55, 55, 55};
#else
int currentAng[DOF] = {-30, -80, -45, 0,
                       0, 0, 0, 0,
                       75, 75, 75, 75,
                       -55, -55, -55, -55};
int previousAng[DOF] = {-30, -80, -45, 0,
                        0, 0, 0, 0,
                        75, 75, 75, 75,
                        -55, -55, -55, -55};
#endif
int zeroPosition[DOF] = {};
int calibratedZeroPosition[DOF] = {};

int8_t servoCalib[DOF] = {0, 0, 0, 0,
                          0, 0, 0, 0,
                          0, 0, 0, 0,
                          0, 0, 0, 0};

int16_t imuOffset[9] = {0, 0, 0,
                        0, 0, 0,
                        0, 0, 0};

float expectedRollPitch[2];
float RollPitchDeviation[2];
float currentAdjust[DOF] = {};
int slope = 1;
```

Nybble

Bittle

These default values match those found in the "rest" skill for each model.

Note that joint indexes #10,11 and #14,15 have opposite sign due to the different leg configuration of Nybble vs. Bittle

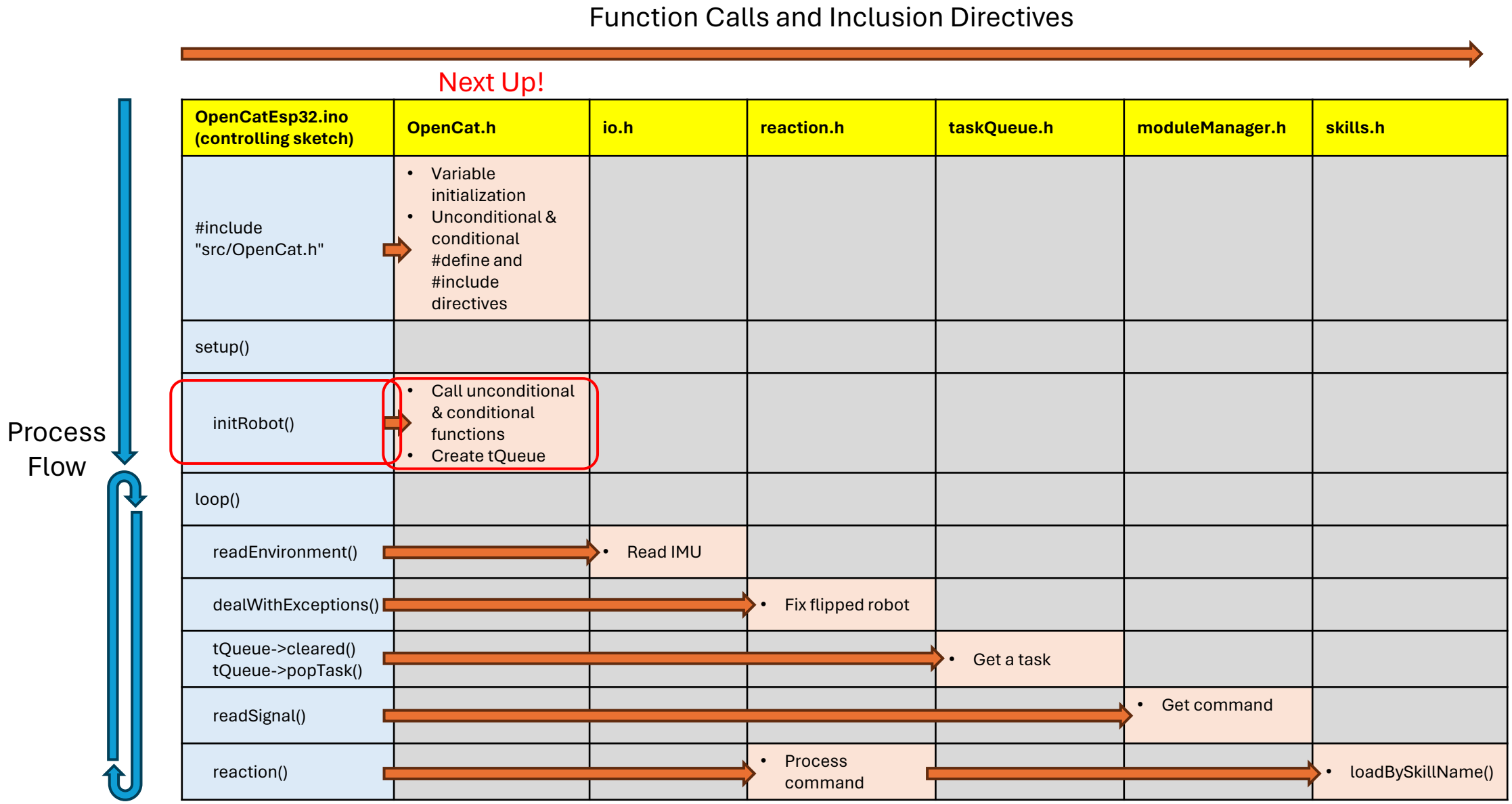
OpenCat.h: unconditional & conditional includes[1]

- Unconditional (always included) header files
  - tools.h, QList/QList.h, taskQueue.h
  - sound.h, I2cEEPROM.h, espServo.h, motion.h, randomMind.h, io.h
  - skill.h, moduleManager.h, reaction.h, qualityAssurance.h
- Conditional (option dependent) header files inclusions
  - #ifdef BT_BLE #include "bleUart.h"
  - #ifdef GYRO_PIN #include "imu.h"
  - #ifdef IR_PIN #include "infrared.h"
  - #ifdef NEOPIXEL_PIN #include "led.h"
- In repo commit "fdae2e19", there *were* conditional inclusions in OpenCat.h, for the following header files:
  - camera.h, voice.h, gesture.h, pir.h, doubleTouch.h, doubleLight.h, doubleInfraredDistance.h
- *However, these have now been moved to moduleManager.h*

```
479  #include "tools.h"
480  #include "QList/QList.h"
481  #include "taskQueue.h"
482
483  #include "sound.h"
484  #include "I2cEEPROM.h"
485  #ifdef BT_BLE
486  #include "bleUart.h"
487  #endif
488  #ifdef GYRO_PIN
489  #include "imu.h"
490  #endif
491  #ifdef IR_PIN
492  #include "infrared.h"
493  #endif
494  #include "espServo.h"
495  #include "motion.h"
496  #include "randomMind.h"
497  #include "io.h"
498
499  #include "skill.h"
500  #include "moduleManager.h"
501  #ifdef NEOPIXEL_PIN
502  #include "led.h"
503  #endif
504  #include "reaction.h"
505  #include "qualityAssurance.h"
```

1. updated based on repo commit "1a008994" 2024-05-17

"Sketch" Process Map



OpenCat.h: initRobot() function

- Beeps notification.
- Wire.begin initializes I2C bus communication.
- Displays initial message in the serial monitor window.
- Calls functions to initialize the robot.

```
488  void initRobot() {
489    beep(20);
490    Wire.begin();
491    SoftwareVersion = SoftwareVersion + BOARD + "_" + DATE;
492    PTL('k');
493    PTLF("Flush the serial buffer...");
494    PTL("\n* Start *");
495    printToAllPorts(MODEL);
496    PTF("Software version: ");
497    printToAllPorts(SoftwareVersion);
498    soundState = i2c_eeprom_read_byte(EEPROM_BOOTUP_SOUND_STATE);
499    buzzerVolume = max(byte(0), min(byte(10), i2c_eeprom_read_byte(EEPROM_BUZZER_VOLUME
500    PTF("Buzzer volume: ");
501    PT(buzzerVolume);
502    PTL("/10");
503    i2cDetect();
504    i2cEepromSetup();
```

*The macro directives GYRO_PIN, BT_BLE, and BT_SPP are conditional, but they were defined above, so the guarded functions **are** called.*

```
505  #ifdef GYRO_PIN
506    imuSetup();
507  #endif
508  #ifdef BT_BLE
509    bleSetup();
510  #endif
511  #ifdef BT_SPP
512    blueSspSetup();
513  #endif
514    servoSetup();
515    lastCmd[0] = '\0';
516    newCmd[0] = '\0';
517    skill = new Skill();
518    skillList = new SkillList();
519    for (byte i = 0; i < randomMindListLength; i++) {
520      randomBase += choiceWeight[i];
521    }
```

The servoSetup() function is unconditional so it is always called.

OpenCat.h: initRobot() function (cont.)

- Calls functions to initialize the robot (cont.)

*The macro directives NEOPIXEL_PIN, PWM_LED_PIN, VOLTAGE, and IR_PIN are conditional, but they were defined above, so the guarded functions **are** called.*

```
523  #ifdef NEOPIXEL_PIN
524    ledSetup();
525  #endif
526  #ifdef PWM_LED_PIN
527    pinMode(PWM_LED_PIN, OUTPUT);
528  #endif
529  #ifdef VOLTAGE
530    while (lowBattery())
531      ;
532  #endif
533
534  #ifdef IR_PIN
535    irrecv.enableIRIn();
536  #endif
537
538    QA();
539    i2c_eeprom_write_byte(EEPROM_BIRTHMARK_ADDRESS, BIRTHMARK);  // finish the test and mark the board as initialized
540
541  #ifdef CAMERA
542    cameraSetup();
543  #endif
544    // #ifdef ULTRASONIC
545    rgbUltrasonicSetup();
546  // #endif
547  #ifdef GESTURE
548    gestureSetup();
549  #endif
```

Perform quality assurance tests during setup.

In repo commit "1a008994" 2024-05-17, these conditional directives are now in moduleManager.h

OpenCat.h: initRobot() function (cont.)[1]

- Calls functions to initialize the robot (cont.)

