DynamiKontrol

Taehee Lee

Jun 28, 2021

CONTENTS:

1 Installation **3**
- 1.1 Install Python . . . 3
- 1.2 Install DynamiKontrol Package in pip . . . 3

2 dynamikontrol package **5**
- 2.1 Submodules . . . 5
- 2.2 dynamikontrol.LED module . . . 5
- 2.3 dynamikontrol.Module module . . . 6
- 2.4 dynamikontrol.Motor module . . . 7
- 2.5 dynamikontrol.Switch module . . . 10
- 2.6 dynamikontrol.Timer module . . . 11
- 2.7 Module contents . . . 12

3 Face Tracking Camera **13**
- 3.1 Explanation . . . 13
- 3.2 Source Code . . . 14

4 Lunch Roulette **17**
- 4.1 Source Code . . . 17

5 Dial GUI **19**
- 5.1 Source Code . . . 19

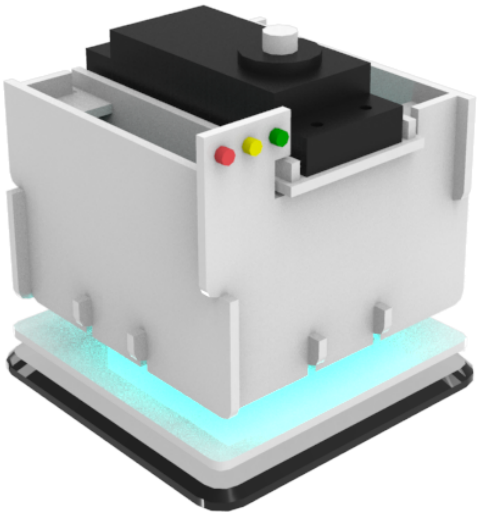
6 Thermometer **21**
- 6.1 Source Code . . . 22

7 IoT Door Lock **23**
- 7.1 Source Code . . . 23

8 Indices and tables **25**

Python Module Index **27**

Index **29**



DynamiKontrol is a API for controlling DynamiKontrol module as motors, LEDs or so on. You can find our modules at https://dk.m47rix.com. Please contact us matrix.ai.solution@gmail.com

CHAPTER

ONE

INSTALLATION

1.1 Install Python

First download and install Python from here:

https://www.python.org/downloads

or use Anaconda:

https://www.anaconda.com/products/individual

1.2 Install DynamiKontrol Package in pip

To install DynamiKontrol package via pip, simply execute in terminal:

```
pip install -U DynamiKontrol
```

This installs the latest version from pypi.

CHAPTER

TWO

DYNAMIKONTROL PACKAGE

2.1 Submodules

2.2 dynamikontrol.LED module

class dynamikontrol.LED.**LED**(*module*)

Bases: `object`

LED submodule class.

```
from dynamikontrol import Module
import time

module = Module()

module.led.blink(color='r') # blink red
module.led.toggle(color='g') # toggle green

while True:
    module.led.on(color='y') # turn on yellow
    time.sleep(0.1)

    module.led.off(color='y') # turn off yellow
    time.sleep(0.1)

module.disconnect()
```

Parameters **module** (*object*) – Module object.

blink(*color='all'*, *on_delay=0.256*, *off_delay=0.256*)

Blink the LED light periodically.

Parameters

- **color** (*str, optional*) – Color of the LED light. `r`, `y` or `g`. Defaults to `all`.
- **on_delay** (*float, optional*) – Delay time for turned-on status. `on_delay` must be between `0.0` to `65.0` in second. Defaults to `0.256`.
- **off_delay** (*float, optional*) – Delay time for turned-off status. `off_delay` must be between `0.0` to `65.0` in second. Defaults to `0.256`.

off(*color='all'*)
Turn off the LED light

Parameters **color** (*str, optional*) – Color of the LED light. `r`, `y` or `g`. Defaults to `all`.

on(*color='all'*)
Turn on the LED light.

Parameters **color** (*str, optional*) – Color of the LED light. `r`, `y` or `g`. Defaults to `all`.

toggle(*color='all'*)
Toggle the LED light. Turn off while the light on status and turn on while the light off.

Parameters **color** (*str, optional*) – Color of the LED light. `r`, `y` or `g`. Defaults to `all`.

