edurov Documentation

Release 0.0.5

Trolllabs

Jul 12, 2018

Contents

Stream camera feed from a Raspberry Pi camera to any web browser on the network. Control the robot with your keyboard directly in the browser.

The eduROV project is all about spreading the joy of technology and learning. The eduROV is being developed as a DIY ROV kit meant to be affordable and usable by schools, hobbyists, researchers and others as they see fit. We are committed to be fully open-source, both software and hardware-wise, everything we develop will be available to you. Using other open-source and or open-access tools and platforms.

GitHub https://github.com/trolllabs/eduROV

PyPI https://pypi.org/project/edurov/

Documentation http://edurov.readthedocs.io

Engage eduROV https://www.edurov.no/

CHAPTER 1

Main features

1. **Low video latency**

   You can stream HD video from the Raspberry Pi camera to any unit on the same network with a video delay below 200ms.

2. **No setup required**

   The package works by displaying the video feed and other content in a web browser. This means that you can use any device to display your interface.

3. **Very easy to use**

   With the exception of Pyro4 (which is installed automatically), edurov doesn't require any other packages or software. Everything is written in python and html. 4 lines of code is everything needed to get started!

4. **Highly customizable**

   Since you define the html page yourself, you can make it look and work exactly the way you want. Use css and javascript as much as you want.

5. **True parallelism**

   Need to control motors, read sensor values and display video feed at the same time? edurov can spawn your functions on multiple CPU cores while still maintaining the possibility to share variables.

CHAPTER 2

Prerequisites

- eduROV requires python 3, if you don't have python installed, you can download it here: https://www.python.org/downloads/
- the camera on the raspberry pi has to be enabled, see https://www.raspberrypi.org/documentation/configuration/camera.md

CHAPTER 3

Installation

Run the following commands in a terminal on the Raspberry Pi.:

```
sudo pip3 install edurov
```

For a more in depth description visit the official documentation.

CHAPTER 4

Usage

4.1 Engage eduROV submersible

On the Raspberry Pi, run the following command:

```
edurov-web
```

This will start the web server and print the ip where the web page can be viewed, e.g. `Visit the webpage at 192.168.0.197:8000`.

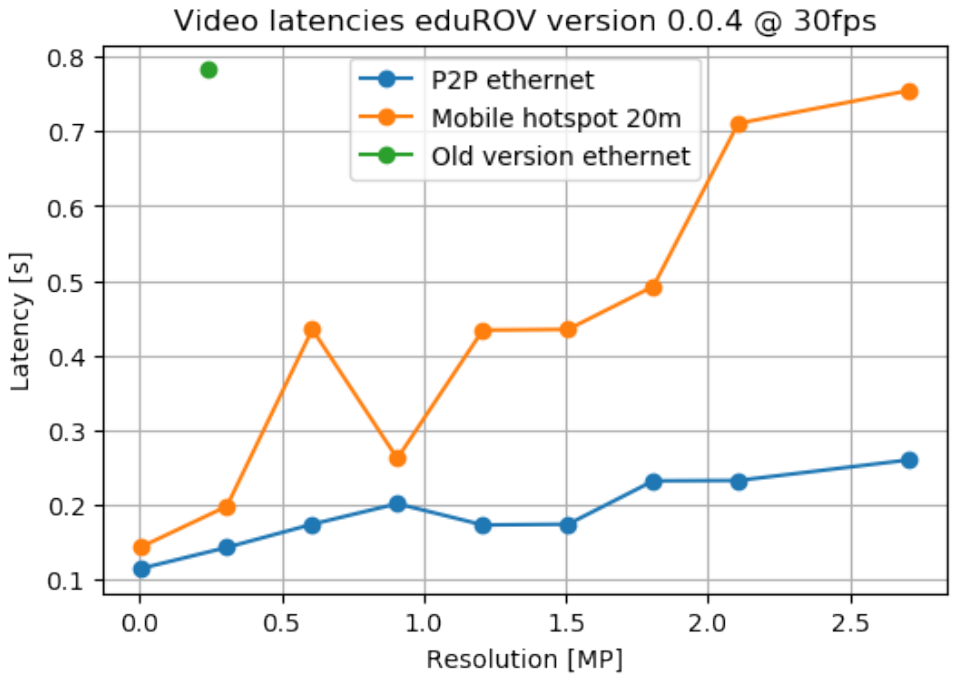
4.2 Create your own

The eduROV package includes multiple classes and functions to facilitate easy robot communication with video feed. It will get you up and running in a matter of minutes. Visit the official documentation for a *getting started*, examples and API.

CHAPTER 5

Performance

The eduROV package were created with a strong focus on keeping the latency at a minimum. When deploying on a wireless network the actual performance will vary depending on factors such as distance, interference and hardware.



MP	Resolution
0.007	100x75
0.307	640x480
0.608	900x675
0.907	1100x825
1.208	1268x951
1.508	1417x1063
1.808	1552x1164
2.107	1676x1257
2.708	1900x1425

CHAPTER 6

Author

The package is created by Martin Løland as part of the master thesis at Norwegian University of Science and Technology 2018

6.1 Installation

6.1.1 Raspbian

First, you will need a raspberry pi with an operating system running on it. Visit the official software guide for a step by step guide on how to do that..

6.1.2 Remote control

In most cases it is more practical to control the Raspberry Pi using another computer. The two most popular methods are with either SSH or VNC.

