web.py

```
"""
Sever classes used in the web method
"""

import io
import json
import logging
import os
import socketserver
import time
from http import server
from threading import Condition

import Pyro4

from edurov.utils import server_ip, detect_pi, warning

if detect_pi():
    import picamera


class StreamingOutput(object):
    """Defines output for the picamera, used by request server
"""

    def __init__(self):
        self.frame = None
        self.buffer = io.BytesIO()
        self.condition = Condition()
        self.count = 0

    def write(self, buf):
        if buf.startswith(b'\xff\xd8'):
            # New frame, copy the existing buffer's content and
notify all
            # clients it's available
            self.buffer.truncate()
            with self.condition:
                self.frame = self.buffer.getvalue()
                self.condition.notify_all()
            self.buffer.seek(0)
            self.count += 1
        return self.buffer.write(buf)


class RequestHandler(server.BaseHTTPRequestHandler):
```

web.py

```
45      """Request server, handles request from the browser"""
46      output = None
47      keys = None
48      rov = None
49      base_folder = None
50      index_file = None
51      custom_response = None
52
53      def do_GET(self):
54          if self.path == '/':
55              self.redirect('/index.html', redir_type=301)
56          elif self.path == '/stream.mjpg':
57              self.serve_stream()
58          elif self.path.startswith('/http') or self.path.
    startswith('/www'):
59              self.redirect(self.path[1:])
60          elif self.path.startswith('/keyup'):
61              self.send_response(200)
62              self.end_headers()
63              self.keys.keyup(key=int(self.path.split('=')[1]))
64          elif self.path.startswith('/keydown'):
65              self.send_response(200)
66              self.end_headers()
67              self.keys.keydown(key=int(self.path.split('=')[1]))
68          elif self.path.startswith('/sensor.json'):
69              self.serve_rov_data('sensor')
70          elif self.path.startswith('/actuator.json'):
71              self.serve_rov_data('actuator')
72          elif self.path.startswith('/stop'):
73              self.send_response(200)
74              self.end_headers()
75              self.rov.run = False
76          else:
77              path = os.path.join(self.base_folder, self.path[1:]
    )
78              if os.path.isfile(path):
79                  self.serve_path(path)
80              elif self.custom_response:
81                  response = self.custom_response(self.path)
82                  if response:
83                      if response.startswith('redirect='):
84                          new_path = response[response.find('=')
    + 1:]
85                          self.redirect(new_path)
86                      else:
87                          self.serve_content(response.encode('utf
```

```
 87 -8'))
 88                 else:
 89                     warning(message='Bad response. {}. custom
    '
 90                                     'response function
    returned nothing'
 91                                 .format(self.requestline), filter=
    'default')
 92                     self.send_404()
 93             else:
 94                 warning(message='Bad response. {}. Could not
    find {}'
 95                         .format(self.requestline, path),
    filter='default')
 96                 self.send_404()
 97
 98     def do_POST(self):
 99         self.send_404()
100
101     def serve_content(self, content, content_type='text/html'):
102         self.send_response(200)
103         self.send_header('Content-Type', content_type)
104         self.send_header('Content-Length', len(content))
105         self.end_headers()
106         self.wfile.write(content)
107
108     def serve_path(self, path):
109         if '.css' in path:
110             content_type = 'text/css'
111         elif '.js' in path:
112             content_type = 'text/javascript'
113         else:
114             content_type = 'text/html'
115         with open(path, 'rb') as f:
116             content = f.read()
117         self.serve_content(content, content_type)
118
119     def redirect(self, path, redir_type=302):
120         self.send_response(redir_type)
121         self.send_header('Location', path)
122         self.end_headers()
123
124     def send_404(self):
125         self.send_error(404)
126         self.end_headers()
127
```

