moblabpy

Release 0.1.0

Moblabpy Official

Apr 19, 2021

CONTENTS:

CHAPTER ONE

WHAT IS MOBLABPY

Moblabpy is a software toolbox designed to emulate the working of a communication channel and enable the conducting of communication related experiments in associated courses, System View of Communications: from Signals to Packets or Error Control Coding and Information Theory. Moblabpy gives users the ability to do experiments without dedicated hardware and software license, but with their laptop (or desktop) computers and mobile phones anywhere at any time. This toolbox is built upon Python as a module package. Python 3.9 and the Moblabpy module are required on the computer. “IP Camera Lite” for iPhones and iOS devices, or “IP Camera” for Android devices are also required (Both developed by ShenYao). Moblabpy acts as a control panel of a visible light communication system. The computer display acts as a transmitter and the phone camera as a receiver. Users can experiment the effect of noise and error correcting codes have in communication.

CHAPTER

TWO

GET STARTED

CHAPTER

THREE

PYTHON AND MOBLABPY INSTALLATION GUIDE

Installing and managing packages in Python is complicated, there are a number of alternative solutions for most tasks. This guide tries to give the reader a sense of the best (or most popular) solutions, and give clear recommendations. It focuses on users of Python and the Moblabpy on common operating systems and hardware.

3.1 PYTHON AND MOBLABPY INSTALLATION (WINDOWS USERS)

1. Check if Python is already installed (You can type *python –version* on cmd). If you have already installed Python, you can skip the next two steps, otherwise, please follow the next step to install Python.
2. Install Python here and select the most recent version.
3. Follow the instruction of the installer and finish installation. Make sure to tick the option *Add Python 3.x (the version that you download) to PATH* at the start of the installer window.
4. After the installation finished, press `Win` key, type cmd and hit `Enter` .
5. Type `pip install moblabpy` in the cmd to install Moblabpy.

3.2 PIP IS NOT RECONGNIZED AS AN INTERNAL OR EXTERNAL COMMAND

If you see the `pip is not recognized as an internal or external command` when you type `pip install moblabpy` in the cmd. It means that you forgot to add Python in PATH during the installation of Python. You can add it by the following step:

1. Press `Win + R` , type `%AppData%` and hit `Enter`
2. Back out one directory from Roaming
3. Navigate to `Local/Programs/Python/Python39/Scripts`
4. Copy the file path by selecting the whole directory in the address bar
5. Press `Win`, type Control Panel and hit `Enter`
6. Navigate to System and Security → System → Advanced System Setting → Environment Variables
7. Select Path in the top box and Edit
8. Select New and paste the file path copied from step 4

3.3 PYTHON AND MOBLABPY INSTALLATION (MAC USERS)

1. Same as step 1 to 3 of window users (terminal for mac users)
2. After the installation finished, press `Cmd` and the space bar simultaneously, type `terminal` and hit `Enter`.
3. Type `pip install moblabpy` in the cmd to install Moblabpy.

3.4 PIP: COMMAND NOT FOUND

If you see the `pip: command not found` when you type `pip install moblabpy` in the terminal. It means that you forgot to add Python in PATH during the installation of Python. You can add it by the following step:

1. Type `which python3` in the terminal and the path of where python is located will be shown
2. Type `echo $PATH` in the terminal and it will show the path of your machine to look for command
3. Navigate back to home file by typing `cd`
4. Type `nano .bash_profile` in the terminal and hit `Enter`
5. Type `PATH="path/of/bin/folder/that/you/install/your/python:${PATH}"` and `export PATH` in the next line and save the changes
6. restart the terminal and it should be fine now

3.5 MORE DETAILS

For more details, you can go to MAC if you are mac users or WINDOWS if you are windows user.

CHAPTER

FOUR

MOBLABPY DOCUMENTATION

class PROPS

A class that contains all the configurations of the Moblabpy module

FPS: int = 15

The frame per second of the generated video.

BPS: int = 16

The number of color matrix in each frame. It can be set by `set_BPS()` .

ROW: int = 4

The number of row of color matrix in each frame. `ROW` must be equal to `COL` and they are defined by the square root of `BPS` .