6.1.3 Update system

Make sure that your Raspberry Pi is up to date:

```
sudo apt-get update
sudo apt-get dist-upgrade
```

6.1.4 Python version

The edurov package requires python 3. If python 3 si not your default python version (check by running `python --version`), you can either (1) change the default python version, or (2) use pip3 and python3 instead.

1. **Change default python version**

   Take a look at this page.

2. **Use pip3 and python3**

   If you don't want to make any changes, you can call `pip3` instead of `pip` and `python3` instead of `python`. This will use version 3 when installing and running python scripts instead.

6.1.5 Install using pip

Install edurov, sudo rights are needed to enable console scripts:

```
sudo pip install edurov
```

6.1.6 Static IP

If you are remotely connected to the Pi it can be very useful with a static ip so that you can find the Pi on the network. How you should configure this depends how your network is setup. A guide can be found here.

6.1.7 Start at system startup

If you want the edurov-web command to run automatically when the raspberry pi has started. Run the following command:

```
sudo nano /etc/rc.local
```

Then add the following line to the bottom of the screen, but *before* the line that says `exit 0`:

```
edurov-web &
```

Exit and save by pressing CTRL+C, y, ENTER. The system then needs to be rebooted:

```
sudo shutdown -r now
```

6.2 Engage eduROV

6.2.1 Terminal command

By calling `edurov-web` in the terminal the edurov-web example will be launched. This command also supports multiple flags that can be displayed by running `edurov-web -h`

-r	resolution, use format WIDTHxHEIGHT (default 1024x768)
-fps	framerate for the camera (default 30)
-port	which port the server should serve it's content (default 8000)
-d	set to print debug information

Example

```
edurov-web -r 640x480 -fps 10
```

Will then set the the video to 640x480 @ 10 fps

6.3 Getting started

Tip: If you came here to find out how to to use the Engage ROV submersible, the Engage eduROV page is probably for you. If you instead plan to create your own ROV or make some kind of modifications, you are in the right place.

Note: Not all details at explained on this page. You should check the API page for more information on the classes, methods and parameters when you need.

On this page we will walk through the features example, one feature at a time. This example was created with the intention of describing all the features of the edurov package. Let's get started!

6.3.1 Displaying the video feed

There are two main parts needed in any edurov project. First, it's the python file that creates the *WebMethod* class and starts serving the server. Secondly, a index.html file that describes how the different objects will be displayed in the browser.

In the two code blocks underneath you can see how simple they can be created. The index.html file needs to be called exactly this. We use the `os.path()` library to ensure correct file path description.

Listing 1: features.py

```
import os
from edurov import WebMethod

# Create the WebMethod class
web_method = WebMethod(
    index_file=os.path.join(os.path.dirname(__file__), 'index.html'),
)
# Start serving the web page, blocks the program after this point
web_method.serve()
```

The index.html file must have an img element with `src="stream.mjpg"`. The server will then populate this image with the one coming from the camera.

Listing 2: index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Features</title>
</head>
<body>
        <img src="stream.mjpg">
</body>
</html>
```

Our file structure now looks like this:

```
project
├── features.py
└── index.html
```

If you wanted to have a security camera system this is all you had to do. If you instead want to control you robot through the browser or display other information, keep reading.

6.3.2 Moving a robot

This section will let us control the ROV from within the web browser. In computer technology there is something called *parallelism*. It basically means that the CPU does multiple things at the same time in different processes. This is an important feature of the edurov package as it let's us do many things without interrupting the video feed. (It wouldn't be very practical if the video stopped each time we moved the robot).

Reading keystrokes

First, we have to ask the browser to send us information when keys are pressed. We do this by including `keys.js` inside the `index.html` file. We have put it inside a folder called *static* as this is the convention for these kind of files.

Listing 3: index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Features</title>
    <script src="./static/keys.js"></script>
</head>
<body>
        <img src="stream.mjpg">
</body>
</html>
```

Listing 4: /static/keys.js

```
var last_key;

document.onkeydown = function(evt) {
    evt = evt || window.event;
    if (evt.keyCode != last_key){
        last_key = evt.keyCode;
        send_keydown(evt.keyCode);
    }
}

document.onkeyup = function(evt) {
    last_key = 0;
    send_keyup(evt.keyCode);
}

function send_keydown(keycode){
    var xhttp = new XMLHttpRequest();
    xhttp.open("GET", "/keydown="+keycode, true);
    xhttp.setRequestHeader("Content-Type", "text/html");
```

(continues on next page)

(continued from previous page)

```
    xhttp.send(null);
}

function send_keyup(keycode){
    var xhttp = new XMLHttpRequest();
    xhttp.open("GET", "/keyup="+keycode, true);
    xhttp.setRequestHeader("Content-Type", "text/html");
    xhttp.send(null);
}
```

Controlling motors (or anything)

In this example we will not show how to move the motors, instead the program will print out which arrow key you are pressing. You can then change the code to do whatever you want!