web.py

```
128     def serve_rov_data(self, data_type):
129         values = ''
130         if data_type == 'sensor':
131             values = json.dumps(self.rov.sensor)
132         elif data_type == 'actuator':
133             values = json.dumps(self.rov.actuator)
134         else:
135             warning('Unable to process data_type {}'.format(
    data_type))
136         content = values.encode('utf-8')
137         self.serve_content(content, 'application/json')
138 
139     def serve_stream(self):
140         self.send_response(200)
141         self.send_header('Age', 0)
142         self.send_header('Cache-Control', 'no-cache, private')
143         self.send_header('Pragma', 'no-cache')
144         self.send_header('Content-Type',
145                          'multipart/x-mixed-replace; boundary=
    FRAME')
146         self.end_headers()
147         try:
148             while True:
149                 with self.output.condition:
150                     self.output.condition.wait()
151                     frame = self.output.frame
152                 self.wfile.write(b'--FRAME\r\n')
153                 self.send_header('Content-Type', 'image/jpeg')
154                 self.send_header('Content-Length', len(frame))
155                 self.end_headers()
156                 self.wfile.write(frame)
157                 self.wfile.write(b'\r\n')
158         except Exception as e:
159             logging.warning(
160                 'Removed streaming client %s: %s',
161                 self.client_address, str(e))
162 
163     def log_message(self, format, *args):
164         return
165 
166 
167 class WebpageServer(socketserver.ThreadingMixIn, server.
    HTTPServer):
168     """Threaded HTTP server, forwards request to the
    RequestHandlerClass"""
169     allow_reuse_address = True
```

```
170     daemon_threads = True
171
172     def __init__(self, server_address, RequestHandlerClass,
    stream_output,
173                  rov_proxy, keys_proxy, index_file=None, debug
    =False,
174                  custom_response=None):
175         self.start = time.time()
176         self.debug = debug
177         RequestHandlerClass.output = stream_output
178         RequestHandlerClass.rov = rov_proxy
179         RequestHandlerClass.keys = keys_proxy
180         RequestHandlerClass.base_folder = os.path.abspath(
181             os.path.dirname(index_file))
182         RequestHandlerClass.index_file = index_file
183         RequestHandlerClass.custom_response = custom_response
184         super(WebpageServer, self).__init__(server_address,
185
    RequestHandlerClass)
186
187     def __enter__(self):
188         return self
189
190     def __exit__(self, exc_type, exc_val, exc_tb):
191         print('Shutting down http server')
192         if self.debug:
193             finish = time.time()
194             frame_count = self.RequestHandlerClass.output.count
195             print('Sent {} images in {:.1f} seconds at {:.2f}
    fps'
196                   .format(frame_count,
197                           finish - self.start,
198                           frame_count / (finish - self.start))
    )
199
200
201 def start_http_server(video_resolution, fps, server_port,
    index_file,
202                       debug=False, custom_response=None):
203     if debug:
204         print('Using {} @ {} fps'.format(video_resolution, fps)
    )
205
206     with picamera.PiCamera(resolution=video_resolution,
207                            framerate=fps) as camera, \
208             Pyro4.Proxy("PYRONAME:ROVSyncer") as rov, \
```

web.py

```
209             Pyro4.Proxy("PYRONAME:KeyManager") as keys:
210         stream_output = StreamingOutput()
211         camera.start_recording(stream_output, format='mjpeg')
212         try:
213             with WebpageServer(server_address=('', server_port)
    ,
214                                RequestHandlerClass=
    RequestHandler,
215                                stream_output=stream_output,
216                                debug=debug,
217                                rov_proxy=rov,
218                                keys_proxy=keys,
219                                index_file=index_file,
220                                custom_response=custom_response
    ) as s:
221                 print('Visit the webpage at {}'.format(
    server_ip(server_port)))
222                 s.serve_forever()
223         finally:
224             print('closing web server')
225             camera.stop_recording()
226
```

core.py

```
import os
import subprocess
import time
from multiprocessing import Process

from edurov.sync import start_sync_classes
from edurov.utils import warning, preexec_function, detect_pi
from edurov.web import start_http_server

if detect_pi():
    import Pyro4

class WebMethod(object):
    """
    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``
    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
    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
```