```
564     tQueue = new TaskQueue();
565
566     // ...
577
578     loadBySkillName("rest");  //must have to avoid memory crash. need to check why.
579     ..........................//allCalibratedPWM(currentAng); alone will lead to crash
580     delay(500);
581
582     initModuleManager();
583   #ifdef GYRO_PIN
584     //read_IMU();  //ypr is slow when starting up. leave enough time between IMU initialization and this reading
585     if (!moduleActivatedQfunction(EXTENSION_DOUBLE_LIGHT) && !moduleActivatedQfunction(EXTENSION_DOUBLE_TOUCH)
586         && !moduleActivatedQfunction(EXTENSION_GESTURE) && !moduleActivatedQfunction(EXTENSION_DOUBLE_IR_DISTANCE)
587         && !moduleActivatedQfunction(EXTENSION_CAMERA) && !moduleActivatedQfunction(EXTENSION_ULTRASONIC))
588       tQueue->addTask((exceptions) ? T_CALIBRATE : T_REST, "");
589   #endif
590     PTL("Ready!");
591     beep(24, 50);
592     idleTimer = millis();
593   }
```

The "tQueue" TaskQueue object is created here.

Commented out code not shown.

moduleManager now handles the modules

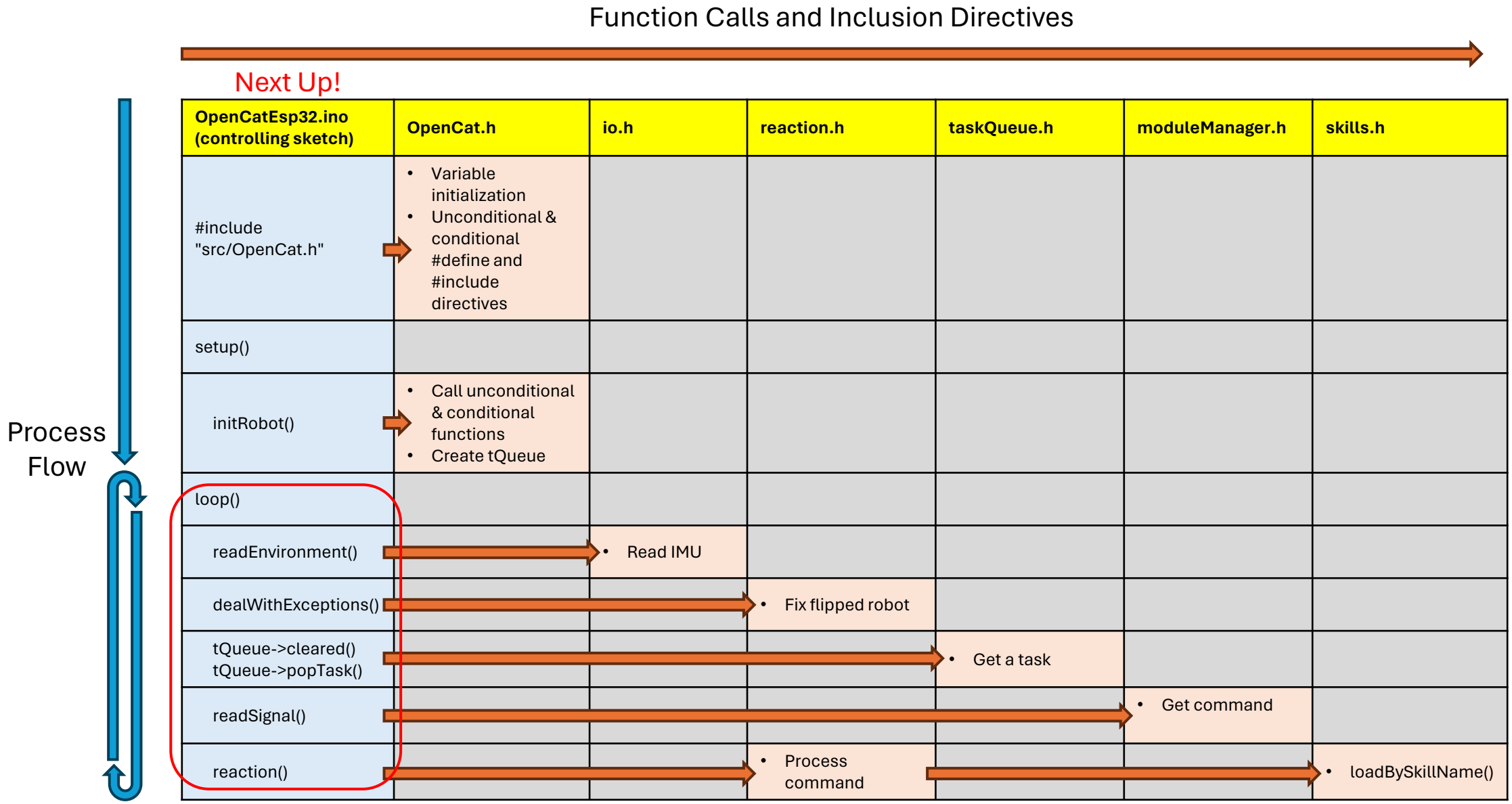
New moduleActivatedQfunction() returns true if the specified module is enabled.

- If the Gyro is presenting an exception (because the robot is on its side) then set the skill to calibration.
- Otherwise, set the skill to the rest posture.

Insert other setup() robot commands here. For example:
tQueue->addTask('k', "up"); //queue "kup" command to start robot in "stand-up" posture

1. updated based on repo commit "1a008994" 2024-05-17

"Sketch" Process Map



OpenCatEsp32.ino: loop() function

- The loop() function is the Arduino function that is run continuously after setup() finishes.
  - If VOLTAGE is defined (currently only for the BiBoard2), then the "check for" lowBattery() function in reaction.h is called.
  - readEnvironment(), in io.h, is called to...
    - read from the IMU [via read_IMU() in imu.h] - more info on a later slide
    - read sound [via read_sound() in io.h]
    - read GPS [via read_GPS() in io.h]

    Of these, only read_IMU() has code.
  - dealWithExceptions(), in reaction.h, is called to deal with 4 exception values:
    - -1 = robot dropped; -2 = robot flipped over; -3 & -4 = cases when robot is manually pushed or rotated. However, only the exception -2 is currently used.
  - The cleared() function of the tQueue object is called.
    - If it returns False (there is at least one task object in the task queue) then a task in the task queue is performed via the popTask() function of tQueue.
    - Else, readSignal(), in moduleManager.h, is called to get a command.
  - playLight() in led.h is called if the NEOPIXEL_PIN macro directive is defined
  - reaction(), in reaction.h, is called to process the command.

We will examine circled items in the next series of slides

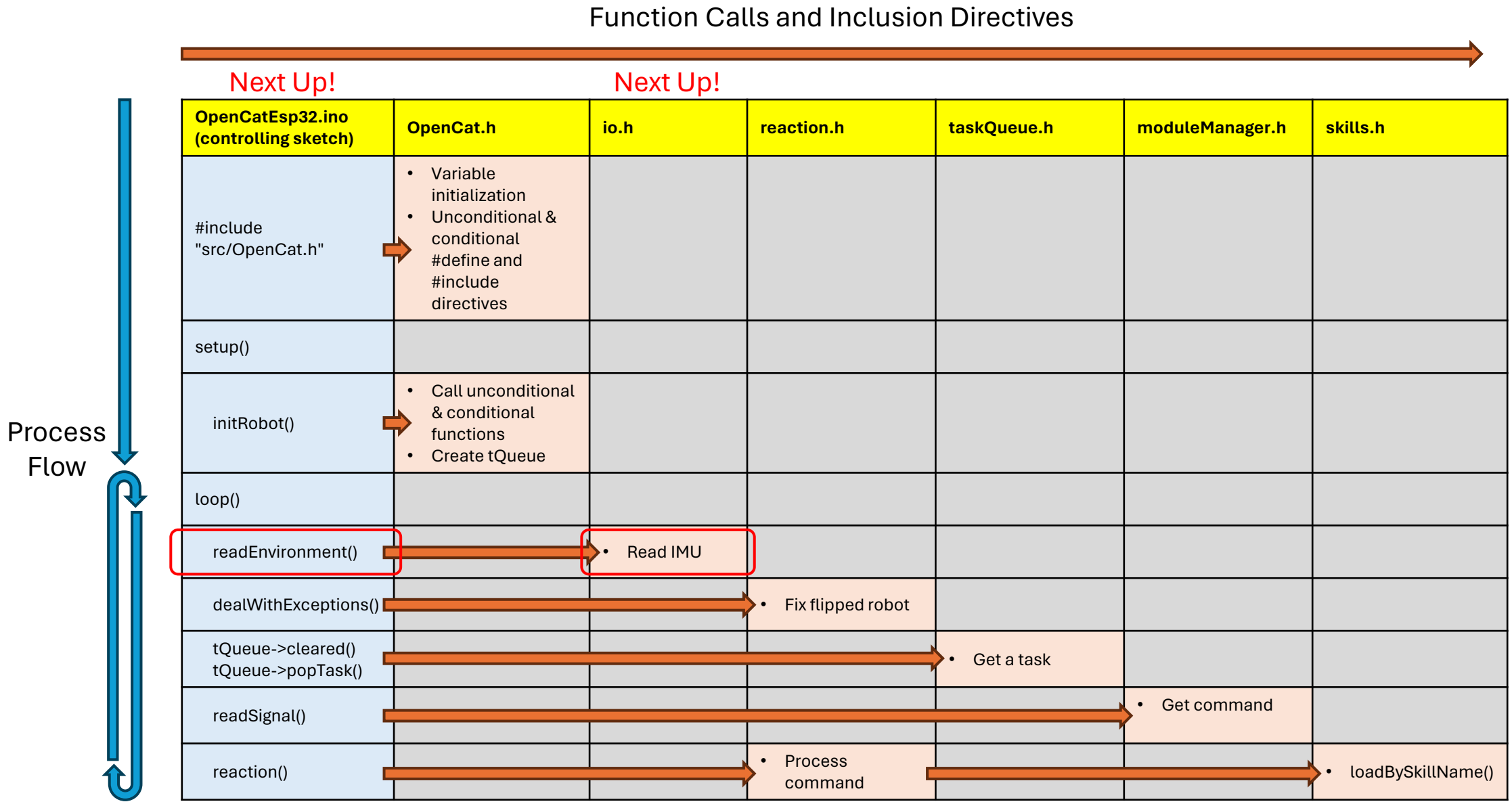
```
47  void loop() {
48  #ifdef VOLTAGE
49    lowBattery();
50  #endif