Listing 5: features.py

```
1  import os
2  import Pyro4
3  from edurov import WebMethod
4
5  def control_motors():
6      """Will be started in parallel by the WebMethod class"""
7      with Pyro4.Proxy("PYRONAME:KeyManager") as keys:
8          with Pyro4.Proxy("PYRONAME:ROVSyncer") as rov:
9              while rov.run:
10                 if keys.state('K_UP'):
11                     print('Forward')
12                 elif keys.state('K_DOWN'):
13                     print('Backward')
14                 elif keys.state('K_RIGHT'):
15                     print('Right')
16                 elif keys.state('K_LEFT'):
17                     print('left')
18
19 # Create the WebMethod class
20 web_method = WebMethod(
21     index_file=os.path.join(os.path.dirname(__file__), 'index.html'),
22     runtime_functions=control_motors,
23 )
24 # Start serving the web page, blocks the program after this point
25 web_method.serve()
```

On line 22 we are telling the *WebMethod* that `control_motors` should be a `runtime_function`. This starts the function in another process and shuts it down when we stop the ROV. For more information visit the API page. Since this function is running in another process it needs to communicate with the server. It does this by the help of `Pyro4` (line 2). We then connect to the `KeyManager` and `ROVSyncer` on line 7-8. This let's us access the variables we need.

The resulting file structure:

```
project
├── features.py
├── index.html
```

(continues on next page)

(continued from previous page)

```
└── static
    └── keys.js
```

6.3.3 Making it pretty

At this point our web page is very boring. It is white with one image. Since it's a html file we can add whatever we want to it! This time we are adding a header, a button to stop the server and some information. In addition we are adding some styling that will center the content and make it look nicer.

Listing 6: index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Features</title>
    <link rel="stylesheet" type="text/css" href="./static/style.css">
    <script src="./static/keys.js"></script>
</head>
<body>
    <main>
        <h2>Welcome to the features example</h2>
        <img src="stream.mjpg">
        <p>
            <a href="stop">Stop server</a>
        </p>
        <p>
            Use arrow keys to print statements in the terminal window.
        </p>
    </main>
</body>
</html>
```

Listing 7: /static/style.css

```
body {
    margin: 0;
    padding: 0;
    font-family: Verdana;
}
a {
    text-decoration: none;
}
img {
    width: 100%;
    height: auto;
}
main{
    width: 700px;
    margin-top: 20px;
    margin-left: auto;
    margin-right: auto;
}
```

```
project
├── features.py
├── index.html
└── static
    ├── keys.js
    └── style.css
```

6.3.4 Displaying sensor values

Coming soon

6.3.5 Custom responses

In some cases you want to display information in the browser that you want to create yourself in a python function. The *WebMethod* has a parameter exactly for this purpose.

Listing 8: features.py

```
import os
import subprocess

import Pyro4

from edurov import WebMethod


def my_response(not_used, path):
    """Will be called by the web server if it not able to process by itself"""
    if path.startswith('/cpu_temp'):
        cmds = ['/opt/vc/bin/vcgencmd', 'measure_temp']
        return subprocess.check_output(cmds).decode()
    else:
        return None


def control_motors():
    """Will be started in parallel by the WebMethod class"""
    with Pyro4.Proxy("PYRONAME:KeyManager") as keys:
        with Pyro4.Proxy("PYRONAME:ROVSyncer") as rov:
            while rov.run:
                if keys.state('K_UP'):
                    print('Forward')
                elif keys.state('K_DOWN'):
                    print('Backward')
                elif keys.state('K_RIGHT'):
                    print('Right')
                elif keys.state('K_LEFT'):
                    print('left')


# Create the WebMethod class
web_method = WebMethod(
    index_file=os.path.join(os.path.dirname(__file__), 'index.html'),
    runtime_functions=control_motors,
```

(continues on next page)

(continued from previous page)

```
37      custom_response=my_response
38  )
39  # Start serving the web page, blocks the program after this point
40  web_method.serve()
```

Listing 9: index.html

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4       <title>Features</title>
5       <link rel="stylesheet" type="text/css" href="./static/style.css">
6       <script src="./static/keys.js"></script>
7       <script src="./static/extra.js"></script>
8   </head>
9   <body>
10      <main>
11          <h2>Welcome to the features example</h2>
12          <img src="stream.mjpg">
13          <p>
14              <a href="stop">Stop server</a>
15              <button onclick="cpuTemp()">Display CPU temp</button>
16          </p>
17          <p>
18              Use arrow keys to print statements in the terminal window.
19          </p>
20      </main>
21  </body>
22  </html>
```

Listing 10: /static/extra.js

```
1   function cpuTemp(){
2       var xhttp = new XMLHttpRequest();
3       xhttp.onreadystatechange = function() {
4           if (this.readyState == 4 && this.stat == 200) {
5               alert('The CPU temperature is '+this.responseText);
6       };
7       xhttp.open("GET", "cpu_temp", true);
8       xhttp.send();
9   }
```

As an example we have created a button in `index.html` (line 15) which calls a function in `extra.js` that asks the server what the CPU temperature is. The new .js file is included as usual (`index.html` (line 7)). On line 7 in `extra.js` we send a GET request with a value of *cpu_temp*. The server does not know how it should answer this request, but since we have defined a `custom_response` (line 37) in `features.py` the request is forwarded to this function and we can create the response our self!

Note that this function needs to accept *two* parameters whereas the last one is path that is requested. If the path starts with `/cpu_temp` we can return the value, else return `None`.

```
project
├── features.py
├── index.html
└── static
    ├── keys.js
```

(continues on next page)

(continued from previous page)

```
    ├── style.css
    └── extra.js
```

6.3.6 Adding more pages

Coming soon.

6.4 Examples

Tip: The following examples can be downloaded from the eduROV examples folder.

6.4.1 Minimal working code

This is a bare minimum example so that the image stream and nothing more can be seen in the browser. A great starting point if you want to expand the functionality yourself.

