Arduino LOKII-CE Programming Guide ver 1.2

Centek International Ltd.
Version 1.0.2
Tue April 23 2024

Table of Contents

Summary

void **connect** ()
Initials the LOKII system with the SPI bus.

void **setCameraMode** (int cameraState)
sets LOKII camera running mode for image processing function.

int **clearLCD** ()
clears text on LOKII's LCD.

int **lcdPrint** (String showString, int lineNum)
shows text on LOKII's LCD.

int **lcdPrintAll** (String showString1, String showString2, String showString3, String showString4, String showString5)
shows texts on multiple rows of LOKII's LCD.

int **waitForFaceResult** (int faceState)
waits for face detection result.

int **getFaceResult** (int attributeType)
gets attribute of the face detection result.

int **waitForBlobResult** (bool isBlocking)
waits for color blob detection result.

void **registerColor** ()
registers a custom color for color detection.

int **getBlobResult** (int blobIndex, int attributeType)
gets the attribute of the color detection result.

int **getBlobCount** ()
gets color blob detected count.

int **recordVideo** (String filename)
starts recording video from camera.

int **stopRecordVideo** ()
stops the video recording from camera.

int **playVideo** (String filename)
plays a H.264 video on the LCD screen.

int **checkVideoStatus** ()
checks video status.

int **stopPlayVideo** ()
stops the video playback on screen.

int **takePhoto** (String filename)
takes a photo.

int **displayPhoto** (String filename)
displays a jpeg photo.

void **startSpeechRecognize** (int wordgroupIndex)
starts/stops speech recognition.

int **waitForSpeechResult** ()
waits for the result of speech recognition (BLOCKING) and gets the array index of the recognized keyword.

int **getSpeechResult** ()
gets the array index of the recognized keyword (NON-BLOCKING).

int **createSDGroup** (int groupIndex, int numKeywords)
creates a speaker dependent (SD) custom speech recognition keyword groups.

int **trainSDkeyword** (int groupIndex, int keywordsIndex)
trains a speaker dependent (SD) keyword on a created keywords group.

int **checkSDComplete** (int groupIndex)
checks if speaker dependent keyword group training is completed.

void **setVolume** (int vol)
sets the sound output volume level of LOKII.

void **playTTS** (String text, int voiceType, int speed, int pitch, int emotion)
plays the english text string using LOKII built-in text-to-speech engine.

int **playSoundFile** (String filename, bool isBlocking)
plays a wav or mp3 sound file stored in the LOKII internal flash storage.

void **stopSound** ()
stops sound playback.

int **checkAudioStatus** ()
checks the audio status.

int **recAudio** (String filename)
starts audio recording.

int **stopRecAudio** (void)
stops audio recording.

int **playMIDI** (int note)
plays MIDI note.

void **setMIDIBPM** (int bpmIn)
sets MIDI BPM duration.

void **setSmartDeviceAdress** (int id)
sets SMART_SERVO address.

int **getSmartDeviceCount** (void)
gets the number of smart devices connected to LOKII.

void **showSmartDeviceStatus** (bool isTrue)
gets the number of smart devices coneected to LOKII.

int **setRCServo** (int motorid, int speed, int position)
moves RC servo (LOKII servo driver exclusive function).

int8_t **setRCServoCompletionTime** (int motorid, int16_t timeMS, int position)
moves RC servo to a desired position within a requested time (LOKII servo driver exclusive function).

int **setDCMotor** (int motorid, int speed, int direction)
moves DC motor (LOKII motor driver exclusive function).

bool **waitForQRResult** (bool isBlocking)
starts QR code detection.

void **getQRResult** (char *qrString)
get recognized QR code's string result.

int **waitForGestureResult** (bool isBlocking)
starts gesture detection.

int **getGestureResult** (int handIndex, int attributeType)
gets hand attributes from the cache after waitForGestureResult(bool isBlocking) function is activated.

int **waitForPostureResult** (bool isBlocking)
starts posture detection.

int **getPostureResult** (int poseIndex, int attributeType)
gets human attributes from the cache after waitForPostureResult(bool isBlocking) function is activated.

IntKeypointTrim **getPostureFeatures** (int poseIndex, int ptIndex)
gets human posture feature points from the cache after waitForPostureResult(bool isBlocking) function is activated.

int **initExtSoftSPI** (int softCSPin, int softCS1Pin, int softMISOPin, int softMOSIPin, int softCLKPin)
Connects LOKII I/O Extender with SMART_ARDUINO.

int **extIOSPISelfTest** ()
SPI Test on LOKII I/O Extender.

void **setGPIO** (uint8_t ioNum, uint8_t in_or_out)
sets a LOKII I/O Extender digital pin to INPUT or OUTPUT.

uint8_t **readGPIO** (uint8_t ioNum)
Reads status from a LOKII I/O Extender pin.

void **writeGPIO** (uint8_t ioNum, uint8_t low_or_high)
Writes a LOKII I/O Extender digital pin.

void **syncGPIO** (void)
synchronize LOKII I/O Extender status.

int32_t **gamepadDirection** (int32_t direction, int32_t digitalORanalog)
reads LOKII Remote Control gamepad Joystick status read LOKII Remote Control gamepad Joystick status.

int32_t **gamepadKey** (int32_t button, int32_t pressedORreleased)
reads LOKII Remote Control gamepad key status.

void **gamepadAll** (int32_t *keys)
reads all LOKII Remote Control gamepad status.

Function Documentation

void connect()

Initials the LOKII system with the SPI bus.

void setCameraMode(int cameraState)

sets LOKII camera running mode for image processing function.

sets LOKII camera to perform image processing functions.

