LOKII CircuitPython

Anson Lee, Edward Chan

Apr 23, 2024

CONTENTS:

CHAPTER

ONE

LOKII CIRCUITPYTHON

LOKII CircuitPython API provides access to LOKII-CE boards functions through the SMART_ARDUINO board circuitpython interface.

Fig. 1: SMART_ARDUINO

1.1 Features

- Face detection
- Color blob detection
- QR code recognition
- Text-To-Speech
- Speech Recognition (Preset keyword groups)
- Speaker Dependent keywords training

- SMART_DEVICE control (SMART_RC, DC motors)
- WIFI Remote Control
- SMART_IO Extender (Extender for additional Digital Input/Output, Analog input)

1.2 SMART_ARDUINO pinout

The SMART_ARDUINO provides following hardware PINs for circuitpython programming which share similar layout as Adafruit feather nrf52840 express board.

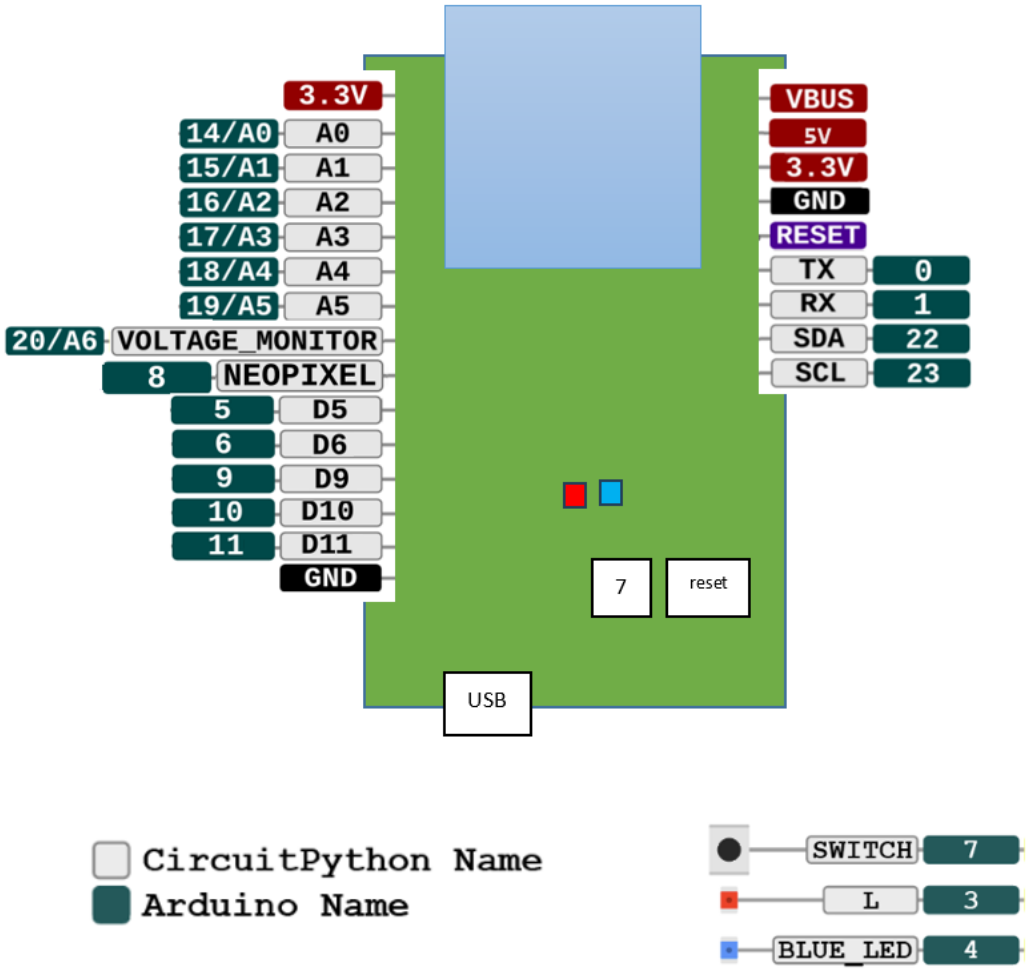


Fig. 2: SMART_ARDUINO pin assignment

1.3 Basic Usage

To use LOKII-CE provided functions from Circuitpython programming language, you can stack SMART_ARDUINO (with CircuitPython firmware) on top of SMART_SHIELD. Then switch SMART_SHIELD's switch-2 to ON position to enable CircuitPython programming.

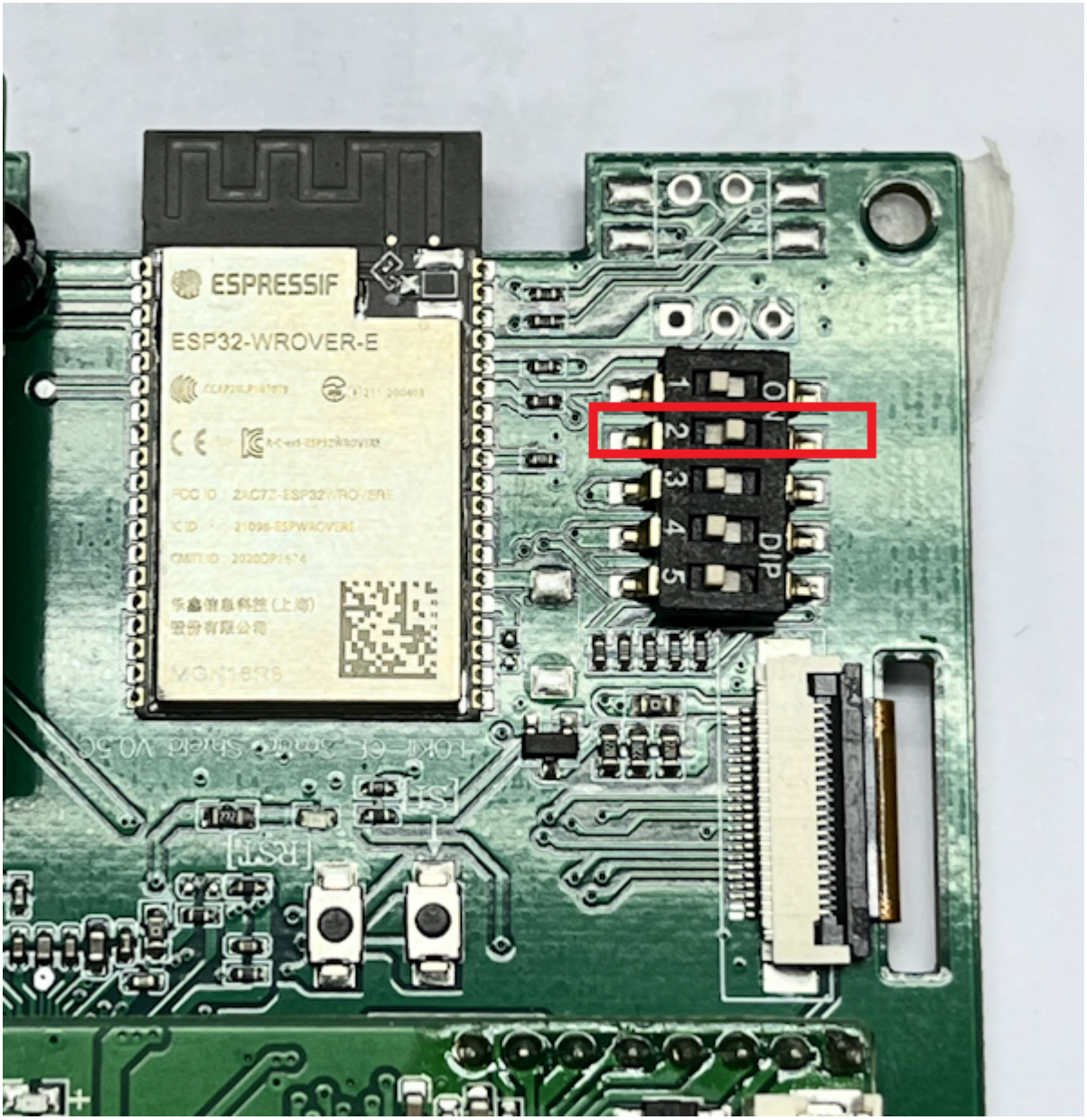


Fig. 3: SMART_SHIELD switch to enable (Arduino/CircuitPython programming)

After connecting SMART_ARDUINO with a Window PC installed with Mu Editor. You can open Mu Editor to start the python programming by importing the LOKII library using these code segment.