Listing 11: minimal.py

```python
from os import path

from edurov import WebMethod

web_method = WebMethod(
    index_file=path.join(path.dirname(__file__), 'index.html')
)
web_method.serve()
```

Listing 12: index.html

```html
<!DOCTYPE html>
<html>
<head>
    <title>Minimal</title>
</head>
<body>
    <img src="stream.mjpg" style="transform:rotate(180deg)">
    <a href="stop">Stop Server</a>
</body>
</html>
```

```
project
├── minimal.py
└── index.html
```

6.4.2 Features

An example created to explain most of the features in the edurov package. See the *Getting started* page in the official documentation for a full walkthrough.

Listing 13: features.py

```
import os
import subprocess

import Pyro4

from edurov import WebMethod


def my_response(not_used, path):
    """Will be called by the web server if it not able to process by itself"""
    if path.startswith('/cpu_temp'):
        cmds = ['/opt/vc/bin/vcgencmd', 'measure_temp']
        return subprocess.check_output(cmds).decode()
    else:
        return None


def control_motors():
    """Will be started in parallel by the WebMethod class"""
    with Pyro4.Proxy("PYRONAME:KeyManager") as keys:
        with Pyro4.Proxy("PYRONAME:ROVSyncer") as rov:
            while rov.run:
                if keys.state('K_UP'):
                    print('Forward')
                elif keys.state('K_DOWN'):
                    print('Backward')
                elif keys.state('K_RIGHT'):
                    print('Right')
                elif keys.state('K_LEFT'):
                    print('left')


# Create the WebMethod class
web_method = WebMethod(
    index_file=os.path.join(os.path.dirname(__file__), 'index.html'),
    runtime_functions=control_motors,
    custom_response=my_response
)
# Start serving the web page, blocks the program after this point
web_method.serve()
```

Listing 14: index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Features</title>
    <link rel="stylesheet" type="text/css" href="./static/style.css">
    <script src="./static/keys.js"></script>
    <script src="./static/extra.js"></script>
</head>
<body>
    <main>
        <h2>Welcome to the features example</h2>
        <img src="stream.mjpg">
```

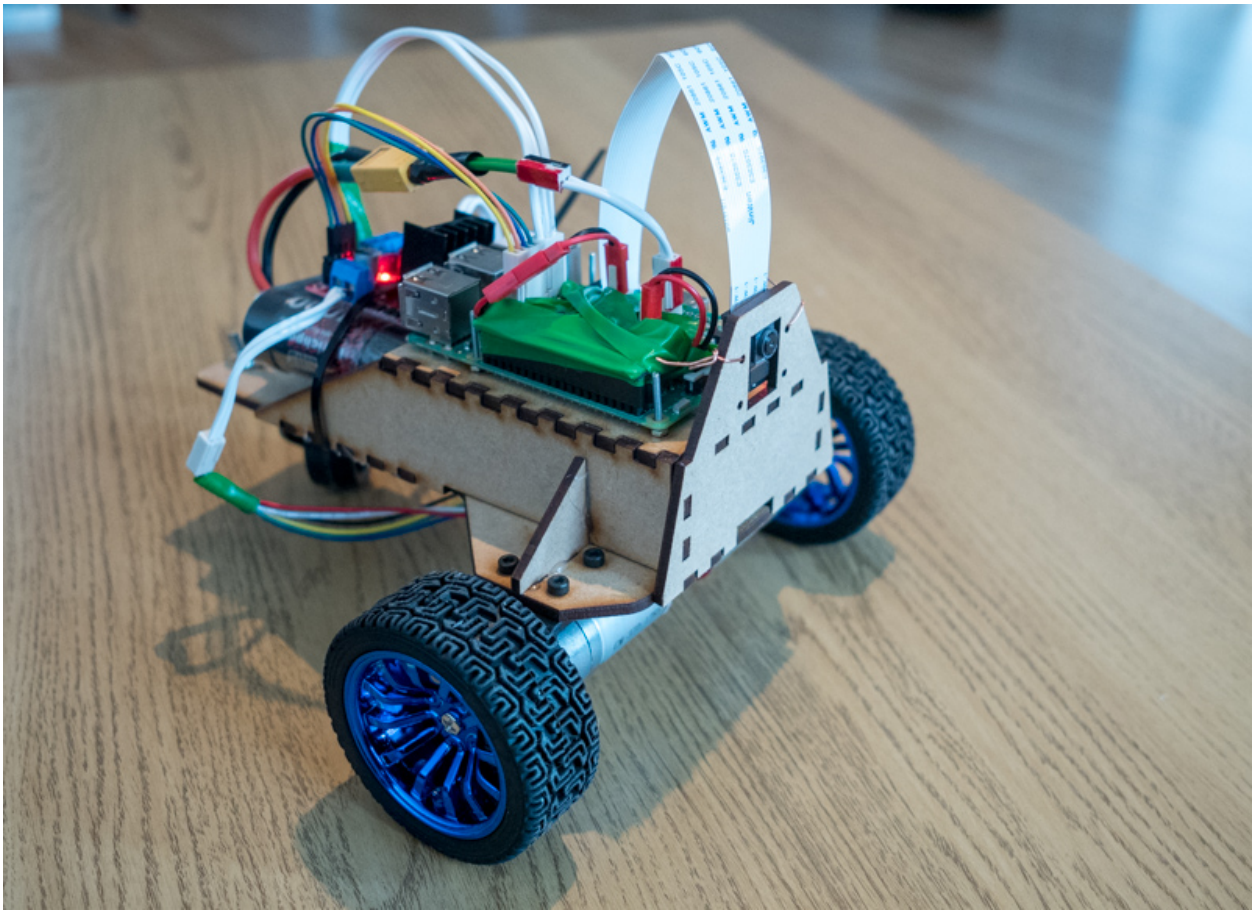
(continues on next page)

(continued from previous page)