Parameters

cameraState	defines the image processing function.
	`L_CAM_RECOGNIZE_RGB, performs Red/Green/Blue color detection mode.`
	`L CAM RECOGNIZE CUSTOM, performs self-trained color detection mode.`
	`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, multi-media mode.`
	`L_GESTURE, performs gesture detection mode.`
	`L_POSTURE, performs posture detection mode.`

int clearLCD()

clears text on LOKII's LCD.

Clears text on LOKII's LCD.

Returns

returns 1 - success, 0 - fail.

int lcdPrint(String showString, int lineNum)

shows text on LOKII's LCD.

shows text on LOKII's LCD.

Parameters

showString	a string, maximum length is 32.
lineNum	row number between 1 - 14.

Returns

returns 1 - success, 0 - fail.

int lcdPrintAll(String showString1, String showString2, String showString3, String showString4, String showString5)

shows texts on multiple rows of LOKII's LCD.

shows texts on multiple rows of LOKII's LCD.

Parameters

showString1	a string on row 10, maximum length is 32.
showString2	a string on row 11, maximum length is 32.
showString3	a string on row 12, maximum length is 32.
showString4	a string on row 13, maximum length is 32.
showString5	a string on row 14, maximum length is 32.

Returns

returns 1 - success, 0 - fail.

int waitForFaceResult(int faceState)

waits for face detection result.

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. Only human front face can be detected.

Parameters

faceState	defines the face detection behavior.

if faceState = L_OFF or false, this function will return immediately regardless of face detected (NON-BLOCKING).

if faceState = L_ON or true, this function will hold until a face is detected (BLOCKING).

Returns

returns face size, 0 - face not detected (in NON-BLOCKING mode only).

int getFaceResult(int attributeType)

gets attribute of the face detection result.

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

Parameters

attributeType	Following attributeTypes can be specified:
	`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.`
	All the above face detected atributeType are based on the LCD screen coordinate system with 320 x 240 pixel.

Returns

returns the face detected attribute result.

int waitForBlobResult(bool isBlocking)

waits for color blob detection result.

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	defines the color blob detection behavior.

if isBlocking = L_OFF or false, this function will return immediately regardless of color blob detected (NON-BLOCKING).

if isBlocking = L_ON or true, 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.

void registerColor()

registers a custom color for color detection.

trains LOKII to recognize the color hue value from camera center region. The trained custom color can then be recognized by setCameraMode(L_CAM_RECOGNIZE_CUSTOM) function. To have a positive training result, make sure: 1) the color object is not reflective. 2) the color object size is around 30. 3) the color object is posited at the centre of the LCD screen.

int getBlobResult(int blobIndex, int attributeType)

gets the attribute of the color detection result.

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

Parameters

blobIndex	the index (starting from zero) for the color blob (The index should be smaller than the total color object detected).

attributeType	Following atributeTypes can be specified:
	`L_XPOS, x coordinate of the color blob centre`
	`L_YPOS, y coordinate of the color blob centre`
	`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`
	All the above color atributeTypes are based on the LCD screen coordinate system with 320 x 240 pixel

Returns

returns one of the color attributes.

int getBlobCount()

gets color blob detected count.

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

Returns

returns the number of color blob in the cache.

int recordVideo(String filename)

starts recording video from camera.

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 type, video name must be ended with ".avi". E.g "video.avi".

Returns

returns 1 - success, 0 - fail.

int stopRecordVideo()

stops the video recording from camera.

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

Returns

returns 0 for success.

int playVideo(String filename)

plays a H.264 video on the LCD screen.

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	the input video name with file extension ".avi". E.g "video.avi".

Returns

returns 1 for success, 0 for error.

int checkVideoStatus()

checks video status.

checks if LOKII is playing a video or not.

Returns

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

int stopPlayVideo()

stops the video playback on screen.

stops the video playback.

Returns

returns 0 for success.

int takePhoto(String filename)

takes a photo.

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

Parameters

filename	string type, the jpeg photo filename. E.g "a.jpg".

Returns

returns 1 - success, 0 - fail.

int displayPhoto(String filename)

displays a jpeg photo.

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

Parameters

filename	the jpeg photo filename. E.g "a.jpg".

Returns

returns 1 - success, 0 - fail.

void startSpeechRecognize(int wordgroupIndex)

starts/stops speech recognition.

Performs speech recognition for trained word groups in Speaker Independent mode (SI) (index can be 1-10). Performs speech recognition for trained word groups in Speaker Dependent mode (SD) (index can be 11 - 20). 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	the index for word groups: 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).

int waitForSpeechResult()

waits for the result of speech recognition (BLOCKING) and gets the array index of the recognized keyword.

Uses with 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.

int getSpeechResult()

gets the array index of the recognized keyword (NON-BLOCKING).

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.

int createSDGroup (int groupIndex, int numKeywords)

creates a speaker dependent (SD) custom speech recognition keyword groups.

creates a speech recognition with a keyword groups index (limited to 11-20). Number of keywords affects the accuracy of speech recognition. Not more than 5 keywords is recommended. This speech recognition is speaker dependent for the training data. Any new keygroup groups will contain the keywords of the number group automatically, so the index of new keywords starting from 11.

Parameters

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

Returns

returns 1 - success , 0 - fail.

int trainSDkeyword (int groupIndex, int keywordsIndex)

trains a speaker dependent (SD) keyword on a created keywords group.

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 this function asking user to speak each keywords by playTTS each time for it to sample ambient sounds. It will perform three training sessions in order to complete the sampling for each keyword. 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	the group index of the created keywords group (11-20 inclusively).
keywordsIndex	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).

int checkSDComplete(int groupIndex)

checks if speaker dependent keyword group training is completed.

