МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Системы реального времени на основе Linux»
Тема: Простая программа-писатель

Студент гр. 2303 _____ Канушин М.С.
Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2018

Цель работы.

Разработать программу, моделирующую робота и лабиринт. Робот должен управляться пользователем через технологию publisher-subscriber. Робот перемещается только по свободным клеткам, перемещение через препятствия(стены) не допускается.

Основные теоретические положения.

У объекта класса NodeHandle

```
ros::NodeHandle <node_handle_name>;
```

есть метод, реализующий механизм отправки сообщений в топик с именем <topic_name>.

Это делается при помощи команды

```
ros::Publisher <publisher_name> = <node_handle_name>.advertise<msg_type>("<top
```

где <topic_name> это имя топика, через который будут общаться publisher и subscriber; а <size> - размер буфера сообщений (а треугольные скобочки после advertize - это конкретизация шаблонной функции).

В данном случае имя топика можно получить, узнав, на какой топик подписан turtlesim_node. Тип сообщения можно узнать, выведя информацию о топике.

Информацию о том, из каких полей состоит сообщение можно узнать командой

```
rosmsg show <msg_type>
```

В тексте программы необходимо создать объект класса <msg_type> и наполнить его содержимым. Информативными являются поля msg.linear.x и msg.angular.z. Остальные поля сообщения не учитываются при обработке.

После того, как сообщение сформировано, его можно отправить в топик командой

```
<publisher_name>.publish (msg)
```

В разделе может быть приведено описание исследуемых физических явлений (с иллюстрациями), основные теоретические положения (в том числе – математический аппарат, описывающий исследуемые явления), схемы измерений, сведения об используемом при проведении работы лабораторном оборудовании.

Разработка программы.

Для отправки сообщений роботу будет использоваться единственный топик - *controller*, отвечающий за команды перемещения.

Узлы.

- controller - узел, считывающий команды пользователя и отправляющий их в топик *controller*
- listener - узел, отвечающий за прием сообщений из топика *controller* и производящий перемещение робота в случае, если это возможно.

Классы.

- KeyPublisher - класс, реализующий функциональность узла *controller*
- LevelMap - класс, хранящий карту лабиринта и местоположение робота на ней.
- Robot - класс, реализующий функциональность робота. Робот определят свое местоположение на карте, считывает из топика команды и выполняет их в случае, если это возможно.

Выводы.

В ходе выполнения данной лабораторной работы были изучена структура проекта в среде ROS и разработана программа, использующая технологию publisher-subscriber.

Код программы.

```
#!/usr/bin/env python

import rospy
from pynput.keyboard import Key, Listener, KeyCode
from std_msgs.msg import String


class KeyPublisher:
        KEY_Q = KeyCode(char='q')
        KEY_W = KeyCode(char='w')
        KEY_A = KeyCode(char='a')
        KEY_S = KeyCode(char='s')
        KEY_D = KeyCode(char='d')
        KEY_P = KeyCode(char='p')
        keys = (KEY_Q, KEY_W, KEY_A, KEY_S, KEY_D, KEY_P, Key.esc)

        def __init__(self, topic):
                self.topic = topic
                self.pub = rospy.Publisher(topic, String, queue_size=10)
                rospy.init_node('key_talker', anonymous=True)
```

```
        def start(self):
                def __on_press(key):
                        if key in self.keys:
                                rospy.loginfo('got key %s', key)
                                if key == self.KEY_W:
                                        self.pub.publish('up')
                                if key == self.KEY_S:
                                        self.pub.publish('down')
                                if key == self.KEY_A:
                                        self.pub.publish('left')
                                if key == self.KEY_D:
                                        self.pub.publish('right')
                                if key == self.KEY_P:
                                        self.pub.publish('print')

                def __on_release(key):
                        if key in (Key.esc, self.KEY_Q):
                                return False

                rospy.loginfo('Controller with topic `%s` launched',
                          self.topic)
                rospy.loginfo('w, a, s, d to move')
                rospy.loginfo('esc, q to exit')
                rospy.loginfo('p to print level map')

                with Listener(on_press=__on_press,
                          on_release=__on_release) as listener:
                        listener.join()


if __name__ == '__main__':
        try:
                talker = KeyPublisher('controller')
                talker.start()
        except rospy.ROSInterruptException:
                pass
```

```
#!/usr/bin/env python

import rospy
import rospkg
from std_msgs.msg import String


class LevelMap:
        def __init__(self, map_file):
                self.map = []
                with open(map_file) as f:
                        for line in f:
                                tmp = []
                                for char in line:
                                        tmp.append(char)
                                self.map.append(tmp)
```

```
        def free(self, x, y):
                return self.map[x][y] == '0'

        def print_map(self, x, y):
                self.map[x][y] = '*'
                res = '\nLevel map:\n'
                for line in self.map:
                        res += ''.join(line)
                res += '* - robot\n'
                res += '1 - wall\n'
                res += '0 - empty space\n'
                rospy.loginfo(res)
                self.map[x][y] = '0'


class Robot:
        def __init__(self, name, x=0, y=0):
                self.name = name
                self.x = x
                self.y = y
                self.map = None

        def load_map(self, local_map):
                self.map = local_map

        def move(self, dx, dy):
                new_x = self.x + dx
                new_y = self.y + dy
                if self.map.free(new_x, new_y):
                        rospy.loginfo('I am %s, I am moving to (%d,%d)', self.name,
                                new_x, new_y)
                        self.x = new_x
                        self.y = new_y
                else:
                        rospy.loginfo('I am %s, cannot move there', self.name)

        def show_location(self):
                self.map.print_map(self.x, self.y)

        def subscribe(self, topic):
                def callback(data):
                        rospy.loginfo('I am %s, my id is %s, I received %s',
                                self.name,
                                rospy.get_caller_id(),
                                data.data)
                        if data.data == 'up':
                                self.move(-1, 0)

                        if data.data == 'left':
                                self.move(0, -1)
```

```
                        if data.data == 'right':
                                self.move(0, 1)

                        if data.data == 'down':
                                self.move(1, 0)

                        if data.data == 'print':
                                self.show_location()

                rospy.init_node('listener', anonymous=True)
                rospy.Subscriber(topic, String, callback)
                rospy.spin()


if __name__ == '__main__':
        rospack = rospkg.RosPack()
        res_path = rospack.get_path('lab_1') + '/res/'
        level = LevelMap('{}map.txt'.format(res_path))

        andy = Robot('Andy', 4, 3)
        andy.load_map(level)
        andy.subscribe('controller')
```