```
        <p>
            <a href="stop">Stop server</a>
            <button onclick="cpuTemp()">Display CPU temp</button>
        </p>
        <p>
            Use arrow keys to print statements in the terminal window.
        </p>
    </main>
</body>
</html>
```

```
project
├── features.py
├── index.html
└── static
    ├── keys.js
    ├── extra.js
    └── style.css
```

6.4.3 Wireless RC car with camera feed



Create your very own wireless RC car with camera! The streaming video can be viewed in a browser on any device on the same network, it is controlled by using the arrow keys on the keyboard.

Bill of materials

Name	Price USD	Comment
Raspberry Pi Zero WH	18	A full size board can also be used
Raspberry Pi Camera Module V2	33	
DC 6V 210RPM Geard Motor Wheel Kit	23	found on eBay
L298N Dual H Bridge Motor Controller Board	1.8	found on eBay
DC-DC 5V 12V Step Down Module Converter 3A	1.6	found on eBay
Total	**76**	

In addition you will need a swivel wheel, M3/M2.5 bolts and nuts, cables and connectors, 12V battery and a car frame. The car frame used in the picture above was cut from 3mm MDF with a laser cutter and can be found here.

CAD files

Visit https://grabcad.com/library/772279

```
project
├── rc_car.py
├── index.html
├── electronics.py
└── static
    └── keys.js
```

6.4.4 Engage eduROV

This example is used to control the ROV used in the eduROV project, see www.edurov.no.

Listing 15: start.py

```
import os
import time

import Pyro4

from edurov import WebMethod
from edurov.utils import detect_pi, serial_connection, send_arduino, \
    receive_arduino, free_drive_space, cpu_temperature

if detect_pi():
    from sense_hat import SenseHat


def valid_arduino_string(arduino_string):
    if arduino_string:
        if arduino_string.count(':') == 2:
            try:
                [float(v) for v in arduino_string.split(':')]
                return True
            except:
                return False
    return False
```

(continues on next page)

(continued from previous page)

```
def arduino():
    lastState = '0000'
    ser = serial_connection()
    # 'letter': [position, value]
    config = {'w': [0, 1],
              's': [0, 2],
              'a': [1, 1],
              'q': [1, 2],
              'd': [2, 1],
              'e': [2, 2]}
    with Pyro4.Proxy("PYRONAME:KeyManager") as keys:
        with Pyro4.Proxy("PYRONAME:ROVSyncer") as rov:
            keys.set_mode(key='l', mode='toggle')
            while rov.run:
                dic = keys.qweasd_dict
                states = [0, 0, 0, 0]
                for key in config:
                    if dic[key]:
                        states[config[key][0]] = config[key][1]
                states[3] = int(keys.state('l'))
                state = ''.join([str(n) for n in states])
                if state != lastState:
                    lastState = state
                    if ser:
                        send_arduino(msg=state, serial_connection=ser)
                    else:
                        print(state)
                if ser:
                    arduino_string = receive_arduino(serial_connection=ser)
                    if valid_arduino_string(arduino_string):
                        v1, v2, v3 = arduino_string.split(':')
                        rov.sensor = {
                            'tempWater': float(v1),
                            'pressureWater': float(v2),
                            'batteryVoltage': float(v3)
                        }


def senser():
    sense = SenseHat()
    with Pyro4.Proxy("PYRONAME:ROVSyncer") as rov:
        while rov.run:
            orientation = sense.get_orientation()
            rov.sensor = {'temp': sense.get_temperature(),
                          'pressure': sense.get_pressure() / 10,
                          'humidity': sense.get_humidity(),
                          'pitch': orientation['pitch'],
                          'roll': orientation['roll'] + 180,
                          'yaw': orientation['yaw']}


def system_monitor():
    with Pyro4.Proxy("PYRONAME:ROVSyncer") as rov:
        while rov.run:
            rov.sensor = {'free_space': free_drive_space(),
                          'cpu_temp': cpu_temperature()}
```

(continues on next page)

(continued from previous page)

```
            time.sleep(10)


def main(video_resolution='1024x768', fps=30, server_port=8000, debug=False):
    web_method = WebMethod(
        index_file=os.path.join(os.path.dirname(__file__), 'index.html'),
        video_resolution=video_resolution,
        fps=fps,
        server_port=server_port,
        debug=debug,
        runtime_functions=[arduino, senser, system_monitor]
    )
    web_method.serve()


if __name__ == '__main__':
    main()
```

Listing 16: index.html

```
<html>
<head>
    <title>eduROV</title>
    <script src="./static/dynamic.js"></script>
    <script src="./static/general.js"></script>
    <script src="./static/keys.js"></script>
    <link rel="shortcut icon" href="favicon.ico" type="image/x-icon">
    <link rel="icon" href="favicon.ico" type="image/x-icon">
    <link rel="stylesheet" type="text/css" href="./static/style.css">
    <link rel="stylesheet" type="text/css" href="./static/bootstrap.css">
</head>
<body onload="set_size()">