COL: int = 4

Same as `ROW` .

INFO_BIT_SIZE: int = 50

The height and width of each color matrix. It is defined by `ROW` divides 200.

START_BIT: str = "0"

The string representation of start bit and end bit of the video.

END_BIT: str = "0"

Same as `START_BIT` .

VID_SOURCE: str = "./16bps.mp4"

The video name of the generated video. It will be set by `set_BPS()` automatically if py:attr:*BPS* is modified.

SCALE: int = 2

The scale of the window size of the video player embedded in the `MobLabPy` .

set_BPS(*bps*)

Set the `BPS` of the video. It will also change the value of `ROW`, `COL`, `INFO_BIT_SIZE` and `VID_SOURCE` automatically.

Parameters **bps** (`int`) – The BPS of the video

class MobLabPy

A class that allows users to connect their ip camera with the program to simulate a telecommunication system. It use the python `multiprocessing` modules to reduce the time delay of decoding images and capturing images. The program uses `multiprocessing.Pipe` object to send images between the `send_frame()` and `recv_frame()` .

__init__(*self*, *ip_address*, *bit_mess*, *props=PROPS*)

Initialize the member variable of the `MobLabPy` object.

Parameters

- **ip_address** (*str*) – The ip address of the IP camera
- **bit_mess** (*str*) – The transmitted bit sequence
- **props** – A *PROPS* class

start (*self*)
Start getting frames from the ip camera and send it to the decoding function.

get_bit_seq (*self*)
Return the bit sequence of the :py:class`MobLabPy` object.

Returns bit_seq The received bit sequence

Return type list(char)

img_to_bit_seq (*frame*, *pt1=()*, *pt2=()*, *props=PROPS*)
Convert black and white color to "0" and "1" respectively.

Parameters

- **frame** (numpy.ndarray) – Image.
- **pt1** (*tuple(int, int)*) – Vertex of the area bounded by qr codes, defaults to ()
- **pt2** (*tuple(int, int)*) – Vertex of the area bounded by qr codes opposite to pt1, defaults to ()
- **props** (*PROPS*) – The *PROPS* object that contains the configurations of the program

Returns bit_seq, pt1, pt2 Decoded bit sequence and vertices of the area bounded by qr codes in opposite direction.

Return type tuple(str, tuple(int, int), tuple(int, int))

Raises **IndexError** – If it fails to calculate the mean of the RGB of the color matrix

check_orientation (*frame*, *pt1*, *pt2*)
Check the orientation of the image according to the qr codes in the corner.

Parameters

- **frame** (numpy.ndarray) – Image
- **pt1** (*tuple(int, int)*) – Vertex of the area bounded by qr codes.
- **pt2** (*tuple(int, int)*) – Vertex of the area bounded by qr codes opposite to pt1.

Return result Number of rotation needed to be performed

Return type int

Raises **IndexError** – If it fails to locate the qr codes

point_rotation (*pt1*, *pt2*, *width*, *rotation*)
Perform the rotation of vertices of the color matrix area.

Parameters

- **pt1** (*tuple(int, int)*) – Vertex of the color matrix
- **pt2** (*tuple(int, int)*) – Vertex of the color matrix opposite to pt1
- **width** – Width of the image that contains the color matrix
- **rotation** (*int*) – Number of rotation that needed to be performed

Type int

Returns pt1, pt2 Vertices of the color matrix area after rotation

Type tuple(tuple(int, int), tuple(int, int))

`find_corner` (*frame*, *pt1=()*, *pt2=()*)
Find the vertex of the color matrix surrounded by but exclude qr codes.

Parameters

- **`frame`** (`numpy.ndarray`) – Image
- **`pt1`** (*`tuple(int, int)`*) – Vertex of the area include qr codes, defaults to ()

Prarm pt2 Vertex of the area include qr codes opposite to pt1, defaults to ()

Returns size, pt1, pt2 size of the color matrix area and its vertices in opposite direction

Return type tuple(int, tuple(int, int), tuple(int, int))

`find_min_and_max` (*points*, *min_pt*, *max_pt*)
Find the minimum and maximum xy-points in a list and the provides points.

