Coverage for **/home/pi/banyan-bot-blue/banyan_assets/robot_control.py** : 86%

103 statements | 93 run | 10 missing | 0 excluded | 8 partial

```
"""

 Copyright (c) 2016-2019 Alan Yorinks All right reserved.

 Python Banyan is free software; you can redistribute it and/or
 modify it under the terms of the GNU AFFERO GENERAL PUBLIC LICENSE
 Version 3 as published by the Free Software Foundation; either
 or (at your option) any later version.
 This library is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 General Public License for more details.

 You should have received a copy of the GNU AFFERO GENERAL PUBLIC LICENSE
 along with this library; if not, write to the Free Software
 Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA

THIS IS A PLACE HOLDER FOR THE ACTUAL CODE TO FOLLOW


"""
from __future__ import unicode_literals

import argparse
import signal
import time
import sys
from python_banyan.banyan_base import BanyanBase


# noinspection PyMethodMayBeStatic
class RobotControl(BanyanBase):
    """
    This class accepts robot commands and translates them
    to motor control messages.

    It also subscribes to receive robot sensor updates to
    autonomously change course if a bumper is hit
    """

    def __init__(self, back_plane_ip_address=None, subscriber_port='43125',
                 publisher_port='43124', process_name=None, loop_time=0.01,
                 publish_to_ui_topic=None,
                 publish_to_hardware_topic=None, subscribe_from_ui_topic=None,
                 subscribe_from_hardware_topic=None, additional_subscriber_list=None,
                 forward_speed=80, turn_speed=60, speed_scale_factor=100):
        """

        :param back_plane_ip_address: ip address for backplane
        :param subscriber_port:
        :param publisher_port:
        :param process_name:
        :param loop_time:
        :param publish_to_ui_topic: topic when publishing messages towards the UI
        :param publish_to_hardware_topic: topic when publishing messages towards the hardware
        :param subscribe_from_ui_topic: topic to receive info from UI
        :param subscribe_from_hardware_topic: topic to receive info from hardware
        :param additional_subscriber_list: additional subscription topics
        :param forward_speed: motor speed to go forward or reverse
        :param turn_speed: turning motor speed
        :param speed_scale_factor: speed scaling

        """
        # save input parameters as instance variables
        self.back_plane_ip_address = back_plane_ip_address
        self.subscriber_port = subscriber_port
        self.publisher_port = publisher_port
        self.process_name = process_name
        self.loop_time = loop_time
        self.additional_subscriber_list = additional_subscriber_list
        self.forward_speed = forward_speed
        self.turn_speed = turn_speed
        self.speed_scaling_factor = speed_scale_factor

        # initialize the parent class
        super(RobotControl, self).__init__(back_plane_ip_address=self.back_plane_ip_address,
                                           process_name=self.process_name,
                                           subscriber_port=self.subscriber_port,
                                           publisher_port=self.publisher_port,
                                           loop_time=self.loop_time)

        # set subscription topics
        self.subscribe_from_ui_topic = subscribe_from_ui_topic
        self.set_subscriber_topic(self.subscribe_from_ui_topic)

        self.subscribe_from_hardware_topic = subscribe_from_hardware_topic
        self.set_subscriber_topic(self.subscribe_from_hardware_topic)

        # if caller specified a list of additional subscription topics, subscribe to those
        if self.additional_subscriber_list is not None:                                        90 ↛ 9
            for topic in self.additional_subscriber_list:
                self.set_subscriber_topic(topic)

        # save the publishing topics
        self.publish_to_hardware_topic = publish_to_hardware_topic
        self.publish_to_ui_topic = publish_to_ui_topic

        # Avoidance control active or not.
        # This will prevent the user from moving the robot if
        # the avoidance maneuver is in progress.
        self.avoidance_active = False