57    //  //— read environment sensors (low level)
58    readEnvironment();
59    //  //— special behaviors based on sensor events
60    dealWithExceptions();  // low battery, fall over, lifted, etc.
```

```
61    if (!tQueue->cleared()) {
62      tQueue->popTask();
63    } else {
64      readSignal();

72  #ifdef NEOPIXEL_PIN
73    playLight();
74  #endif
75    reaction();
76  }
```

"Sketch" Process Map



io.h: readEnvironment() function

- The bool gyroBalanceQ is true when the Gyro is enabled.
- The read_IMU() function (in imu.h) reads a Gyro packet to update the float *ypr (yaw, pitch, roll) array.
  - The *ypr array is used extensively in the source code to check and respond to the robot orientation.

in io.h

```
24  void readEnvironment() {
25  #ifdef GYRO_PIN
26    if (gyroBalanceQ && !(frame % imuSkip))
27      imuUpdated = read_IMU();
28  #endif
29    read_sound();
30    read_GPS();
31  }
```

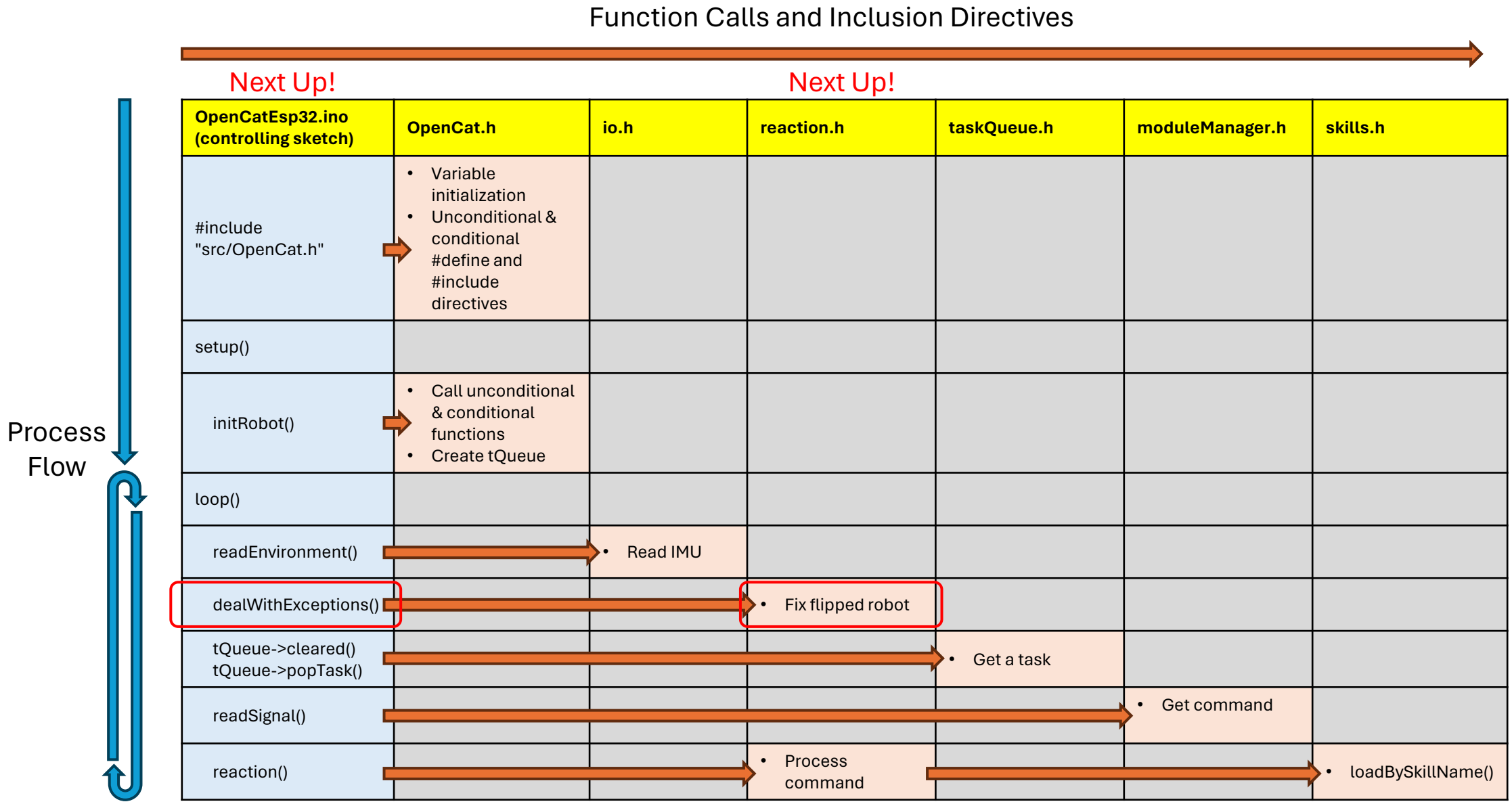
Currently, does nothing.

in imu.h *(not shown in process map)*

```
283  bool read_IMU() {
284    if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) { // Get the Latest packet
285      //display Euler angles in degrees
286      mpu.dmpGetQuaternion(&q, fifoBuffer);
287      mpu.dmpGetAccel(&aa, fifoBuffer);
288      mpu.dmpGetEuler(euler, &q);
289      mpu.dmpGetGravity(&gravity, &q);
290      mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
291      mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
292      mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
293
294      for (byte i = 0; i < 3; i++) { ... }
302      if (printGyro)
303        print6Axis();
304      //exceptions = aaReal.z < 0 && fabs(ypr[2]) > 85;  //the second condi
305
306      // ...
316      if (ARZ < 0 && fabs(ypr[2]) > 85)  //  exceptions = aaReal.z < 0;
317        exceptions = -2;                 // ...
323      else exceptions = 0;
324      //however, its change is very slow.
325      for (byte m = 0; m < 3; m++) { ... }
331      return true;
332    }
333    return false;
334  }
```

from Gyro packet

"Sketch" Process Map



reaction.h: dealWithExceptions() function

- This code has 4 exceptions but only "case -2" does anything - will only show that part.