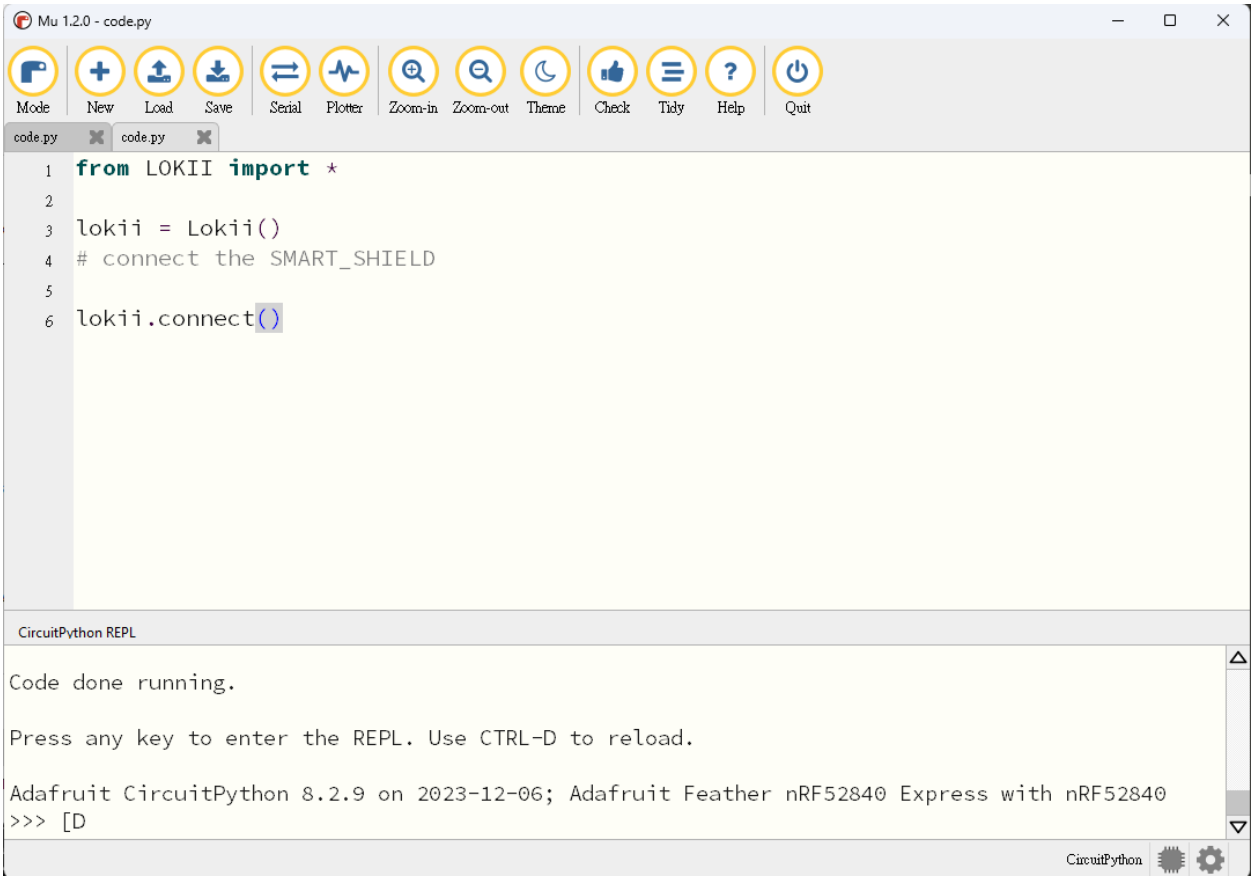


Fig. 4: python code to initialize LOKII SPI connection with SMART_SHIELD

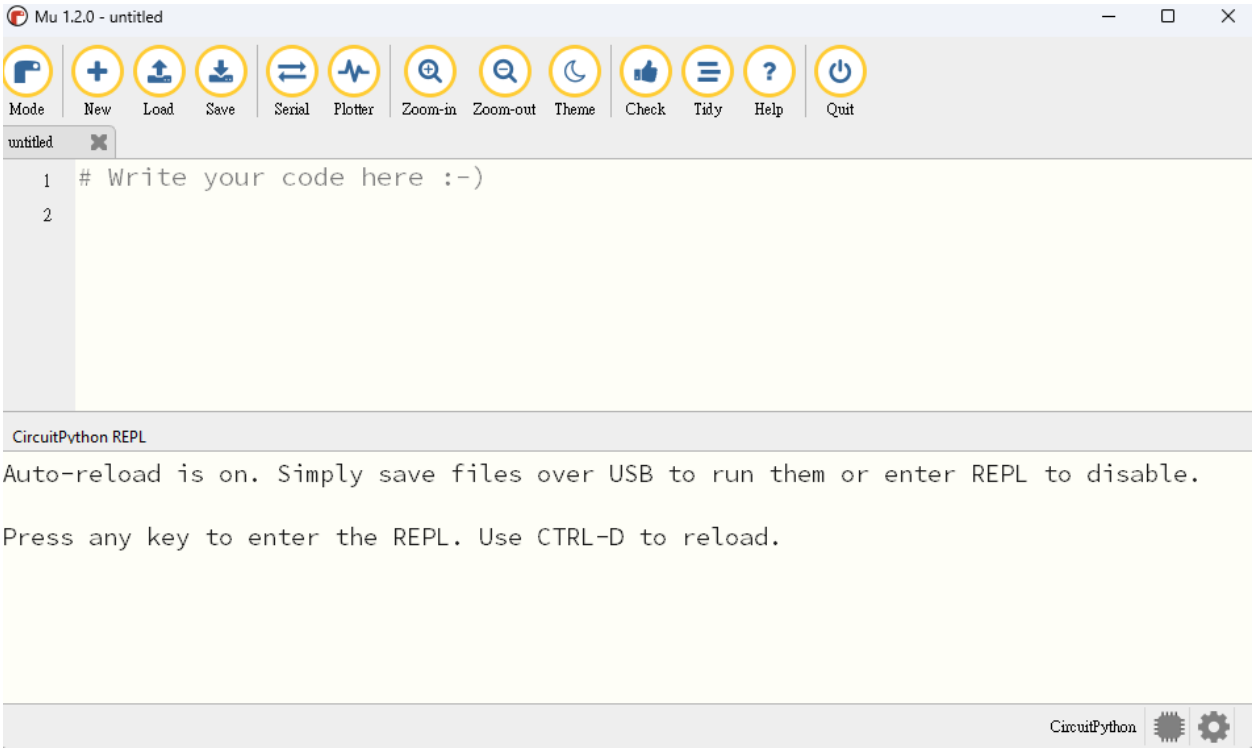
P.S. Mu Editor can be download from https://codewith.mu/

1.4 SMART_ARDUINO CircuitPython firmware update

To update SMART_ARDUINO with CircuitPython firmware (SMART_ARDUINO can either be flashed with Arduino or CircuitPython firmware), you can follow these procedures:

1. Connect SMART_ARDUINO to a Window PC with a USB cable.

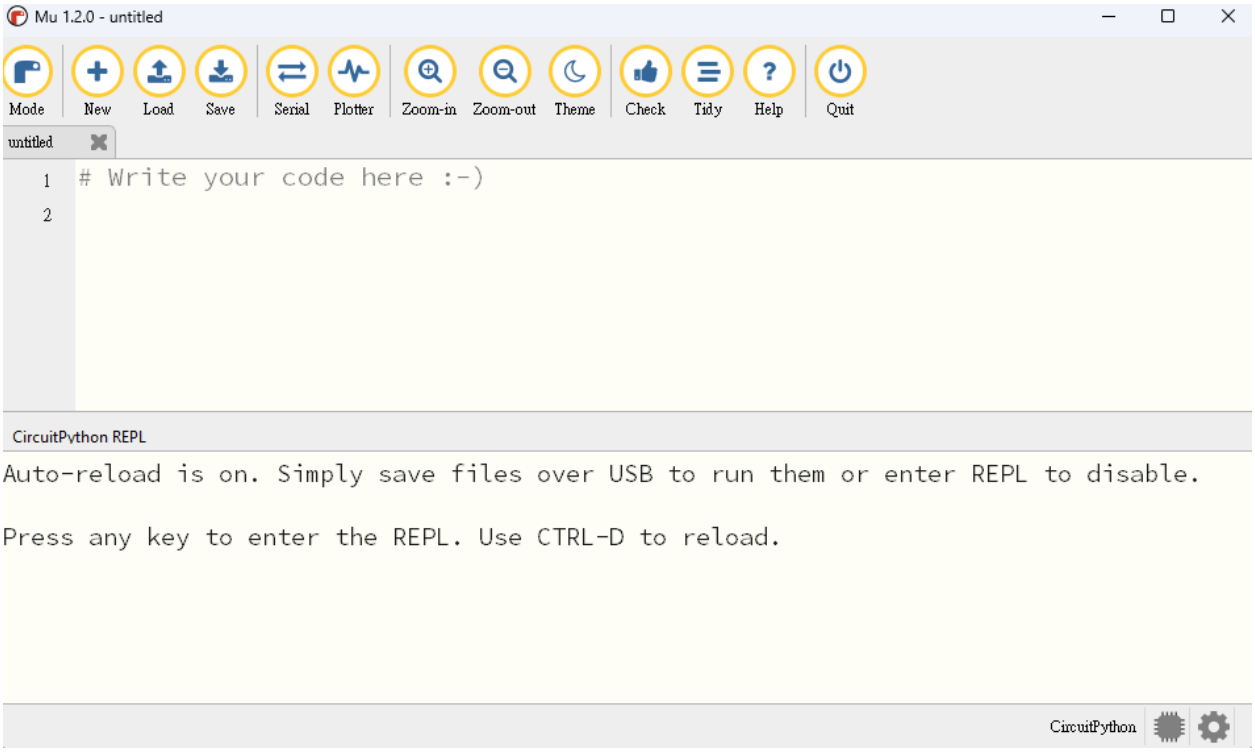
2. Press reset button twice to enter Bootloader mode. The Window PC will show a drive with name "FTHR840BOOT"



3. Download CircuitPython 8.2.10 UF2 from https://circuitpython.org/board/feather_nrf52840_express/

4. Copy and paste the UF2 file to the "FTHR840BOOT" drive. Then, SMART_ARDUINO will refresh itself automatically and a new drive called "CIRCUITPY" will be shown in the Window PC. Once the "CIRCUITPY" shown, it is ready for Circuitpython programming.