2.3 dynamikontrol.Module module

class dynamikontrol.Module.**Module**(*serial_no=None*, *debug=False*)
Bases: `object`

Module class.

```
from dynamikontrol import Module

module = Module(serial_no) # specify the module by serial number

# Print module serial number
print('Serial number: %s' % (module.get_serial_no(),)

module.disconnect()
```

Parameters

- **serial_no** (*str*) – Specify serial number of the module.
- **debug** (*bool*) – print debug messages.

connect()
Connect to the module.

disconnect()
Close the connection of the module. Must include `module.disconnect()` at the end of the code so that module can close connection properly.

get_fw_version()
Get firmware version of the connected module.

Returns Device firmware version.

Return type str

get_id()
Get ID of the connected module.

Returns Module ID.

Return type int

get_serial_no()
Get serial number of the connected module.

Returns Serial number.

Return type str

get_time()
Get device time of the connected module.

Returns Device time.

Return type datetime

send(*data*)
Send the data to the module

Parameters **data** (*bytearray of int*) – Data to send.

set_default_switch_operation(*on*)
Set either turn default switch operation on or not. Motor is spinning when switch is pressed by default.

Parameters **on** (*bool*) – Set default operation on or off.

2.4 dynamikontrol.Motor module

class dynamikontrol.Motor.**BLDC**(*module*)
Bases: object

BLDC(Speed) motor submodule class.

```
from dynamikontrol import Module
import time

module = Module()

module.motor.speed(1000)
time.sleep(5)
module.motor.stop()

module.disconnect()
```

Parameters **module** (*object*) – Module object.

get_speed(*func*, *unit='rpm'*)
Get speed of the motor asynchronously.

```
from dynamikontrol import Module
import time

module = Module()

module.motor.speed(4000, period=10)

def get_speed_cb(speed):
    print('Current Speed', speed)

for i in range(60):
    time.sleep(0.5)
```

(continues on next page)

(continued from previous page)

```
    module.motor.get_speed(func=get_speed_cb)

module.disconnect()
```

Parameters

- **func** (*function*) – Callback function when getting speed from the motor.
- **unit** (*str, optional*) – Speed unit must be one of `rpm`, `deg/s` and `rad/s`. Defaults to `'rpm'`.

speed(*speed*, *period=None*, *unit='rpm'*, *func=None*, *args=()*, *kwargs={}*)

Control speed of the motor.

Parameters

- **speed** (*int*) – If `speed > 0` spins along clockwise, otherwise spins along counter clockwise.
- **period** (*int, optional*) – Control period. `period` must be between `0.0` to `65.0` in second. Defaults to `None`.
- **unit** (*str, optional*) – Speed unit must be one of `rpm`, `deg/s` and `rad/s`. Defaults to `'rpm'`.
- **func** (*function, optional*) – Callback function when motor has been stopped. Defaults to `None`.
- **args** (*tuple, optional*) – args for callback function. Defaults to `()`.
- **kwargs** (*dict, optional*) – kwargs for callback function. Defaults to `{}`.

stop()

Stop the motor immediately.

class dynamikontrol.Motor.**Motor**(*module*)

Bases: `object`

angle(**args*, ***kwargs*)

get_offset(**args*, ***kwargs*)

get_speed(**args*, ***kwargs*)

set_offset(**args*, ***kwargs*)

speed(**args*, ***kwargs*)

stop(**args*, ***kwargs*)

class dynamikontrol.Motor.**Servo**(*module*)

Bases: `object`

Servo(Angle) motor submodule class.

```
from dynamikontrol import Module
import time

module = Module()

module.motor.angle(0)
```

(continues on next page)

(continued from previous page)

```
time.sleep(2)

while True:
    module.motor.angle(45)
    time.sleep(2)

    def cb(string):
        print(string)

    module.motor.angle(-45, func=cb, args=('hello',)) # print 'hello' when motor␣
↪stopped at -45 degree.
    time.sleep(2)

module.disconnect()
```

Parameters **module** (*object*) – Module object.

angle(*angle*, *period=None*, *func=None*, *args=()*, *kwargs={}*)
Control the angle of motor.

Parameters

- **angle** (*int*) – If `angle > 0` moves along clockwise, otherwise moves along counter clockwise. `angle` must be between `-85` to `85` in degrees.
- **period** (*float, optional*) – Control period. `period` must be between `0.0` to `65.0` in second. Defaults to `None`.
- **func** (*function, optional*) – Callback function when motor has been stopped. Defaults to `None`.
- **args** (*tuple, optional*) – args for callback function. Defaults to `()`.
- **kwargs** (*dict, optional*) – kwargs for callback function. Defaults to `{}`.

get_offset()
Get offset of the motor.