        # Motor control payloads
        # Here we build a look-up table that maps commands received from the
        # the GUI to motor commands.
        # The 'X' value is internal and represents any of the stop motor commands
        # (that is a lower case command from the UI)
        # noinspection PyPep8,PyPep8,PyPep8,PyPep8,PyPep8,PyPep8,PyPep8
        self.motor_control_payloads = [
            # stop
            {'X':
                [
                    {'command': 'dc_motor_forward', 'motor': 1, 'speed': 0.0},
                    {'command': 'dc_motor_forward', 'motor': 2, 'speed': 0.0}
                ]
            },

            # forward
            {'U':
                [
                    {'command': 'dc_motor_forward', 'motor': 1,
                     'speed': self.forward_speed / self.speed_scaling_factor},

                    {'command': 'dc_motor_forward', 'motor': 2, 'speed': self.forward_speed / self.speed_scaling_factor
                ]
            },

            # reverse
            {'D':
                [
                    {'command': 'dc_motor_reverse', 'motor': 1, 'speed': -(self.forward_speed /
                                                                           self.speed_scaling_factor)},
                    {'command': 'dc_motor_reverse', 'motor': 2, 'speed': -(self.forward_speed /
                                                                           self.speed_scaling_factor)}

                ]
            },

            # left
            {'R':
                [
                    {'command': 'dc_motor_forward', 'motor': 1, 'speed': self.forward_speed /
                                                                         self.speed_scaling_factor},
                    {'command': 'dc_motor_forward', 'motor': 2, 'speed': self.turn_speed / self.speed_scaling_factor}
                ]
            },

            # right
            {'L':
                [
                    {'command': 'dc_motor_forward', 'motor': 1, 'speed': self.turn_speed /
                                                                         self.speed_scaling_factor},
                    {'command': 'dc_motor_forward', 'motor': 2, 'speed': self.forward_speed / self.speed_scaling_factor
                ]
            },

            # spin right
            {'S':
                [
                    {'command': 'dc_motor_forward', 'motor': 1,
                     'speed': self.forward_speed / self.speed_scaling_factor},
                    {'command': 'dc_motor_reverse', 'motor': 2, 'speed': -(self.forward_speed /
                                                                           self.speed_scaling_factor)}
                ]
            },

            # spin left
            {'W':
                [
                    {'command': 'dc_motor_reverse', 'motor': 1,
                     'speed': -(self.forward_speed / self.speed_scaling_factor)},
                    {'command': 'dc_motor_forward', 'motor': 2, 'speed': self.forward_speed / self.speed_scaling_factor
                ]
            }

        ]
        # set bumper switch inputs
        payload = {'command': 'set_mode_digital_input_pullup', 'pin': 0}
        self.publish_payload(payload, self.publish_to_hardware_topic)
        payload = {'command': 'set_mode_digital_input_pullup', 'pin': 1}
        self.publish_payload(payload, self.publish_to_hardware_topic)

        # start up the Banyan receive_loop
        self.receive_loop()

    def incoming_message_processing(self, topic, payload):
        """
        Incoming message processing routed from the receive_loop

        :param topic: Message Topic string.

        :param payload: Message Data.
```

```
193         """
194         # Handle messages from the UI
195         if topic == self.subscribe_from_ui_topic:
196             # throw away commands if in avoidance mode
197             if not self.avoidance_active:                                   197 ↛ exi
198                 self.motion_control(payload)
199         # Handle messages from the hardware
200         elif topic == self.subscribe_from_hardware_topic:                  200 ↛ 20
201             self.avoidance_control(payload)
202         else:
203             raise RuntimeError('Unknown topic received: ', topic)
204 
205     def motion_control(self, payload):
206         """
207         Motor control
208         :param payload:
209         :return:
210         """
211         # Get the key into the motor command table.
212         key = payload['command']
213         motor_commands = None
214 
215         # If the key is a lower case letter, than that means to stop.
216         # Assign a virtual key of 'X' for the lookup.
217         if key.islower():
218             key = 'X'
219 
220         # Find the messages for the key command and publish
221         # the commands to the motor controller.