```
1   void dealWithExceptions() {
2   #ifdef GYRO_PIN
3     if (gyroBalanceQ && exceptions) {  // the gyro reaction switch can be toggled on/off by the 'g' token
4       switch (exceptions) {
          :
          :
17        case -2:
18        {
19          PTL("EXCEPTION 2");
20          soundFallOver();
21          //  for (int m = 0; m < 2; m++)
22          //    meow(30 - m * 12, 42 - m * 12, 20);
23          token = 'k';
24          manualHeadQ = false;
25          strcpy(newCmd, "rc");
26          newCmdIdx = -2;
27          // tQueue->addTaskToFront('k', "rc");
28          break;
29        }
```

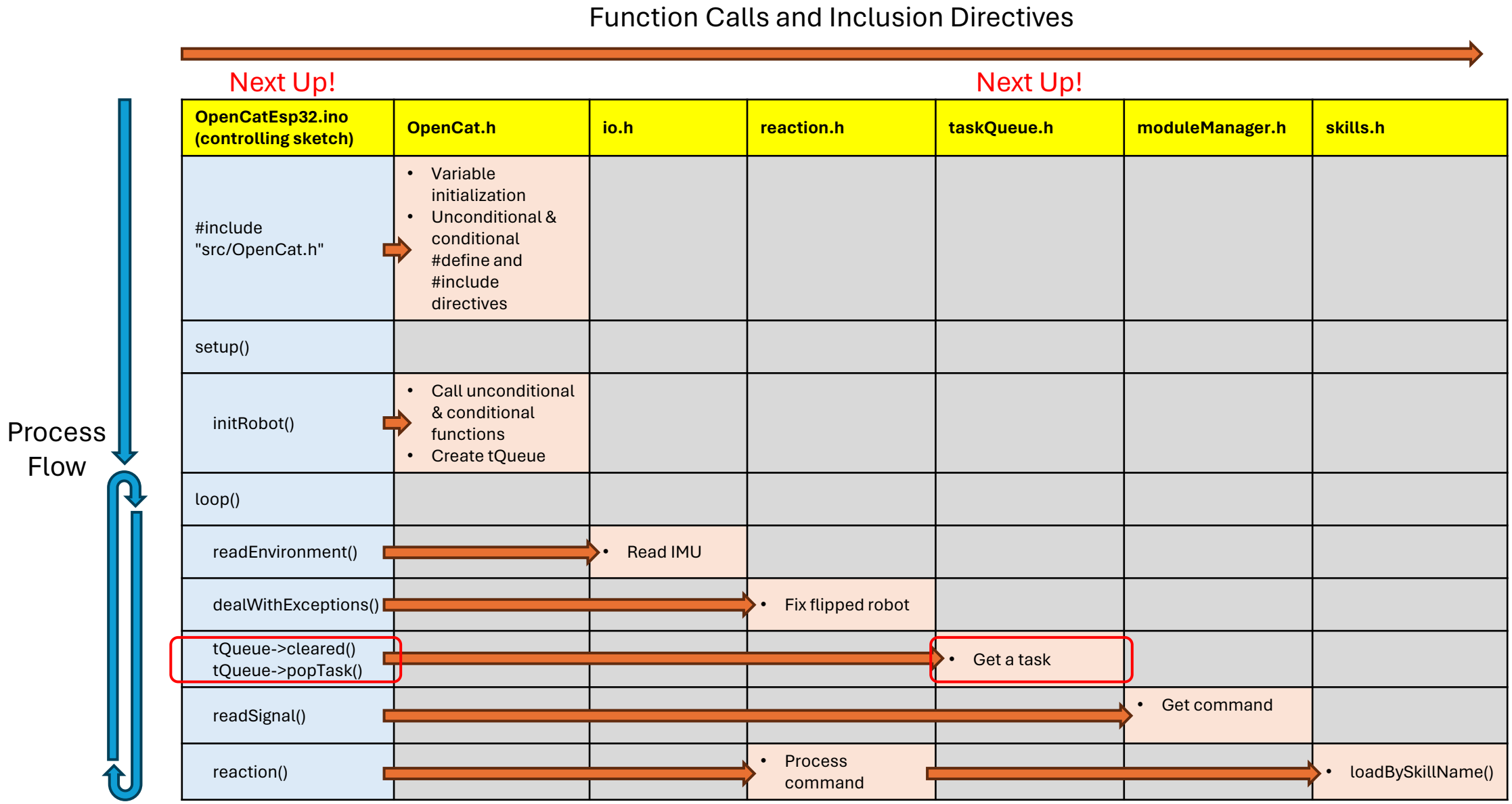
Exception "case -2" is a fall ("flipped robot") detected by the Gyro.
It triggers the recovery ("rc") skill.

After the recovery attempt, read_IMU()
is called to get fresh Gyro readings...

```
          :
          :
101     if (exceptions != -4)
102       print6Axis();
103     read_IMU();  // flush the IMU to avoid static readings and infinite loop
104
105     if (tQueue->lastTask == NULL) {
106       if (strcmp(lastCmd, "") && strcmp(lastCmd, "lnd") && *strGet(newCmd, -1) != 'L' && *strGet(lastCmd, -1) != 'R') {
107         PTH("save last task", lastCmd);
108         tQueue->lastTask = new Task('k', lastCmd);
109       }
110     }
111   }
112   // if (tQueue->cleared() && runDelay <= delayException)
113   //   runDelay = delayPrevious;
114   #endif
115   }
```

then tQueue is used to requeue the last task

"Sketch" Process Map



taskQueue.h: Class Task and the tQueue object

Class Task creates a task object of generic type (using C++ template).

- Initialize taskTimer and taskInterval.
- Initialize tkn (token character), parameters (array), paraLength (length of that array) and dly (task delay). Note: dly (and parameter d) are apparently not used.
- Class constructor parameters are t, p and d.
  - char t (= token character) assigned to tkn.
  - T* p (= pointer to parameter aka newCmd which can contain ascii or binary characters).
  - int d (= task delay) assigned to dly.
- Class constructor code is
  - Sets paraLength based on 'p', depending on the token character (A-Z vs. other = a-z, digits, special characters).
  - Dimension parameters (array).
  - Use copy function arrayNCPY() from tools.h to copy from p to parameters (array).
  - Set last character of parameters to '~" (for capital letter tokens) or to "\0" for all other tokens.
    - Distinguish between binary and ascii commands

Capital letters are used for binary tokens, so called because they are meant to accept binary parameters.

Binary parameters use the '~' character as a terminator in newCmd.

ASCII parameters use the '\0' character as a terminator in newCmd.