Parameters

- **`points`** (*`list(int)`*) – A list of points
- **`min_pt`** (*`list(int)`*) – The minimum points provided
- **`max_pt`** (*`list(int)`*) – The maximum points provided

Returns min_pt, max_pt The minimum and maximum xy-points

Return type tuple(intuple(int, int)t, tuple(int, int))

`all_zeros_or_ones` (*bit_seq*)
Check if the bit sequence contains only 0 or 1.

Parameters **`bit_seq`** (*`str`*) – The bit sequence needed to be checked

Returns "True" if the bit sequence only contains 0 or 1, "False" otherwise

Return type bool

Raises **`ValueError`** – it input contains non numerical values

`str_to_ascii_seq` (*my_str='Apple'*)
Convert each character in a string to its corresponding ascii code.

Parameters **`my_str`** (*`str`*) – The string wanted to be converted, defaults to *Apple*

Return ascii_arr A list of integer that contains all the ascii code of each character in my_str

Return type list(int)

`ascii_seq_to_bit_seq` (*ascii_arr*)
Convert a list of ascii code to its binary representation.

Parameters **`ascii_arr`** (*`list(int)`*) – A list that contains ascii code

Return bit_arr An integer binary list

Return type list(int)

`bit_seq_to_ascii_seq` (*bit_arr*)
Perform byte conversion of an array.

Parameters **`bit_arr`** (*`list(int)`*) – An integer binary array

Return ascii_arr A list contains all the corresponding ascii code

Return type list(int)

`ascii_seq_to_str`(*ascii_arr*)
Convert a list of ascii code to its corresponding character.

Parameters **`ascii_arr`** (*`list(int)`*) – A list that contains ascii code

Return my_str The character format of the ascii list

Return type str

`append_zeros`(*bit_seq*, *props=PROPS*, *is_str=False*)
Append zeros until the length of the input equals to the products of desired length.

Parameters

- **`bit_seq`** (*`list(int)`*) – Integer binary array
- **`props`** (*`PROPS`*) – *`PROPS`* object that contains the configurations of the program.

Return bit_arr An integer binary array which length is the product of the second parameter

Return type list(int)

`generate_img`(*dir_name*, *my_str='Hello World'*, *if_bin=False*, *props=PROPS*)
Generate images that contain color matrix of my_str parameter, with black represents 0 and white represents 1, and paste them with the qr codes (finder patterns) together. The generated images will be saved at the dir_name directory.

Parameters

- **`dir_name`** (*`str`*) – The directory name used to save the image generated
- **`my_str`** (*`str, list(int)`*) – The integer binary array or string which will be convert to color code formats, defaults to "Hello World"
- **`if_bin`** (*`bool`*) – *True* if my_str is an integer binary array, *False* otherwise, defaults to *False*
- **`props`** (*`PROPS`*) – the *`PROPS`* object that contains the properties of the program

`generate_res`()
Generate the res folder which contains all the head and end messages, start and end signals and finder patterns.

`generate_video`(*video_name='16bps.mp4'*, *mystr='Hello World'*, *if_bin=False*, *props=PROPS*)
Generate a video which contains start and end messages, start and end signals, finder patterns and color matrix of the mystr parameter, with desired fps.

Parameters

- **`video_name`** (*`str`*) – The name of the generated video in mp4 format
- **`mystr`** (*`str, list(int)`*) – The message that will be converted to color code, defaults to "Hello World"
- **`if_bin`** (*`bool`*) – *True* if mystr is an integer binary array, *False* otherwise, defaults to *False*
- **`props`** (*`PROPS`*) – the *`PROPS`* object that contains the properties of the program

`recv_frame`(*fps*, *is_pilot_bit_found*, *recver*, *bit_seq*, *flag*, *bit_mess*, *props=PROPS*)
Funtion that received frames from the sender function. It does all the manipulation of the captured iamge here, such as detecting start and end signals, checking rotations and decoding color codes back to 0 and 1.