core.py

```
39  returned value
40          starts with ``redirect=`` followed by a path, the
    browser wil redirect
41          the user to this path. The callable must accept two
    parameters whereas
42          the second one is the requested path
43
44      Examples
45      --------
46      >>> import os
47      >>> from edurov import WebMethod
48      >>>
49      >>> file = os.path.join(os.path.dirname(__file__), 'index.
    html', )
50      >>> web_method = WebMethod(index_file=file)
51      >>> web_method.serve()
52      """
53      def __init__(self, index_file, video_resolution='1024x768',
    fps=30,
54                   server_port=8000, debug=False,
    runtime_functions=None,
55                   custom_response=None):
56
57          self.res = video_resolution
58          self.fps = fps
59          self.server_port = server_port
60          self.debug = debug
61          self.run_funcs = self._valid_runtime_functions(
    runtime_functions)
62          self.cust_resp = self._valid_custom_response(
    custom_response)
63          self.index_file = self._valid_index_file(index_file)
64
65      def _valid_custom_response(self, custom_response):
66          if custom_response:
67              if not callable(custom_response):
68                  warning('custom_response parameter has to be a
    callable '
69                          'function, not type {}'.format(type(
    custom_response)))
70                  return None
71          return custom_response
72
73      def _valid_runtime_functions(self, runtime_functions):
74          if runtime_functions:
75              if callable(runtime_functions):
```

core.py

```
76                  runtime_functions = [runtime_functions]
77              elif isinstance(runtime_functions, list):
78                  for f in runtime_functions:
79                      if not callable(f):
80                          warning(
81                              'Parameter runtime_functions has
   to be a function '
82                              'or a list of functions, not {}'.
   format(type(f)))
83              else:
84                  warning('Parameter runtime_functions has to be
   a function '
85                          'or a list of functions, not {}'
86                          .format(type(runtime_functions)))
87          return runtime_functions
88
89      def _valid_index_file(self, file_path):
90          if not 'index.html' in file_path:
91              warning('The index files must be called "index.html
   ')
92          if os.path.isfile(file_path):
93              return os.path.abspath(file_path)
94          else:
95              warning('could not find "{}", needs absolute path'
96                      .format(file_path))
97          return None
98
99      def serve(self, timeout=None):
100         """
101         Will start serving the web page defined by the
   index_file parameter
102
103         Parameters
104         ----------
105         timeout : int, optional
106             if set, the web page will only be served for that
   many seconds
107             before it automatically shuts down
108
109         Notes
110         -----
111         This method will block the rest of the script.
112         """
113         start = time.time()
114         name_server = subprocess.Popen('pyro4-ns', shell=False,
115                                        preexec_fn=
```

core.py

```
115 preexec_function)
116         time.sleep(2)
117         pyro_classes = Process(target=start_sync_classes)
118         pyro_classes.start()
119         time.sleep(4)
120         web_server = Process(
121             target=start_http_server,
122             args=(self.res, self.fps, self.server_port, self.
    index_file,
123                   self.debug, self.cust_resp))
124         web_server.daemon = True
125         web_server.start()
126         processes = []
127         if self.run_funcs:
128             for f in self.run_funcs:
129                 p = Process(target=f)
130                 p.daemon = True
131                 p.start()
132                 processes.append(p)
133 
134         with Pyro4.Proxy("PYRONAME:ROVSyncer") as rov:
135             try:
136                 while rov.run:
137                     if timeout:
138                         if time.time() - start >= timeout:
139                             break
140             except KeyboardInterrupt:
141                 pass
142             finally:
143                 print('Shutting down')
144                 web_server.terminate()
145                 rov.run = False
146                 if self.run_funcs:
147                     for p in processes:
148                         p.join(3)
149                 pyro_classes.terminate()
150                 name_server.terminate()
151 
```

sync.py

```
"""
Synchronizing the state of ROV and controller
"""

import os
import time

import Pyro4


class Key(object):
    """Manages the state of a specific key on the keyboard"""

    def __init__(self, KeyASCII, ASCII, common, keycode, mode='hold'):
        self.state = False
        self.KeyASCII = KeyASCII
        self.ASCII = ASCII
        self.common = common
        self.mode = mode
        if keycode:
            self.keycode = int(keycode)
        else:
            self.keycode = None

    def keydown(self):
        if self.mode == 'toggle':
            self.state = not self.state
        else:
            self.state = True

    def keyup(self):
        if self.mode != 'toggle':
            self.state = False

    def __str__(self):
        return str(vars(self))


@Pyro4.expose
class KeyManager(object):
    """
    Keeps control of all user input from keyboard.

    Examples
    --------
```