5. You can open the Mu editor, the board connection should be enabled at the bottom right corner.



1.5 SMART_ARDUINO CircuitPython Library Installation

Connect SMART_ARDUINO board (with CircuitPython firmware) with a PC. Then copy

- LOKII.mpy
- LOKISPI.mpy
- LOKIIExtender.mpy
- adafruit_bitbangio.mpy

into “CIRCUITPY/lib” directory

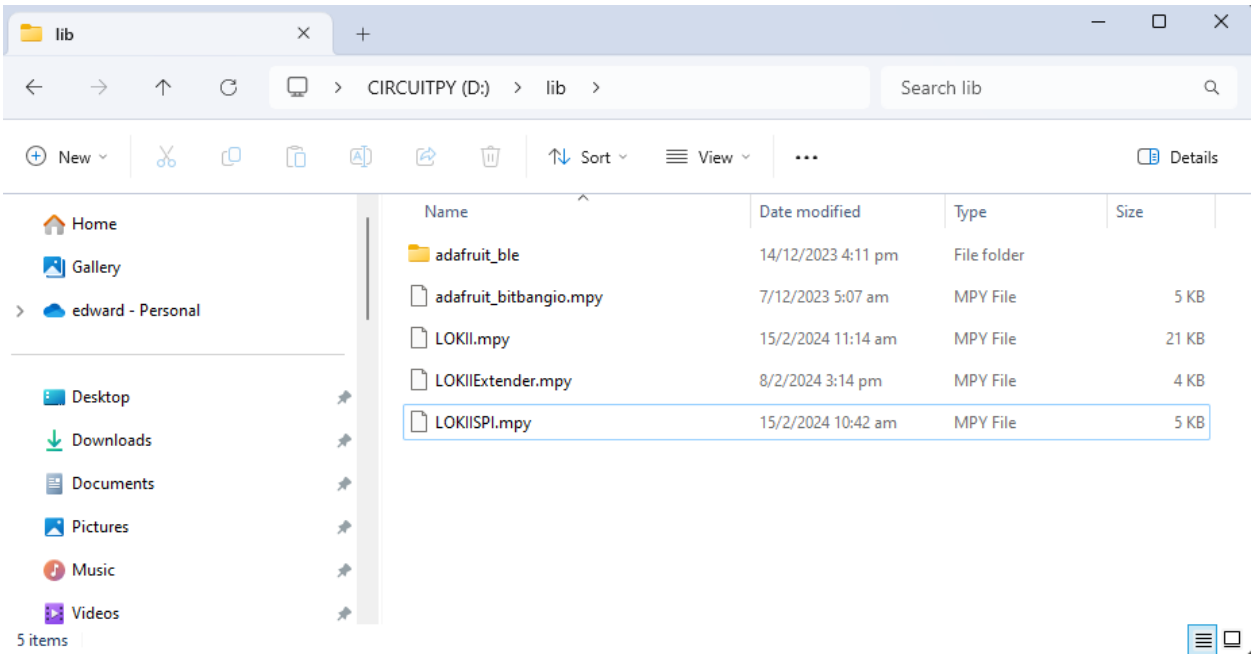


Fig. 5: LOKII Circuitpython library copying path

You can visit www.btobsteam.com to check for latest LOKII CircuitPython libraries or visit www.circuitpython.org/libraries for other hardware and third-parties libraries support.

1.6 Support

You are always welcome to visit www.btobsteam.com/ to check for latest news and library update.

If you are having issues, please free fee to email us: btobsteam@centek.com.hk

CHAPTER

TWO

LOKII CIRCUITPYTHON API REFERENCE

The main class to invoke LOKII functions from SMART_SHIELD.

class LOKII.**Lokii**

LOKII provide methods to invoke LOKII-CE functions, such as image processing, sound processing, multi-media functions, motor control, and read/write I/O devices.

connect()

Initials the LOKII system with the SPI bus. Check version of Lokii circuitpython library.

playMIDI(*note*)

plays MIDI note for a second.

Parameters

note (*Integer*) – MIDI values (0 - 59);

- 0 = NOTE_C1.
- 1 = NOTE_C1S.
- 2 = NOTE_D1.
- 3 = NOTE_E1b.
- 4 = NOTE_E1.
- 5 = NOTE_F1.
- 6 = NOTE_F1S.
- 7 = NOTE_G1.
- 8 = NOTE_G1S.
- 9 = NOTE_A2.
- 10 = NOTE_B2b.
- 11 = NOTE_B2.
- 12 = NOTE_C2.
- 13 = NOTE_C2S.
- 14 = NOTE_D2.
- 15 = NOTE_E2b.
- 16 = NOTE_E2.
- 17 = NOTE_F2.
- 18 = NOTE_F2S.

- 19 = NOTE_G2.
- 20 = NOTE_G2S.
- 21 = NOTE_A3.
- 22 = NOTE_B3b.
- 23 = NOTE_B3.
- 24 = NOTE_C3.
- 25 = NOTE_C3S.
- 26 = NOTE_D3.
- 27 = NOTE_E3b.
- 28 = NOTE_E3.
- 29 = NOTE_F3.
- 30 = NOTE_F3S.
- 31 = NOTE_G3.
- 32 = NOTE_G3S.
- 33 = NOTE_A4.
- 34 = NOTE_B4b.
- 35 = NOTE_B4.
- 36 = NOTE_C4.
- 37 = NOTE_C4S.
- 38 = NOTE_D4.
- 39 = NOTE_E4b.
- 40 = NOTE_E4.
- 41 = NOTE_F4.
- 42 = NOTE_F4S.
- 43 = NOTE_G4.
- 44 = NOTE_G4S.
- 45 = NOTE_A5.
- 46 = NOTE_B5b.
- 47 = NOTE_B5.
- 48 = NOTE_C5.
- 49 = NOTE_C5S.
- 50 = NOTE_D5.
- 51 = NOTE_E5b.
- 52 = NOTE_E5.
- 53 = NOTE_F5.
- 54 = NOTE_F5S.

- 55 = NOTE_G5.
- 56 = NOTE_G5S.
- 57 = NOTE_A6.
- 58 = NOTE_B6b.
- 59 = NOTE_B6.

`checkAudioStatus()`

checks the audio status.

Returns
returns the audio status: 1 = audio is playing, 0 = no audio is playing, -1 = command sent fails.

Return type
Integer

`setVolume(volume)`

sets the sound output volume level from 0-100.

Parameters
volume (*Integer*) – volume level (0-100).

`playTTS(text, voiceType=0, speed=5, pitch=5, emotion=0)`

play text-to-speech with specified voice, speed, pitch, emotion settings for an input text.

Parameters

- **text** (*String*) – english text.
- **voiceType** (*Integer*) – Character's voice;
  - 0 = L_DEFAULT.
  - 1 = L_MAN.
  - 2 = L_OLDMAN.
  - 3 = L_OLDWOMAN.
  - 4 = L_BOY.
  - 5 = L_YOUNGGIRL.
  - 6 = L_ROBOT.
  - 7 = L_GIANT.
  - 8 = L_DWARF.
  - 9 = L_ALIENT.
- **speed** (*Integer*) – from 1 - 10.
- **pitch** (*Integer*) – from 1 - 10.
- **emotion** (*Integer*) –
  - 0 = E_NATURAL.
  - 1 = E_FRIENDLY.
  - 2 = E_ANGRY.
  - 3 = E_FURIOUS.

- 4 = E_DRILL.
- 5 = E_SCARED.
- 6 = E_EMOTIONAL.
- 7 = E_WEEPY.
- 8 = E_EXCITED.
- 9 = E_SURPRISED.
- 10 = E_SAD.
- 11 = E_DISGUSTED.
- 12 = E_WHISPER.

`getSpeechResult()`

Uses with startSpeechRecognize(int wordgroupIndex) to get the array index of the recognized keyword immediately, i.e non-Blocking.

Returns
returns an array index of the keyword or -1 for no speech recognized.

Return type
Integer

`startSpeechRecognize(`*wordgroupIndex*`)`

starts/stops speech recognition by selecting trained word groups. Please wait for 2 seconds for this function to sample ambient sounds before speaking any keywords. This function can't be used with functions that have an audio output simultaneously.

Parameters
wordgroupIndex (*Integer*) – trained word group index from 1 - 20. 0 stops the speech recognition.

- 1 is number group for ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10"].
- 2 is action group for ["Action", "Move", "Turn", "Run", "Look", "Attack", "Stop", "Hello"].
- 3 is movement group for ["Turn Left", "Turn Right", "Move Up", "Move Down", "Go Forward", "Go Backward"].
- 4 is command group for ["Tell me a joke", "play me a song", "stop the music", "take a photo", "show me a photo", "track my face", "follow the ball", "record motor motion", "playback motor", "list commands"].
- 11-20 starts speech recognition in custom training data (Speaker Dependent).

`waitForSpeechResult()`

waits for speech recognition (BLOCKING) result and gets the array index of the recognized keyword. This method is called after startSpeechRecognize(int wordgroupIndex) to get the array index of the recognized keyword when the process of speech recognition is completed, i.e Blocking.

Returns
returns an array index of the keyword or -1 for no speech recognized.

Return type
Integer

`setCameraMode`(*cameraState*)

sets LOKII camera to perform image processing functions or multimedia functions.