```
1   long taskTimer = 0;
2   long taskInterval = -1;
3   class Task {
4   public:
5     char tkn;
6     char* parameters;
7     int paraLength;
8     int dly;
9     template<typename T> Task(char t, T* p, int d = 0)
10      : tkn{ t }, dly{ d } {
11      paraLength = (tkn >= 'A' && tkn <= 'Z') ? strlenUntil(p, '~') : strlen((char*)p);
12      parameters = new char[paraLength + 1];
13      arrayNCPY(parameters, p, paraLength);
14      parameters[paraLength] = (tkn >= 'A' && tkn <= 'Z') ? '~' : '\0';
15      //PTL("create task");
16      //info();
17    };
18    ~Task() {
19      //if (paraLength)
20      delete[] parameters;
21    };
22    void info() {
23      printCmdByType(tkn, parameters);
24    }
25  };
```

Create class destructor and info() function

taskQueue.h: Class TaskQueue

- Class TaskQueue inherits from QList (a generic linked list class)
  - A TaskQueue object is a linked list that holds items of type Task.
  - lastTask is initially NULL but later holds the most recent Task completed.
  - addTaskToFront() and addTask() puts a Task item at the front and back of the list, respectively.
  - createTask() creates and adds example Tasks to the TaskQueue
  - The cleared() function, called in the OpenCatEsp32.ino loop() function:
    - returns true (no Tasks in the list) when:
      TaskQueue size is 0 AND "time since last Task completion" is > current taskInterval

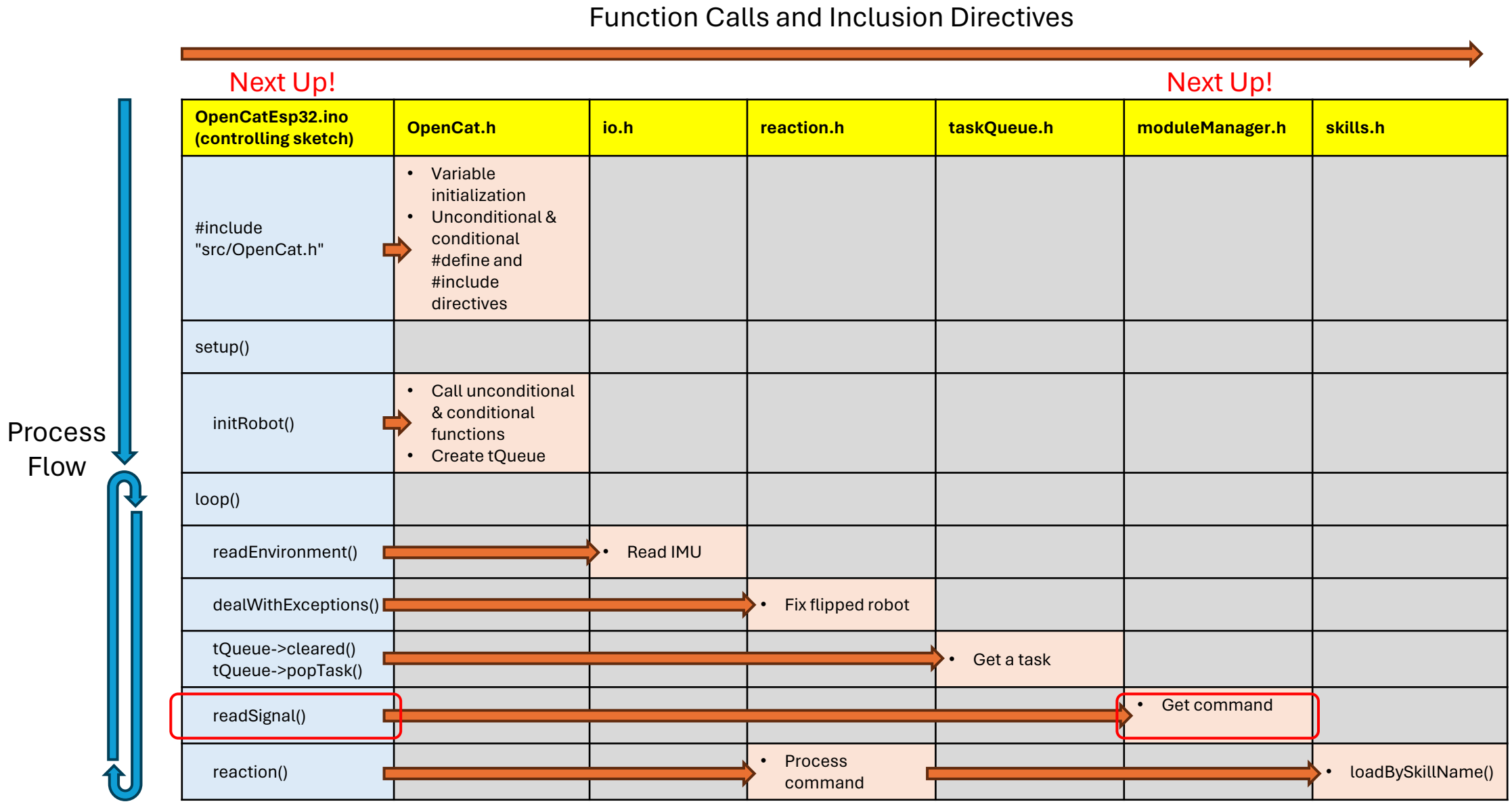
```
27  class TaskQueue : public QList<Task*> {
28  public:
29    Task* lastTask;
30    TaskQueue() {
31      PTLF("TaskQ");
32      lastTask = NULL;
33    };
34    template<typename T> void addTask(char t, T* p, int d = 0) {
35      // PTH("add ", p);
36      this->push_back(new Task(t, p, d));
37    }
38    template<typename T> void addTaskToFront(char t, T* p, int d = 0) {
39      PTH("add front", p);
40      this->push_front(new Task(t, p, d));
41    }
42    void createTask() {  //this is an example task
43      this->addTask('k', "vtF", 2000);
44      this->addTask('k', "up");
45    }
46    bool cleared() {
47      return this->size() == 0 && long(millis() - taskTimer) > taskInterval;
48    }
```

taskQueue.h: Class TaskQueue (cont.)

- The popTask() function, called in the OpenCatEsp32.ino loop() function:
  - Calls loadTaskInfo() while getting a Task from the front of the TaskQueue.
    - loadTaskInfo() places Task information into global variables (token, cmdLen, taskInterval, newCmd, taskTimer, newCmdIdx).
  - The Task is then popped off the front (and thereby removed from the TaskQueue).

```
49    void loadTaskInfo(Task* t) {
50      token = t->tkn;
51      cmdLen = t->paraLength;
52      taskInterval = t->dly;
53      arrayNCPY(newCmd, t->parameters, cmdLen);
54      newCmd[cmdLen] = (token >= 'A' && token <= 'Z') ? '~' : '\0';
55      taskTimer = millis();
56      newCmdIdx = 5;
57    }
58    void popTask() {
59      if (long(millis() - taskTimer) > taskInterval) {
60        if (this->size() > 0) {
61          loadTaskInfo(this->front());
62          this->pop_front();
63          //PTL("Use pop");
64        }
65      }
66    }
67  };
68  TaskQueue* tQueue;
```

"Sketch" Process Map



moduleManager.h: readSignal() function[1]

- Conditional directives select the source to receive the command
  - Then uses one of the several functions, read_infrared(), read_serial(), readBle(), or read_voice(), as appropriate to get (directly or via interpretation) the command (token plus token parameters) from that source.
- Read from whatever sensors are enabled.

Read serial source: USB or Bluetooth Classic SPP (Serial Port Profile)

```
306  void readSignal() {
307    byte moduleIndex = activeModuleIdx();
308  #ifdef IR_PIN
309    read_infrared();  //  newCmdIdx = 1
310  #endif
311    read_serial();  //  newCmdIdx = 2
312  #ifdef BT_BLE
313    detectBle();  //  newCmdIdx = 3;
314    readBle();
315  #endif
316
317  #ifdef VOICE
318    if (moduleList[moduleIndex] == EXTENSION_VOICE)
319      read_voice();
320  #endif
321
322    long current = millis();
323    if (newCmdIdx)
324      idleTimer = millis() +
325  #ifdef DOUBLE_INFRARED_DISTANCE
326                0
327  #else
328                IDLE_TIME
329  #endif
330      ;
```

newCmdIdx is set to the values indicated

1. updated based on repo commit "1a008994" 2024-05-17

moduleManager.h: readSignal() function (cont.)[1]

- Read from whatever sensors are enabled (cont.).

```
331     ··else·if·(token·!=·T_CALIBRATE·&&·token·!=·T_SERVO_FOLLOW·&&·token·!=·T_SERVO_FEEDBACK·&&·current·-·idleTimer·>·0)·{
332     ····if·(moduleIndex·==·-1)··//·no·active·module
333     ····    return;
334
335     #ifdef·CAMERA
336     ····if·(moduleList[moduleIndex]·==·EXTENSION_CAMERA)
337     ····    read_camera();
338     #endif
339     #ifdef·ULTRASONIC
340     ····if·(moduleList[moduleIndex]·==·EXTENSION_ULTRASONIC)·{
341     ····    readRGBultrasonic();
342     ····}
343
344     #endif
345     #ifdef·GESTURE
346     ····if·(moduleList[moduleIndex]·==·EXTENSION_GESTURE)
347     ····    read_gesture();
348     #endif
349     #ifdef·PIR
350     ····if·(moduleList[moduleIndex]·==·EXTENSION_PIR)
351     ····    read_PIR();
352     #endif
```

```
353     #ifdef·DOUBLE_TOUCH
354     ····if·(moduleList[moduleIndex]·==·EXTENSION_DOUBLE_TOUCH)
355     ····    read_doubleTouch();
356     #endif
357     #ifdef·DOUBLE_LIGHT
358     ····if·(moduleList[moduleIndex]·==·EXTENSION_DOUBLE_LIGHT)
359     ····    read_doubleLight();
360     #endif
361     #ifdef·DOUBLE_INFRARED_DISTANCE
362     ····if·(moduleList[moduleIndex]·==·EXTENSION_DOUBLE_IR_DISTANCE)
363     ····    read_doubleInfraredDistance();··//·has·some·bugs
364     #endif
365     #ifdef·TOUCH0
366     ····read_touch();
367     #endif
368     ····// ...
371
372     ····if·(autoSwitch)·{
373     ····    randomMind();·····//·make·the·robot·do·random·demos
374     ····    powerSaver(POWER_SAVER);··//·make·the·robot·rest·after·a·certain·period,
375     ····}
376     ··}
377     }
```

1. updated based on repo commit "1a008994" 2024-05-17

moduleManager.h: read_serial() function as an example[1]

- As a command source example, we look at read_serial(), which is used for serial communication.