sync.py

```
46      >>> import Pyro4
47      >>>
48      >>> with Pyro4.Proxy("PYRONAME:KeyManager") as keys:
49      >>> with Pyro4.Proxy("PYRONAME:ROVSyncer") as rov:
50      >>>     keys.set_mode(key='l', mode='toggle')
51      >>>     while rov.run:
52      >>>         if keys.state('up arrow'):
53      >>>             print('You are pressing the up arrow')
54      >>>         if keys.state('l'):
55      >>>             print('light on')
56      >>>         else:
57      >>>             print('light off')
58
59      Note
60      ----
61      When using the methods below a **key identifier** must be
    used. Either the
62      keycode (int) or the KeyASCII or Common Name (str) from the
    table further
63      down on this page can be used. Using keycode is faster.
64      """
65
66      def __init__(self):
67          self.keys = {}
68          cwd = os.path.dirname(os.path.abspath(__file__))
69          with open(os.path.join(cwd, 'keys.txt'), 'r') as f:
70              for line in f.readlines()[1:]:
71                  KeyASCII = line[0:14].rstrip()
72                  ASCII = line[14:22].rstrip()
73                  common = line[22:44].rstrip()
74                  keycode = line[44:].rstrip()
75                  if keycode:
76                      dict_key = int(keycode)
77                  else:
78                      dict_key = KeyASCII
79                  self.keys.update({
80                      dict_key: Key(KeyASCII, ASCII, common,
    keycode)})
81
82      def set_mode(self, key, mode):
83          """
84          Set the press mode for the key to *hold* or *toggle*
85
86          Parameters
87          ----------
88          key : int or str
```

sync.py

```
89              key identifier as described above
90          mode : str
91              *hold* or *toggle*
92          """
93          self._get(key).mode = mode
94
95      def set(self, key, state):
96          """
97          Set the state of the key to True or False
98
99          Parameters
100         ----------
101         key : int or str
102             key identifier as described above
103         state : bool
104             *True* or *False*
105         """
106         self._get(key).state = bool(state)
107
108     def _get(self, key_idx, make_exception=True):
109         """
110         Returns the Key object identified by *key_idx*
111
112         Parameters
113         ----------
114         key_idx : int or str
115             key identifier as described above
116         make_exception : bool, optional
117             As default an exception is raised if the key is not
    found, this
118             behavior can be changed be setting it to *False*
119
120         Returns
121         -------
122         key : Key object
123             list items is *namedtuple* of type *LiItem*
124         """
125         if key_idx in self.keys:
126             return self.keys[key_idx]
127         elif isinstance(key_idx, str):
128             for dict_key in self.keys:
129                 if key_idx in [self.keys[dict_key].common,
130                                self.keys[dict_key].KeyASCII]:
131                     return self.keys[dict_key]
132         if make_exception:
133             raise ValueError('Could not find key {}'.format(
```

sync.py

```
133 key_idx))
134         else:
135             return None
136 
137     def state(self, key):
138         """
139         Returns the state of *key*
140 
141         Parameters
142         ----------
143         key : int or str
144             key identifier as described above
145 
146         Returns
147         -------
148         state : bool
149             *True* or *False*
150         """
151         return self._get(key).state
152 
153     def keydown(self, key, make_exception=False):
154         """
155         Call to simulate a keydown event
156 
157         Parameters
158         ----------
159         key : int or str
160             key identifier as described above
161         make_exception : bool, optional
162             As default an exception is raised if the key is not
    found, this
163             behavior can be changed be setting it to *False*
164         """
165         btn = self._get(key, make_exception=make_exception)
166         if btn:
167             btn.keydown()
168 
169     def keyup(self, key, make_exception=False):
170         """
171         Call to simulate a keyup event
172 
173         Parameters
174         ----------
175         key : int or str
176             key identifier as described above
177         make_exception : bool, optional
```