Parameters

`cameraState` (*Integer*) – defines the image processing functions

- L_CAM_RECOGNIZE_RGB, performs Red/Green/Blue/custom color detection mode.
- L_CAM_RECOGNIZE_CUSTOM, recognize camera center region as a new custom color.
- L_CAM_FACE_DETECT, performs frontal face detection mode.
- L_CAM_PREVIEW, sets camera to preview mode, no image processing is performed.
- L_CAM_QRCODE, performs QR code scan mode.
- L_CAM_DIGITALVIDEO_MODE, performs multi-media functions, such as taking photo or taking video.
- L_GESTURE, performs gesture detection mode.
- L_POSTURE, performs posture detection mode.

`getFaceResult`(*attributeType*)

gets the face detection result from the cache after waitForFaceResult(int faceState) function is activated.

Parameters

`attributeType` (*Integer*) – Face result attributes;

- L_XPOS, x coordinate of the face centre.
- L_YPOS, y coordinate of the face centre.
- L_WIDTH, width of the face boundary.
- L_HEIGHT, height of the face centre.

Returns

returns one of the following face detected attributes above. All attributes are based on the LCD screen coordinate system with 320 x 240 pixel.

Return type

Integer

`waitForFaceResult`(*isBlocking*)

waits and returns face detection result after setCameraMode(L_CAM_FACE_DETECT) is activated. After this function call, a copy of the face result will be cached.

Parameters

`isBlocking` (*Integer*) – defines the face detection behavior. 0, this function will return the result immediately (NON-BLOCKING). 1, this function will hold until until a face is detected (BLOCKING).

Returns

face area size (in pixel) or 0 when face not detected (in NON-BLOCKING mode only).

Return type

Integer

`getBlobCount`()

gets number of color blob detected result from the cache after waitForBlobResult(isBlocking) function is activated.

Returns

returns the number of color blob in the cache.

Return type
Integer

`waitForBlobResult`(*isBlocking*)

waits and returns color blob detection result after setCameraMode(L_CAM_RECOGNIZE_RGB) or setCameraMode(L_CAM_RECOGNIZE_CUSTOM) is activated. After this function call, a copy of the color result will be cached.

Parameters
isBlocking (*boolean*) – defines the color blob detection behavior. 0, this function will return immediately regardless of color blob detected (NON-BLOCKING). 1, this function will hold until at least one color objected is detected (BLOCKING).

Returns
returns number of color object detected, 0 - nothing detected, -1 - command sent fails.

Return type
Integer

`getBlobResult`(*blobIndex*, *attributeType*)

gets the color blob detected result from the cache after waitForBlobResult(bool isBlocking) function is activated. The color blob results are already sorted in descending order, i.e. blob in index 0 is always the biggest color blob detected in current environment.

Parameters

- **blobIndex** (*Integer*) – the index (starting from zero) for the color blob (The index should be smaller than the total color object detected).
- **attributeType** (*Integer*) – Color object attributes;
  - L_XPOS, x coordinate of the color blob frame (upperleft corner).
  - L_YPOS, y coordinate of the color blob frame (upperleft corner).
  - L_WIDTH, width of the color blob boundary.
  - L_HEIGHT, height of the color blob centre.
  - L_COLOR, color of the color blob, such as L_RED_COLOR, L_GREEN_COLOR, L_BLUE_COLOR, L_CUSTOM_COLOR.

Returns
returns one of the color attributes above. All the attributes are based on the LCD screen coordinate system with 320 x 240 pixel.

Return type
Integer

`getQRResult`()

Get QR code's string result.

Returns
returns QR's string.

Return type
String

`waitForQRResult`(*isBlocking=True*)

starts QR code detection.

Parameters
isBlocking (*boolean*) – False - non-blocking (returns the result immediately from current

live camera image), True - blocking (starts QR code detection and waits until QR code is recognized).

Returns
returns 1 - success, 0 - no QR code scanned.

Return type
Integer

`waitForGestureResult`(*isBlocking*)

starts gesture detection.

Parameters
isBlocking (*boolean*) – False - non-blocking (returns the result immediately from current live camera image), True - blocking (returns the result when a hand is detected).

Returns
returns number of hands detected - success, 0 - no hands detected.

Return type
Integer

`getGestureResult`(*handIndex*, *attributeType*)

gets hand attributes from the cache after waitForGestureResult(bool isBlocking) function is activated.

Parameters

- **handIndex** (*Integer*) – the index (starting from zero)
- **attributeType** (*Integer*) – hand attributes
  - HAND_XPOS, x coordinate of the frame (upperleft corner).
  - HAND_YPOS, y coordinate of the frame (upperleft corner).
  - HAND_WIDTH, width of the frame.
  - HAND_HEIGHT, height of the frame.
  - HAND_FINGERCOUNT, number of hands.

Returns
returns one of the attributes above. All the attributes are based on the LCD screen coordinate system with 320 x 240 pixel.

Return type
Integer

`waitForPostureResult`(*isBlocking*)

starts posture detection.

Parameters
isBlocking (*boolean*) – False - non-blocking (returns the result immediately from current live camera image), True - blocking (returns the result when a human is detected).

Returns
returns number of people detected - success, 0 - no humans detected.

Return type
Integer

`getPostureResult`(*poseIndex*, *attributeType*)

gets human attributes from the cache after waitForPostureResult(bool isBlocking) function is activated.

Parameters

- **poseIndex** (*Integer*) – index (starting from zero)
- **attributeType** (*Integer*) – human attributes
  - POSE_XPOS, x coordinate of the frame (upperleft corner).
  - POSE_YPOS, y coordinate of the frame (upperleft corner).
  - POSE_WIDTH, width of the frame.
  - POSE_HEIGHT, height of the frame.
  - POSE_LEFTHANDCODE, position of left hand.
  - POSE_RIGHTHANDCODE, position of right hand.

The coordinate system for POSE_LEFTHANDCODE/POSE_RIGHTHANDCODE:

1 | 2 | 3

4 | 5 | 6

7 | 8 | 9 (LCD Screen)

Returns

returns one of the attributes above. All the attributes are based on the LCD screen coordinate system with 320 x 240 pixel, except for POSE_LEFTHANDCODE and POSE_RIGHTHANDCODE.

Return type

Integer

`getPostureFeatures`(*poseIndex*, *ptIndex*)

gets human posture feature points from the cache after waitForPostureResult(bool isBlocking) function is activated.

Parameters

- **poseIndex** (*Integer*) – index (starting from zero)
- **ptIndex** (*Integer*) – human body
  - POSE_XPOS, x coordinate of the frame (upperleft corner).
  - POSE_YPOS, y coordinate of the frame (upperleft corner).
  - POSE_WIDTH, width of the frame.
  - POSE_HEIGHT, height of the frame.
  - POSE_LEFTHANDCODE, position of left hand.
  - POSE_RIGHTHANDCODE, position of right hand.
  - ES_NOSE, coordinates of nose.
  - ES_L_EYE, coordinates of left eye.
  - ES_R_EYE, coordinates of right eye.
  - ES_L_EAR, coordinates of left ear.
  - ES_R_EAR, coordinates of right ear.
  - ES_L_SHOULDER, coordinates of left shoulder.

- ES_R_SHOULDER, coordinates of right shoulder.
- ES_L_ELBOW, coordinates of left elbow.
- ES_R_ELBOW, coordinates of right elbow.
- ES_L_WRIST, coordinates of left wrist.
- ES_R_WRIST, coordinates of right wrist.
- ES_L_HIP, coordinates of left hip.
- ES_R_HIP, coordinates of right hip.
- ES_L_KNEE, coordinates of left knee.
- ES_R_KNEE, coordinates of right knee.
- ES_L_ANKLE, coordinates of left ankle.
- ES_R_ANKLE, coordinates of right ankle

Returns
returns coordinates (x,y) of one of the attributes above. All the attributes are based on the LCD screen coordinate system with 320 x 240 pixel.

Return type
tuple

`setDCMotor`(*motorid*, *speed*, *direction*)

moves DC motor attached on SMART_POWER.

Parameters

- **motorid** (*Integer*) – 0 - 3 for SMART_POWER connection.
- **speed** (*Integer*) – 0 - 100.
- **direction** (*Integer*) – 0 - clockwise, 1 - anti-clockwise.

Returns
returns 0 - success, -1 - fail.

Return type
Integer

`setallDCMotors`(*motorid*, *speed1*, *speed2*, *speed3*, *speed4*)

moves all DC motors connected to SMART_POWER at the same time.

Parameters

- **motorid** (*Integer*) – the first motor id.
- **speed1** (*Integer*) – speed of the first DC motor (0 - 20), 0 = the fastest, 1 = the slowest.
- **speed2** (*Integer*) – speed of the second DC motor(0 - 20), 0 = the fastest, 1 = the slowest.
- **speed3** (*Integer*) – speed of the third DC motor (0 - 20), 0 = the fastest, 1 = the slowest.
- **speed4** (*Integer*) – speed of the fourth DC motor (0 - 20), 0 = the fastest, 1 = the slowest.

`setRCServo`(*motorid*, *speed*, *position*)

moves RC servo (E.g MG90S, MG996R) attached on SMART_RC board.

Parameters

- **motorid** (*Integer*) – the servo id.

- **speed** (*Integer*) – speed (0 - 20), 0 = the fastest, 1 = the slowest.
- **position** (*Integer*) – 0, power off. 1 - 250, servo positions.