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

Parameters

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

Returns

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

void setVolume(int vol)

sets the sound output volume level of LOKII.

sets the sound output volume level between 0-100 inclusively.

Parameters

vol	volume level (0-100).

void playTTS (String text, int voiceType, int speed, int pitch, int emotion)

plays the english text string using LOKII built-in text-to-speech engine.

converts english text string to speech with a specified voice, speed, pitch, emotion settings.

Parameters

text	english text string.

voiceType	voice can be one of following values:
	`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	speed can be 1 - 10 inclusively.
pitch	pitch can be 1 - 10 inclusively.

emotion	emotion can be one of the following values:
	`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`

int playSoundFile(String filename, bool isBlocking)

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

plays a wav sound file by specifying the sound filename.

Parameters

filename	sound file name, e.g "a.mp3".
isBlocking	0 - non-blocking, 1 - blocking (waits until the sound file playback completed).

Returns

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

void stopSound()

stops sound playback.

stops sound playback.

int checkAudioStatus()

checks the audio status.

Returns

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

int recAudio(String filename)

starts audio recording.

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	filename string, e.g "a.wav".

Returns

returns 0 - fail, 1 - success.

int stopRecAudio()

stops audio recording.

Returns

returns 0 for success.

int playMIDI(int note)

plays MIDI note.

plays MIDI note for a second.

Parameters

note	the MIDI integer 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 If the note value is negative, LOKII will play a silence.

void setMIDIBPM(int bpmIn)

sets MIDI BPM duration.

sets MIDI BPM between 30 - 100.

Parameters

bpmIn	the BPM value.

void setSmartDeviceAdress(int id)

sets SMART_SERVO address.

Sets SMART_SERVO pins id. ID must be greater than 7. The new id will be assigned to the first pin (J1), id+1 for the next pin. SMART_SERVO must connect to LOKII independently when using this function.

Parameters

id	servo id.

int getSmartDeviceCount()

gets the number of smart devices connected to LOKII.

Prints out Smart devices id and total number in the serial log.

Returns

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

void showSmartDeviceStatus (bool isTrue)

Shows smart devices' status connected to LOKII.

Shows smart devices id, their types and attributes on LCD.

Parameters

isTrue	display ON/OFF.

int setRCServo(int motorid, int speed, int position)

moves RC servo (LOKII servo driver exclusive function).

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

Parameters

motorid	the servo id.
speed	speed (0 - 20), 0 = the fastest, 1 = the slowest.
position	0, power off. 1 - 250, servo position.

Returns

returns 0 - success, -1 - fail.

int8_t setRCServoCompletionTime(int motorid, int16_t timeMS, int position)

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

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

Parameters

motorid	the servo id.
timeMS	time (ms), 16-bit integer.
position	0, power off. 1 - 250, servo position.

Returns

returns 0 - success, other value - fail.

int setDCMotor(int motorid, int speed, int direction)

moves DC motor (LOKII motor driver exclusive function).

moves DC motor attached on SMART_POWER or SMART_DC board.

Parameters

motorid	0 - 3 for SMART_POWER connection, other id are configurable when attached on SMART_DC board.
speed	0 - 100.
direction	0 - clockwise, 1 - anti-clockwise.

Returns

returns 0 - success, -1 - fail.

bool waitForQRResult(bool isBlocking)

starts QR code detection.

blocking = starts QR code detection and waits until QR code is recognized. Non-blocking returns the result immediately from current live camera image.

Parameters

isBlocking	0 = false - non-blocking, 1 = true - blocking.

Returns

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

void getQRResult (char * qrString)

get recognized QR code's string result.

Get QR code's string result. Caller should allocate at least 128 bytes character buffer for the qrString variable in order to get the QR result.

Parameters

qrString	char array pointer. The recognized result will be passed into this memory address.

int waitForGestureResult (bool 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.

int getGestureResult (int handIndex, int attributeType)

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

Parameters

handIndex	index (starting from zero).
attributeType	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.

int waitForPostureResult (bool 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.

int getPostureResult (int poseIndex, int attributeType)

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

Parameters

poseIndex	index (starting from zero)
attributeType	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.

IntKeypointTrim getPostureFeatures (int poseIndex, int ptIndex)

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

Parameters

poseIndex	index (starting from zero).
ptIndex	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 IntKeypointTrim attributes x and y of the attributes above. All the attributes are based on the LCD screen coordinate system with 320 x 240 pixel.

int initExtSoftSPI (int softCSPin, int softCS1Pin, int softMISOPin, int softMOSIPin, int softCLKPin)

Connects LOKII I/O Extender with SMART_ARDUINO.

a method initiates a communication bridge between SMART_ARDUINO and LOKII I/O Extender (IO Extender 2)

Parameters

softCSPin	the corresponding SMART_ARDUINO's pin connects to SPI_CS of I/O Extender.
softCS1Pin	the corresponding SMART_ARDUINO's pin connects to SPI_CS_1 of I/O Extender for reading analog pins.
softMISOPin	the corresponding SMART_ARDUINO's pin connects to SPI_MISO of I/O Extender.
softMOSIPin	the corresponding SMART_ARDUINO's pin connects to SPI_MOSI of I/O Extender.
softCLKPin	the corresponding SMART_ARDUINO's pin connects to SPI_SCK of I/O Extender.

Returns

returns 0 - success.

int extIOSPISelfTest()

SPI Test on LOKII I/O Extender.

SPI communication test on connection between LOKII and I/O Extender.

Returns

returns 0 - success, -1 - error.

void setGPIO(uint8_t ioNum, uint8_t in_or_out)

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

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

Parameters

ioNum	a pin number of I/O Extender from 0 to 11. 0~11 => pin D0~D11.
in_or_out	0 - input, 1 - output.

uint8_t readGPIO(uint8_t ioNum)

Reads status from a LOKII I/O Extender pin.

a method reads status of a LOKII I/O Extender pin from SMART_ARDUINO cache.

Parameters

ioNum	a pin number of I/O Extender from 0 to 15. 0~11 => pin D0~D11, 0 or 1 or invalid value 0xFFFF. 12~15 => pin A0~A3, 0~4095 or invalid value 0xFFFF.

Returns

returns status of the pin.

void writeGPIO(uint8_t ioNum, uint8_t low_or_high)

Writes a LOKII I/O Extender digital pin.

a method writes a LOKII I/O Extender digital pin.

Parameters

ioNum	a pin number of I/O Extender from 0 to 11. 0~11 => pin D0~D11.
low_or_high	0 = low. 1 = high.

void syncGPIO()

synchronize LOKII I/O Extender status.

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.

int32_t gamepadDirection(int32_t direction, int32_t digitalORanalog)

reads LOKII Remote Control gamepad Joystick status.

reads LOKII Remote Control gamepad Joystick status.

Parameters

direction	32 bits integer. 0 = joystick X1, 1 = joystick Y1, 2 = joystick X2, 3 = joystick Y2.
digitalORanalog	0 = reads digital value, 1 = reads analog value.

int32_t gamepadKey(int32_t button, int32_t pressedORreleased)

reads LOKII Remote Control gamepad key status.

reads LOKII Remote Control gamepad key status.

Parameters

button	32 bits 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	defines high state of a button.

Returns

returns input status.

void gamepadAll(int32_t * keys)

reads all LOKII Remote Control gamepad status.

stores gamepad input value in a buffer. keys[0] = X1 value, keys[1] = Y1 value, keys[2] = X2 value, keys[3] = Y2 value, key[4] = buttons value.

Parameters

keys	32 bits integer array buffer with a length of 5.

Example Documentation

Clamp_and_Arm_with_Object_Demo.ino

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.

```
#include <stdio.h>
#include <lokii.h>
LOKII lokii;

int const Turnable_Mounting = 10;
int const Arm_Up_and_Down = 9;
int const Arm_Backward_and_Forward = 8;
int const Clamp = 11;
int times = 0;

void Normal()
{
  lokii.setRCServo(Clamp,5,99);
  delay(1000);
  lokii.setRCServo(Arm_Backward_and_Forward,5,169);
  delay(1000);
  lokii.setRCServo(Arm_Up_and_Down,5,5);
  delay(1000);
  lokii.setRCServo(Turnable_Mounting,5,125);
  delay(1000);
}

void setup() {
  // put your setup code here, to run once:
  while(!Serial);
  lokii.connect();
  lokii.setCameraMode(L_CAM_PREVIEW);
  Normal();
}

void loop() {
  // put your main code here, to run repeatedly:
  if (times == 0)
  {
    lokii.setRCServo(Arm_Up_and_Down,5,1);
    delay(1000);
    lokii.setRCServo(Turnable_Mounting,5,200);
    delay(1000);
    lokii.setRCServo(Arm_Up_and_Down,5,70);
    delay(1000);
    lokii.setRCServo(Arm_Backward_and_Forward,5,200);
    delay(1000);
    lokii.setRCServo(Clamp,5,40);
    delay(1000);
    lokii.setRCServo(Arm_Backward_and_Forward,5,169);
    delay(1000);
    lokii.setRCServo(Arm_Up_and_Down,5,1);
    delay(1000);
    lokii.setRCServo(Turnable_Mounting,5,50);
    delay(1000);
    lokii.setRCServo(Arm_Up_and_Down,5,70);
    delay(1000);
```

```
    lokii.setRCServo(Arm_Backward_and_Forward,5,200);
    delay(1000);
    Normal();
    times = 1;
  }
}
```

Color_Detection.ino

This example setups LOKII into RGB color detection mode and retrieves the color object and its attributes detected by waitForBlobResult(1) in the blocking mode.

```
#include <lokii.h>
#include <stdio.h>
LOKII lokii;
int PosX, posY, width, height, color;

void setup() {
  // put your setup code here, to run once:
  lokii.connect();
  lokii.setCameraMode(L_CAM_RECOGNIZE_RGB);
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  lokii.waitForBlobResult(true);
  PosX = lokii.getBlobResult(0, L_XPOS);
  posY = lokii.getBlobResult(0, L_YPOS);
  width = lokii.getBlobResult(0, L_WIDTH);
  height = lokii.getBlobResult(0, L_HEIGHT);
  color = lokii.getBlobResult(0, L_COLOR);
  Serial.println(PosX); //check output on Serial
}
```