sync.py

```
178             As default an exception is raised if the key is not
    found, this
179             behavior can be changed be setting it to *False*
180         """
181         btn = self._get(key, make_exception=make_exception)
182         if btn:
183             btn.keyup()
184 
185     @property
186     def qweasd_dict(self):
187         """
188         Dictionary with the state of the letters q, w, e, a, s
    and d
189         """
190         state = {
191             'q': self._get(81).state,
192             'w': self._get(87).state,
193             'e': self._get(69).state,
194             'a': self._get(65).state,
195             's': self._get(83).state,
196             'd': self._get(68).state,
197         }
198         return state
199 
200     @property
201     def arrow_dict(self):
202         """
203         Dictionary with the state of the keys *up arrow*, *down
    arrow*,
204         *left arrow* and *right arrow*
205         """
206         state = {
207             'up arrow': self._get(38).state,
208             'down arrow': self._get(40).state,
209             'left arrow': self._get(37).state,
210             'right arrow': self._get(39).state,
211         }
212         return state
213 
214 
215 @Pyro4.expose
216 class ROVSyncer(object):
217     """
218     Holds all variables for ROV related to control and sensors
219 
220     Examples
```

sync.py

```
221     --------
222     >>> import Pyro4
223     >>>
224     >>> with Pyro4.Proxy("PYRONAME:ROVSyncer") as rov:
225     >>>   while rov.run:
226     >>>       print('The ROV is still running')
227     """
228
229     def __init__(self):
230         self._sensor = {'time': time.time()}
231         self._actuator = {}
232         self._run = True
233
234     @property
235     def sensor(self):
236         """
237         Dictionary holding sensor values
238
239         :getter: Returns sensor values as dict
240         :setter: Update sensor values with dict
241         :type: dict
242         """
243         return self._sensor
244
245     @sensor.setter
246     def sensor(self, values):
247         self._sensor.update(values)
248         self._sensor['time'] = time.time()
249
250     @property
251     def actuator(self):
252         """
253         Dictionary holding actuator values
254
255         :getter: Returns actuator values as dict
256         :setter: Update actuator values with dict
257         :type: dict
258         """
259         return self._actuator
260
261     @actuator.setter
262     def actuator(self, values):
263         self._actuator.update(values)
264         self._actuator['time'] = time.time()
265
266     @property
```

sync.py

```
267     def run(self):
268         """
269         Bool describing if the ROV is still running
270 
271         :getter: Returns bool describing if the ROV is running
272         :setter: Set to False if the ROV should stop
273         :type: bool
274         """
275         return self._run
276 
277     @run.setter
278     def run(self, bool_):
279         self._run = bool_
280 
281 
282 def start_sync_classes():
283     """Registers pyro classes in name server and starts request
    loop"""
284     rov = ROVSyncer()
285     keys = KeyManager()
286     with Pyro4.Daemon() as daemon:
287         rov_uri = daemon.register(rov)
288         keys_uri = daemon.register(keys)
289         with Pyro4.locateNS() as ns:
290             ns.register("ROVSyncer", rov_uri)
291             ns.register("KeyManager", keys_uri)
292         daemon.requestLoop()
293 
294 
295 if __name__ == "__main__":
296     start_sync_classes()
297 
```