```
239  void read_serial(){
240    Stream *serialPort = NULL;
241    // String source;
242  #ifdef BT_SSP
243    if (SerialBT.available()) {  //give BT a higher priority over wired serial
244      serialPort = &SerialBT;
245      // source = "BT";
246    } else
247  #endif
248    if (moduleActivatedQ[0] && Serial2.available()) {
249      serialPort = &Serial2;
250    } else if (Serial.available()) {
251      serialPort = &Serial;
252      // source = "SER";
253    }
```

Check for Bluetooth serial communication

Check for serial communication to Voice module

Finally, check for USB serial communication

1. updated based on repo commit "1a008994" 2024-05-17

moduleManager.h: read_serial() function as an example (cont.)[1]

- Read token then read any parameters that follow.

```
254   if (serialPort) {
255     token = serialPort->read();
256     lowerToken = tolower(token);
257     newCmdIdx = 2;
258     delay(1);                                                    //leave enough time for serial read
259     terminator = (token >= 'A' && token <= 'Z') ? '~' : '\n';  //capitalized tokens use binary encoding for long data commands
260                                                                  //'~' ASCII code = 126; may introduce bug when the angle is 126 so only use angles <= 125
261     serialTimeout = (token == T_SKILL_DATA || lowerToken == T_BEEP) ? SERIAL_TIMEOUT_LONG : SERIAL_TIMEOUT;
262     lastSerialTime = millis();
263     do {
264       if (serialPort->available()) {
265         // ...
267         do {
268           if ((token == T_SKILL || lowerToken == T_INDEXED_SIMULTANEOUS_ASC || lowerToken == T_INDEXED_SEQUENTIAL_ASC)
269               && cmdLen >= spaceAfterStoringData || cmdLen > BUFF_LEN) {
270             PTH("Cmd Length: ", cmdLen);
271             PTF("OVF");
272             beep(5, 100, 50, 5);
273             do {
274               serialPort->read();
275             } while (serialPort->available());
276             printToAllPorts(token);
277             token = T_SKILL;
278             strcpy(newCmd, "up");
279             cmdLen = 2;
280             return;
281           }
282           newCmd[cmdLen++] = serialPort->read();
283         } while (serialPort->available());
284         lastSerialTime = millis();
285       }
286     } while (newCmd[cmdLen - 1] != terminator && long(millis() - lastSerialTime) < serialTimeout);  //the lower case tokens are encoded in ASCII and car
```

Get token

Assign the terminator, based on token (uppercase vs. not-uppercase)

Outer "do ... while" loop continues until the proper terminator is found (or until timeout occurs).

Middle "do ... while" loop checks for overflow (too many characters after token) before adding next parameter character to newCmd

Wrapped this line so it fits on the page

Inner "do ... while" loop clear the serial buffer when overflow occurs.

Set the command to something safe when the overflow occurs

Add next parameter character to newCmd then loop to recheck for overflow

1. updated based on repo commit "1a008994" 2024-05-17

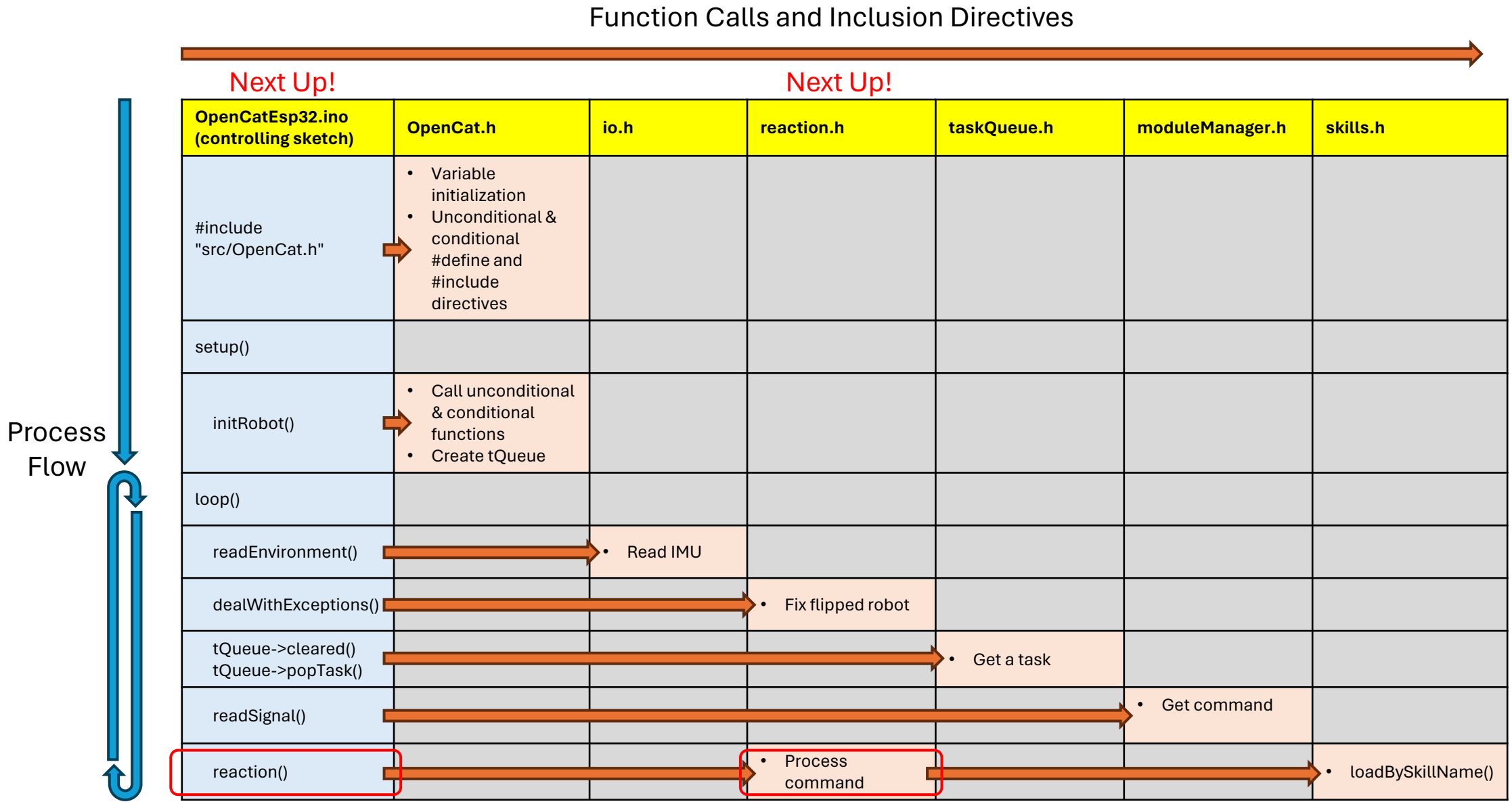
moduleManager.h: read_serial() function as an example (cont.)[1]

- Check for non-uppercase tokens, excepting 'X', 'R', 'W' tokens, to delete undesired '\r' and '\n' characters.
  - Terminate newCmd with '\0' (string terminator) in such cases.
  - Otherwise, use the '~' terminator.