<div class="grid-container">
    <div class="d-none d-md-block side-panel " style="display:none;">
        <div class="card bg-light cinema">
            <h5 class="card-header">Sensors</h5>
            <div class="card-body">
                <h5>ROV</h5>
                <table class="table table-hover table-sm">
                    <tbody>
                    <tr>
                        <th scope="row">Temperature</th>
                        <td id="temp"></td>
                        <td>&#8451</td>
                    </tr>
                    <tr>
                        <th scope="row">Pressure</th>
                        <td id="pressure"></td>
                        <td>kPa</td>
                    </tr>
                    <tr>
                        <th scope="row">Humidity</th>
                        <td id="humidity"></td>
                        <td>%</td>
                    </tr>
```

(continues on next page)

(continued from previous page)

```
                    <tr>
                        <th scope="row">Pitch</th>
                        <td id="pitch"></td>
                        <td>&#176</td>
                    </tr>
                    <tr>
                        <th scope="row">Roll</th>
                        <td id="roll"></td>
                        <td>&#176</td>
                    </tr>
                    <tr>
                        <th scope="row">Yaw</th>
                        <td id="yaw"></td>
                        <td>&#176</td>
                    </tr>
                    </tbody>
                </table>
                <h5>Water</h5>
                <table class="table table-sm">
                    <tbody>
                    <tr>
                        <th scope="row">Temperature</th>
                        <td id="tempWater"></td>
                        <td>&#8451</td>
                    </tr>
                    <tr>
                        <th scope="row">Pressure</th>
                        <td id="pressureWater"></td>
                        <td>kPa</td>
                    </tr>
                    </tbody>
                </table>
            </div>
        </div>
        <div class="card bg-light cinema">
            <h5 class="card-header">System</h5>
            <div class="card-body">
                <table class="table table-sm">
                    <tbody class="table-borderless">
                    <tr id="voltageTr">
                        <th scope="row">Battery</th>
                        <td id="batteryVoltage"></td>
                        <td>V</td>
                    </tr>
                    <tr id="diskTr">
                        <th scope="row">Disk space</th>
                        <td id="free_space"></td>
                        <td>MB</td>
                    </tr>
                    <tr id="cpuTr">
                        <th scope="row">CPU temp</th>
                        <td id="cpu_temp"></td>
                        <td>&#8451</td>
                    </tr>
                    </tbody>
                </table>
            </div>
```

(continues on next page)

(continued from previous page)

```
        </div>
    </div>
    <div class="center-panel">
        <img id="image" src="stream.mjpg">
        <img class="rollOverlay" id="rollOverlay" src="./static/roll.png">
    </div>
    <div class="d-none d-md-block side-panel">
        <div class="card bg-light cinema">
            <h5 class="card-header">Options</h5>
            <div class="card-body">
                <button type="button" onclick="toggle_armed()" id="armBtn"
                        class="btn btn-outline-success btn-sm btn-block"
                        title="Use this to arm the robot">
                    Arm
                </button>
                <button type="button" onclick="rotate_image()"
                        class="btn btn-outline-primary btn-sm btn-block"
                        title="Will rotate the video 180 degrees">
                    Flip video
                </button>
                <button type="button" onclick="toggle_roll()" id="rollBtn"
                        class="btn btn-outline-primary btn-sm btn-block active"
                        title="Toggle the roll indicator on/off">
                    Roll
                </button>
                <button type="button" onclick="toggle_cinema()"
                        class="btn btn-outline-primary btn-sm btn-block"
                        title="Toggle cinema mode which hides everything except video
↪">
                    Cinema
                </button>
                <button type="button" onclick="set_update_frequency()"
                        class="btn btn-outline-primary btn-sm btn-block"
                        title="Changes the sensor update frequency to desired value">
                    Sensor frequency
                </button>
                <button type="button" onclick="toggle_light()" id="lightBtn"
                        class="btn btn-outline-warning btn-sm btn-block"
                        title="Toggle the light on the ROV on/off">Light
                </button>
                <button type="button" onclick="stop_rov()"
                        class="btn btn-outline-danger btn-sm btn-block"
                        title="Stops the ROV, this page will stop working">
                    Shutdown
                </button>
            </div>
        </div>
        <div class="card bg-light cinema">
            <h5 class="card-header">Hotkeys</h5>
            <div class="card-body">
                <table class="table table-sm">
                    <tbody>
                    <tr>
                        <td><b>F11</b></td>
                        <td>Fullscreen</td>
                    </tr>
                    <tr>
```

(continues on next page)

(continued from previous page)

```
                        <td><b>L</b></td>
                        <td>Lights</td>
                    </tr>
                    <tr>
                        <td><b>C</b></td>
                        <td>Cinema</td>
                    </tr>
                    <tr>
                        <td><b>ENTER</b></td>
                        <td>Arm</td>
                    </tr>
                    </tbody>
                </table>
            </div>
        </div>