Returns Offset of the motor in degree.

Return type float

set_offset(*angle*)
Set offset of the motor. If the motor angle is inclined slightly even angle set to 0, you can adjust offset of the motor manually.

Parameters **angle** (*float*) – Offset of the motor in degree. e.g) 17.5

2.5 dynamikontrol.Switch module

class dynamikontrol.Switch.**Switch**(*module*)

Bases: object

Switch submodule class.

```
from dynamikontrol import Module
import time

module = Module()

def callback(string, angle):
    print(string)
    module.motor.angle(angle)

module.switch.press(callback, ('Switched to on', 85,))
module.switch.release(callback, ('Switched to off', 0,))

while True:
    time.sleep(1)

module.disconnect()
```

Parameters **module** (*object*) – Module object.

off(*func*, *args=()*, *kwargs={}*, *ch=0*)

Define callback function when switch is set to off status.

Parameters

- **func** (*function*) – Callback function.
- **args** (*tuple, optional*) – args. Defaults to ().
- **kwargs** (*dict, optional*) – kwargs. Defaults to {}.
- **ch** (*int, optional*) – Switch channel number. Must be 0 or 1. Defaults to 0.

on(*func*, *args=()*, *kwargs={}*, *ch=0*)

Define callback function when switch is set to on status.

Parameters

- **func** (*function*) – Callback function.
- **args** (*tuple, optional*) – args. Defaults to ().
- **kwargs** (*dict, optional*) – kwargs. Defaults to {}.
- **ch** (*int, optional*) – Switch channel number. Must be 0 or 1. Defaults to 0.

press(**args*, ***kwargs*)

Define callback function when switch is pressed. Exactly same as on method.

release(**args*, ***kwargs*)

Define callback function when switch is released. Exactly same as off method.

2.6 dynamikontrol.Timer module

class dynamikontrol.Timer.**Timer**

Bases: object

General timer class.

```
from dynamikontrol import Module, Timer
import time

t1 = Timer()
t2 = Timer()

module = Module()

t1.callback_at(func=module.led.toggle, args=('r',), at='2021-03-02 19:46:30',␣
↪interval=0.1)

t2.callback_after(func=module.led.toggle, args=('g',), after=1, interval=0.1)

time.sleep(5)

t1.stop()
t2.stop()

module.disconnect()
```

callback_after(*func*, *args=()*, *kwargs={}*, *after=0*, *interval=None*)

Call the callback function after specific time.

Parameters

- **func** (*function*) – Callback function.
- **args** (*tuple, optional*) – args. Defaults to ().
- **kwargs** (*dict, optional*) – kwargs. Defaults to {}.
- **after** (*int, optional*) – Callback delay time in seconds. Defaults to 0.
- **interval** (*int, optional*) – Callback interval time in seconds. Defaults to None.

callback_at(*func*, *args=()*, *kwargs={}*, *at=None*, *interval=None*)

Call the callback function at specific time.

Parameters

- **func** (*function*) – Callback function.
- **args** (*tuple, optional*) – args. Defaults to ().
- **kwargs** (*dict, optional*) – kwargs. Defaults to {}.
- **at** (*datetime str, optional*) – Callback time in datetime str. e.g) 2021-03-04 21:57:30. Defaults to None.
- **interval** (*[type], optional*) – Callback interval time in seconds. Defaults to None.

stop()

Stop the timer.

2.7 Module contents

CHAPTER

THREE

FACE TRACKING CAMERA

Face tracking algorithm is not included in DynamiKontrol package. So we use Mediapipe face tracking solution for face tracking and OpenCV for streaming webcam.

3.1 Explanation

Let's define `x1`, `x2` are top left point and bottom right point of face bounding box, then `cx` would be center point of the face. Mediapipe face tracking solution returns relative coordinates of the input image, so range of `cx` would be 0.0 to 1.0. For instance, if your face is located in exact center of the image, `cx` is 0.5.

```
cx = (x1 + x2) / 2 # center of the face

if cx < 0.4: # left -> clockwise
    angle += ANGLE_STEP
    module.motor.angle(angle)
elif cx > 0.6: # right -> counter clockwise
```

(continues on next page)

(continued from previous page)

```
    angle -= ANGLE_STEP
    module.motor.angle(angle)
```

If `cx < 0.4`, face is located in the left of the image, camera should move to left, and it means the motor should move to clockwise(+). Otherwise vice versa.