DC_Motor.ino

This examples 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 in 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 in the DC motor port on the other sides of SMART_POWER board.

```
#include <lokii.h>
#include <stdio.h>
LOKII lokii;
int times;

void setup() {
  // put your setup code here, to run once:
  lokii.connect();
  lokii.setCameraMode(L_CAM_PREVIEW);
  Serial.begin(9600);
  times = 1;
}

void loop() {
  // put your main code here, to run repeatedly:
  while(times == 1) //run only once
  {
    times = times + 1;
    lokii.playTTS("Go forward.", 0, 5, 5, 0);
    for(int i = 0; i < 15; i++)
    {
      lokii.setDCMotor(0,26,0); //forward
      lokii.setDCMotor(1,26,0);
      lokii.setDCMotor(2,26,0);
      lokii.setDCMotor(3,26,0);
    }
    lokii.playTTS("Turn left.", 0, 5, 5, 0);
    for(int i = 0; i < 3; i++)
    {
      lokii.setDCMotor(0,26,0); //left
      lokii.setDCMotor(1,26,0);
      lokii.setDCMotor(2,0,0);
      lokii.setDCMotor(3,0,0);
    }
    lokii.playTTS("Turn Right.", 0, 5, 5, 0);
    for(int i = 0; i < 3; i++)
    {
      lokii.setDCMotor(0,0,0); //right
      lokii.setDCMotor(1,0,0);
      lokii.setDCMotor(2,26,0);
      lokii.setDCMotor(3,26,0);
    }
    lokii.playTTS("Go backward.", 0, 5, 5, 0);
    for(int i = 0; i < 15; i++)
    {
      lokii.setDCMotor(0,26,1); //backward
      lokii.setDCMotor(1,26,1);
      lokii.setDCMotor(2,26,1);
      lokii.setDCMotor(3,26,1);
    }
  }
  lokii.setDCMotor(0,0,0);
  lokii.setDCMotor(1,0,0);
  lokii.setDCMotor(2,0,0);
  lokii.setDCMotor(3,0,0);
}
```

Face_Detection.ino

This example setups LOKII into Face detection mode and retrieves the biggest human front face and its attributes detected by using waitForFaceResult(1) in the blocking mode.

```
#include <lokii.h>
#include <stdio.h>
LOKII lokii;
int faceResult, PosX, posY, width, height;

void setup() {
  // put your setup code here, to run once:
  lokii.connect();
  lokii.setCameraMode(L_CAM_FACE_DETECT);
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  lokii.waitForFaceResult(1);
  PosX = lokii.getFaceResult(L_XPOS);
  posY = lokii.getFaceResult(L_YPOS);
  width = lokii.getFaceResult(L_WIDTH);
  height = lokii.getFaceResult(L_HEIGHT);
  Serial.println(PosX); //check output on Serial
}
```

Keyword_Group_Training.ino

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

```
#include <lokii.h>

volatile int SD_index;
volatile int numKeywords;
volatile int result;

LOKII lokii;


String numberList[] = {"orange", "apple", "mango", "banana", "grape"};
String List[] = {"zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine",
"ten", "orange", "apple", "mango", "banana", "grape"};


void setup(){
  Serial.begin(9600);

  lokii.connect();

  SD_index = 11;
  numKeywords = 5;
  result = 0;
}

void loop(){

  int speechResult;

  // create speaker depentent group
  lokii.createSDGroup(SD_index, numKeywords);
  for (int i = (0); i <= (numKeywords - 1); i = i + (1)) {

      lokii.playTTS(  (String) "Said Your word" + (String) " " , L_DEFAULT ,5 ,5 ,E_NATURAL);
      Serial.println(String("Train speech index") + String(String(i, DEC)));
      //delay(500);
      result = lokii.trainSDkeyword(SD_index,i);
      Serial.println(String(" result = ") + String(String(result, DEC)));
  }
  result = lokii.checkSDComplete(SD_index);
  if (result == 0) {
    Serial.println("Create SD group success!");

  } else {
    Serial.println("Create SD group Fail!");
  }


  // start recognize speaker depentent group
  while (1)
  {
      lokii.startSpeechRecognize(SD_index);
      speechResult = lokii.waitForSpeechResult();
      if(speechResult>=0 && speechResult <= (10+numKeywords))
      {
        lokii.playTTS(List[speechResult] , L_DEFAULT ,5 ,5 ,E_NATURAL);
      }
  }

}
```

playmidi.ino

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)

```
/*
  songnote

  [Prerequisite]
  This program requires a speaker attached to LOKII Arduino Shield.

  [Function]
  Set a smart ID to the serial device attached in LOKII Arduino Shield.
  (Make sure only one smart device is attached)
  User shall enter a smart ID number [ 1-99] to the serial monitor connected in Arduino
  when prompt to set the smart ID.


  Written by Markuz Lee
  10-1-2020

*/


#include <lokii.h>

LOKII lokii;

int 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,
};




int openMos[] =
{ BEAT_1, NOTE_1, NOTE_2, NOTE_3, NOTE_1, NOTE_1, NOTE_2, NOTE_3, NOTE_1, NOTE_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,
};


int 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

} ;



// 1 = C // Do // C4?
// 2 = D // Re
// 3 = E // Mi
// 4 = F
// 5 = G
// 6 = A
// 7 = B


void setup() {

  Serial.begin(9600);
  //while (!Serial);

  lokii.connect();
  Serial.println("starting E!");
  lokii.setVolume(100);

}

void loop() {

  // beat per minute
  //lokii.setMIDIBPM(90);

  // play single note Middle Major C for 1 beat
  // lokii.playMIDI(BEAT_1);
  // lokii.playMIDI(NOTE_C);


  // set LOKII to 90 beat per minute
  lokii.setMIDIBPM(90);

  // play "Twinkle Twinkle Little Star"
  for (unsigned int i = (1); i <= (sizeof(TwinkleLittleStar) / sizeof(TwinkleLittleStar[0]));
i = i + (1)) {
    lokii.playMIDI(TwinkleLittleStar[(int)(i - 1)]);
  }


  // set LOKII to 300 beat per minute
  lokii.setMIDIBPM(300);
  // play "It's a Small World"
  for (int i = (1); i <= (sizeof(smallWorld)/sizeof(smallWorld[0])); i = i + (1)) {
    lokii.playMIDI(smallWorld[(int)(i - 1)]);
  }


  // set LOKII to 100 beat per minute
  lokii.setMIDIBPM(100);
  // play "Open Mos"
  for (int i = (1); i <= (sizeof( openMos   )/sizeof(openMos[0])); i = i + (1)) {
     lokii.playMIDI( openMos[(int)(i - 1)]);
  }

  delay(10000);

}
```