    </div>
</div>

</body>
</html>
```

```
project
├── entry.py
├── start.py
├── index.html
└── static
    ├── keys.js
    ├── general.js
    ├── dynamic.js
    ├── roll.png
    ├── bootstrap.css
    └── style.css
```

6.5 API

Tip: If you are having a hard time, you can always have a look at the examples page where the classes, methods and parameters are used in practice.

6.5.1 WebMethod

class edurov.core.**WebMethod**(*index_file*, *video_resolution='1024x768'*, *fps=30*, *server_port=8000*, *debug=False*, *runtime_functions=None*, *custom_response=None*)

Starts a video streaming from the rasparry pi and a webserver that can handle user input and other requests.

Parameters

- **index_file** (*str*) – Absolute path to the frontpage of the webpage, must be called `index.html`. For more information, see *Displaying the video feed*.
- **video_resolution** (*str, optional*) – A string representation of the wanted video resolution in the format WIDTHxHEIGHT.

- **fps** (*int, optional*) – Wanted framerate, may not be achieved depending on available resources and network.
- **server_port** (*int, optional*) – The web page will be served at this port
- **debug** (*bool, optional*) – If set True, additional information will be printed for debug purposes.
- **runtime_functions** (*callable or list, optional*) – Should be a callable function or a list of callable functions, will be started as independent processes automatically. For more information, see *Controlling motors (or anything)*.
- **custom_response** (*callable, optional*) – If set, this function will be called if default web server is not able to handle a GET request, should return a str or None. If returned value starts with `redirect=` followed by a path, the server will redirect the browser to this path. The callable must accept two parameters whereas the second one is the requested path. For more information, see *Custom responses*.

Examples

```
>>> import os
>>> from edurov import WebMethod
>>>
>>> file = os.path.join(os.path.dirname(__file__), 'index.html', )
>>> web_method = WebMethod(index_file=file)
>>> web_method.serve()
```

serve(*timeout=None*)
Will start serving the web page defined by the index_file parameter

Parameters **timeout** (*int, optional*) – if set, the web page will only be served for that many seconds before it automatically shuts down

Notes

This method will block the rest of the script.

6.5.2 ROVSyncer

class edurov.sync.**ROVSyncer**
Holds all variables for ROV related to control and sensors

Examples

```
>>> import Pyro4
>>>
>>> with Pyro4.Proxy("PYRONAME:ROVSyncer") as rov:
>>>   while rov.run:
>>>       print('The ROV is still running')
```

actuator
Dictionary holding actuator values

Getter Returns actuator values as dict

Setter Update actuator values with dict

Type dict

run
Bool describing if the ROV is still running

Getter Returns bool describing if the ROV is running

Setter Set to False if the ROV should stop

Type bool

sensor
Dictionary holding sensor values

Getter Returns sensor values as dict

Setter Update sensor values with dict

Type dict

6.5.3 KeyManager

class edurov.sync.**KeyManager**
Keeps control of all user input from keyboard.

Examples

```
>>> import Pyro4
>>>
>>> with Pyro4.Proxy("PYRONAME:KeyManager") as keys:
>>> with Pyro4.Proxy("PYRONAME:ROVSyncer") as rov:
>>>     keys.set_mode(key='l', mode='toggle')
>>>     while rov.run:
>>>         if keys.state('up arrow'):
>>>             print('You are pressing the up arrow')
>>>         if keys.state('l'):
>>>             print('light on')
>>>         else:
>>>             print('light off')
```

Note: When using the methods below a **key identifier** must be used. Either the keycode (int) or the KeyASCII or Common Name (str) from the table further down on this page can be used. Using keycode is faster.

arrow_dict
Dictionary with the state of the keys *up arrow*, *down arrow*, *left arrow* and *right arrow*

keydown(*key*, *make_exception=False*)
Call to simulate a keydown event

Parameters

- **key** (*int or str*) – key identifier as described above

- **make_exception** (*bool, optional*) – As default an exception is raised if the key is not found, this behavior can be changed be setting it to *False*

keyup (*key*, *make_exception=False*)
Call to simulate a keyup event

Parameters

- **key** (*int or str*) – key identifier as described above
- **make_exception** (*bool, optional*) – As default an exception is raised if the key is not found, this behavior can be changed be setting it to *False*

qweasd_dict
Dictionary with the state of the letters q, w, e, a, s and d

set (*key*, *state*)
Set the state of the key to True or False

Parameters

- **key** (*int or str*) – key identifier as described above
- **state** (*bool*) – *True* or *False*

set_mode (*key*, *mode*)
Set the press mode for the key to *hold* or *toggle*

Parameters

- **key** (*int or str*) – key identifier as described above
- **mode** (*str*) – *hold* or *toggle*

state (*key*)
Returns the state of *key*

Parameters **key** (*int or str*) – key identifier as described above

Returns **state** – *True* or *False*

Return type bool

Keys table

```
KeyASCII      ASCII   Common Name           Keycode
K_BACKSPACE   \b      backspace             8
K_TAB         \t      tab                   9
K_CLEAR               clear
K_RETURN      \r      return                13
K_PAUSE               pause
K_ESCAPE      ^[      escape                27
K_SPACE               space                 32
K_EXCLAIM     !       exclaim
K_QUOTEDBL    "       quotedbl
K_HASH        #       hash
K_DOLLAR      $       dollar
K_AMPERSAND   &       ampersand
K_QUOTE               quote
K_LEFTPAREN   (       left parenthesis
K_RIGHTPAREN  )       right parenthesis
K_ASTERISK    *       asterisk
```

(continues on next page)

(continued from previous page)

K_PLUS	+	plus sign	
K_COMMA	,	comma	
K_MINUS	-	minus sign	
K_PERIOD	.	period	
K_SLASH	/	forward slash	
K_0	0	0	48
K_1	1	1	49
K_2	2	2	50
K_3	3	3	51
K_4	4	4	52
K_5	5	5	53
K_6	6	6	54
K_7	7	7	55
K_8	8	8	56
K_9	9	9	57
K_COLON	:	colon	
K_SEMICOLON	;	semicolon	
K_LESS	<	less-than sign	
K_EQUALS	=	equals sign	
K_GREATER	>	greater-than sign	
K_QUESTION	?	question mark	
K_AT	@	at	
K_LEFTBRACKET	[	left bracket	
K_BACKSLASH	\	backslash	
K_RIGHTBRACKET	]	right bracket	
K_CARET	^	caret	
K_UNDERSCORE	_	underscore	
K_BACKQUOTE	`	grave	
K_a	a	a	65
K_b	b	b	66
K_c	c	c	67
K_d	d	d	68
K_e	e	e	69
K_f	f	f	70
K_g	g	g	71
K_h	h	h	72
K_i	i	i	73
K_j	j	j	74
K_k	k	k	75
K_l	l	l	76
K_m	m	m	77
K_n	n	n	78
K_o	o	o	79
K_p	p	p	80
K_q	q	q	81
K_r	r	r	82
K_s	s	s	83
K_t	t	t	84
K_u	u	u	85
K_v	v	v	86
K_w	w	w	87
K_x	x	x	88
K_y	y	y	89
K_z	z	z	90
K_DELETE		delete	
K_KP0		keypad 0	
K_KP1		keypad 1	

(continues on next page)

(continued from previous page)

```
K_KP2                keypad 2
K_KP3                keypad 3
K_KP4                keypad 4
K_KP5                keypad 5
K_KP6                keypad 6
K_KP7                keypad 7
K_KP8                keypad 8
K_KP9                keypad 9
K_KP_PERIOD   .      keypad period
K_KP_DIVIDE   /      keypad divide
K_KP_MULTIPLY *      keypad multiply
K_KP_MINUS    -      keypad minus
K_KP_PLUS     +      keypad plus
K_KP_ENTER    \r     keypad enter
K_KP_EQUALS   =      keypad equals
K_UP                 up arrow             38
K_DOWN               down arrow           40
K_RIGHT              right arrow          39
K_LEFT               left arrow           37
K_INSERT             insert               45
K_HOME               home                 36
K_END                end                  35
K_PAGEUP             page up              33
K_PAGEDOWN           page down            34
K_F1                 F1
K_F2                 F2
K_F3                 F3
K_F4                 F4
K_F5                 F5
K_F6                 F6
K_F7                 F7
K_F8                 F8
K_F9                 F9
K_F10                F10
K_F11                F11
K_F12                F12
K_F13                F13
K_F14                F14
K_F15                F15
K_NUMLOCK            numlock
K_CAPSLOCK           capslock
K_SCROLLOCK          scrollock
K_RSHIFT             right shift
K_LSHIFT             left shift
K_RCTRL              right control
K_LCTRL              left control
K_RALT               right alt
K_LALT               left alt
K_RMETA              right meta
K_LMETA              left meta
K_LSUPER             left Windows key
K_RSUPER             right Windows key
K_MODE               mode shift
K_HELP               help
K_PRINT              print screen
K_SYSREQ             sysrq
K_BREAK              break
```

(continues on next page)

(continued from previous page)

```
K_MENU                  menu
K_POWER                 power
K_EURO                  Euro
```

6.5.4 Utilities

Different utility functions practical for ROV control

edurov.utils.**cpu_temperature**()
Checks and returns the on board CPU temperature

Returns **temperature** – the temperature

Return type float

edurov.utils.**free_drive_space**(*as_string=False*)
Checks and returns the remaining free drive space

Parameters **as_string** (*bool, optional*) – set to True if you want the function to return a formatted string. 4278 -> 4.28 GB

Returns **space** – the remaining MB in float or as string if *as_string=True*

Return type float or str

edurov.utils.**receive_arduino**(*serial_connection*)
Returns a message received over *serial_connection*

Expects that the message received starts with a 6 bytes long number describing the size of the remaining data. "0x000bhello there" -> "hello there".

Parameters **serial_connection** (*object*) – the `serial.Serial` object you want to use for receiving

Returns **msg** – the message received or None

Return type str or None

edurov.utils.**receive_arduino_simple**(*serial_connection*, *min_length=1*)
Returns a message received over *serial_connection*

Same as `receive_arduino` but doesn't expect that the message starts with a hex number.

Parameters

- **serial_connection** (*object*) – the `serial.Serial` object you want to use for receiving
- **min_length** (*int, optional*) – if you only want that the function to only return the string if it is at least this long.

Returns **msg** – the message received or None

Return type str or None

edurov.utils.**send_arduino**(*msg*, *serial_connection*)
Send the *msg* over the *serial_connection*

Adds a hexadecimal number of 6 bytes to the start of the message before sending it. "hello there" -> "0x000bhello there"

Parameters

- **msg** (*str or bytes*) – the message you want to send
- **serial_connection** (*object*) – the `serial.Serial` object you want to use for sending

`edurov.utils.`**send_arduino_simple**(*msg*, *serial_connection*)
Send the *msg* over the *serial_connection*

Same as `send_arduino`, but doesn't add anything to the message before sending it.

Parameters

- **msg** (*str or bytes*) – the message you want to send
- **serial_connection** (*object*) – the `serial.Serial` object you want to use for sending

`edurov.utils.`**serial_connection**(*port='/dev/ttyACM0'*, *baudrate=115200*, *timeout=0.05*)
Establishes a serial connection

Parameters

- **port** (*str, optional*) – the serial port you want to use
- **baudrate** (*int, optional*) – the baudrate of the serial connection
- **timeout** (*float, optional*) – read timeout value

Returns **connection** – a `serial.Serial` object if successful or None if not

Return type class or None

Python Module Index

e

Index