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!

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.

features.py¶

```python
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.

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.

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.

index.html¶

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

/static/keys.js¶

```javascript
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");
    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!

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
└── static
    └── keys.js
```

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.

index.html¶

```
 1  <!DOCTYPE html>
```

```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>
```

/static/style.css¶

```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
```

Displaying sensor values

Coming soon

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.

features.py¶

```python
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()
```

index.html¶

```html
<!DOCTYPE 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>
```

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