

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа компьютерных технологий и информационных систем

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

Дисциплина: Программное обеспечение встраиваемых систем

Тема: Разработка описания робота

Выполнил
студент гр. 5140901/21501

<подпись>

А.М. Кобыжев

Преподаватель

<подпись>

Г.С. Васильянов

«___» _____ 2023 г.

Санкт-Петербург

2023

СОДЕРЖАНИЕ

1.	Цели работы	3
2.	Задание.....	3
3.	Ход работы	3
3.1.	Описание робота	3
3.2.	Описание узла управления роботом с клавиатуры	4
3.2.1.	Запуск робота	7
3.3.	Описание узла управления роботом в автоматическом режиме (ик- сенсоры)	8
3.3.1.	Запуск робота	11
4.	Выводы	13

1. ЦЕЛИ РАБОТЫ

Разработка узла управления роботом с клавиатуры, а также узла, управляющего движением робота в автоматическом режиме при помощи датчиков дистанции.

2. ЗАДАНИЕ

- Разработать узел управления роботом с клавиатуры.
- Разработать узел, управляющий движением робота в автоматическом режиме.
 - Расположение ноды для автоматического режима – *myrobot/myrobot_teleop/nodes/auto_teleop*;
 - запуск – *teleop_auto.launch*
- Робот должен избегать препятствий используя датчики дистанции
- Используемая карта – *testwalls*; запуск – *gazebo_testwalls.launch*

3. ХОД РАБОТЫ

Репозиторий с исходным кодом для данной лабораторной работы можно посмотреть по ссылке:

https://github.com/alexneveskiy/urdf_labs/tree/lab3

3.1. Описание робота

В предыдущей лабораторной работе обнаружилось, что тело робота при увеличении или уменьшении линейной скорости качалось вперёд или назад, что не позволило бы нормально сделать данную лабораторную работу при автоматическом управлении при помощи ик-сенсоров. Поэтому было принято решение поднять колёсную базу робота до такой высоты, чтобы между нижней точкой колеса и тела робота оставался 1 мм, при этом колёса должны касаться пола. Таким образом, робот всё равно будет качаться, однако это уже не будет никак мешать работе ик-сенсоров, потому что тело робота всегда будет практически параллельно полу с минимальными отклонениями. Также все три ик-сенсора были подняты на верхнюю часть тела робота, чтобы сенсоры ложно не срабатывали при измерении дистанции при минимальном наклоне,

так как при старом положении они бы находились практически на уровне пола. Обновлённая модель робота представлена на рис. 3.1.

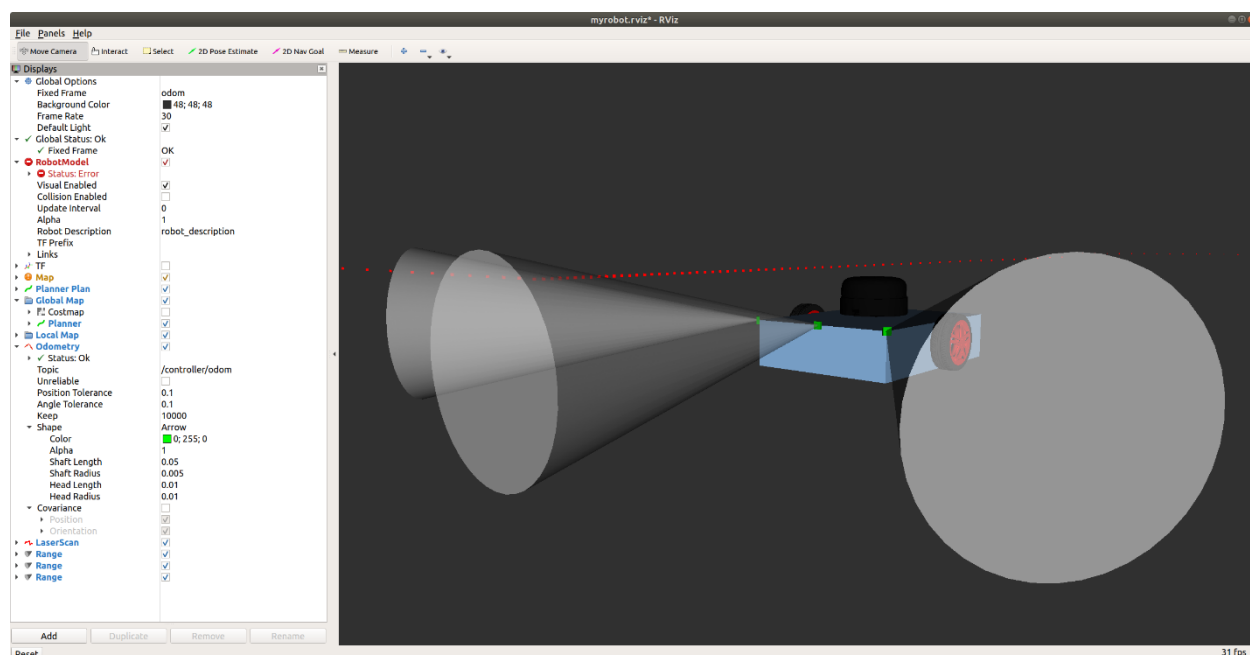


Рис. 3.1. Обновлённая модель робота

3.2. Описание узла управления роботом с клавиатуры

Для описания узла управления роботом с клавиатуры использована заготовка скрипта *keyboard_teleop*, в которой описывается работа с топином *cmd_vel* для задания линейной и угловой скоростей, а также получение данных от клавиатуры. Описание узла представлено в листинг 3.1.

В представленном скрипте содержится класс *KeyboardTeleop*, при инициализации которого происходит инициализация ноды *keyboard_teleop*, регистрация публишера для отправки сообщений топику, отвечающему за управление роботом, а также задание начальных значений полей для линейной и угловой скоростей.

Для запуска управления роботом вызывается функция *run()*, в которой сначала инициализируется слушатель нажатия кнопок на клавиатуре с двумя колбэками: за нажатие отвечает *on_key_press()*, а за отпускание - *on_key_release()*. Далее задаётся частота опроса (10 Гц) и в бесконечном цикле происходит обновление состояния робота – функция *update_key_state()*.

Колбэки *on_key_press()* и *on_key_release()* имеют одинаковую структуру, только первый задаёт значения линейной (клавиши W и S) и угловой (клавиши A и D) скорости, а второй их обнуляет.

В функции обновления состояния робота *update_key_state()* сначала считается линейная и угловая скорость робота в данный момент времени путём умножения полученных величин на максимальную скорость, после чего создаётся сообщение типа *Twist*, куда задаются полученные значения скоростей, и отправляется публишером в ранее описанный топик.

Листинг 3.1. Описание узла управления роботом с клавиатуры

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
from pynput.keyboard import KeyCode, Listener

MINICAR_MAX_LIN_VEL = 1
MINICAR_MAX_ANG_VEL = 2

class KeyboardTeleop:
    def __init__(self):
        rospy.init_node('keyboard_teleop')

        cmd_vel_topic = rospy.get_param('/keyboard_teleop/cmd_vel', '/cmd_vel')
        rospy.loginfo('Cmd Vel topic: ' + cmd_vel_topic)

        self.pub = rospy.Publisher(cmd_vel_topic, Twist, queue_size=10)

        self.key_state_vel = 0
        self.key_state_ang = 0
        self.target_linear_vel = 0.0
        self.target_angular_vel = 0.0
        self.keyboard_listener = None

    def run(self):
        self.keyboard_listener = Listener(on_press=self.on_key_press,
on_release=self.on_key_release)
        self.keyboard_listener.start()
        rate = rospy.Rate(10)
        try:
            while not rospy.is_shutdown():
                self.update_key_state()
                rate.sleep()
        except rospy.ROSInterruptException:
            pass
```

```

self.keyboard_listener.stop()
self.keyboard_listener.join()

def update_key_state(self):
    self.target_linear_vel = MINICAR_MAX_LIN_VEL * self.key_state_vel
    self.target_angular_vel = MINICAR_MAX_ANG_VEL * self.key_state_ang
    twist = Twist()
    twist.linear.x = self.target_linear_vel; twist.linear.y = 0.0; twist.linear.z = 0.0
    twist.angular.x = 0.0; twist.angular.y = 0.0; twist.angular.z =
self.target_angular_vel
    self.pub.publish(twist)

def on_key_press(self, key):
    if key == KeyCode.from_char('w'):
        self.key_state_vel = 1

    if key == KeyCode.from_char('s'):
        self.key_state_vel = -1

    if key == KeyCode.from_char('a'):
        self.key_state_ang = 1

    if key == KeyCode.from_char('d'):
        self.key_state_ang = -1

    self.update_key_state()
    return True

def on_key_release(self, key):
    if key == KeyCode.from_char('w') and self.key_state_vel != 0:
        self.key_state_vel = 0

    if key == KeyCode.from_char('s') and self.key_state_vel != 0:
        self.key_state_vel = 0

    if key == KeyCode.from_char('a') and self.key_state_ang != 0:
        self.key_state_ang = 0

    if key == KeyCode.from_char('d') and self.key_state_ang != 0:
        self.key_state_ang = 0

    self.update_key_state()
    return True

if __name__=="__main__":
    teleop = KeyboardTeleop()
    teleop.run()

```

Также стоит упомянуть сложности, с которыми я столкнулся при реализации узла управления. Для того, чтобы считывать данные с клавиатуры

В rviz мы можем наблюдать траекторию робота, после нажатых кнопок, которые были представлены ранее в окне терминала, как показано на рис. 3.3.

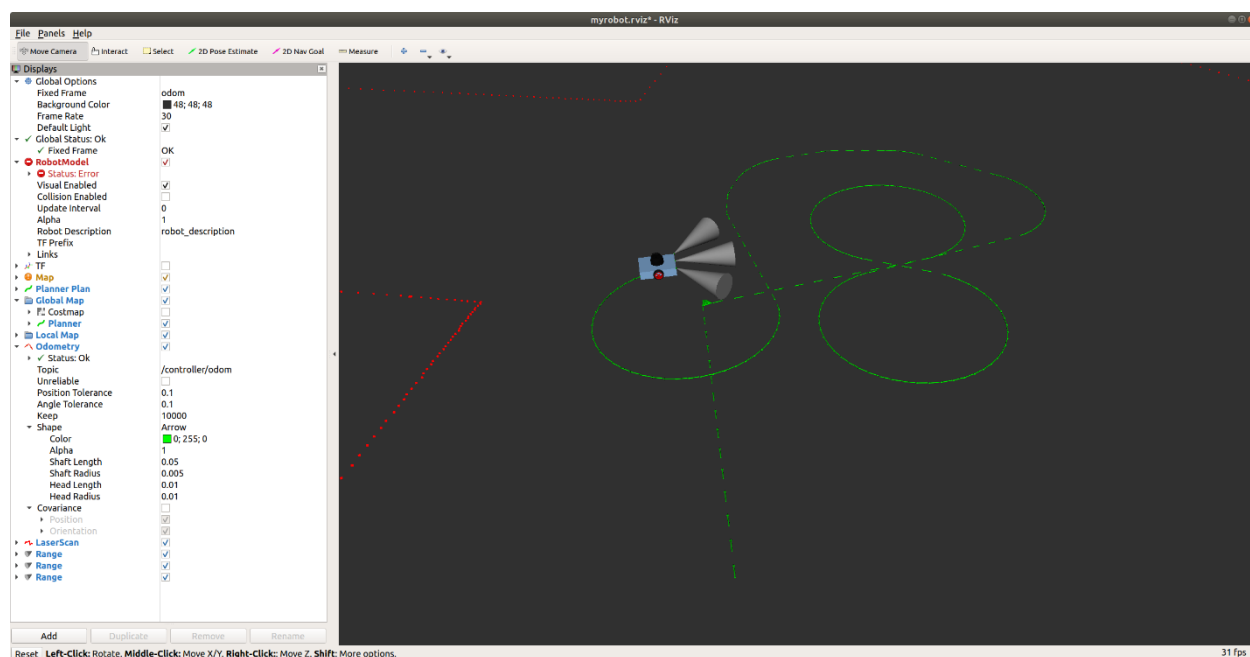


Рис. 3.3. Окно rviz с траекторией движения робота

3.3. Описание узла управления роботом в автоматическом режиме (ик-сенсоры)

Для описания узла управления роботом в автоматическом режиме при помощи ик-сенсоров модифицирован ранее описанный скрипт *keyboard_teleop*. Описание узла произведено в файле *ir_auto_teleop*, код которого представлен в листинг 3.2.

Представленный скрипт имеет схожую структуру с описанным ранее. При инициализации класса *IrAutoTeleop* происходит инициализация ноды *ir_auto_teleop*, регистрация публишера для отправки сообщений топику, отвечающему за управление роботом, трёх подписчиков на получение данных о расстоянии с ик-сенсоров, а также задание начальных значений полей для линейной и угловой скоростей, расстояний для левого, центрального и правого сенсоров.

Для запуска управления роботом вызывается функция *run()*, в которой сначала задаётся частота опроса (10 Гц), счётчики для подсчёта тактов срабатывания угловых и центрального датчика, а также флаг, отвечающий за

движения робота в обратном направлении. Далее в бесконечном цикле происходит обновление состояния робота – функция *update_state()* и описывается логика работы управления роботом с помощью ик-сенсоров.

Сначала происходит проверка на то, что один из боковых сенсоров имеет расстояние до препятствия меньше заданного расстояния (34 сантиметра). Если расстояние меньше, то увеличивается счётчик подсчёта тактов для угловых сенсоров и определяется угловая скорость в зависимости от того, какой из датчиков имеет наименьшее расстояние до препятствия, иначе угловая скорость и счётчик обнуляются. Далее происходит проверка расстояния для центрального датчика. Если расстояние меньше, то линейная скорость устанавливается в 0 и повышается счётчик подсчёта тактов для центрального сенсора, иначе линейная скорость устанавливается в 1 и счётчик обнуляется. В конце происходит проверка на достижение одного из счётчиков заданного предела (50 тактов). При положительной проверке флаг обратного движения ставится в истину, иначе в ложь. Если флаг положительный, то линейная скорость устанавливается в -1 (задний ход) и в зависимости от конкретного значения угловой скорости в данный момент высчитывается будущее направление робота. Если угловая скорость не равна нулю, то устанавливается обратное значение, чтобы робот при движении назад поворачивал в сторону датчика, имеющего меньшее расстояние до препятствия, иначе выбирается угловая скорость равная 1 для случаев, когда у робота сработал только центральный ик-сенсор, например при столкновении с препятствием острой формы.

Листинг 3.2. Описание узла управления роботом в автоматическом режиме

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
from sensor_msgs.msg import Range

MINICAR_MAX_LIN_VEL = 1
MINICAR_MAX_ANG_VEL = 2
IR_CENTER_TARGET_RANGE = 0.34
IR_SIDE_TARGET_RANGE = 0.34
```

```

CLOSE_DIST_TARGET = 50

LEFT_IR_SENSOR = 'ir_sensor_front_0'
CENTER_IR_SENSOR = 'ir_sensor_front_1'
RIGHT_IR_SENSOR = 'ir_sensor_front_2'

LEFT_IR_TOPIC = '/myrobot/sensor/front_0'
CENTER_IR_TOPIC = '/myrobot/sensor/front_1'
RIGHT_IR_TOPIC = '/myrobot/sensor/front_2'

class IrAutoTeleop:
    def __init__(self):
        rospy.init_node('ir_auto_teleop')

        cmd_vel_topic = rospy.get_param('/ir_auto_teleop/cmd_vel', '/cmd_vel')
        rospy.loginfo('Cmd Vel topic: ' + cmd_vel_topic)

        self.pub = rospy.Publisher(cmd_vel_topic, Twist, queue_size=10)

        self.range_left = 0
        self.range_center = 0
        self.range_rigth = 0

        self.sub_left = rospy.Subscriber(LEFT_IR_TOPIC, Range, callback=self.callback)
        self.sub_center = rospy.Subscriber(CENTER_IR_TOPIC, Range, callback=self.callback)
        self.sub_right = rospy.Subscriber(RIGHT_IR_TOPIC, Range, callback=self.callback)

        self.state_vel = 0
        self.state_ang = 0
        self.target_linear_vel = 0.0
        self.target_angular_vel = 0.0

    def run(self):
        rate = rospy.Rate(10)
        close_side_dist_counter = 0
        close_center_dist_counter = 0
        reverse_flag = False
        try:
            while not rospy.is_shutdown():
                self.update_state()
                rate.sleep()
                print self.range_left, self.range_center, self.range_rigth
                if self.range_left < IR_SIDE_TARGET_RANGE or self.range_rigth <
IR_SIDE_TARGET_RANGE:
                    close_side_dist_counter += 1
                    if self.range_left < self.range_rigth:
                        self.state_ang = -1
                    else:
                        self.state_ang = 1
                else:
                    self.state_ang = 0

```

```

        close_side_dist_counter = 0

        if self.range_center < IR_CENTER_TARGET_RANGE:
            self.state_vel = 0
            close_center_dist_counter += 1
        else:
            self.state_vel = 1
            close_center_dist_counter = 0

        if close_side_dist_counter >= CLOSE_DIST_TARGET or close_center_dist_counter
>= CLOSE_DIST_TARGET:
            reverse_flag = True
        else:
            reverse_flag = False

        if reverse_flag:
            self.state_vel = -1
            if self.state_ang != 0:
                self.state_ang = 1
            else:
                self.state_ang = -self.state_ang
    except rospy.ROSInterruptException:
        pass

def update_state(self):
    self.target_linear_vel = MINICAR_MAX_LIN_VEL * self.state_vel
    self.target_angular_vel = MINICAR_MAX_ANG_VEL * self.state_ang
    twist = Twist()
    twist.linear.x = self.target_linear_vel; twist.linear.y = 0.0; twist.linear.z = 0.0
    twist.angular.x = 0.0; twist.angular.y = 0.0; twist.angular.z =
self.target_angular_vel
    self.pub.publish(twist)

def callback(self, msg):
    if msg.header.frame_id == LEFT_IR_SENSOR:
        self.range_left = msg.range
    elif msg.header.frame_id == CENTER_IR_SENSOR:
        self.range_center = msg.range
    elif msg.header.frame_id == RIGHT_IR_SENSOR:
        self.range_rigth = msg.range

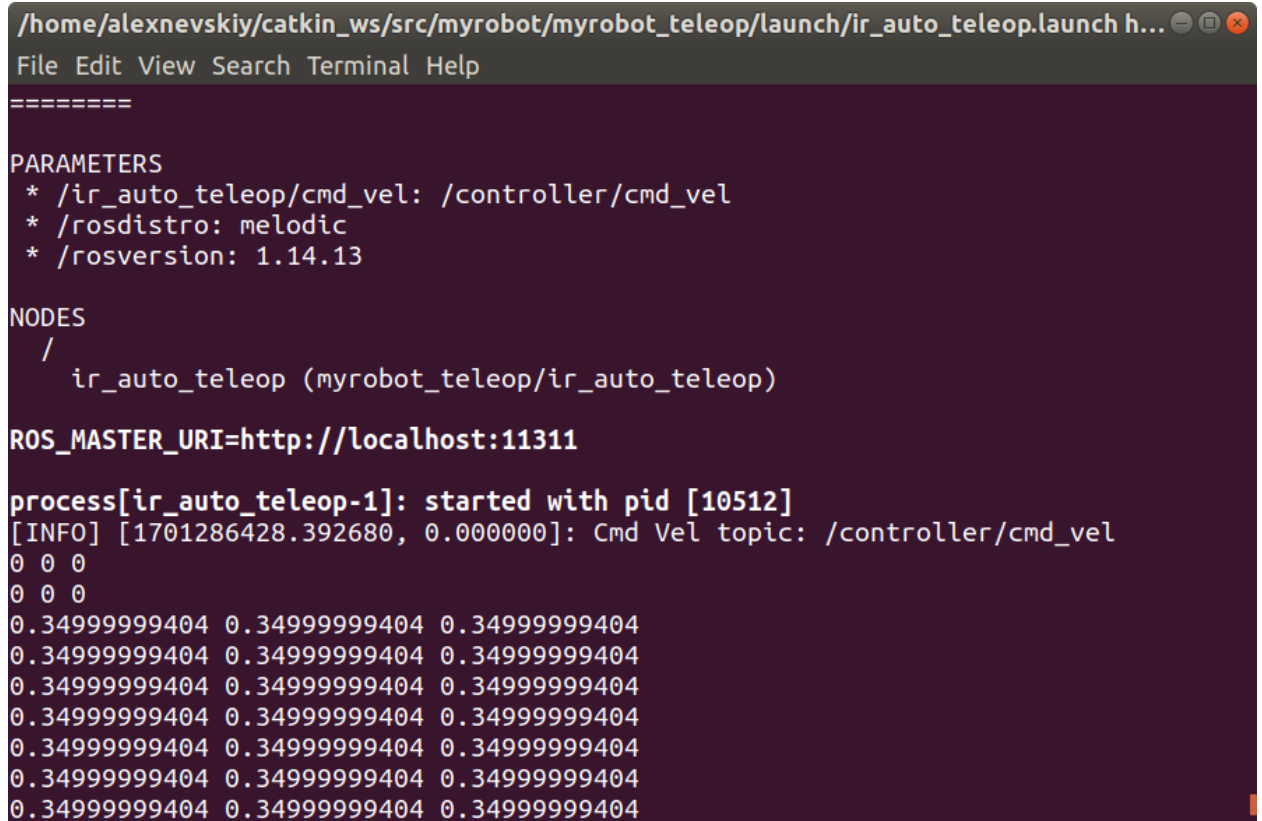
if __name__=="__main__":
    teleop = IrAutoTeleop()
    teleop.run()

```

3.3.1. Запуск работа

Для запуска работа необходимо снова запустить контроллер, симулятор и rviz. После инициализации всех компонентов необходимо запустить узел для управления роботом в автоматическом режиме при помощи *roslaunch*

myrobot_teleop ir_auto_teleop.launch, предварительно создав launch-файл по аналогии с *teleop_keyboard.launch*. Запуск узла в терминале представлен на рис. 3.4. В терминале также отображаются расстояния до препятствий с каждого ик-сенсора на каждый такт работы.



```

/home/alexnevskiy/catkin_ws/src/myrobot/myrobot_teleop/launch/ir_auto_teleop.launch h...
File Edit View Search Terminal Help
=====
PARAMETERS
* /ir_auto_teleop/cmd_vel: /controller/cmd_vel
* /roscdistro: melodic
* /rosversion: 1.14.13

NODES
/
  ir_auto_teleop (myrobot_teleop/ir_auto_teleop)

ROS_MASTER_URI=http://localhost:11311

process[ir_auto_teleop-1]: started with pid [10512]
[INFO] [1701286428.392680, 0.000000]: Cmd Vel topic: /controller/cmd_vel
0 0 0
0 0 0
0.34999999404 0.34999999404 0.34999999404
0.34999999404 0.34999999404 0.34999999404
0.34999999404 0.34999999404 0.34999999404
0.34999999404 0.34999999404 0.34999999404
0.34999999404 0.34999999404 0.34999999404
0.34999999404 0.34999999404 0.34999999404
0.34999999404 0.34999999404 0.34999999404
0.34999999404 0.34999999404 0.34999999404

```

Рис. 3.4. Окно терминала с узлом управления в автоматическом режиме

В *rviz* мы можем наблюдать траекторию робота, после некоторого времени работы автоматического управления, как показано на рис. 3.5. Видно, что робот при приближении к стене направляется в противоположную сторону от неё.

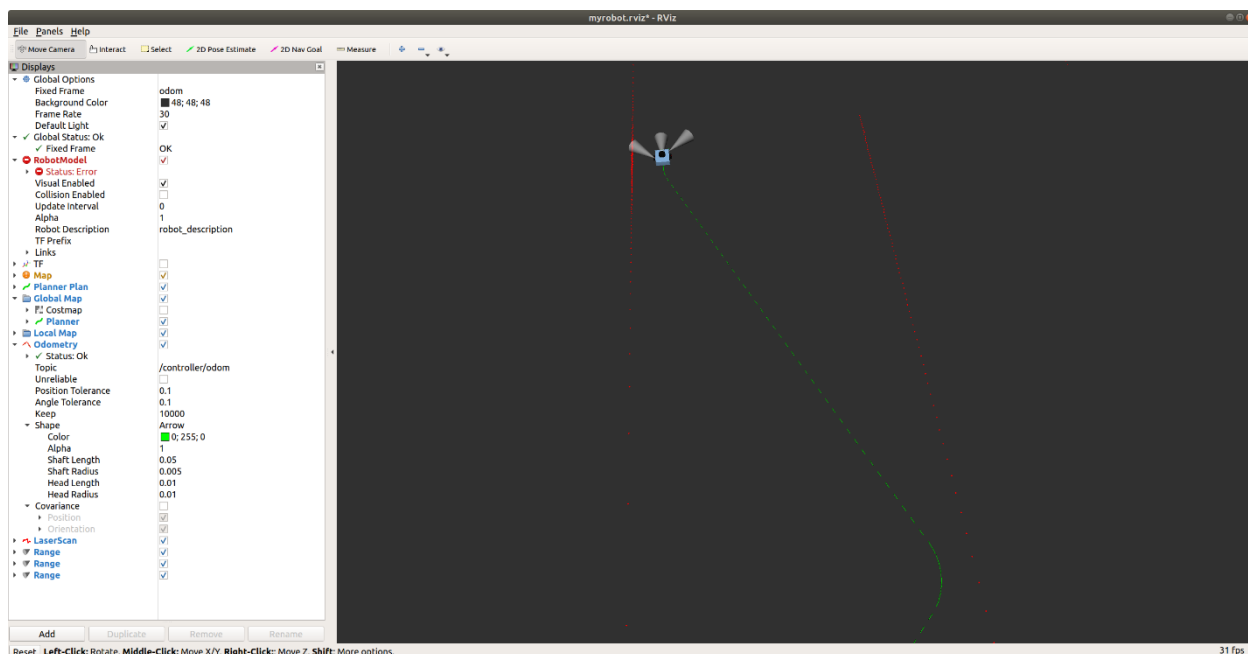


Рис. 3.5. Окно rviz с траекторией движения робота в автоматическом режиме

Пример управления роботом в автоматическом режиме в gazebo представлен на рис. 3.6.

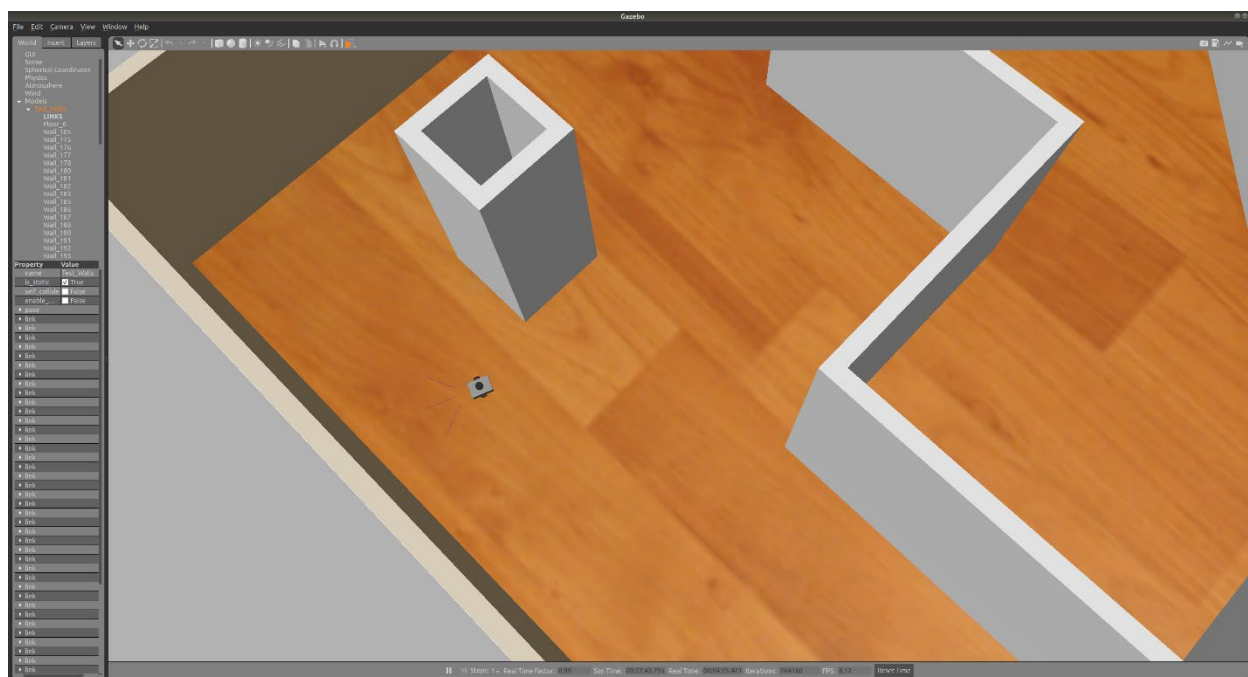


Рис. 3.6. Окно gazebo с автоматическим управлением роботом

4. ВЫВОДЫ

В ходе выполнения данной лабораторной работы произведено описание узла управления роботом с клавиатуры. Для этого написан скрипт на языке

Python, в котором производится считывание нажатий клавиш с клавиатуры, и в зависимости от нажатой клавиши (W – вперёд, S – назад, A – влево, D – вправо) задаётся либо линейная скорость, либо угловая, либо обе одновременно.

Помимо этого произведено описание узла управления роботом в автоматическом режиме при помощи ик-сенсоров, скрипт для которого также написан на языке Python. В нём производится считывание расстояний от трёх сенсоров до препятствий (максимальное расстояние ик-сенсора равняется 35 сантиметрам), и в зависимости от того, какой из них имеет наименьшее расстояние до препятствия (менее 34 сантиметров), вычисляется линейная или угловая скорости. Предложенный алгоритм автоматического управления предусматривает выезд из любых тупиковых ситуаций, а также случаи, когда робот упирается в острые углы, расстояние до которых фиксирует только центральный сенсор.