Returns
returns 0 - success, -1 - fail.

Return type
Integer

`setRCServoCompletionTime`(*motorid*, *timeMS*, *position*)

moves RC servo to a desired position within a requested time.

Parameters

- **motorid** (*Integer*) – the servo id.
- **timeMS** (*Integer*) – time (ms).
- **position** (*Integer*) – 0, power off. 1 - 250, servo positions.

Returns
returns 0 - success, other value - fail.

Return type
Integer

`setSmartDeviceAddress`(*id*)

Sets SMART_DEVICE id for the SMART_DEVICE connected in SMART_BUS. To run this function, make sure only one SMART_DEVICE board is attached in SMART_BUS. If the SMART_DEVICE is a composite SMART_DEVICE, i.e. SMART_RC board, the id will be assigned to the first socket (J1), and the remaining socket J2, J3 ,J4 will be automatically assigned with (id+1), (id+2),(id+3) respectively.

Parameters
id (*Integer*) – servo id. ID must be greater than 7.

`getSmartDeviceCount`()

Gets number of Smart devices attached in SMART_BUS.

Returns
returns number of smart device in the bus, -1 - error.

Return type
Integer

`showSmartDeviceStatus`(*isTrue*)

Show smart devices id, types and attributes connected in SMART_BUS and display them on LCD.

Parameters
isTrue (*boolean*) – display ON/OFF.

`recordVideo`(*filename*)

Starts H.264 video recording in 480x360 resolution and saves it to internal flash storage. The maximum recording duration is 60 seconds. Video recording can't proceed when the storage is less than 30 MB.

Parameters
filename (*String*) – video name must be ended with ".avi". E.g "video.avi".

Returns
returns 1 - success, 0 - fail.

Return type
Integer

playVideo(*filename*)

plays a H.264 video file from the LOKII internal flash storage. This function can only play H.264 video encoded by LOKII-CE boards or through the LOKII-CE video conversion tool.

Parameters
filename (*String*) – the input video name with file extension ".avi". E.g "video.avi".

Returns
returns 1 for success, 0 for error.

Return type
Integer

stopRecordVideo()

stops the video recording, the recorded video will be saved in LOKII internal flash storage.

Returns
returns 0 for success.

Return type
Integer

stopPlayVideo()

stops the video playback.

Returns
returns 0 for success.

Return type
Integer

checkVideoStatus()

checks if LOKII is playing a video or not.

Returns
returns status: 1 - a video is playing, 0 - no video is playing.

Return type
Integer

takePhoto(*filename*)

takes a photo with 1280x720 resolution in JPEG format and saves it in LOKII internal flash storage.

Parameters
filename (*String*) – a jpeg photo filename. E.g "a.jpg".

Returns
1 - success, 0 - fail.

Return type
Integer

displayPhoto(*filename*)

displays a jpeg photo stored in the LOKII internal flash to the LCD Screen. The maximum resolution of jpeg file is 1280x720.

Parameters
filename (*String*) – the jpeg photo filename. E.g "a.jpg".

Returns
returns 1 - success, 0 - fail.

Return type
Integer

`recAudio`(*filename*)

starts audio recording and save it to LOKII internal flash storage. The maximum recording duration is 60 seconds. Audio recording can't proceed when the storage is less than 10 MB.

Parameters
filename (*String*) – filename string, e.g "a.wav".

Returns
returns 0 - fail, 1 - success.

Return type
Integer

`stopRecAudio`()

stops audio recording.

Returns
returns 0 for success.

Return type
Integer

`playSoundFile`(*filename*, *isBlocking*)

plays a wav or mp3 sound file stored in the LOKII internal flash storage.

Parameters

- **filename** (*String*) – sound file name, e.g "a.mp3".
- **isBlocking** (*boolean*) – False - non-blocking, True - blocking (waits until the sound file playback completed).

Returns
return status: 1 = sound file is playing, no sound file is playing.

Return type
Integer

`stopSound`()

Stops sound playback.

`createSDGroup`(*groupIndex*, *numKeywords*)

creates a speech recognition with a keyword groups index (limited to 11-20). Number of keywords affects the accuracy of speech recognition. No more than 5 keywords is recommended. This speech recognition is speaker dependent for the training data. Any new keyword groups will contain the keywords of the "number group" automatically, so the index of new keywords starting from 11.

Parameters

- **groupIndex** (*Integer*) – the index for training word groups (11-20 inclusively).
- **numKeywords** (*Integer*) – the number of keywords needed to train (5 maximum).

Returns
returns 1 - success , 0 - fail.

Return type
Integer

trainSDkeyword(*groupIndex*, *keywordsIndex*)

trains a speaker dependent audio on a keywords group created by createSDGroup(int groupIndex, int numKeywords) using LOKII built-in microphone. This speech recognition is speaker dependent for the training data. Please wait for 2 seconds after calling this function before speaking out the voice. It will perform three training sessions in order to complete the sampling for each keyword phrase. The whole keyword group needs to resample if any keywords is failed to sample. At the end, user can use checkSDComplete() to double-check to make sure the corresponding keyword group is created successfully.

Parameters

- **groupIndex** (*Integer*) – the group index of the created keywords group (11-20 inclusively).
- **keywordsIndex** (*Integer*) – the array index of the keyword.

Returns

returns 1 - complete , 0 - incomplete (it is recommended to train for at least 3 audio samples to get a better training result).

Return type

Integer

checkSDComplete(*groupIndex*)

checks if created keyword group training is completed, i.e. all the keywords audio training is completed in the SD keyword group.

Parameters

groupIndex (*Integer*) – the index of the created keyword group (11-20 inclusively).

Returns

1 - complete (All keywords are well-trained), 0 - incomplete.

Return type

Integer

clearLCD()

Clears text on LOKII's LCD.

Returns

returns 1 - success, 0 - fail.

Return type

Integer

lcdPrint(*showString*, *lineNum*)

Shows text on LOKII's LCD.

Parameters

- **showString** (*String*) – maximum length is 32.
- **lineNum** (*Integer*) – a row number between 1 - 14.

Returns

returns 1 - success, 0 - fail.

Return type

Integer

lcdPrintAll(*showString1*, *showString2*, *showString3*, *showString4*, *showString5*)

shows texts on multiple rows of LOKII's LCD.

Parameters

- **showString1** (*String*) – a string on row 10, maximum length is 32.
- **showString2** (*String*) – a string on row 11, maximum length is 32.
- **showString3** (*String*) – a string on row 12, maximum length is 32.
- **showString4** (*String*) – a string on row 13, maximum length is 32.
- **showString5** (*String*) – a string on row 14, maximum length is 32.

Returns
returns 1 - success, 0 - fail.

Return type
Integer

gamepadDirection(*direction*, *digitalORanalog*)

reads LOKII Remote Control gamepad Joystick status.

Parameters

- **direction** (*Integer*) – 0 = joystick X1, 1 = joystick Y1, 2 = joystick X2, 3 = joystick Y2.
- **digitalORanalog** (*Integer*) – 0 = reads digital value, 1 = reads analog value.

Returns
0, 1 or -1 for digital settings, -100~100 for analog settings.

Return type
Integer

gamepadKey(*button*, *pressedORreleased*)

reads LOKII Remote Control gamepad key status

Parameters

- **button** (*Integer*) – 0 = button A, 1 = button B, 2 = button X, 3 = button Y, 4 = button F1, 5 = button F2, 6 = button F3, 7 = button F4.
- **pressedORreleased** (*Integer*) – defines high state of a button. 0 = pressed button turns on, 1 = buttons turns on when released.

Returns
returns input status.

Return type
Boolean

class SmartExtender.**smartExtender**

SmartExtender provides functions for controlling SMART_IO Extender 2 via SMART_ARDUINO.

initExtSoftSPI(*softCSPin=board.D11*, *softCS1Pin=board.D5*, *softMISOPin=board.D10*, *softMOSIPin=board.D9*, *softCLKPin=board.D6*)

Initiates a SPI communication bridge between SMART_ARDUINO and LOKII I/O Extender (IO Extender 2). Five hardware PINs are required for SMART_ARDUINO to connect with SMART_IO Extender. For example, D11, D5,D6, D9, D10. Hardware SPI pins and D13 are occupied by LOKII-CE.

Parameters

- **softCSPin** (*board*) – The corresponding SMART_ARDUINO's pin connects to SPI_CS of I/O Extender
- **softCS1Pin** (*board*) – The corresponding SMART_ARDUINO's pin connects to SPI_CS_1 of I/O Extender for reading analog pins

- **softMISOPin** (*board*) – The corresponding SMART_ARDUINO's pin connects to SPI_MISO of I/O Extender
- **softMOSIPin** (*board*) – The corresponding SMART_ARDUINO's pin connects to SPI_MOSI of I/O Extender
- **softCLKPin** (*board*) – The corresponding SMART_ARDUINO's pin connects to SPI_SCK of I/O Extender

Returns
0 - success

Return type
Integer

extIOSPISelfTest()

Test smartExtender connection. A SPI communication test on connection between SMART_IO Extender and SMART_ARDUINO.

Returns
0 - success, -1 - fail

Return type
Integer

writeGPIO(*ioNumIn*, *low_or_high*)

Writes a LOKII I/O Extender digital pin.