222         for record in range(0, len(self.motor_control_payloads)):
223             if key in self.motor_control_payloads[record]:
224                 motor_commands = self.motor_control_payloads[record]
225                 payload = motor_commands[key][0]
226                 self.publish_payload(payload, self.publish_to_hardware_topic)
227                 payload2 = motor_commands[key][1]
228                 self.publish_payload(payload2, self.publish_to_hardware_topic)
229 
230         # In case the command is not found in the table
231         if motor_commands is None:                                          231 ↛ 23
232             raise RuntimeError('Motor Command Not Found: ', key)
```

```

    def avoidance_control(self, payload):
        """
        Initiate avoidance procedure
        :param payload:
        """

        # The value returned is 0 when the bumper switch is activated
        if not payload['value']:
            # set the avoidance active flag
            self.avoidance_active = True
            # Publish the motor commands for avoidance maneuver
            payload1 = {'command': 'dc_motor_reverse', 'motor': 1, 'speed': -(self.forward_speed /
                                                                             self.speed_scaling_factor)}
            payload2 = {'command': 'dc_motor_reverse', 'motor': 2, 'speed': -(self.forward_speed /
                                                                             self.speed_scaling_factor)}
            self.publish_payload(payload1, self.publish_to_hardware_topic)
            self.publish_payload(payload2, self.publish_to_hardware_topic)
            # let motors run for one second
            time.sleep(1)

            # turn motors off
            payload1 = {'command': 'dc_motor_reverse', 'motor': 1, 'speed': 0}
            payload2 = {'command': 'dc_motor_reverse', 'motor': 2, 'speed': 0}
            self.publish_payload(payload1, self.publish_to_hardware_topic)
            self.publish_payload(payload2, self.publish_to_hardware_topic)

            # clear the avoidance active flag
            self.avoidance_active = False


def robot_control():
    """
    Launcher for robot control
    """
    parser = argparse.ArgumentParser()

    parser.add_argument("-b", dest="back_plane_ip_address", default="None",
                        help="None or IP address used by Back Plane")
    parser.add_argument("-d", dest="publish_to_hardware_topic", default="to_hardware",
                        help="Publishing topic for hardware commands")
    parser.add_argument("-f", dest="forward_speed", default="80",
                        help="Forward and Reverse Motor Speed")
    parser.add_argument("-g", dest="turn_speed", default="60",
                        help="Turning Motor Speed")
    parser.add_argument("-k", dest="speed_scale_factor", default="100",
                        help="Speed scaling factor")
    parser.add_argument("-l", dest="additional_subscriber_list",
                        default=["report"], nargs="+",
                        help="Banyan topics space delimited: topic1 topic2 "
                             "topic3")
    parser.add_argument("-n", dest="process_name", default="Robot Control",
                        help="Set process name in banner")
    parser.add_argument("-p", dest="publisher_port", default='43124',
                        help="Publisher IP port")
    parser.add_argument("-r", dest="publish_to_ui_topic", default="to_ui",
                        help="Publishing topic for report messages")
    parser.add_argument("-s", dest="subscriber_port", default='43125',
                        help="Subscriber IP port")
    parser.add_argument("-t", dest="loop_time", default=".01",
                        help="Event Loop Timer in seconds")
    parser.add_argument("-u", dest="subscribe_from_ui_topic", default="from_bt_gateway",
                        help="Topic From User Interface")
    parser.add_argument("-v", dest="subscribe_from_hardware_topic", default="report_from_hardware",
                        help="Topic From Hardware")

    args = parser.parse_args()
    if args.back_plane_ip_address == 'None':                                        300 ↛ 30
        args.back_plane_ip_address = None

    if args.process_name == 'None':                                                 303 ↛ 30
        args.process_name = None

    if args.additional_subscriber_list == ['None']:                                 306 ↛ 30
        args.additional_subscriber_list = None

    kw_options = {
        'back_plane_ip_address': args.back_plane_ip_address,
        'publisher_port': args.publisher_port,
        'subscriber_port': args.subscriber_port,
        'process_name': args.process_name,
        'loop_time': float(args.loop_time),
        'additional_subscriber_list': args.additional_subscriber_list,
        'publish_to_hardware_topic': args.publish_to_hardware_topic,
        'publish_to_ui_topic': args.publish_to_ui_topic,
        'subscribe_from_ui_topic': args.subscribe_from_ui_topic,
        'subscribe_from_hardware_topic': args.subscribe_from_hardware_topic,
        'forward_speed': int(args.forward_speed),
        'turn_speed': int(args.turn_speed),
        'speed_scale_factor': float(args.speed_scale_factor)
    }

    try:
        app = RobotControl(**kw_options)
    except KeyboardInterrupt:
        sys.exit()

    # noinspection PyUnusedLocal
    def signal_handler(sig, frame):
        print("Control-C detected. See you soon.")
        app.clean_up()
        sys.exit(0)

    # listen for SIGINT
    signal.signal(signal.SIGINT, signal_handler)
    signal.signal(signal.SIGTERM, signal_handler)


if __name__ == '__main__':                                                              341 ↛ exi
    robot_control()
```

« index   coverage.py v4.5.3, created at 2019-08-19 20:18