3.2 Source Code

```
import cv2
import mediapipe as mp
from dynamikontrol import Module

ANGLE_STEP = 1

module = Module()

mp_drawing = mp.solutions.drawing_utils
mp_face_detection = mp.solutions.face_detection

face_detection = mp_face_detection.FaceDetection(
    min_detection_confidence=0.7)

cap = cv2.VideoCapture(0)
angle = 0 # motor current angle

while cap.isOpened():
    ret, img = cap.read()
    if not ret:
        break

    img = cv2.flip(img, 1) # mirror image

    results = face_detection.process(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

    if results.detections:
        for detection in results.detections:
            mp_drawing.draw_detection(img, detection)

            x1 = detection.location_data.relative_bounding_box.xmin # left side of face␣
↪bounding box
            x2 = x1 + detection.location_data.relative_bounding_box.width # right side␣
↪of face bounding box

            cx = (x1 + x2) / 2 # center of the face

            if cx < 0.4: # left -> clockwise
                angle += ANGLE_STEP
                module.motor.angle(angle)
            elif cx > 0.6: # right -> counter clockwise
                angle -= ANGLE_STEP
```

(continues on next page)

(continued from previous page)

```
                module.motor.angle(angle)

            cv2.putText(img, '%d deg' % (angle), org=(10, 30), fontFace=cv2.FONT_HERSHEY_
↪SIMPLEX, fontScale=1, color=255, thickness=2)

            break

    cv2.imshow('Face Cam', img)
    if cv2.waitKey(1) == ord('q'):
        break

cap.release()
face_detection.close()
module.disconnect()
```

CHAPTER

FOUR

LUNCH ROULETTE

A Machine select your lunch menu with no worries.

4.1 Source Code

```
from dynamikontrol import Module
import time, random

module = Module()

direction = 1 # initial direction, clockwise
start_time = time.time()
stop_time = 5 # stop after 5 seconds
stop_angle = random.randint(-80, 80) # random angle

while True:
    if time.time() - start_time > stop_time:
        module.motor.angle(angle=stop_angle)
        print(f'Motor stopped at {stop_angle} degree')
        break

    direction = direction * -1 # change direction
    module.motor.angle(angle=direction * 80)
    time.sleep(0.6)

module.led.blink()
module.disconnect()
```

CHAPTER

FIVE

DIAL GUI

Simple dial GUI example using PyQt5.

5.1 Source Code

Install PyQt5

```
pip install PyQt5
```

```
from dynamikontrol import Module
from PyQt5.QtWidgets import *
import sys

class Dial(QWidget):
    def __init__(self, module):
        QWidget.__init__(self)
        self.module = module

        self.dial = QDial()
        self.dial.setRange(-150, 150)
        self.dial.setNotchesVisible(True)
        self.dial.valueChanged.connect(self.dialMoved)

        layout = QGridLayout()
        layout.addWidget(self.dial)
        self.setLayout(layout)

        self.setGeometry(500, 500, 500, 500)

    def dialMoved(self):
        self.module.motor.angle(self.dial.value())
        print(f'Dial value {self.dial.value()}')

if __name__ == '__main__':
    app = QApplication(sys.argv)

    screen = Dial(Module())
    screen.show()
```

(continues on next page)

(continued from previous page)