recordAudio.ino

An audio will be recorded for 10 seconds and saved to a file called "b.wav" in LOKII internal flash. Then, the recorded sound file will be playback by LOKII.

```
#include <lokii.h>
#include <stdio.h>
LOKII lokii;
int times;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  while(!Serial);
  lokii.connect();
  lokii.setCameraMode(L_CAM_PREVIEW);
  times = 1;
}

void loop() {
  // put your main code here, to run repeatedly:
  while(times == 1) //play only once
  {
    times = times + 1;
    Serial.println("test");
    Serial.println(lokii.checkAudioStatus());
    lokii.playTTS("Record an Audio", 0, 5, 5, 0);
    lokii.recAudio("b.wav");
    delay(10000);
    lokii.stopRecAudio();
    lokii.playTTS("audio saved", 0, 5, 5, 0);
    delay(1000);
    lokii.playSoundFile("b.wav",1);
    delay(10000);
    lokii.stopSound();
  }
}
```

recordVideo.ino

This program will take a picture and record a video for 5 seconds and the files will be stored in LOKII internal flash. Then, it will playback the video and display a photo.

```
#include <lokii.h>
#include <stdio.h>
LOKII lokii;
int times;
void setup() {
  // put your setup code here, to run once:
  lokii.connect();
  lokii.setCameraMode(L_CAM_PREVIEW);
  times = 1;
}

void loop() {
  // put your main code here, to run repeatedly:
  while(times == 1) //play only once
  {
    times = times + 1;
    lokii.playTTS("Take a photo", 0, 5, 5, 0);
    lokii.takePhoto("a.jpg");
    lokii.recordVideo("video6.avi");
    delay(5000);
    lokii.stopRecordVideo();
    lokii.playTTS("Video saved", 0, 5, 5, 0);
    delay(3000);
    lokii.playVideo("video6.avi");
    delay(5000);
    lokii.stopPlayVideo();
    lokii.displayPhoto("a.jpg");
  }
}
```

SoundDirection_blocking.ino

LOKII detects the sound direction and print out the sound directed detected.

```
#include <lokii.h>
#include <stdio.h>
LOKII lokii;
int times;
void setup() {
  // put your setup code here, to run once:
  lokii.connect();
  lokii.setCameraMode(L_CAM_PREVIEW);
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
   int i = lokii.waitForSoundDirection(1);
   Serial.println(i);
}
```

Speech_Recognition.ino

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

```
#include <lokii.h>
#include <stdio.h>
LOKII lokii;

int speechResult;
String command_group[10] = {"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"};

void setup() {
  // put your setup code here, to run once:
  lokii.connect();
  lokii.setCameraMode(L_CAM_PREVIEW);
}

void loop() {
  // put your main code here, to run repeatedly:
  lokii.startSpeechRecognize(4);
  speechResult = lokii.waitForSpeechResult();
  if(speechResult >= 0 && speechResult <=9)
  {
    lokii.startSpeechRecognize(0);
    lokii.playTTS(command_group[speechResult],0,5,5,0);
  }
}
```

TTS.ino

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.

```
#include <lokii.h>
#include <stdio.h>
LOKII lokii;
int times;
void setup() {
  // put your setup code here, to run once:
  lokii.connect();
  lokii.setCameraMode(L_CAM_PREVIEW);
  times = 1;
}

void loop() {
  // put your main code here, to run repeatedly:
  while(times == 1) //play only once
  {
    times = times + 1;
   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);
  }
}
```

BusExtTest.ino

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

```
#include <lokii.h>


LOKII lokii;

#define EXTAD_CS  5
#define EXTIO_CS   11
#define EXT_MOSI  9
#define EXT_MISO 10
#define EXT_SCK   6

unsigned long StartTime;
int needWithLokii = 1;
IOExtender ioExt;

// IOExtender class functions prototype
// ioNum : 0~11 = D0 ~ D11


// software spi pins init
//void initN55SoftSPI(int softCSPin, int softMISOPin, int softMOSIPin, int softCLKPin);

// set IO to INPUT or OUTPUT
// in_or_out : 0 = INPUT PULL HIGH, 1 = OUTPUT LOW
//void setGPIO(uint8_t ioNum, uint8_t in_or_out);

// read single IO status by IO number
//uint8_t readGPIO(uint8_t ioNum);

// low_or_high : 0 = LOW, 1=HIGH
//void writeGPIO(uint8_t ioNum , uint8_t low_or_high);



void initExt(void)
{
   //ioExt.initN55SoftSPI(EXTIO_CS, EXT_MISO, EXT_MOSI, EXT_SCK);
   ioExt.initExtSoftSPI(EXTIO_CS, EXTAD_CS, EXT_MISO, EXT_MOSI, EXT_SCK);
}

void writeGPIO(uint8_t ioNum , uint8_t buf)
{
   ioExt.writeGPIO(ioNum, buf);
}

void setGPIO(uint8_t ioNum, uint8_t in_or_out)
{
   ioExt.setGPIO(ioNum, in_or_out);
}

void syncGPIO(void)
{
   ioExt.syncGPIO();
}

uint16_t readGPIO(uint8_t ioNum)
{
   return ioExt.readGPIO(ioNum);
}

int extenderSPISelfTest(void)
{
   return ioExt.extIOSPISelfTest();
```

```
}

int lcdPrint(String showString, int lineNum)
{
   if (needWithLokii == 1)
      return lokii.lcdPrint(showString, lineNum);
   else
      return 0;
}



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

  Serial.begin(115200);
  while (!Serial);

  delay(100);// first print need delay after Serial.begin
  Serial.println(F("start BusExtender test"));

  initExt();

  // optional
  if (extenderSPISelfTest() == LOKI_REPLY_OK)
     Serial.println(F("EXTIO SPI test SUCCESS"));
  else
     Serial.println(F("EXTIO SPI test FAIL"));


  // work with LOKII class test
  if (needWithLokii == 1)
     lokii.connect();


#if 0
   pinMode(EXT_MOSI, OUTPUT);
   digitalWrite(EXT_MOSI, HIGH);
   pinMode(EXT_MISO, OUTPUT);
   digitalWrite(EXT_MISO, HIGH);
   pinMode(EXT_SCK, OUTPUT);
   digitalWrite(EXT_SCK, HIGH);
#endif

}


void loop() {

  int i;
  int changed;

  // readGPIO now return 16bit number!
  uint16_t cur[4];
  uint16_t last[4] = {0xFF, 0xFF, 0xFF, 0xFF};

  uint16_t adValue[4];

  delay(100);

#if 1

   for (i = 0; i < 4; i++)
   {
      setGPIO(i,0);   // D0~D3 , 0=INPUT PULL HIGH, 1=OUTPUT LOW
   }

   while (true)
   {

      //StartTime = millis();
```

```
      syncGPIO(); // read AD
      //Serial.println(millis()-StartTime);

      changed = 0;
      for (i = 0; i < 4; i++)
      {
         cur[i] = readGPIO(i);
         if ( last[i] == 1 &&
             cur[i] == 0)
         {
            Serial.print(F("D"));
            Serial.print(i);
            Serial.println(F(" = LOW"));

            lcdPrint("LOW ", 5);
            changed = 1;
         }
         if ( last[i] == 0 &&
             cur[i] == 1)
         {
            Serial.print(F("D"));
            Serial.print(i);
            Serial.println(F(" = HIGH"));
            lcdPrint("HIGH", 5);
            changed = 1;
         }
         last[i] = cur[i];
      }

      // 0~4095 or invalid value 0xFFFF
      // joystick center position 2048
      #define INACTIVE_RANGE 300
      #define CENTER_POS   2048
      for (i = 0; i < 4; i++)
      {
         adValue[i] = (readGPIO(12 + i)); // return AD value which read from syncGPIO()
      }
      for (i = 0; i < 4; i++)
      {
         if (adValue[i] < CENTER_POS-INACTIVE_RANGE ||
            adValue[i] > CENTER_POS+INACTIVE_RANGE)
         {
            Serial.print(F(" A0="));
            Serial.print(adValue[0]);
            Serial.print(F(" A1="));
            Serial.print(adValue[1]);
            Serial.print(F(" A2="));
            Serial.print(adValue[2]);
            Serial.print(F(" A3="));
            Serial.print(adValue[3]);
            Serial.println(F(" "));

            changed = 1;
            break;
         }
      }

      if (changed)
         delay(10); // debounce, no debounce will make the print buffer overflow
  }

#endif

}
```

QRdetect.ino

LOKII is looking for a QR code image (QR Code Version 2 format), print out the QR code's message on LCD and speak out using Text-To-Speech function.

```
#include <lokii.h>

#define BLOCKING 1
#define NOBLOCKING 0

//void getQRResult(char *qrString );
//bool waitForQRResult(bool isBlocking);


unsigned long StartTime = 0;

char qrCode[128];

LOKII lokii;

void QRDetectSample() {


  if (lokii.waitForQRResult() == true) {

    qrCode[0] = 0;
    lokii.getQRResult(qrCode);
    lokii.setCameraMode(L_CAM_PREVIEW);
    lokii.playTTS(qrCode, L_DEFAULT ,5 ,5 ,E_NATURAL);
    Serial.println(qrCode);
    lokii.setCameraMode(L_CAM_QRCODE);


  }
}

void setup(){

  Serial.begin(115200);

  lokii.connect();

}

void loop(){

  lokii.playTTS("QR Code Demo" , L_DEFAULT ,5 ,5 ,E_NATURAL);
  delay(3000);
  lokii.setCameraMode(L_CAM_QRCODE);
  while (true)
  {
     QRDetectSample();
  }


}
```

getSettingTest.ino

This example shows how to read Blockly gamepad input by using the following functions.

```
#include <lokii.h>
LOKII lokii;
void setup()
{
   int i;
   Serial.begin(115200);
   while(!Serial);
   lokii.connect();
}
void loop()
{
   int i;
   int32_t key[8];
   int32_t oldKey[8];
   oldKey[0] = -1;
   oldKey[1] = -1;
   oldKey[2] = -1;
   oldKey[3] = -1;
   oldKey[4] = -1;
      for (i = 0; i  < 8; i++)
      {
         key[i] = lokii.gamepadKey(i, 0);
         Serial.print(" KEY ");
         Serial.print(i+1);
         Serial.print(" ");
         Serial.println(key[i]);
      }
      for (i = 0; i  < 5; i++)
      {
         key[i] = lokii.gamepadDirection(i, 1);
         Serial.print(" KEY DIRECTION ");
         Serial.print(i+1);
         Serial.print(" ");
         Serial.println(key[i]);
         oldKey[i] = key[i];
      }
   while (true)
   {
      #if 0
      for (i = 0; i  < 5; i++)
      {
         key[i] = lokii.gamepadDirection(i, 1);
         if (key[i] != oldKey[i] && oldKey[i] != -1)
         {
            Serial.print(" KEY ");
            Serial.print(i+1);
            Serial.print(" ");
            Serial.println(key[i]);
         }
         oldKey[i] = key[i];
      }
      #endif //delay(100);
      lokii.gamepadAll(key);
      for (i = 0; i  < 5; i++)
      {
         if (key[i] != oldKey[i] && oldKey[i] != -1)
         {
            Serial.print(" KEY ");
            Serial.print(i+1);
            Serial.print(" ");
            Serial.println(key[i]);
         }
         oldKey[i] = key[i];
      }
   }
}
```

gestureDetectTest.ino

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.

```
#include <lokii.h>
LOKII lokii;
volatile int handCount;
volatile int handX;
volatile int handY;
volatile int handW;
volatile int handH;
volatile int handFingerCount;
void handDetectionSample()
{
  char str1[50];
  char str2[50];
  char str3[50];
  char str4[50];
  char str5[50];
  handCount = lokii.waitForGestureResult(true);
  if (handCount > 0)
  {
    for (int i = (0); i <= (handCount - 1); i = i + (1))
    {
      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);
      sprintf(str1, "Hand[%d] x = %d", i , handX);
      Serial.println(str1);
      sprintf(str2, "Hand[%d] y = %d", i , handY);
      Serial.println(str2);
      sprintf(str3, "Hand[%d] width = %d", i , handW);
      Serial.println(str3);
      sprintf(str4, "Hand[%d] height = %d", i , handH);
      Serial.println(str4);
      sprintf(str5, "Hand[%d] fingerCount = %d", i , handFingerCount);
      Serial.println(str5);
      if (i == 0)
      {
        lokii.lcdPrint(str1, 3);
        lokii.lcdPrint(str2, 4);
        lokii.lcdPrint(str3, 5);
        lokii.lcdPrint(str4, 6);
        lokii.lcdPrint(str5, 7);
      }
      else
      {
        lokii.lcdPrintAll(str1, str2, str3, str4, str5);
      }
    }
  }
  else
  {
    lokii.clearLCD();
  }
}
void setup()
{
   Serial.begin(115200);
   lokii.connect();
}
void loop() {
  lokii.setCameraMode(L_GESTURE);
  while (true) {
    handDetectionSample();
  }
}
```

gesturePoseDetect.ino

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.

```
#include <lokii.h>
LOKII lokii;
inline const char * typeStr (int   var) { return " int "; }
/*
// HEAD
#define ES_NOSE         0
#define ES_L_EYE        1
#define ES_R_EYE        2
#define ES_L_EAR        3
#define ES_R_EAR        4
// Upper body
#define ES_L_SHOULDER    5
#define ES_R_SHOULDER    6
#define ES_L_ELBOW       7
#define ES_R_ELBOW       8
#define ES_L_WRIST       9
#define ES_R_WRIST      10
// bottom body
#define ES_L_HIP        11
#define ES_R_HIP        12
#define ES_L_KNEE       13
#define ES_R_KNEE       14
#define ES_L_ANKLE      15
#define ES_R_ANKLE      16
*/
volatile int poseCount;
volatile int poseX;
volatile int poseY;
volatile int poseW;
volatile int poseH;
volatile int poseLeftHandCode;
volatile int poseRightHandCode;
IntKeypointTrim poseNosePos;
String printout = "abc";
void poseDetectionSample() {
  char str1[50];
  char str2[50];
  char str3[50];
  char str4[50];
  char str5[50];
  char str6[50];
  char str7[50];
  poseCount = lokii.waitForPostureResult(true);
  if (poseCount > 0)
  {
    for (int i = (0); i <= (poseCount - 1); i = i + (1)) {
      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);
      sprintf(str1, "pose[%d] x = %d", i , poseX);
      sprintf(str2, "pose[%d] y = %d", i , poseY);
      sprintf(str3, "pose[%d] width = %d", i , poseW);
      sprintf(str4, "pose[%d] height = %d", i , poseH);
      sprintf(str5, "pose[%d] LH Code = %d", i , poseLeftHandCode);
      sprintf(str6, "pose[%d] RH Code = %d", i , poseRightHandCode);
      sprintf(str7, "NoseX = %d NoseY %d", poseNosePos.x, poseNosePos.y);
      Serial.println(str1);
      Serial.println(str2);
      Serial.println(str3);
      Serial.println(str4);
      Serial.println(str5);
```

```
    Serial.println(str6);
    Serial.println(str7);
    sprintf(str7, "%d,%d,", poseNosePos.x, poseNosePos.y);
    printout += String(str7);
    if (i == 0)
    {
      lokii.lcdPrint(str1, 2);
      lokii.lcdPrint(str2, 3);
      lokii.lcdPrint(str3, 4);
      lokii.lcdPrint(str4, 5);
      lokii.lcdPrint(str5, 6);
      lokii.lcdPrint(str6, 7);
      lokii.lcdPrint(str7, 8);
    }
    else
    {
      lokii.lcdPrint(str1, 9);
      lokii.lcdPrintAll(str2, str3, str4, str5, str6);
    }
   }
  }
  else
  {
    lokii.clearLCD();
  }
}
void setup() {
   Serial.begin(115200);
   lokii.connect();
}
void loop() {
  lokii.setCameraMode(L_POSTURE);
  while (true) {
    poseDetectionSample();
  }
}
```