Parameters

- **ioNumIn** (*Integer*) – a pin number of I/O Extender from 0 to 11. 0~11 => pin D0~D11
- **low_or_high** (*Integer*) – 0 = low. 1 = high.

setGPIO(*ioNum*, *in_or_out*)

Sets a LOKII I/O Extender digital pin to INPUT or OUTPUT.

Parameters

- **ioNum** (*Integer*) – a pin number of I/O Extender from 0 to 11. 0~11 => pin D0~D11
- **in_or_out** (*Integer*) – 0 for input. 1 for output

readGPIO(*ioNumIn*)

Reads status of a LOKII I/O Extender pin from SMART_ARDUINO cache

Parameters
ioNumIn (*Integer*) – a pin number of I/O Extender from 0 to 15. 0~11 => pin D0~D11, 12~15 => pin A0~A3.

Returns
0 or 1 for digital pin, 0~4095 for analog pin, 0xFFFF if value is invalid.

Return type
Integer

syncGPIO()

Synchronize the current LOKII I/O Extender I/O status with SMART_ARDUINO, so that subsequent reading of the I/O pins can be directly copied from the SMART_ARDUINO cache.

CHAPTER

THREE

EXAMPLE PROGRAMS

3.1 Setup

To run the examples, make sure you have stacked the SMART_ARDUINO board (flashed with Circuitpyton firmware) with the LOKII SMART_SHIELD. Then copy the Lokii.mpy into "lib" folder under the circuitpython mass storage drive.

3.2 Examples

3.2.1 QR Code Detection

This program sets LOKII-CE board into QR detection mode and recognize the QR code (Version 2) from the camera. When the code is detected, it will speak out the content using the built-in Text-To-Speech function.

```
################################
# QRCode demo
# support QR Code Version 2 (Model 2)
# https://qr.calm9.com/en/
################################
import time
from LOKII import *
lokii = Lokii()


def QRDetectSample():
    qrCode = ""
    if (lokii.waitForQRResult() is True):
        qrCode = lokii.getQRResult()
        lokii.setCameraMode(L_CAM_PREVIEW)
        lokii.playTTS(qrCode, L_DEFAULT, 5)
        print(qrCode)
        lokii.setCameraMode(L_CAM_QRCODE)

lokii.connect()

lokii.playTTS("QR Code Demo", L_DEFAULT, 5)
time.sleep(3.0)
lokii.setCameraMode(L_CAM_QRCODE)
```

(continues on next page)

(continued from previous page)

```
while True:
    QRDetectSample()
    time.sleep(0.1)
```

3.2.2 Audio Recording Demo

This program sets LOKII-CE board to record 10 seconds audio from the built-in microphone. Then it will playback the audio recorded from the speaker.

```
##################################
# Record and playback audio demo
##################################
import time
from LOKII import *

lokii = Lokii()
lokii.connect()

lokii.setCameraMode(L_CAM_PREVIEW)


print(lokii.checkAudioStatus())

lokii.playTTS("Record an Audio", 0, 5, 5, 0)
lokii.recAudio("b.wav")
time.sleep(10.0)
lokii.stopRecAudio()

lokii.playTTS("audio saved", 0, 5, 5, 0)
time.sleep(1)

lokii.playSoundFile("b.wav",1)
time.sleep(10)
lokii.stopSound()
```

3.2.3 Video Recording and Take Photo

This program sets LOKII-CE board into DV mode to take a photo and record a video. Following each action, it will show the result on the LCD.

```
##################################
# record video /photo  and playback demo
##################################
import time
from   LOKII import *


lokii = Lokii()
lokii.connect()
```

(continues on next page)

(continued from previous page)

```
lokii.setCameraMode(L_CAM_DIGITALVIDEO_MODE)
time.sleep(3.0)


lokii.playTTS("Take a photo", 0, 5, 5, 0)
time.sleep(1.0)
lokii.takePhoto("a.jpg")

lokii.playTTS("Display photo for 5 seconds", 0, 5, 5, 0)
lokii.displayPhoto("a.jpg")
time.sleep(5.0)

lokii.playTTS("Record Video", 0, 5, 5, 0)
lokii.recordVideo("video6.avi")
time.sleep(5.0)
lokii.stopRecordVideo()

lokii.playTTS("play back Video", 0, 5, 5, 0)
lokii.playVideo("video6.avi")

time.sleep(5.0)
lokii.stopPlayVideo()
```

3.2.4 Blockly Gamepad Demo

This program shows how LOKII-CE reads Gamepad inputs from Blockly via SMART_ARDUINO. To go to the Blockly interface of your LOKII-CE board, enter the broadcast address on a browser.

```
import time
from LOKII import *

lokii = Lokii()
lokii.connect()

lokii.setCameraMode(L_CAM_PREVIEW)
while True:
    x1 = lokii.gamepadDirection(0,1)
    y1 = lokii.gamepadDirection(1,1)
    x2 = lokii.gamepadDirection(2,1)
    y2 = lokii.gamepadDirection(3,1)

    a = lokii.gamepadKey(0,0)
    b = lokii.gamepadKey(1,0)
    x = lokii.gamepadKey(2,0)
    y = lokii.gamepadKey(3,0)
    f1 = lokii.gamepadKey(4,0)
    f2 = lokii.gamepadKey(5,0)
    f3 = lokii.gamepadKey(6,0)
    f4 = lokii.gamepadKey(7,0)
```

(continues on next page)

(continued from previous page)

```
    print("Gamepad X1:{} Y1:{} X2:{} Y2:{}".format(x1,y1, x2,y2)  )
    print("GameKey A:{} B:{} X:{} Y:{} F1: {} F2:{} F3: {} F4: {}".format(a,b,x,y,f1,f2,
↪f3,f4)  )
    time.sleep(0.5)
```

3.2.5 Plot Analog Input

This program shows how to plot analog values read from an analog pin on Mu Editor.

```
#########################################################
#  Analog Input Demo
#  Connect board.A0, board.A1 with either "3.3V" or "GND",
#  open the Mu editor's Plotter to view the AD value change
#########################################################

import time
import board
from analogio import AnalogIn

# AD value varies from 0 - 65535  for Voltage 0 - 3.3V
analog_in0 = AnalogIn(board.A0)
analog_in1 = AnalogIn(board.A1)



while True:
    print( (analog_in0.value, analog_in1.value))
    time.sleep(0.1)
```

3.2.6 LED Demo

This program shows how to turn LED on and off.

```
#########################################################
#  Toggle Red and Blue LED in the SMART_ARDUINO board
#########################################################

import time
import digitalio
import board

# board.L  or  board.LED  refer to Red LED
redLed = digitalio.DigitalInOut(board.L)
redLed.direction = digitalio.Direction.OUTPUT

# board.BLUE_LED refer to Blue LED
blueLed = digitalio.DigitalInOut(board.BLUE_LED)
blueLed.direction = digitalio.Direction.OUTPUT
```

(continues on next page)

(continued from previous page)

```
while True:
    redLed.value = True
    blueLed.value = False
    time.sleep(0.5)
    blueLed.value = True
    redLed.value = False
    time.sleep(0.5)
```

3.2.7 Switchbutton Demo

This program shows how to read a button of SMART_ARDUINO.

```
##########################################################
#  Read "SWITCH" button state with debouncing
##########################################################

import board
from digitalio import DigitalInOut, Direction, Pull

btn = DigitalInOut(board.SWITCH)
btn.direction = Direction.INPUT
btn.pull = Pull.UP

prev_state = btn.value

while True:
    cur_state = btn.value
    if cur_state != prev_state:
        if not cur_state:
            print("BTN is down")
        else:
            print("BTN is up")

    prev_state = cur_state
```

3.2.8 Read Digital Inputs Demo

This program configures IO pins as pull-down digital inputs to read the pins value.

```
##########################################################
#  Simple I/O Test
##########################################################
import time
import digitalio
import board
```

(continues on next page)

(continued from previous page)

```
############################################################
# These PINs are working as simple I/O pins
############################################################
A0 = digitalio.DigitalInOut(board.A0)
A1 = digitalio.DigitalInOut(board.A1)
A2 = digitalio.DigitalInOut(board.A2)
A3 = digitalio.DigitalInOut(board.A3)
A4 = digitalio.DigitalInOut(board.A4)
A5 = digitalio.DigitalInOut(board.A5)
A6 = digitalio.DigitalInOut(board.VOLTAGE_MONITOR)
D8 = digitalio.DigitalInOut(board.NEOPIXEL)
D5 = digitalio.DigitalInOut(board.D5)
D6 = digitalio.DigitalInOut(board.D6)
D9 = digitalio.DigitalInOut(board.D9)
D10 = digitalio.DigitalInOut(board.D10)
PinList     = [A0, A1, A2, A3, A4, A5, A6, D8, D5, D6, D9, D10]
PinNameList = ["A0", "A1", "A2", "A3", "A4", "A5", "A6", "D8", "D5", "D6", "D9", "D10"]

# set all  pins as simple input with internal Pull low resistor
for pin in PinList:
    pin.direction = digitalio.Direction.INPUT
    pin.pull = digitalio.Pull.DOWN



# You can connect SMART_ARDUINO "3.3V" to one of these pins
# to trigger the values
print("Checking PIN status")
while True:
    i = 0
    for pin in PinList :
        if (pin.value) :
            print("{} is {}".format(  PinNameList[i], pin.value) )

        #print("{} is {}".format(pin, pin.value) )
        i = i + 1
    time.sleep(0.5)
```