keys.txt

	KeyASCII	ASCII	Common Name	Keycode
1	K_BACKSPACE	\b	backspace	8
2	K_TAB	\t	tab	9
3	K_CLEAR		clear	
4	K_RETURN	\r	return	13
5	K_PAUSE		pause	
6	K_ESCAPE	^[	escape	27
7	K_SPACE		space	32
8	K_EXCLAIM	!	exclaim	
9	K_QUOTEDBL	"	quotedbl	
10	K_HASH	#	hash	
11	K_DOLLAR	$	dollar	
12	K_AMPERSAND	&	ampersand	
13	K_QUOTE		quote	
14	K_LEFTPAREN	(	left parenthesis	
15	K_RIGHTPAREN	)	right parenthesis	
16	K_ASTERISK	*	asterisk	
17	K_PLUS	+	plus sign	
18	K_COMMA	,	comma	
19	K_MINUS	-	minus sign	
20	K_PERIOD	.	period	
21	K_SLASH	/	forward slash	
22	K_0	0	0	48
23	K_1	1	1	49
24	K_2	2	2	50
25	K_3	3	3	51
26	K_4	4	4	52
27	K_5	5	5	53
28	K_6	6	6	54
29	K_7	7	7	55
30	K_8	8	8	56
31	K_9	9	9	57
32	K_COLON	:	colon	
33	K_SEMICOLON	;	semicolon	
34	K_LESS	<	less-than sign	
35	K_EQUALS	=	equals sign	
36	K_GREATER	>	greater-than sign	
37	K_QUESTION	?	question mark	
38	K_AT	@	at	
39	K_LEFTBRACKET	[	left bracket	
40	K_BACKSLASH	\	backslash	
41	K_RIGHTBRACKET	]	right bracket	
42	K_CARET	^	caret	
43	K_UNDERSCORE	_	underscore	
44	K_BACKQUOTE	`	grave	
45	K_a	a	a	65

```
 47 K_b            b       b                     66
 48 K_c            c       c                     67
 49 K_d            d       d                     68
 50 K_e            e       e                     69
 51 K_f            f       f                     70
 52 K_g            g       g                     71
 53 K_h            h       h                     72
 54 K_i            i       i                     73
 55 K_j            j       j                     74
 56 K_k            k       k                     75
 57 K_l            l       l                     76
 58 K_m            m       m                     77
 59 K_n            n       n                     78
 60 K_o            o       o                     79
 61 K_p            p       p                     80
 62 K_q            q       q                     81
 63 K_r            r       r                     82
 64 K_s            s       s                     83
 65 K_t            t       t                     84
 66 K_u            u       u                     85
 67 K_v            v       v                     86
 68 K_w            w       w                     87
 69 K_x            x       x                     88
 70 K_y            y       y                     89
 71 K_z            z       z                     90
 72 K_DELETE               delete
 73 K_KP0                  keypad 0
 74 K_KP1                  keypad 1
 75 K_KP2                  keypad 2
 76 K_KP3                  keypad 3
 77 K_KP4                  keypad 4
 78 K_KP5                  keypad 5
 79 K_KP6                  keypad 6
 80 K_KP7                  keypad 7
 81 K_KP8                  keypad 8
 82 K_KP9                  keypad 9
 83 K_KP_PERIOD    .       keypad period
 84 K_KP_DIVIDE    /       keypad divide
 85 K_KP_MULTIPLY  *       keypad multiply
 86 K_KP_MINUS     -       keypad minus
 87 K_KP_PLUS      +       keypad plus
 88 K_KP_ENTER     \r      keypad enter
 89 K_KP_EQUALS    =       keypad equals
 90 K_UP                   up arrow              38
 91 K_DOWN                 down arrow            40
 92 K_RIGHT                right arrow           39
```

keys.txt

```
 93 K_LEFT                left arrow             37
 94 K_INSERT              insert                 45
 95 K_HOME                home                   36
 96 K_END                 end                    35
 97 K_PAGEUP              page up                33
 98 K_PAGEDOWN            page down              34
 99 K_F1                  F1
100 K_F2                  F2
101 K_F3                  F3
102 K_F4                  F4
103 K_F5                  F5
104 K_F6                  F6
105 K_F7                  F7
106 K_F8                  F8
107 K_F9                  F9
108 K_F10                 F10
109 K_F11                 F11
110 K_F12                 F12
111 K_F13                 F13
112 K_F14                 F14
113 K_F15                 F15
114 K_NUMLOCK             numlock
115 K_CAPSLOCK            capslock
116 K_SCROLLOCK           scrollock
117 K_RSHIFT              right shift
118 K_LSHIFT              left shift
119 K_RCTRL               right control
120 K_LCTRL               left control
121 K_RALT                right alt
122 K_LALT                left alt
123 K_RMETA               right meta
124 K_LMETA               left meta
125 K_LSUPER              left Windows key
126 K_RSUPER              right Windows key
127 K_MODE                mode shift
128 K_HELP                help
129 K_PRINT               print screen
130 K_SYSREQ              sysrq
131 K_BREAK               break
132 K_MENU                menu
133 K_POWER               power
134 K_EURO                Euro
```