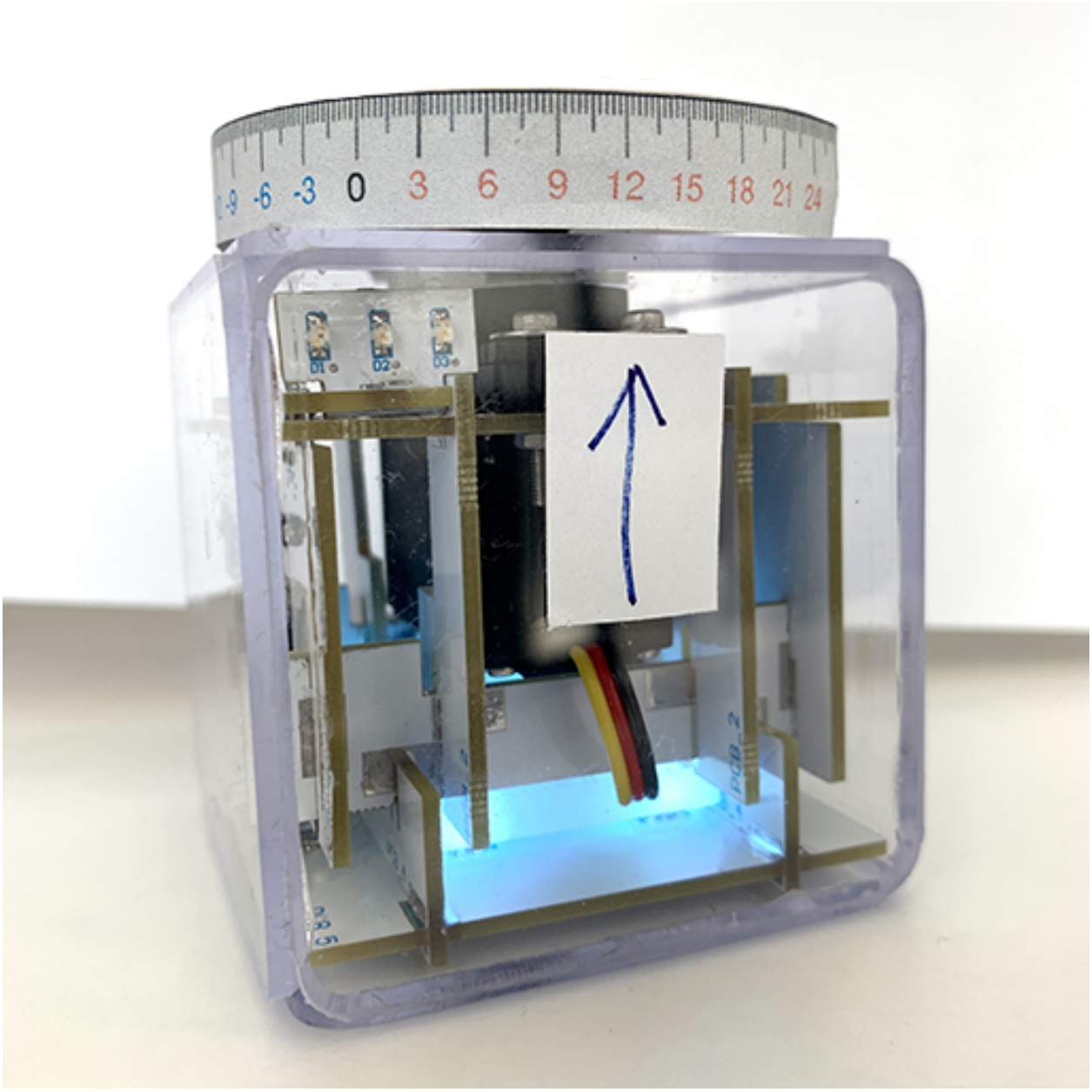
```
    sys.exit(app.exec_())
```

CHAPTER

SIX

THERMOMETER

Simple IoT analogue thermometer using weather API. It is around 13 degrees Celsius in the image below.



You can download a graduated ruler from this link.

https://github.com/TheMatrixGroup/DynamiKontrol/raw/master/examples/thermometer.pdf

6.1 Source Code

```
from dynamikontrol import Module
import requests, time
from datetime import datetime

module = Module()

# weather of Seoul, Korea
lat = 37.566536  # latitude
lon = 126.977966 # longitude
url = f'https://fcc-weather-api.glitch.me/api/current?lat={lat}&lon={lon}'

while True:
    res = requests.get(url).json()
    temp = res['main']['temp']

    print(f'{datetime.now()} temperature {temp} degree')

    angle = int(temp * 10 / 3)
    module.motor.angle(angle)

    time.sleep(60)

module.disconnect()
```

CHAPTER

SEVEN

IOT DOOR LOCK

You can build IoT smart door lock system that can be integrated in the mobile app and the web app.

In this example, we use Flask web framework. You can also use Node.js and other languages and frameworks.