3.2.9 Color Detection Demo

This example setups LOKII-CE into RGB color detection mode and retrieves the color object and its attributes detected by waitForBlobResult(True) in the blocking mode.

```
################################
# Color detection demo
################################

import time
import time
```

(continues on next page)

(continued from previous page)

```
from LOKII import *

lokii = Lokii()
lokii.connect()
lokii.setCameraMode(L_CAM_RECOGNIZE_RGB)



colorNames = [ "BLACK", "Red", "Green", "Blue" ]

while True:
    # wait for a color blob detection found
    lokii.waitForBlobResult(True)
    # get the color blob attributes
    PosX = lokii.getBlobResult(0, L_XPOS)
    PosY = lokii.getBlobResult(0, L_YPOS)
    width = lokii.getBlobResult(0, L_WIDTH)
    height = lokii.getBlobResult(0, L_HEIGHT)
    colorCode = lokii.getBlobResult(0, L_COLOR)
    print(
        "Biggest Color object detected: (x,y):({},{}) (w,h):({},{}) color:{}".format(
        PosX,
        PosY,
        width,
        height,
        colorNames[colorCode]
        )
    )
    time.sleep(0.1)
```

3.2.10 RC Servo Demo

This example shows the usage of setRCServo() functions to control RC servos using SMART_RC board (J1,J2,J3,J4 port). The program moves the arm to the right hand side and uses the claw to pick up an object. Then, the arm will move to the other side and place the object. Before running the program, please make sure the LOKII arm and claw RC servos are connected to the right RC servo ports. i.e. SMART_SERVO: J1 = ID 8, J2 = ID 9, J3 = ID 10, J4 = ID 11. In this case, the servo (J4) controls the claw , the servo (J3) controls the arm moving left and right, the servo (J2) controls the arm moving up and down and the servo (J1) controls the arm moving forward and backward respectively.

```
################################
# Robotic ARM demo (SMART_RC) board
################################

import time
from LOKII import *
```

(continues on next page)

(continued from previous page)

```
# define the robotic ARM RC servo smartId here
Turnable_Mounting = 10
Arm_Up_and_Down = 9
Arm_Backward_and_Forward = 8
Clamp = 11


def Normal():
        lokii.setRCServo(Clamp,5,99)
        time.sleep(1.0)
        lokii.setRCServo(Arm_Backward_and_Forward,5,169)
        time.sleep(1.0)
        lokii.setRCServo(Arm_Up_and_Down,5,5)
        time.sleep(1.0)
        lokii.setRCServo(Turnable_Mounting,5,125)


lokii = Lokii()
lokii.connect()

lokii.setCameraMode(L_CAM_PREVIEW)

# move to normal position
Normal()

# start the robot arm movement
lokii.setRCServo(Arm_Up_and_Down,5,1)
time.sleep(1.0)
lokii.setRCServo(Turnable_Mounting,5,200)
time.sleep(1.0)
lokii.setRCServo(Arm_Up_and_Down,5,70)
time.sleep(1.0)
lokii.setRCServo(Arm_Backward_and_Forward,5,200)
time.sleep(1.0)
lokii.setRCServo(Clamp,5,40)
time.sleep(1.0)
lokii.setRCServo(Arm_Backward_and_Forward,5,169)
time.sleep(1.0)
lokii.setRCServo(Arm_Up_and_Down,5,1)
time.sleep(1.0)
lokii.setRCServo(Turnable_Mounting,5,50)
time.sleep(1.0)
lokii.setRCServo(Arm_Up_and_Down,5,70)
time.sleep(1.0)
lokii.setRCServo(Arm_Backward_and_Forward,5,200)
time.sleep(1.0)

# back to normal position
Normal()
```

3.2.11 Keyword Group Training Demo

This example only trains five new keywords for three times and registers the trained keywords group at index 11. After the trained keywords is successfully created, LOKII will recognize the keywords and plays the keyword the user spoke if succeed. This new keyword group 11 also contains “”number group” keywords, so the index of new keywords starts from 11. Keyword group 11 will look like List[] below.

```
################################
# Speaker dependent keyword training demo
################################

import time
from LOKII import *


lokii = Lokii()
lokii.connect()
lokii.setCameraMode(L_CAM_PREVIEW)


# train these keywords into keywords group:5
fruitList = ["orange", "apple", "mango", "banana", "grape"]

# final keyword group will become like this
finalCombinedList = ["zero", "one", "two", "three", "four", "five", "six", "seven",
↪"eight", "nine","ten", "orange", "apple", "mango", "banana", "grape"]



# define the speaker dependent storage parameter
SD_index = 11
numKeywords = 5


# create speaker dependent group which append into the default
# numer list: 0-10
lokii.createSDGroup(SD_index, numKeywords)

# train new keywords group from fruitList
for i in range(numKeywords) :
        lokii.playTTS("Say Your word" , L_DEFAULT ,5 ,5 , E_NATURAL)
        print("Train speech index: {}".format(i) )
        result = lokii.trainSDkeyword(SD_index,i)
        print("result =  {}".format(result) )


result = lokii.checkSDComplete(SD_index)
if (result == 0) :
        print("Create SD group success!")
else :
        print("Create SD group Fail!")
```

(continues on next page)

(continued from previous page)

```
# start recognizing the  speaker dependent group
while True:
        lokii.startSpeechRecognize(SD_index)
        speechResult = lokii.waitForSpeechResult()
        if(speechResult>=0 and speechResult <= (10+numKeywords) ) :
                lokii.playTTS(finalCombinedList[speechResult] , L_DEFAULT ,5 ,5 , E_
↪NATURAL)
        time.sleep(0.1)
```

3.2.12 IO Extender Demo

This example shows how to connect LOKII I/O Extender with SMART_ARDUINO, and uses the following functions to read and write GPIO ports and I/O status. Those functions are instance methods of class "IOExtender".

```
################################
# SMART_IO_2 Extender for a analog joy stick demo
################################

from LOKII import *
from LOKIIExtender import *

lokii = Lokii()
smartExt = smartExtender()
lokii.connect()
lokii.setCameraMode(L_CAM_PREVIEW)


# Init the SPI connection with SMART_IO_2 Extender
smartExt.initExtSoftSPI(board.D11, board.D5, board.D10, board.D9, board.D6)
# lokii.extIOSPISelfTest()
smartExt.setGPIO(3,0)

while True:
    smartExt.syncGPIO()
    # SMART_IO_2  PIN 12, 13,14,15 are the analog input pin with ( 0-4095 value)
    Y2 = smartExt.readGPIO(12)
    X2 = smartExt.readGPIO(13)
    Y1 = smartExt.readGPIO(14)
    X1 = smartExt.readGPIO(15)

    print(" X1,Y1: {} {} and X2,Y2: {} {}".format( X1,Y1, X2,Y2) )
```

3.2.13 Speech Recognition Demo

LOKII-CE recognizes the keywords from built-in keywords group 4 and speaks out the recognized keywords using Text-To-Speech functions.

```
################################
# Speech recongition demo
################################
import time
from LOKII import *

lokii = Lokii()
lokii.connect()

command_group = ["Tell me a joke", "play me a song", "stop the music", "take a photo",
↪"show me a photo", "track my face", "follow the ball", "record motor motion",
↪"playback motor","list commands"]
lokii.setCameraMode(L_CAM_PREVIEW)

while True:
        lokii.startSpeechRecognize(4)
        speechResult = lokii.waitForSpeechResult()
        if(speechResult >= 0 and  speechResult <=9) :
                lokii.startSpeechRecognize(0)
                lokii.playTTS(command_group[speechResult],0,5,5,0)
                time.sleep(0.1)
```

3.2.14 DC Motors Demo

This example moves 4 DC motors in a clockwise and anti-clockwise direction using 4 DC motor ports on the SMART_POWER boards. DC motors with id 0 and 1 are connected to the DC motor port near the SMART_BUS port on the SMART_POWER. On the other hand, DC motors with id 1 and 2 are connected to the DC motor port on the other sides of SMART_POWER board.

```
################################
# DC Motor demo (SMART_POWER)
################################
from LOKII import *

lokii = Lokii()
lokii.connect()
lokii.setCameraMode(L_CAM_PREVIEW)

lokii.playTTS("Go forward.", 0, 5, 5, 0)
lokii.setDCMotor(0, 26, 0)
lokii.setDCMotor(1, 26, 0)
lokii.setDCMotor(2, 26, 0)
lokii.setDCMotor(3, 26, 0)
time.sleep(1.0)

lokii.playTTS("Turn left", 0, 5, 5, 0)
lokii.setDCMotor(0, 26, 0)
```

(continues on next page)

(continued from previous page)

```
lokii.setDCMotor(1, 26, 0)
lokii.setDCMotor(2, 0, 0)
lokii.setDCMotor(3, 0, 0)
time.sleep(1.0)

lokii.playTTS("Turn Right.", 0, 5, 5, 0)
lokii.setDCMotor(0, 0, 0)
lokii.setDCMotor(1, 0, 0)
lokii.setDCMotor(2, 26, 0)
lokii.setDCMotor(3, 26, 0)
time.sleep(1.0)

lokii.playTTS("Go backward.", 0, 5, 5, 0)
lokii.setDCMotor(0, 26, 1)
lokii.setDCMotor(1, 26, 1)
lokii.setDCMotor(2, 26, 1)
lokii.setDCMotor(3, 26, 1)
time.sleep(1.0)
```

3.2.15 MIDI Demo

Different MIDI note numbers are assigned to the playMIDI function to play "Twinkle Twinkle Little Star", "It's a Small World", "Open Mos" with a frequency set by setMIDIBPM. (Beat Per Minutes)

```
################################
# MIDI notes demo
################################

import time
from LOKII import *

lokii = Lokii()
lokii.connect()

lokii.setCameraMode(L_CAM_PREVIEW)




TwinkleLittleStar = [
   BEAT_1, NOTE_1, NOTE_1, NOTE_5, NOTE_5, NOTE_6, NOTE_6, BEAT_2, NOTE_5,
   BEAT_1, NOTE_4, NOTE_4, NOTE_3, NOTE_3, NOTE_2, NOTE_2, BEAT_2, NOTE_1,
   BEAT_1, NOTE_5, NOTE_5, NOTE_4, NOTE_4, NOTE_3, NOTE_3, BEAT_2, NOTE_2,
   BEAT_1, NOTE_5, NOTE_5, NOTE_4, NOTE_4, NOTE_3, NOTE_3, BEAT_2, NOTE_2,
   BEAT_1, NOTE_1, NOTE_1, NOTE_5, NOTE_5, NOTE_6, NOTE_6, BEAT_2, NOTE_5,
   BEAT_1, NOTE_4, NOTE_4, NOTE_3, NOTE_3, NOTE_2, NOTE_2, BEAT_2, NOTE_1
   ]


openMos = [ BEAT_1, NOTE_1, NOTE_2, NOTE_3, NOTE_1, NOTE_1, NOTE_2, NOTE_3, NOTE_1, NOTE_
```

(continues on next page)

(continued from previous page)

```
↪3, NOTE_4, BEAT_2,NOTE_5,
BEAT_1, NOTE_3, NOTE_4, BEAT_2, NOTE_5,
BEAT_0_5, NOTE_5, NOTE_6, NOTE_5, NOTE_4, BEAT_1, NOTE_3, NOTE_1,
BEAT_0_5, NOTE_5, NOTE_6, NOTE_5, NOTE_4, BEAT_1, NOTE_3, NOTE_1,
BEAT_1, NOTE_2, NOTE_5L, BEAT_2, NOTE_1,
BEAT_1, NOTE_2, NOTE_5L, BEAT_2, NOTE_1,
]


smallWorld = ( [
BEAT_1, NOTE_3L, NOTE_4L, BEAT_2, NOTE_5L, NOTE_3, NOTE_1,
BEAT_1, NOTE_2, NOTE_1, BEAT_2, NOTE_1, NOTE_7L, NOTE_7L, BEAT_1, NOTE_2L, NOTE_3L, BEAT_
↪2, NOTE_4L, NOTE_2, NOTE_7L,
BEAT_1, NOTE_1, NOTE_7L, BEAT_2, NOTE_6L, NOTE_5L, NOTE_5L,
BEAT_1, NOTE_3L, NOTE_4L, BEAT_2, NOTE_5L, BEAT_1, NOTE_1, NOTE_2, BEAT_2, NOTE_3,
BEAT_1, NOTE_2, NOTE_1, BEAT_2, NOTE_6L, BEAT_1, NOTE_2, NOTE_3, BEAT_2, NOTE_4,
BEAT_1, NOTE_3, NOTE_2, BEAT_2, NOTE_5L, NOTE_4, NOTE_3, NOTE_2, BEAT_4, NOTE_1,
BEAT_4, NOTE_P,
BEAT_1, NOTE_1, NOTE_P, NOTE_P, NOTE_1, BEAT_2, NOTE_3, NOTE_1, NOTE_2, BEAT_1, NOTE_P,␣
↪NOTE_2, BEAT_4, NOTE_2,
BEAT_1, NOTE_2, NOTE_P, NOTE_P, NOTE_2, BEAT_2, NOTE_4, NOTE_2, NOTE_3, BEAT_1, NOTE_P,␣
↪NOTE_3, BEAT_4, NOTE_3,
BEAT_1, NOTE_3, NOTE_P, NOTE_P, NOTE_3, BEAT_2, NOTE_5, NOTE_3, NOTE_4, BEAT_1, NOTE_P,␣
↪NOTE_4, BEAT_2, NOTE_4,
BEAT_1, NOTE_3, NOTE_2, BEAT_4, NOTE_5L, NOTE_7L, NOTE_1
] )

# set the volume
lokii.setVolume(100)


# set the MIDI BeatPerMinutes
lokii.setMIDIBPM(90)
for note in TwinkleLittleStar:
    lokii.playMIDI(note)



lokii.setMIDIBPM(300)
for note in smallWorld:
    lokii.playMIDI(note)



# set to BPM 100
lokii.setMIDIBPM(100)
for note in openMos:
        lokii.playMIDI(note)
```

3.2.16 Text-to-Speech Demo

LOKII deploys Text-To-Speech function to speak out different English text using a combination of settings of voice types, speed, pitch and emotion types.

```
################################
# Text-To-Speech demo
################################

import time
from LOKII import *

lokii = Lokii()
lokii.connect()

lokii.setCameraMode(L_CAM_PREVIEW)

# Play Text-To-Speech using     different  character  voice, speed, pitch and emotion
lokii.playTTS("Twinkle, twinkle, little star,", L_DEFAULT, 5, 5, E_NATURAL)
lokii.playTTS("How I wonder what you are!", L_DEFAULT, 5, 5, E_NATURAL)
lokii.playTTS("Up above the world so high,", L_DEFAULT, 5, 5, E_NATURAL)
lokii.playTTS("Like a diamond in the sky", L_DEFAULT, 5, 5, E_NATURAL)
lokii.playTTS("Twinkle, twinkle, little star,", L_YOUNGGIRL, 5, 5, E_FRIENDLY)
lokii.playTTS("How I wonder what you are!", L_MAN, 5, 5, E_EMOTIONAL)
lokii.playTTS("Up above the world so high,", L_BOY, 5, 5, E_EXCITED)
lokii.playTTS("Like a diamond in the sky", L_OLDWOMAN, 5, 5, E_SURPRISED)
```

3.2.17 Face Detection Detection

This example setups LOKII-CE into Face detection mode and retrieves the biggest human front face and its attributes detected by using waitForFaceResult(True) in the blocking mode.

```
################################
# Face detection demo
################################

import time
from LOKII import *

lokii = Lokii()
lokii.connect()
lokii.setCameraMode(L_CAM_FACE_DETECT)

while True:
    # wait for a face
    lokii.waitForFaceResult(True)
    # get the face coordinates:  (x,y, w, h)
    PosX = lokii.getBlobResult(0, L_XPOS)
    PosY = lokii.getBlobResult(0, L_YPOS)
    width = lokii.getBlobResult(0, L_WIDTH)
    height = lokii.getBlobResult(0, L_HEIGHT)
```

(continues on next page)

(continued from previous page)

```
    print(
        "Face detected coorindate (x,y):({},{}) (w,h):({},{})".format(
        PosX,
        PosY,
        width,
        height)

    )
    time.sleep(0.1)
```

3.2.18 Gesture Detection

This example setups LOKII-CE into gesture detection mode and retrieves human hands and their attributes detected by using waitForGestureResult(True) in the blocking mode.

```
from LOKII import *
lokii = Lokii()
lokii.connect()
lokii.setCameraMode(L_GESTURE)

handX = 0
handY = 0
handW = 0
handH = 0
handFingerCount = 0

while True:
    handCount = lokii.waitForGestureResult(True)

    if handCount > 0:
        print(handCount)
        for i in range(handCount):
            handX = lokii.getGestureResult(i,HAND_XPOS)
            handY = lokii.getGestureResult(i,HAND_YPOS)
            handW = lokii.getGestureResult(i,HAND_WIDTH)
            handH = lokii.getGestureResult(i,HAND_HEIGHT)
            handFingerCount = lokii.getGestureResult(i,HAND_FINGERCOUNT)
        print("#################")
        print(handX)
        print(handY)
        print(handW)
        print(handH)
        print(handFingerCount)
        print("#################")
```

3.2.19 Posture Detection

This example setups LOKII-CE into posture detection mode and retrieves human posture and its attributes detected by using waitForPostureResult(True) in the blocking mode.

```
from LOKII import *
lokii = Lokii()
lokii.connect()
lokii.setCameraMode(L_POSTURE)

poseCount = 0
poseX = 0
poseY = 0
poseW = 0
poseH = 0
poseLeftHandCode = 0
poseRightHandCode = 0
poseNosePos = 0


while True:
    poseCount = lokii.waitForPostureResult(True)
    # print(poseCount)
    if poseCount > 0:
        print(poseCount)
        for i in range(poseCount):
            print(i)
            poseX = lokii.getPostureResult( i,POSE_XPOS)
            poseY = lokii.getPostureResult( i,POSE_YPOS)
            poseW = lokii.getPostureResult( i,POSE_WIDTH)
            poseH = lokii.getPostureResult( i,POSE_HEIGHT)
            poseLeftHandCode  = lokii.getPostureResult( i,POSE_LEFTHANDCODE)
            poseRightHandCode = lokii.getPostureResult( i,POSE_RIGHTHANDCODE)
            poseNosePos = lokii.getPostureFeatures(i, ES_NOSE)
        print("#################")
        print(poseX)
        print(poseY)
        print(poseW)
        print(poseH)
        print(poseLeftHandCode)
        print(poseRightHandCode)
        print(poseNosePos)
        print("#################")
```

PYTHON MODULE INDEX

INDEX