utils.py

```
"""
Different utility functions practical for ROV control
"""

import ctypes
import os
import platform
import signal
import socket
import struct
import subprocess
import warnings


def detect_pi():
    return platform.linux_distribution()[0].lower() == 'debian'


if detect_pi():
    import serial
    import fcntl


def is_int(number):
    if isinstance(number, int):
        return True
    else:
        try:
            if isinstance(int(number), int):
                return True
        except ValueError:
            pass
    return False


def resolution_to_tuple(resolution):
    if 'x' not in resolution:
        raise ValueError('Resolution must be in format WIDTHxHEIGHT')
    screen_size = tuple([int(val) for val in resolution.split('x')])
    if len(screen_size) is not 2:
        raise ValueError('Error in parsing resolution, len is not 2')
    return screen_size

```

utils.py

```
44
45 def preexec_function():
46     signal.signal(signal.SIGINT, signal.SIG_IGN)
47
48
49 def valid_resolution(resolution):
50     if 'x' in resolution:
51         w, h = resolution.split('x')
52         if is_int(w) and is_int(h):
53             return resolution
54     warning('Resolution must be WIDTHxHEIGHT')
55
56
57 def server_ip(port):
58     online_ips = []
59     for interface in [b'eth0', b'wlan0']:
60         sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
61         sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,
   1)
62         try:
63             ip = socket.inet_ntoa(fcntl.ioctl(
64                 sock.fileno(),
65                 0x8915,
66                 struct.pack('256s', interface[:15])
67             )[20:24])
68             online_ips.append(ip)
69         except OSError:
70             pass
71         sock.close()
72     return ' or '.join(['{}:{}'.format(ip, port) for ip in
   online_ips])
73
74
75 def check_requirements():
76     if detect_pi():
77         camera = subprocess.check_output(['vcgencmd',
78                                           'get_camera']).
   decode().rstrip()
79         if '0' in camera:
80             warning('Camera not enabled or connected properly')
81             return False
82         else:
83             return True
84     else:
85         warning('eduROV only works on a raspberry pi')
86         return False
```

utils.py

```
 87
 88
 89 def send_arduino(msg, serial_connection):
 90     """
 91     Send the *msg* over the *serial_connection*
 92
 93     Adds a hexadecimal number of 6 bytes to the start of the
    message before
 94     sending it. "hello there" -> "0x000bhello there"
 95
 96     Parameters
 97     ----------
 98     msg : str or bytes
 99         the message you want to send
100     serial_connection : object
101         the :code:`serial.Serial` object you want to use for
    sending
102     """
103     if not isinstance(msg, bytes):
104         msg = str(msg).encode()
105     length = "{0:#0{1}x}".format(len(msg), 6).encode()
106     data = length + msg
107     serial_connection.write(data)
108
109
110 def receive_arduino(serial_connection):
111     """
112     Returns a message received over *serial_connection*
113
114     Expects that the message received starts with a 6 bytes
    long number
115     describing the size of the remaining data. "0x000bhello
    there" -> "hello
116     there".
117
118     Parameters
119     ----------
120     serial_connection : object
121         the :code:`serial.Serial` object you want to use for
    receiving
122
123     Returns
124     -------
125     msg : str or None
126         the message received or None
127     """
```