```
    if (!(token >= 'A' && token <= 'Z') || token == 'X' || token == 'R' || token == 'W') {  // serial monitor is used to send lower cased tokens by users
                                                                                          // delete the unexpected '\r' '\n' if the serial monitor sends line end
      for (int i = cmdLen - 1; i >= 0; i--) {
        if ((newCmd[i] == '\n') || (newCmd[i] == '\r')) {
          newCmd[i] = '\0';
          cmdLen--;
        }
      }
    }
    cmdLen = (newCmd[cmdLen - 1] == terminator) ? cmdLen - 1 : cmdLen;
    newCmd[cmdLen] = (token >= 'A' && token <= 'Z') ? '~' : '\0';
    newCmdIdx = 2;
    // PTH("read_serial, cmdLen = ", cmdLen);
    // printCmdByType(token, newCmd, cmdLen);
  }
}
```

1. updated based on repo commit "1a008994" 2024-05-17

"Sketch" Process Map



reaction.h: reaction() function[1]

First there is some setup work.

- Note:
  newCmdIdx is set by various functions
  e.g. readSignal(), readInfrared(), read_PIR()
  - Value of newCmdIdx is checked in reaction() and readSignal()
  - It must be non-zero to allow access to most of the reaction() functionality

- Value is used in the beep() function
- newCmdIdx = 2 used with VOICE macro
- newCmdIdx = -1 and -2 used in dealWithExceptions
- newCmdIdx = 4 used by powerSaver() function in randomMind.h
- newCmdIdx = 100 used by randomMind() function in randomMind.h
- resetCmd() sets newCmdIdx = 0

If the last token was one of those listed, then turn on the gyro

```
176  void reaction() {
177    if (newCmdIdx) {
178      //PTLF("-----");
179      lowerToken = tolower(token);
180      if (initialBoot) { ... }
186      if (token != T_REST && newCmdIdx < 5)
187        idleTimer = millis();
188      if (newCmdIdx < 5 && lowerToken != T_BEEP && token != T_MEOW && token != T_LISTED_BIN
189          && token != T_INDEXED_SIMULTANEOUS_BIN && token != T_TILT && token != T_READ && token != T_WRITE)
190        beep(15 + newCmdIdx, 5);  //ToDo: check the muted sound when newCmdIdx = -1
191      if (!workingStiffness && (lowerToken == T_SKILL || lowerToken == T_INDEXED_SEQUENTIAL_ASC || lowerToken == T_INDEXED_SIMULTANEOUS_ASC)) { ... }
197      if ((lastToken == T_CALIBRATE || lastToken == T_REST || lastToken == T_SERVO_FOLLOW || !strcmp(lastCmd, "fd")) && token != T_CALIBRATE) { ... }
201      if (token != T_PAUSE && !tStep) { ... }
205    #ifdef ESP_PWM
206      if (token != T_SERVO_FEEDBACK && token != T_SERVO_FOLLOW && measureServoPin != -1) { ... }
212    #endif
```

Initializes the gyro and Random Mind

idleTimer used to know time since last command

Used by T_SERVO_MICROSECOND token

Play note with a frequency that is based on the value of newCmdIdx

Wrapped this line so it fits on the page

Initializes servos

1. updated based on repo commit "1a008994" 2024-05-17

reaction.h: reaction() function (cont.)[1]

Next is this very long switch that responds to every defined token.

```
214     switch (token) {
215       case T_QUERY:
216       {...}
221       case T_NAME:
222       {...}
231       case T_GYRO_FINENESS:
232       case T_GYRO_BALANCE:
233       case T_PRINT_GYRO:
234       case T_VERBOSELY_PRINT_GYRO:
235       case T_RANDOM_MIND:
236       case T_SLOPE:
237       {...}
263       case T_PAUSE:
264       {...}
273       case T_ACCELERATE:
274       {...}
278       case T_DECELERATE:
279       {...}
283       case T_REST:
284       {...}
295       case T_JOINTS:
296       {...}
308       case T_MELODY:
309       {...}
313     #ifdef ULTRASONIC
314       case T_COLOR:
315       {...}
327     #endif
328       case ';':
329       {...}
333       case ':':
334       {...}
```

```
338       case T_SAVE:
339       {...}
348       case T_ABORT:
349       {...}
358       case T_RESET:
359       {...}
363       case T_CALIBRATE:                /
364       case T_INDEXED_SEQUENTIAL_ASC:    /
365       case T_INDEXED_SIMULTANEOUS_ASC:  /
366     #ifdef T_SERVO_MICROSECOND
367       case T_SERVO_MICROSECOND:  //send
368     #endif
369     #ifdef T_SERVO_FEEDBACK
370       case T_SERVO_FEEDBACK:
371       case T_SERVO_FOLLOW:
372     #endif
373       case T_TILT:  //tilt the robot, fo
374       case T_MEOW:  //meow
375       case T_BEEP:  //beep(tone, duratio
376     #ifdef T_TUNER
377       case T_TUNER:
378     #endif
379       {
380         if (token == T_INDEXED_SIMULTANEOUS_ASC && cmdLen == 0)
381           manualHeadQ = false;
382         else{...}
547         break;
548       }
```

```
550       // this block handles array like argumen
551       case T_INDEXED_SEQUENTIAL_BIN:
552       case T_INDEXED_SIMULTANEOUS_BIN:
553       case T_READ:
554       case T_WRITE:
555       {...}
612       case EXTENSION:
613       {...}
640       case T_LISTED_BIN:  //list of all 16 jc
641       {...}
645       case T_BEEP_BIN:
646       {...}
659       case T_TEMP:
660       {...}
669       case T_SKILL_DATA:  //takes in the skil
670       {...}
684       case T_SKILL:
685       {...}
699       case T_TASK_QUEUE:
700       {...}
704       default:
705       {...}
709     }
```

Undocumented
tokens used by setServoP()
in espServo.h to tune servo stiffness

1. updated based on repo commit "1a008994" 2024-05-17

reaction.h: reaction() function (cont.)

- Code for selected tokens.

```
215         case T_GYRO_FINENESS:
216         case T_GYRO_BALANCE:
217         case T_PRINT_GYRO:
218         case T_VERBOSELY_PRINT_GYRO:
219         case T_RANDOM_MIND:
220         case T_SLOPE:
221           {
222             if (token == T_RANDOM_MIND) {
223               autoSwitch = !autoSwitch;
224               token = autoSwitch ? 'Z' : 'z';  // G for activated gyro
225             }
226 #ifdef GYRO_PIN
227             else if (token == T_GYRO_FINENESS) {
228               fineAdjust = !fineAdjust;
229               // imuSkip = fineAdjust ? IMU_SKIP : IMU_SKIP_MORE;
230               runDelay = fineAdjust ? delayMid : delayShort;
231               token = fineAdjust ? 'G' : 'g';  // G for activated gyro
232             } else if (token == T_GYRO_BALANCE) {
233               gyroBalanceQ = !gyroBalanceQ;
234               token = gyroBalanceQ ? 'G' : 'g';  // G for activated gyro
235             } else if (token == T_PRINT_GYRO) {
236               print6Axis();
237             } else if (token == T_VERBOSELY_PRINT_GYRO) {
238               printGyro = !printGyro;
239               token = printGyro ? 'V' : 'v';  // V for verbosely print gyro data
240             } else if (token == T_SLOPE) {
241               slope = -slope;
242               token = slope > 0 ? 'R' : 'r';  // G for activated gyro
243             }
244 #endif
245             break;
246           }
```

Incrementally decrease runDelay.

Incrementally increase runDelay.

```
257         case T_ACCELERATE:
258           {
259             runDelay = max(0, runDelay - 1);
260             break;
261           }
262         case T_DECELERATE:
263           {
264             runDelay = min(delayLong, runDelay + 1);
265             break;
266           }
```

Issue the "rest" command then do shutdown activities.

```
267         case T_REST:
268           {
269             strcpy(newCmd, "rest");
270             if (strcmp(newCmd, lastCmd)) {
271               loadBySkillName(newCmd);
272             }
273             shutServos();
274             gyroBalanceQ = false;
275             manualHeadQ = false;
276             printToAllPorts('g');
277             break;
278           }
```

reaction.h: reaction() function (cont.)

- Code for selected tokens (cont.)
  - The T_Skill is the most used token.
  - We will look at the loadSkillName() function, found in skills.h, next.

```
656         case T_SKILL:
657         {
658           if (!strcmp("x", newCmd)        // x for random skill
659               || strcmp(lastCmd, newCmd)  // won't transform for the same gait.
660               || skill->period <= 1) {    // skill->period can be NULL!
661             // it's better to compare skill->skillName and newCmd.
662             // but need more logics for non skill cmd in between
663             loadBySkillName(newCmd);  // newCmd will be overwritten as dutyAngles then recovered from skill->skillName
664             if (skill->period > 0)
665               printToAllPorts(token);
666             // skill->info();
667           }
668           break;
669         }

670         case T_TASK_QUEUE:
671         {
672           tQueue->createTask();
673           break;
674         }
675         default:
676         {
677           printToAllPorts("Undefined token!");
678           break;
679         }
680       }
```

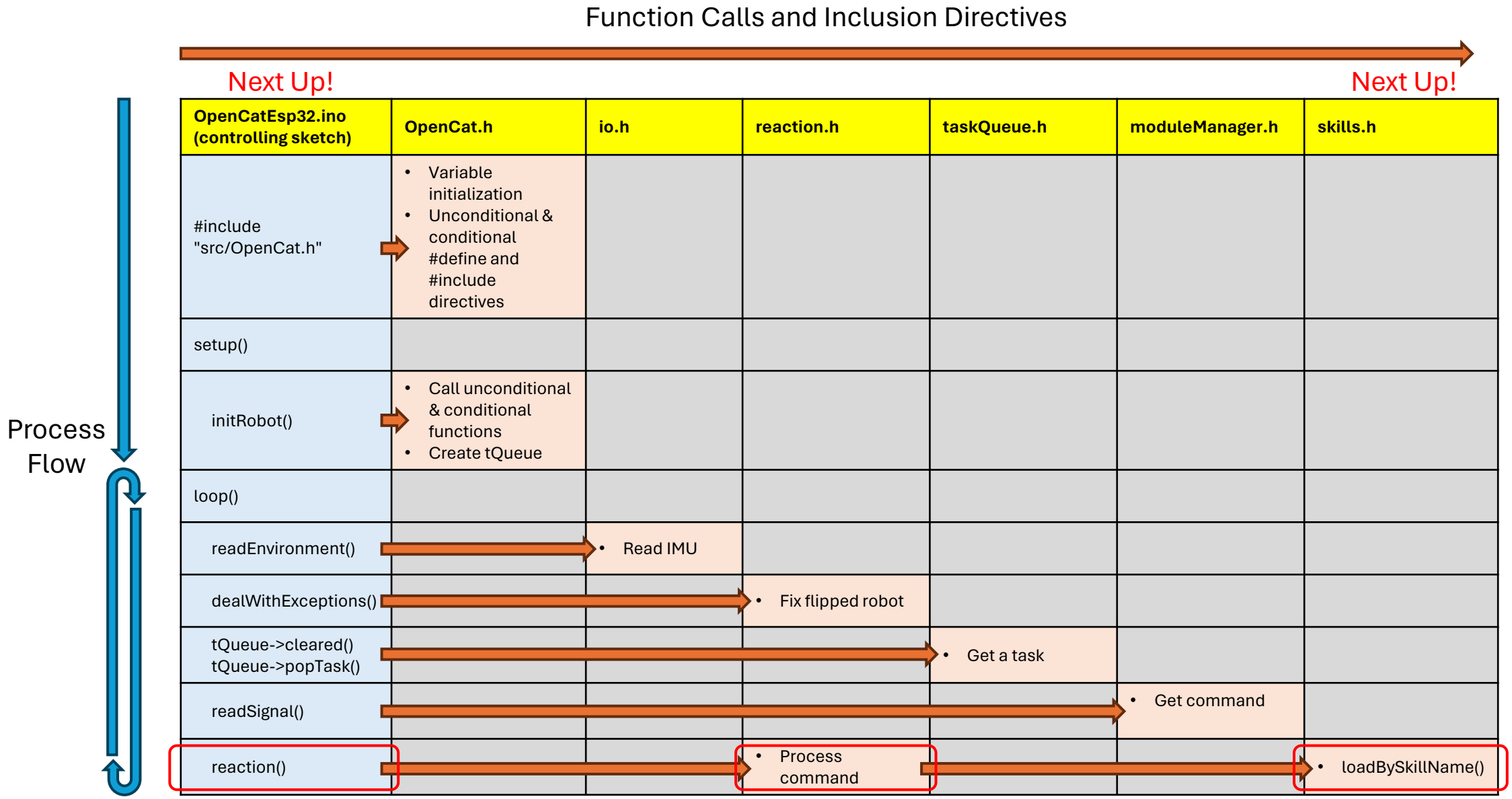
This is a consequence of newCmd having two roles, as stated previously in the "OpenCat.h" section.

At this point, the value of newCmd is the skill name, and it is used to load that skill.

Loads example commands into the task queue (currently skills "vtF" and "up").

Default part of the switch.
This is what happens if the token is unrecognized!

"Sketch" Process Map



skill.h: loadBySkillName() function

```
357  void loadBySkillName(const char* skillName) {  //get lookup information from on-board EEPROM and read the data array from storage
358    int skillIndex = skillList->lookUp(skillName);
359    if (skillIndex != -1) {
360      //if (skill != NULL)
361      //  delete[] skill;
362      char lr = skillName[strlen(skillName) - 1];
363      skill->offsetLR = (lr == 'L' ? 30 : (lr == 'R' ? -30 : 0));
364      skill->buildSkill(skillList->get(skillIndex)->index);
365      strcpy(newCmd, skill->skillName);
366      if (strcmp(newCmd, "calib") && skill->period == 1)
367        protectiveShift = esp_random() % 100 / 10.0 - 5;
368      else
369        protectiveShift = 0;
370  #ifdef GYRO_PIN
371      //keepDirectionQ = (skill->period > 1) ? false : true;
372      thresX = (skill->period > 1) ? 12000 : 8000;
373      thresY = (skill->period > 1) ? 10000 : 6000;
374    //thresZ = (skill->period > 1) ? -8000 : -10000;
375  #endif
376      for (byte i = 0; i < DOF; i++)
377        skill->dutyAngles[i] += protectiveShift;
378      //skill->info();
379      if (lr == 'R' || (lr == 'X' || lr != 'L') && random(100) % 2)
380        skill->mirror();  //randomly mirror the direction of a behavior
381      skill->transformToSkill(skill->nearestFrame());
382  #ifdef NYBBLE
383      for (byte i = 0; i < HEAD_GROUP_LEN; i++)
384        targetHead[i] = currentAng[i] - currentAdjust[i];
385  #endif
386      //    runDelay = delayMid + 2;
387      //skill->info();
388    }
389  }
390
```

Index of skill in PROGMEM (flash memory).

Get the last character of the skill name (use to check for mirrorable skills).

Set the offsetLR, based on that last character (used by mirrored skills like "bkL" and "bkR").

Use the get() function of skillList [a QList of type SkillPreview which holds the list of skills], with the supplied skillIndex, to return a SkillPreview item.
Use the index property of that SkillPreview item to return the skill index which is used in progmemPointer[] by buildSkill().
Note: This returned index "should be" the same as skillIndex above so this line could be written as: skill->buildSkill(skillIndex);

buildSkill() gets the skill from PROGMEM, formats it, and stores it in the skill object.

For mirrorable skills, the default direction is leftward locomotion, 'L', so mirror if "lr" variable is 'R'. Also supports random leftward vs rightward locomotion, when the last character is 'X'.

skill.h: Skill class

- The Skill class is used to create the global *skill* object.
  - As seen on the last slide, the buildSkill() method of this object is used to get skill information from PROGMEM and format it properly for use.

```
74  class Skill {
75  public:
76    char skillName[20];  //use char array instead of String to save memory
77    int8_t offsetLR;
78    int period;  //the period of a skill. 1 for posture, >1 for gait, <-1 for behavior
79    float transformSpeed;
80    byte skillHeader;
81    byte frameSize;
82    int expectedRollPitch[2];  //expected body orientation (roll, pitch)
83    byte angleDataRatio;       //divide large angles by 1 or 2. if the max angle of a skill is >128, all the angls will be divided by 2
84    byte loopCycle[3];         //the looping section of a behavior (starting row, ending row, repeating cycles)
85    byte firstMotionJoint;
86    int8_t* dutyAngles;  //the data array for skill angles and parameters
87
88    Skill(){...}
100
101   void buildSkill(){...}
108
109   void buildSkill(int s) {
110     strcpy(skillName, newCmd);
111     unsigned int pgmAddress = (unsigned int)progmemPointer[s];
112     period = (int8_t)pgm_read_byte(pgmAddress);  //automatically cast to char*
113     for (int i = 0; i < dataLen(period); i++) {
114       newCmd[i] = pgm_read_byte(pgmAddress++);
115     }
116     newCmd[dataLen(period)] = '~';
117     formatSkill();
118   }
```

Skill object properties.

Used when token is 'K'.

Used by loadBySkillName() function in previous slide.

Get skill array and store in newCmd.

Format the skill.

skill.h: Skill class (cont.)

- Here are the methods of the Skill class

```
101     void buildSkill() { ... }
108
109     void buildSkill(int s) { ... }
119
120     ~Skill() { ... }
122
123     int dataLen(int8_t p) { ... }
132
133     void inplaceShift() { ... }
147
148     void formatSkill() { ... }
180
181   #define PRINT_SKILL_DATA
182
183     void info() { ... }
224
225     void mirror() { ... }
241
242     int nearestFrame() { ... }
249
250     void transformToSkill(int frame = 0) { ... }
254
255     void convertTargetToPosture(int* targetFrame) { ... }
270
271     void perform() { ... }
```

Used by loadBySkillName() function

A rather long method to implement the skill, used in the reaction() function of reaction.h

Code Walkthrough Ends!