7.1 Source Code

Server `door_lock.py`

```
from dynamikontrol import Module
from flask import Flask, request, render_template

module = Module()

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/change', methods=['POST'])
def change():
    is_locked = request.form['is_locked']

    if is_locked == 'true':
        module.motor.angle(85)
        module.led.off(color='r')
        module.led.on(color='g')
    else:
        module.motor.angle(-45)
        module.led.off(color='g')
        module.led.on(color='r')

    return {'result': True}

if __name__ == '__main__':
    app.run()
    module.disconnect()
```

Client `templates/index.html`

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no
↪">
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/
↪bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/
↪iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
    <title>IoT Door Lock System</title>
  </head>

  <body>
    <div class="d-flex justify-content-center mt-5">
      <h5>IoT Door Lock System</h5>
    </div>
    <div class="d-flex justify-content-center">
      <div class="custom-control custom-switch">
        <input type="checkbox" class="custom-control-input" id="lockSwitch">
        <label class="custom-control-label" for="lockSwitch">Lock the Door</label>
      </div>
    </div>

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></
↪script>

    <script>
      $(function() {
        $('#lockSwitch').change(function(e) {
          $.ajax({
            method: 'POST',
            url: '/change',
            data: { is_locked: this.checked }
          })
          .done(function(msg) {
            console.log(msg);
          });
        });
      })
    </script>
  </body>
</html>
```

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

dynamikontrol, 12
dynamikontrol.LED, 5
dynamikontrol.Module, 6
dynamikontrol.Motor, 7
dynamikontrol.Switch, 10
dynamikontrol.Timer, 11

INDEX

A

angle() (*dynamikontrol.Motor.Motor method*), 8
angle() (*dynamikontrol.Motor.Servo method*), 9

B

BLDC (*class in dynamikontrol.Motor*), 7
blink() (*dynamikontrol.LED.LED method*), 5

C

callback_after() (*dynamikontrol.Timer.Timer method*), 11
callback_at() (*dynamikontrol.Timer.Timer method*), 11
connect() (*dynamikontrol.Module.Module method*), 6

D

disconnect() (*dynamikontrol.Module.Module method*), 6
dynamikontrol
    module, 12
dynamikontrol.LED
    module, 5
dynamikontrol.Module
    module, 6
dynamikontrol.Motor
    module, 7
dynamikontrol.Switch
    module, 10
dynamikontrol.Timer
    module, 11

G

get_fw_version() (*dynamikontrol.Module.Module method*), 6
get_id() (*dynamikontrol.Module.Module method*), 6
get_offset() (*dynamikontrol.Motor.Motor method*), 8
get_offset() (*dynamikontrol.Motor.Servo method*), 9
get_serial_no() (*dynamikontrol.Module.Module method*), 6
get_speed() (*dynamikontrol.Motor.BLDC method*), 7
get_speed() (*dynamikontrol.Motor.Motor method*), 8
get_time() (*dynamikontrol.Module.Module method*), 7

L

LED (*class in dynamikontrol.LED*), 5

M

module
    dynamikontrol, 12
    dynamikontrol.LED, 5
    dynamikontrol.Module, 6
    dynamikontrol.Motor, 7
    dynamikontrol.Switch, 10
    dynamikontrol.Timer, 11
Module (*class in dynamikontrol.Module*), 6
Motor (*class in dynamikontrol.Motor*), 8

O

off() (*dynamikontrol.LED.LED method*), 5
off() (*dynamikontrol.Switch.Switch method*), 10
on() (*dynamikontrol.LED.LED method*), 6
on() (*dynamikontrol.Switch.Switch method*), 10

P

press() (*dynamikontrol.Switch.Switch method*), 10

R

release() (*dynamikontrol.Switch.Switch method*), 10

S

send() (*dynamikontrol.Module.Module method*), 7
Servo (*class in dynamikontrol.Motor*), 8
set_default_switch_operation() (*dynamikontrol.Module.Module method*), 7
set_offset() (*dynamikontrol.Motor.Motor method*), 8
set_offset() (*dynamikontrol.Motor.Servo method*), 9
speed() (*dynamikontrol.Motor.BLDC method*), 8
speed() (*dynamikontrol.Motor.Motor method*), 8
stop() (*dynamikontrol.Motor.BLDC method*), 8
stop() (*dynamikontrol.Motor.Motor method*), 8
stop() (*dynamikontrol.Timer.Timer method*), 11
Switch (*class in dynamikontrol.Switch*), 10

T

Timer (*class in dynamikontrol.Timer*), 11
toggle() (*dynamikontrol.LED.LED method*), 6