utils.py

```
128     if serial_connection.inWaiting():
129         msg = serial_connection.readline().decode().rstrip()
130         if len(msg) >= 6:
131             try:
132                 length = int(msg[:6], 0)
133                 data = msg[6:]
134                 if length == len(data):
135                     return data
136                 else:
137                     warning('Received incomplete serial string
    : {}'
138                             .format(data), 'default')
139             except ValueError:
140                 pass
141     return None
142 
143 
144 def send_arduino_simple(msg, serial_connection):
145     """
146     Send the *msg* over the *serial_connection*
147 
148     Same as :code:`send_arduino`, but doesn't add anything to
    the message
149     before sending it.
150 
151     Parameters
152     ----------
153     msg : str or bytes
154         the message you want to send
155     serial_connection : object
156         the :code:`serial.Serial` object you want to use for
    sending
157     """
158     if not isinstance(msg, bytes):
159         msg = str(msg).encode()
160     serial_connection.write(msg)
161 
162 
163 def receive_arduino_simple(serial_connection, min_length=1):
164     """
165     Returns a message received over *serial_connection*
166 
167     Same as :code:`receive_arduino` but doesn't expect that the
     message starts
168     with a hex number.
169 
```

utils.py

```
170     Parameters
171     ----------
172     serial_connection : object
173         the :code:`serial.Serial` object you want to use for
    receiving
174     min_length : int, optional
175         if you only want that the function to only return the
    string if it is
176         at least this long.
177 
178     Returns
179     -------
180     msg : str or None
181         the message received or None
182     """
183     if serial_connection.inWaiting():
184         msg = serial_connection.readline().decode().rstrip()
185         if len(msg) >= min_length:
186             return msg
187         else:
188             return None
189 
190 
191 def serial_connection(port='/dev/ttyACM0', baudrate=115200,
    timeout=0.05):
192     """
193     Establishes a serial connection
194 
195     Parameters
196     ----------
197     port : str, optional
198         the serial port you want to use
199     baudrate : int, optional
200         the baudrate of the serial connection
201     timeout : float, optional
202         read timeout value
203 
204     Returns
205     -------
206     connection : class or None
207         a :code:`serial.Serial` object if successful or None if
     not
208     """
209     try:
210         ser = serial.Serial(port, baudrate, timeout=timeout)
211         ser.close()
```

```
212         ser.open()
213         return ser
214     except FileNotFoundError:
215         pass
216     except serial.serialutil.SerialException:
217         pass
218     except ValueError:
219         pass
220     warning(message="""Could not establish serial connection at
    {}\n
221     Try running 'ls /dev/*tty*' to find correct port"""
222             .format(port), filter='default')
223     return None
224
225
226 def warning(message, filter='error', category=UserWarning):
227     warnings.simplefilter(filter, category)
228     warnings.formatwarning = warning_format
229     warnings.warn(message)
230
231
232 def warning_format(message, category, filename, lineno,
233                    file=None, line=None):
234     return 'WARNING:\n  {}: {}\n  File: {}:{}\n'.format(
235         category.__name__, message, filename, lineno)
236
237
238 def free_drive_space(as_string=False):
239     """
240     Checks and returns the remaining free drive space
241
242     Parameters
243     ----------
244     as_string : bool, optional
245         set to True if you want the function to return a
    formatted string.
246         4278 -> 4.28 GB
247
248     Returns
249     -------
250     space : float or str
251         the remaining MB in float or as string if *as_string=
    True*
252     """
253     if platform.system() == 'Windows':
254         free_bytes = ctypes.c_ulonglong(0)
```

utils.py

```
255         ctypes.windll.kernel32.GetDiskFreeSpaceExW(ctypes.
    c_wchar_p('/'),
256                                                    None, None
    ,
257                                                    ctypes.
    pointer(free_bytes))
258         mb = free_bytes.value / 1024 / 1024
259     else:
260         st = os.statvfs('/')
261         mb = st.f_bavail * st.f_frsize / 1024 / 1024
262 
263     if as_string:
264         if mb >= 1000:
265             return '{:.2f} GB'.format(mb / 1000)
266         else:
267             return '{:.0f} MB'.format(mb)
268     else:
269         return mb
270 
271 
272 def cpu_temperature():
273     """
274     Checks and returns the on board CPU temperature
275 
276     Returns
277     -------
278     temperature : float
279         the temperature
280     """
281     cmds = ['/opt/vc/bin/vcgencmd', 'measure_temp']
282     response = subprocess.check_output(cmds).decode()
283     return float(response.split('=')[1].split("'")[0].rstrip())
284 
```

__init__.py

```
1 from .core import WebMethod
2
```