Parameters

- **`fps`** (`multiprocessing.Value`) – A `multiprocessing.Value` object which represents the fps of the camera

- **is_pilot_bit_found** (`multiprocessing.Value`) – A `multiprocessing.Value` object which is *True* if the start signal is detected, *False* otherwise
- **recver** (`multiprocessing.Pipe`) – A `multiprocessing.Pipe` object that received images sent from the sender function
- **bit_seq** (`multiprocessing.Value`) – A `multiprocessing.Value` object whcih is the decoded bit sequence of received images
- **flag** (`multiprocessing.Value`) – A `multiprocessing.Value` object which is the state of the play video button
- **bit_mess** (*str*) – The transmitted bit sequence
- **props** (*PROPS*) – The *PROPS* object that contains the properties of the program

vid_player (*vid_source*, *flag*, *props=PROPS*)

A video player will be appeared if the play video button is clicked

Parameters

- **vid_source** (*str*) – The name of the video to pe played
- **flag** (`multiprocessing.Value`) – A `multiprocessing.Value` object which is the state of the play video button
- **props** (*PROPS*) – The *PROPS* object that contains the properties of the program

send_frame (*fps*, *is_pilot_bit_found*, *sender*, *ip_address*, *props=PROPS*)

A sender function which gets images from the connected camera and send it to the receiver function.

Parameters

- **fps** (`multiprocessing.Value`) – A `multiprocessing.Value` object which represents the fps of the camera
- **is_pilot_bit_found** (`multiprocessing.Value`) – A `multiprocessing.Value` object which is *True* if the start signal is detected, *False* otherwise
- **sender** (`multiprocessing.Pipe`) – A `multiprocessing.Pipe` object that send images captured by the connected camera
- **ip_address** (*str*) – IP address of the camera that is going to be connected
- **props** (*PROPS*) – The :py:class:PROPS object that contains the properties of the program

encode (*D*, *G*)

Perform matrix multiplication on parameter D and G, which is D x G.

Parameters

- **D** (*list (int)*) – A 1 x k matrix
- **G** (`numpy.ndarray`) – A generator matrix

Return C A 1 x n matrix

Return type `numpy.ndarray`

Raises

- **ValueError** – If the number of columns in the first matrix not equals to the number of rows in the second matrix
- **UFuncTypeError** – If both parameters are not integer `numpy.ndarray`

syndrome(*R*, *H*)
Calculate the syndrome of the received bit sequence.

Parameters

- **R** (*list(int)*) – A 1 x n matrix
- **H** (numpy.ndarray) – A parity check matrix

class monitor

__init__(*self*, *master*, *bit_mess*)
Defines the monitor window.

Parameters

- **master** (*Tk*) – Root window
- **bit_mess** (*str*) – The bit sequence to be sent

update_recv(*self*, *bit_seq*)
Update the received and error bits textbox upon call.

Parameters **bit_seq** – The bits that are collected from the receiver

button_toggle(*self*)
Switch the status of the button to True, which indicates it has been pressed.

get_button_status(*self*)
Return the status of the button.

Returns The status of the button, whether it has been pressed

Return type bool

close_windows(*self*)
Destroy the root window.

class VidCap

__init__(*self*, *scale*, *vidsource=0*)
Defines the video to be played and its porperty.

Parameters

- **scale** (*int*) – The scale of which the video is being played. The higher the number, the bigger the video
- **vidsource** (*str*) – The file path of the video

get_fps(*self*)
Return the fps of the video.

Returns The fps of the video

Return type int

get_height(*self*)
Return the height of the video.

Returns The height of the video

Return type int

get_width(*self*)
Return the width of the video.

Returns The width of the video

Return type int

get_frame(*self*)
Get one frame from the video.

Returns ret, frame whether a frame is successfully grabbed; the image from the video, None otherwise.

Return type tuple(`bool`, `numpy.ndarray`)

reset(*self*, *vidsource=0*)
Reset the video source and start from the beginning.

__del__(*self*)
Release the video when the window is closed.

class player

__init__(*self*, *master*, *scale*, *vidsource=0*)
Defines the video player window.

Parameters

- **master** (*Tk*) – Root window
- **scale** (*int*) – The scale of which the video is being played. The higher the number, the bigger the video
- **vidsource** (*str*) – The file path of the video

update(*self*)
Update the video canvas to display the next frame.

restart_vid(*self*)
Replay the video.

close_windows(*self*)
Destroy the root window.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX