Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологии
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Дисциплина: Программное обеспечение встраиваемых систем

Тема: Разработка мобильного робота в среде ROS

Выполнил студент гр. 01502 С.С. Гаспарян

Руководитель, ст. преподаватель Г.С. Васильянов

«16» декабря 2021

Санкт-Петербург
2021

Задание №1

Ознакомиться с работой межпроцессорного взаимодействия узлов ROS. Реализовать публикацию нескольких топиков узлов ROS и выполнить симуляцию в среде rqt.

Решение

В листинге 1 приведен список команд для клонирования проекта из репозитория и сборки проекта для выполнения симуляции.

Листинг 1

```
# Создание папки с проектами и рабочего пространства для ROS
mkdir -p ~/myprojects/catkin_ws/src
# Инициализация рабочего пространства ROS
cd ~/myprojects/catkin_ws/src
catkin_init_workspace
# Клонируем репозиторий
git clone https://github.com/ros/ros_tutorials.git -b noetic-devel
# Установка зависимостей
cd ..
rosdep install --from-paths src --ignore-src -r -y
# Сборка проектов
catkin_make
```

На рисунке 1 представлен запуск из терминала узла-мастера для выполнения публикации сообщений.

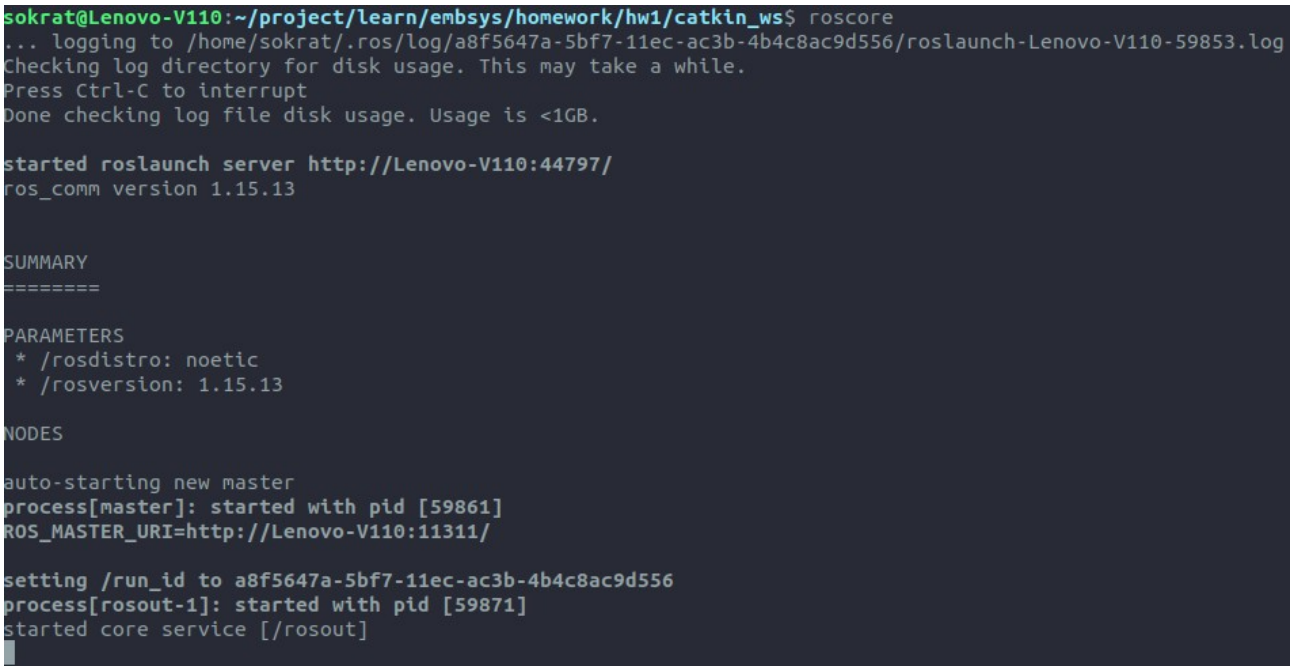

Рис. 1. Запуск узла мастера

Для того, проверки работы передачи между узлами сообщении в среде ROS будет использоваться пример с симуляцией — turtlesim. В данном примере, управляемый объект получает в сообщениях узла координаты и угловое положение в 2D пространстве. Таким образом мы можем проверить отправку сообщении на данный узел. Проверка будет происходить в среде rqt. В rqt будут рассмотрены несколько топиков с сообщениями для управления объектом среды симулирования.

Далее на рисунке 2 представлен запуск из терминала симулятора turtlesim с публикацией узла turtlesim_node. В данном узле задаётся координаты в пространстве для управляемого объекта.

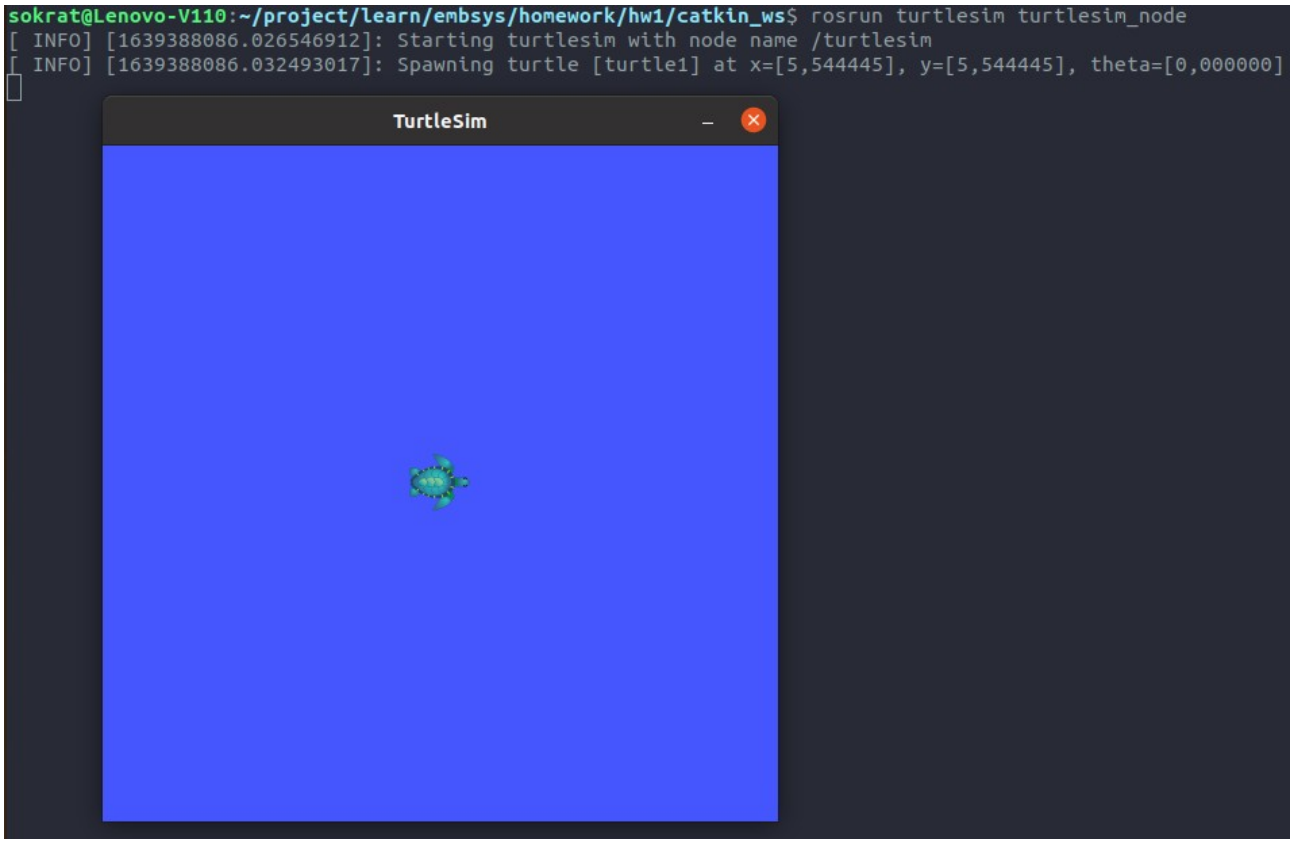

Рис. 2. Запуск среды симулирования turtlesim

Далее на рисунке 3 представлен запуск среды rqt, в котором задан в качестве текущего топика узла линейная и угловая скорость объекта. На рисунке 4 показано использование плагина Topic Monitor, в котором отображаются все активные топики среды ROS. Также из рисунка 4, что для объекта задана линейная скорость = 0.03 м/с и угловая скорость = 0.14 рад/с.

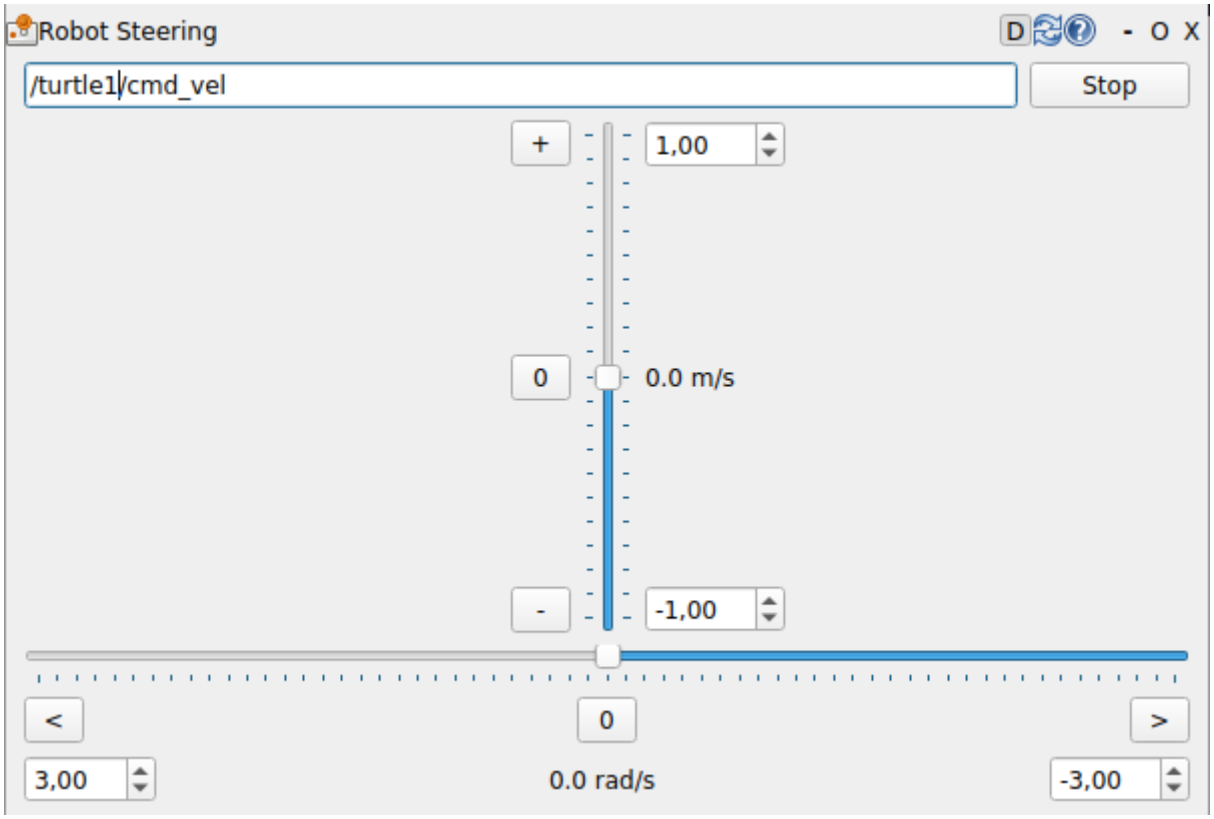

Рис. 3. Запуск среды rqt с топиком turtle1/cmd_vel

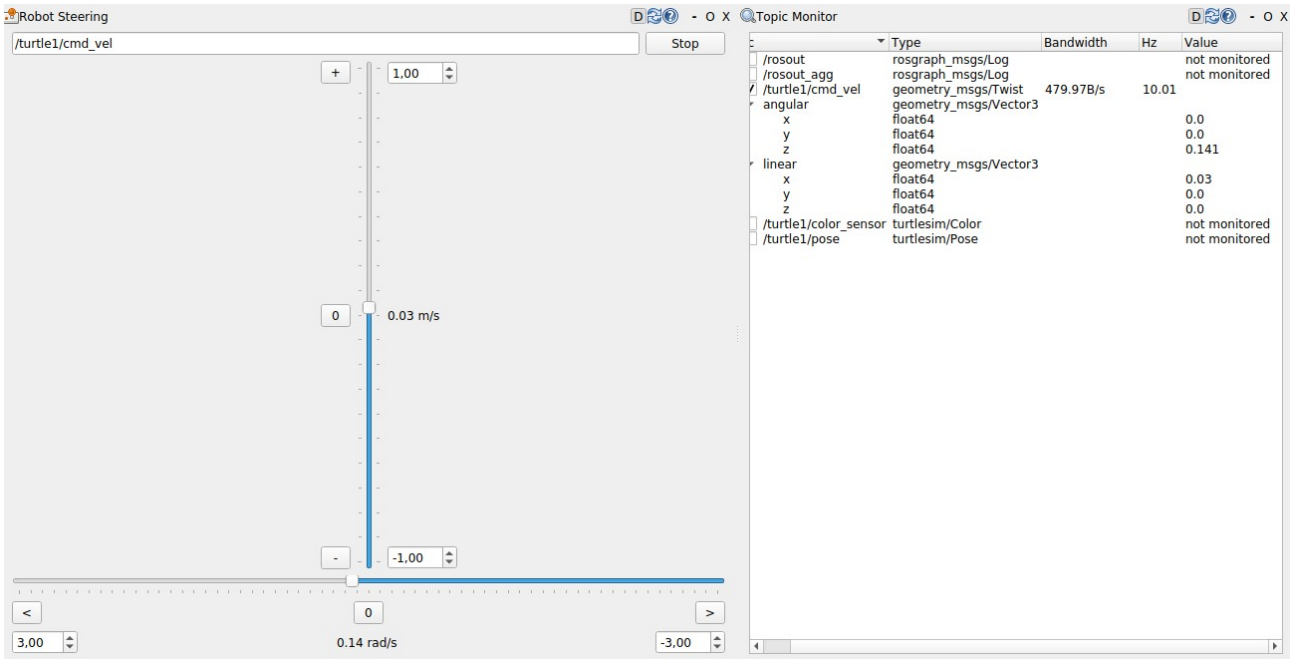

Рис. 4. Среда rqt с активными топиками

Мы можем самостоятельно в ручную публиковать данные для топиков. Сделать это можно несколькими способами. Первый способ был представлен на рисунке 4, где с помощью ползунков была задана скорость для объекта управления. На рисунке 5 представлен другой способ с использованием плагина Message Publisher, в котором вручную задаются параметры для топика узла.

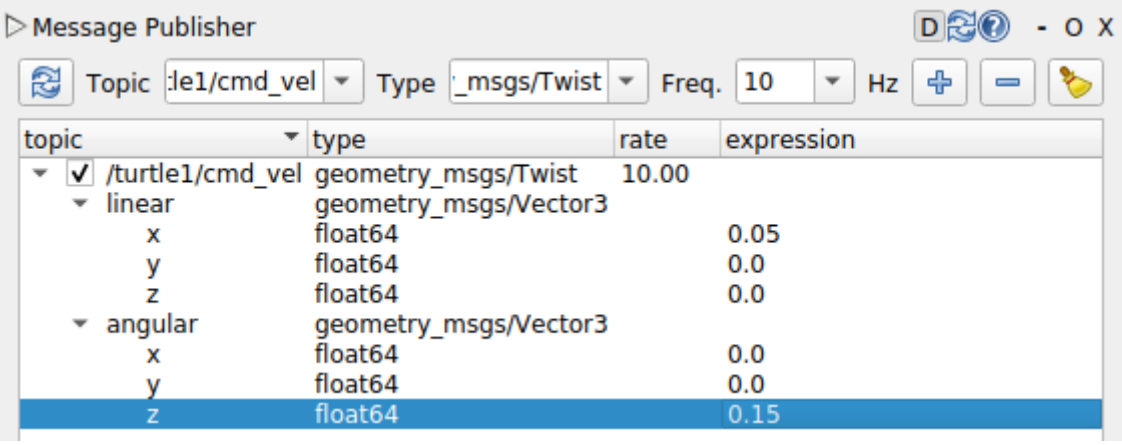

Рис. 5. Использование плагина Message Publisher в rqt

Результат симуляции с заданными параметрами скорости, из рисунка 5, представлен на рисунке 6. Из рисунка видно, что объект управления двигается по окружности с постоянной скоростью, оставляя за собой след пройденного пути. Таким образом, мы видим, что данные параметры передается в узел turtlesim_node из которого далее они считывается в среду симуляции.

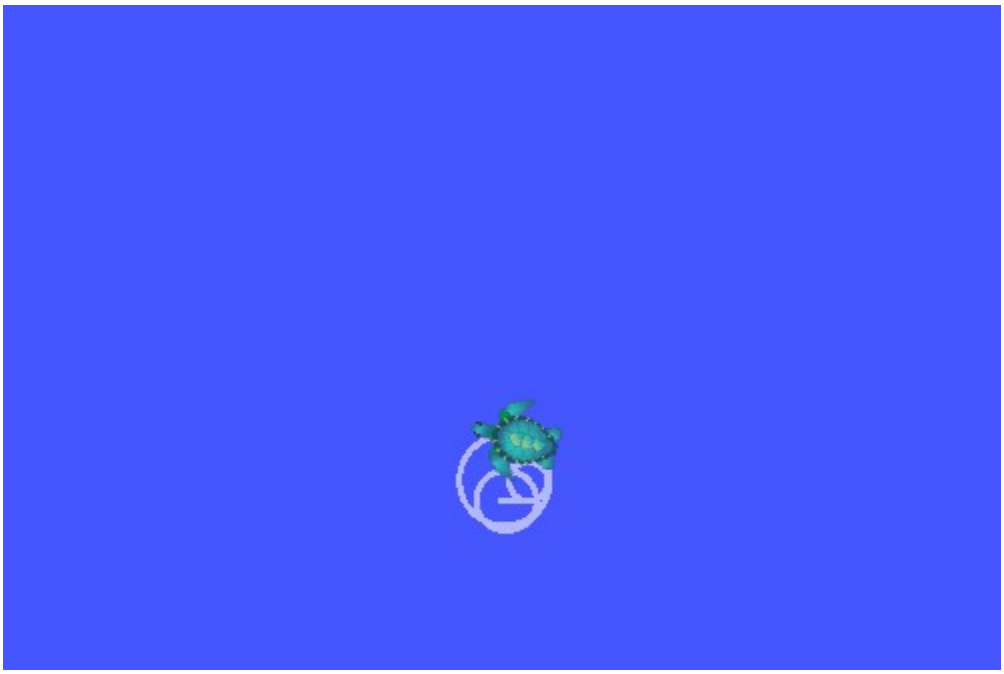

Рис. 6. Результат симуляции, после публикации параметров в сообщения

В дополнение, на рисунке 7 представлен рисунок с графом активных узлов среды ROS, который можно открыть с помощью плагина Node Graph программы rqt. Как видно из рисунка в узле /turtle1 есть несколько топиков, которым можно задавать значение в зависимости от их типа данных. Все эти топики относятся к одному узлу /turtlesim, являющийся нашей средой симуляции.

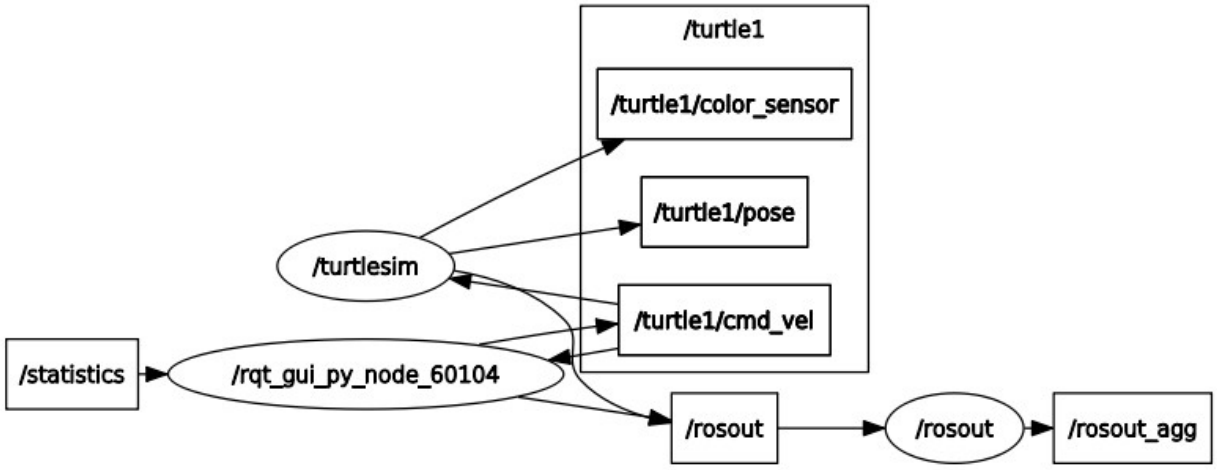

Рис. 7. Граф активных топиков ROS в rqt

Задание №2

Ознакомиться с языковой системой описания роботов URDF и системой макросов xacro. Реализовать с помощью данных систем двухколесного робота. Выполнить симуляцию спроектированного робота в среде Gazebo simulator и rviz.

Решение

В листинге 2 приведено описание двухколесного робота в формате URDF с использованием макросов xacro, который состоит из корпуса — коробки(параллелепипед) и двух цилиндров, представляющих колеса. Тип соединения корпуса и колес — continuous, которое не имеет ограничений при вращений. Также для описания физических свойств робота используется момент инерции для цилиндра и кубойда.

Листинг 2

```
<?xml version="1.0"?>

<robot xmlns:xacro="http://ros.org/wiki/xacro">

    <!-- Constants for robot dimensions -->
   <xacro:property name="PI" value="3.1415926535897931"/>
   <xacro:property name="BIAS" value="0.0001"/>

   <!-- Пустой линк -->
   <link name="dummy">
   </link>

   <!-- корпус -->
   <xacro:macro name="box_inertial" params="length width height *jorigin">
      <link name="base_link">
         <xacro:base_inertial length="${length}" width="${width}" height="$
{height}" mass="${chassis_mass}">
            <xacro:insert_block name="jorigin"/>
         </xacro:base_inertial>
         <visual>
            <xacro:insert_block name="jorigin"/>
            <geometry>
               <box size="${length} ${width} ${height}"/>
```

```
            </geometry>

        </visual>
        <collision>
            <xacro:insert_block name="jorigin"/>
            <geometry>
                <box size="${width} ${length} ${height}"/>
            </geometry>
        </collision>
    </link>

    <joint name="dummy_joint" type="fixed">
        <parent link="dummy"/>
        <child link="base_link"/>
    </joint>

</xacro:macro>

<!-- колёса -->
<xacro:macro name="wheel" params="wheel_prefix parent_link left_right radius
width
*joint_origin">

    <link name="${wheel_prefix}_wheel">
        <xacro:cylinder_inertial length="${width}" mass="${wheel_mass}"
radius="${radius}">
            <origin rpy="${PI/2} 0 0" xyz="${wheelbase_offset}
            ${left_right*wheelbase_length/2} ${radius-chassis_clearance}"/>
        </xacro:cylinder_inertial>
        <visual>
            <origin rpy="${left_right * PI/2} 0 ${PI}" xyz="${wheelbase_offset}
            ${left_right*wheelbase_length/2} ${radius-chassis_clearance}"/>
            <geometry>
              <mesh filename="package://myrobot_description/meshes/tire.dae"
scale="${radius} ${radius} ${width/2}"/>
            </geometry>
        </visual>

        <collision>
            <origin rpy="${PI/2} 0 0" xyz="${wheelbase_offset}
            ${left_right*wheelbase_length/2} ${radius-chassis_clearance}"/>
            <geometry>
              <cylinder length="${width}" radius="${radius}"/>
            </geometry>
```

```
        </collision>
      </link>

      <joint name="${wheel_prefix}_wheel_joint" type="continuous">
        <parent link="${parent_link}"/>
        <child link="${wheel_prefix}_wheel"/>
        <xacro:insert_block name="joint_origin"/>
        <axis rpy="0 0 0" xyz="0 1 0"/>
      </joint>

      <transmission name="${wheel_prefix}_wheel_trans"
type="SimpleTransmission">
        <type>transmission_interface/SimpleTransmission</type>
        <actuator name="${wheel_prefix}_wheel_motor">
          <mechanicalReduction>1</mechanicalReduction>
        </actuator>
        <joint name="${wheel_prefix}_wheel_joint">

<hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
        </joint>
      </transmission>

   </xacro:macro>

   <!-- Инерция -->
   <xacro:macro name="cylinder_inertial" params="radius length mass *origin">
      <inertial>
      <mass value="${mass}"/>
      <xacro:insert_block name="origin"/>
      <inertia ixx="${0.0833333 * mass * (3 * radius * radius + length * length)}"
            ixy="0.0" ixz="0.0" iyy="${0.0833333 * mass * (3 * radius * radius +
length * length)}"
            iyz="0.0" izz="${0.5 * mass * radius * radius}"/>
      </inertial>
   </xacro:macro>

   <xacro:macro name="base_inertial" params="length width height mass *origin">
      <inertial>
      <mass value="${mass}"/>
      <xacro:insert_block name="origin"/>
      <inertia ixx="${0.0833333 * mass * (height*height + length*length)}"
            ixy="0.0" ixz="0.0" iyy="${0.0833333 * mass * (width*width +
length*length)}"
            iyz="0.0" izz="${0.0833333 * mass * (width*width + height*height)}"/>
```

```
      </inertial>
   </xacro:macro>

   <xacro:macro name="base_no_inertial" params="length width height mass
*origin">
      <inertial>
      <mass value="${mass}"/>
      <xacro:insert_block name="origin"/>
      <inertia ixx="$0.0" ixy="0.0" ixz="0.0"
            iyy="0.0" iyz="0.0" izz="$0.0"/>
      </inertial>
   </xacro:macro>

</robot>
```

В листинге 3 приведено создание экземпляров с помощью макросов xacro для описания робота в среде Gazebo simulation. Робот имеет следующие характеристики:

1. Ширина колеса = 0.02
2. Радиус колеса = 0.0325
3. Масса колеса = 1.0
4. Длина корпуса(шасси) = 0.2
5. Ширина корпуса = 0.125
6. Высота корпуса = 0.04
7. Высота посадки = 0.02
8. Масса корпуса = 10.0

Листинг 3

```
<?xml version="1.0"?>
<robot name="myrobot" xmlns:xacro="http://www.ros.org/wiki/xacro">

   <!-- property wheel -->
   <xacro:property name="wheel_radius" value="0.0325"/>
   <xacro:property name="wheel_mass" value="1"/>
   <xacro:property name="wheel_width" value="0.02"/>
   <xacro:property name="wheelbase_length" value="0.16"/>
   <xacro:property name="wheelbase_offset" value="0.0"/>
```

```
  <!-- property chassis -->
  <xacro:property name="chassis_length" value="0.2"/>
  <xacro:property name="chassis_width" value="0.125"/>
  <xacro:property name="chassis_height" value="0.04"/>
  <xacro:property name="chassis_clearance" value="0.02"/>
  <xacro:property name="chassis_mass" value="10.0"/>

  <xacro:include filename="$(find myrobot_description)/urdf/macro.xacro"/>

  <!-- ============= BASE =============== -->
  <xacro:box_inertial length="${chassis_length}" width="${chassis_width}"
height="${chassis_height}">
      <origin xyz="0 0 ${chassis_clearance}"/>
  </xacro:box_inertial>

  <!-- ============= WHEELS =============== -->
  <xacro:wheel wheel_prefix="left" parent_link="base_link" left_right="1"
radius="${wheel_radius}" width="${wheel_width}">
      <joint_origin xyz="${wheelbase_offset} ${wheelbase_length/2} $
{wheel_radius - chassis_clearance}"/>
  </xacro:wheel>

  <xacro:wheel wheel_prefix="right" parent_link="base_link" left_right="-1"
radius="${wheel_radius}" width="${wheel_width}">
      <joint_origin xyz= "${wheelbase_offset} ${-wheelbase_length/2} $
{wheel_radius - chassis_clearance}"/>
  </xacro:wheel>

</robot>
```

На рисунке 8 представлена модель двухколесного робота со всеми link и joint в среде симулирования Gazebo. Как видно из рисунка робот полностью совпадает с описанием в URDF файле. Также на рисунке 9 показан этот же робот в среде симуляцией rviz.

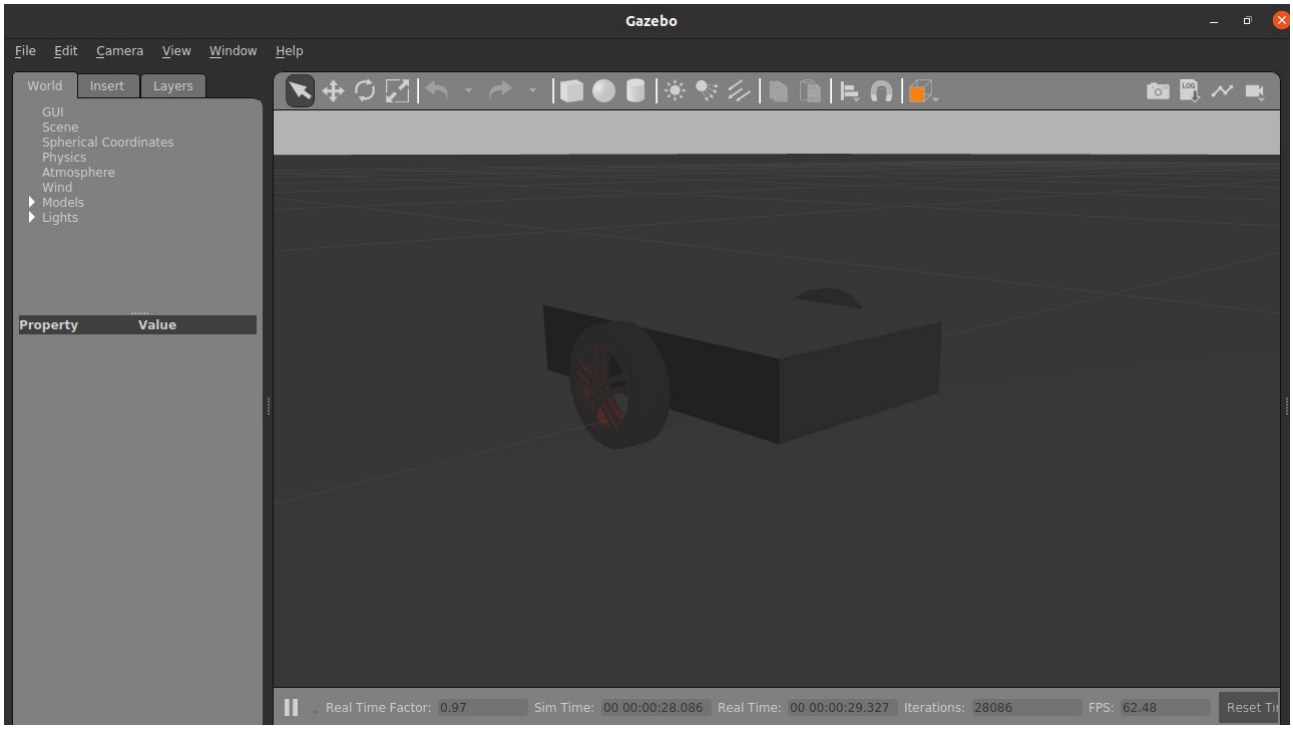

Рис. 8. Модель робота в Gazebo simulation

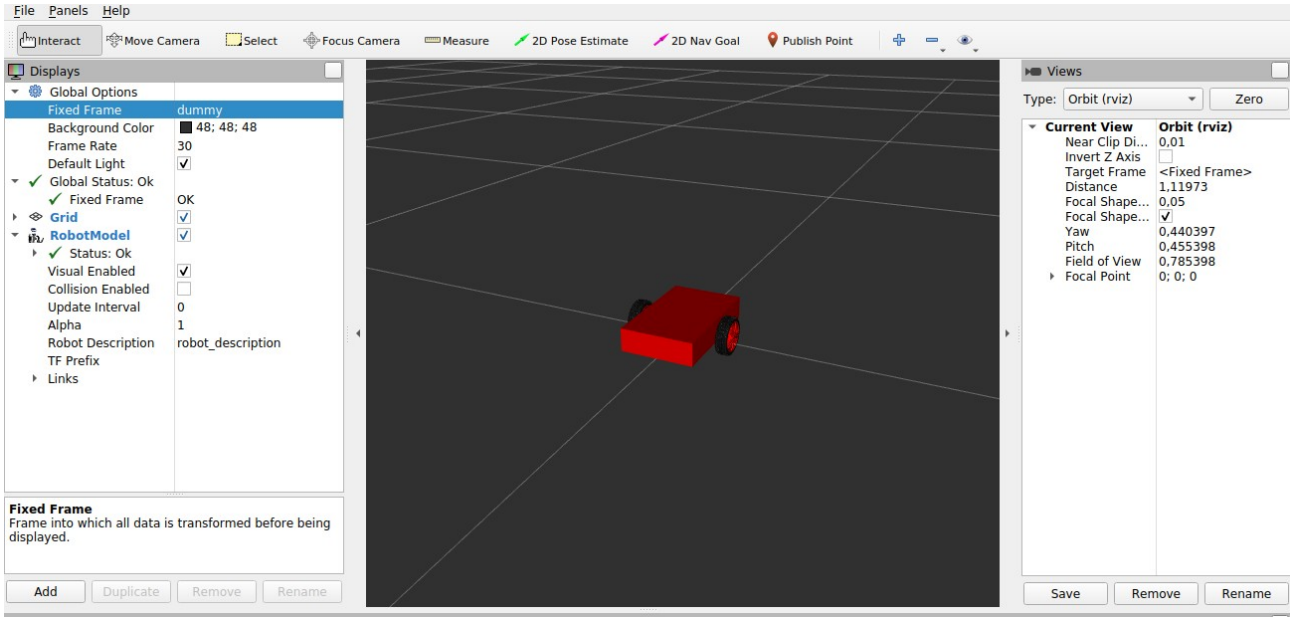

Рис. 9. Модель робота в среде rviz

Задание №3

Реализовать программное управление мобильного робота в ROS с помощью клавиатуры. Реализовать автономное управление роботом по инфракрасным датчикам.

Решение

В листинге 4 представлено URDF описание четырехколесного робота. Два колеса были добавлены к реализации предыдущего задания для улучшенной балансировки корпуса робота. Также, для задних колес были изменены параметры радиуса и массы, для лучшей управляемости. В конце описания добавлены IR датчики, функция которых заключается в обнаружении препятствии на пути робота. Параметры лучей датчика приведены в листинге 5, где минимальный размер луче 0.01 единиц и максимальный 2.35 единиц. Также в конце листинга 5 описаны топики, которые публикуют показания датчиков.

Листинг 4

```
<?xml version="1.0"?>
<robot name="myrobot" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <!-- property wheel -->
  <xacro:property name="wheel_radius" value="0.0325"/>
  <xacro:property name="wheel_mass" value="1"/>
  <xacro:property name="wheel_width" value="0.02"/>
  <xacro:property name="wheelbase_length" value="0.16"/>
  <xacro:property name="wheelbase_offset" value="0.05"/>

  <!-- property chassis -->
  <xacro:property name="chassis_length" value="0.2"/>
  <xacro:property name="chassis_width" value="0.125"/>
  <xacro:property name="chassis_height" value="0.04"/>
  <xacro:property name="chassis_clearance" value="0.02"/>
  <xacro:property name="chassis_mass" value="10.0"/>

  <xacro:include filename="$(find myrobot_description)/urdf/macro.xacro"/>

  <!-- ============ BASE =============== -->
```

```
  <xacro:box_inertial length="${chassis_length}" width="${chassis_width}"
height="${chassis_height}">
    <origin rpy="0 0 0" xyz="0 0 ${chassis_clearance}"/>
  </xacro:box_inertial>


  <!-- ============= WHEELS =============== -->
  <xacro:wheel wheel_prefix="left" parent_link="base_link" left_right="1"
radius="${wheel_radius}" width="${wheel_width}">
    <origin xyz="${wheelbase_offset} ${OFFSET_WHEEL*wheelbase_length} $
{wheel_radius - chassis_clearance}"/>
  </xacro:wheel>

  <xacro:wheel wheel_prefix="right" parent_link="base_link" left_right="-1"
radius="${wheel_radius}" width="${wheel_width}">
    <origin xyz="${wheelbase_offset} ${-OFFSET_WHEEL*wheelbase_length}
${wheel_radius - chassis_clearance}"/>
  </xacro:wheel>

  <xacro:back_wheel wheel_prefix="back_left" parent_link="base_link"
left_right="1" radius="${wheel_radius}" width="${wheel_width}">
    <origin xyz="${-wheelbase_offset} ${OFFSET_WHEEL*wheelbase_length}
${wheel_radius - chassis_clearance}"/>
  </xacro:back_wheel>

  <xacro:back_wheel wheel_prefix="back_right" parent_link="base_link"
left_right="-1" radius="${wheel_radius}" width="${wheel_width}">
    <origin xyz="${-wheelbase_offset} ${-OFFSET_WHEEL*wheelbase_length}
${wheel_radius - chassis_clearance}"/>
  </xacro:back_wheel>

  <!-- ============= IR SENSORS =============== -->
  <xacro:ir_sensor name="front_0" parent="base_link" sx="0.003" sy="0.008"
sz="0.008">
    <origin rpy="0 0 0" xyz="${0.525*chassis_length} 0 ${chassis_clearance}"/>
  </xacro:ir_sensor>

  <xacro:ir_sensor name="front_1" parent="base_link" sx="0.003" sy="0.008"
sz="0.008">
    <origin rpy="0 0 -0.4" xyz="${0.525*chassis_length} ${-0.5*chassis_width} $
{chassis_clearance}"/>
  </xacro:ir_sensor>

  <xacro:ir_sensor name="front_2" parent="base_link" sx="0.003" sy="0.008"
```

```
sz="0.008">
      <origin rpy="0 0 0.4" xyz="${0.525*chassis_length} ${0.5*chassis_width} $
{chassis_clearance}"/>
   </xacro:ir_sensor>

</robot>
```

Листинг 5

```
<xacro:macro name="ir_sensor_gazebo" params="name *topic_name
*frame_name">
      <gazebo reference="ir_sensor_${name}">
         <sensor type="ray" name="ir_sensor_${name}_abc">
            <pose>0 0 0 0 0 0</pose>
            <update_rate>50</update_rate>
            <visualize>true</visualize>
            <always_on>true</always_on>
            <ray>
               <scan>
                  <horizontal>
                     <samples>1</samples>
                     <resolution>1.0</resolution>
                     <min_angle>-0.01</min_angle>
                     <max_angle>0.01</max_angle>
                  </horizontal>
                  <vertical>
                     <samples>1</samples>
                     <resolution>1.0</resolution>
                     <min_angle>-0.01</min_angle>
                     <max_angle>0.01</max_angle>
                  </vertical>
               </scan>
               <range>
                  <min>0.01</min>
                  <max>2.35</max>
                  <resolution>0.1</resolution>
               </range>
            </ray>
            <plugin filename="libgazebo_ros_range.so" name="gazebo_ros_range">
               <gaussianNoise>0.005</gaussianNoise>
               <alwaysOn>true</alwaysOn>
               <updateRate>5</updateRate>
               <xacro:insert_block name="topic_name"/>
               <xacro:insert_block name="frame_name"/>
```

```
            <visualize>true</visualize>
            <radiation>infrared</radiation>
            <fov>0.436</fov>
          </plugin>
        </sensor>
    </gazebo>
  </xacro:macro>

  <xacro:ir_sensor_gazebo name="front_0">
    <topicName>/myrobot/sensor/front_0</topicName>
    <frameName>ir_sensor_front_0</frameName>
  </xacro:ir_sensor_gazebo>
  <xacro:ir_sensor_gazebo name="front_1">
    <topicName>/myrobot/sensor/front_1</topicName>
    <frameName>ir_sensor_front_1</frameName>
  </xacro:ir_sensor_gazebo>
   <xacro:ir_sensor_gazebo name="front_2">
    <topicName>/myrobot/sensor/front_2</topicName>
    <frameName>ir_sensor_front_2</frameName>
  </xacro:ir_sensor_gazebo>
```

В листинге 6 приведена реализация управления роботом с помощью клавиатуры на *Python*, в которой создается узел для публикации данных, где в качестве данных выступает линейная и угловая скорость. Линейная скорость задается с помощью клавиш «*w*» и «*s*», что соответствует управлению роботом вперед и назад. Угловая скорость задается клавишами «*a*» и «*d*», что соответствует управлению роботом влево и вправо.

Листинг 6

```
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.speed_mul = 0.25
        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.speed_mul
        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
        elif key == KeyCode.from_char('s'):
            self.key_state_vel = -1

        if key == KeyCode.from_char('a'):
            self.key_state_ang = 1
        elif 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
        elif 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
        elif key == KeyCode.from_char('d') and self.key_state_ang != 0:
            self.key_state_ang = 0
        elif key == Key.esc:
            return False
        self.update_key_state()
        return True
```

В листинге 7 приведена реализация автономного управления роботом на *Python*. Управление происходит по средствам публикации скорости в узел Publisher, используя для логики управления данные с датчиков. При пересечении датчика заданного порога, у робота пересчитывается линейная и угловая скорость в зависимости от положения датчика. Данные с датчиков приходят с помощью узлов Listener, которые проводят обработку данных в функциях обработчиках «callback_ir_front». Линейная скорость зависит только от переднего датчика, угловая скорость зависит от боковых датчиков. Также, в реализации учитываются случай, когда оба боковых датчика пересекли порог данных.

Листинг 7

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

pub_cmd_vel = None
MINICAR_MAX_LIN_VEL = 1
MINICAR_MAX_ANG_VEL = 2
UPDATE_VEL = 0.25
UPDATE_ANG = 1.1
MIN_RANGE = 0.01
MAX_RANGE = 2.35
```

```
RANGE_THRESHOLD = 1.1
FRONT_RANGE_THRESHOLD = 0.45
GAIN_ANG = 5
range_front1 = MAX_RANGE
range_front2 = MAX_RANGE

twist = Twist()

def update_velocity(angular, velocity):
    global twist
    if (velocity != None):
        target_linear_vel = MINICAR_MAX_LIN_VEL * velocity
        twist.linear.x = target_linear_vel
    if (angular != None):
        target_angular_vel = MINICAR_MAX_ANG_VEL * angular
        twist.angular.z = target_angular_vel
    return


def callback_ir_front0(data):
    global pub_cmd_vel
    range_value = data.range
    is_left = range_front1 <= RANGE_THRESHOLD
    is_right = range_front2 <= RANGE_THRESHOLD
    ### update linear velocity
    if (range_value > FRONT_RANGE_THRESHOLD and not is_left and not
is_right):
        update_velocity(None, UPDATE_VEL)
    else:
        update_velocity(None, 0)

    ### update angular velocity
    if (is_left and is_right):
        if (range_front1 < range_front2):
            update_velocity(GAIN_ANG*UPDATE_ANG, None)
        else:
            update_velocity(-GAIN_ANG*UPDATE_ANG, None)
    elif(is_left and range_front2 > RANGE_THRESHOLD):
        update_velocity(UPDATE_ANG, None)
    elif(is_right and range_front1 > RANGE_THRESHOLD):
        update_velocity(-UPDATE_ANG, None)
    else:
        update_velocity(0, None)
```

```
    ### publish result velocity
    pub_cmd_vel.publish(twist)


def callback_ir_front1(data):
    global range_front1
    range_front1 = data.range

def callback_ir_front2(data):
    global range_front2
    range_front2 = data.range


def callback_cmd_vel(data):
    rospy.loginfo('Robot linear velocity is a %s', data.linear)
    rospy.loginfo('Robot angular velocity is a %s', data.angular)


def run_ir_sensor_reader():
    global pub_cmd_vel
    rospy.init_node('ir_sensors_control', anonymous=True)
    pub_cmd_vel = rospy.Publisher('/controller/cmd_vel', Twist, queue_size=10)
    rospy.Subscriber('/myrobot/sensor/front_0', Range, callback_ir_front0)
    rospy.Subscriber('/myrobot/sensor/front_1', Range, callback_ir_front1)
    rospy.Subscriber('/myrobot/sensor/front_2', Range, callback_ir_front2)

    #rospy.Subscriber('/controller/cmd_vel', Twist, callback_cmd_vel)
    rospy.spin()
    return

def init_velocity():
    global twist
    twist.linear.x = 0.0
    twist.linear.y = 0.0
    twist.linear.z = 0.0
    twist.angular.x = 0.0
    twist.angular.y = 0.0
    twist.angular.z = 0.0

if __name__ == '__main__':
    try:
        init_velocity()
        run_ir_sensor_reader()
    except rospy.ROSInterruptException as err:
```

```
        print('Error! %s' % err)
        pass
```

На рисунке 10 представлена модель четырехколесного робота с тремя IR датчиками(синие лучи) в среде симулирования Gazebo, где в качестве модели мира для симуляции выбрана модель со стенами - «test_walls.world». На рисунка 11, 12, 13 представлено автоматическое перемещение робота по миру. Робот объезжает стены, исключая тем самым столкновения с ними.

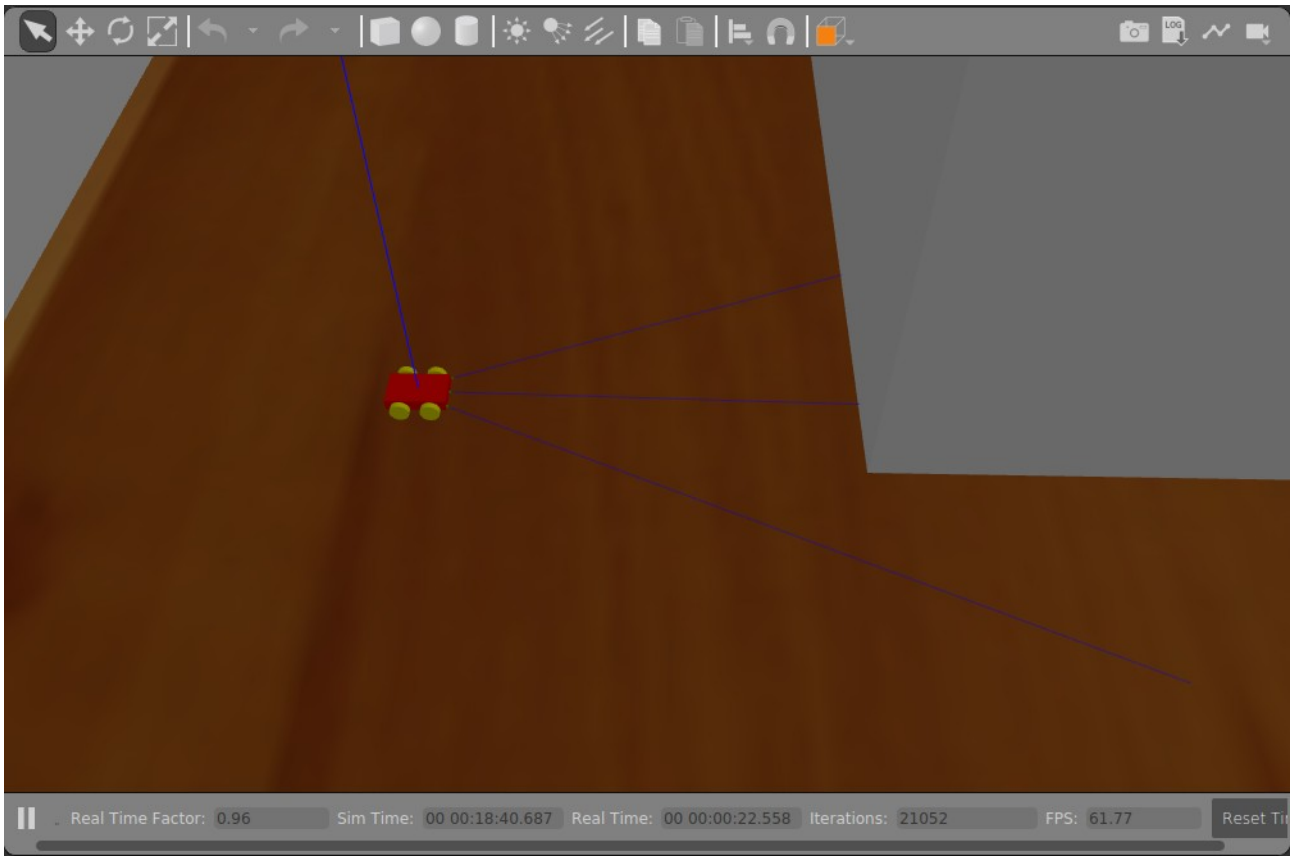

Рис. 10. Модель робота в Gazebo simulation

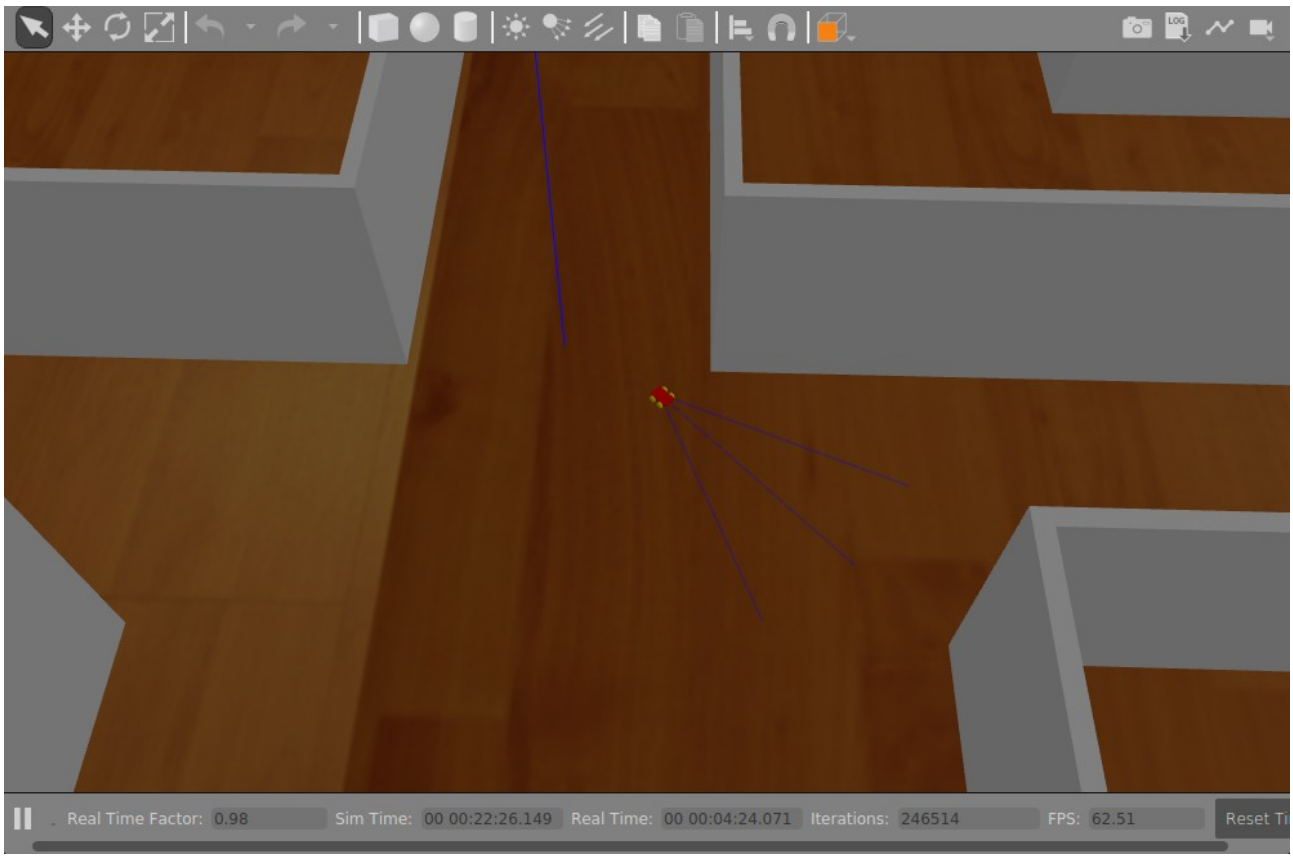

Рис. 11. Перемещение модели робота в Gazebo simulation

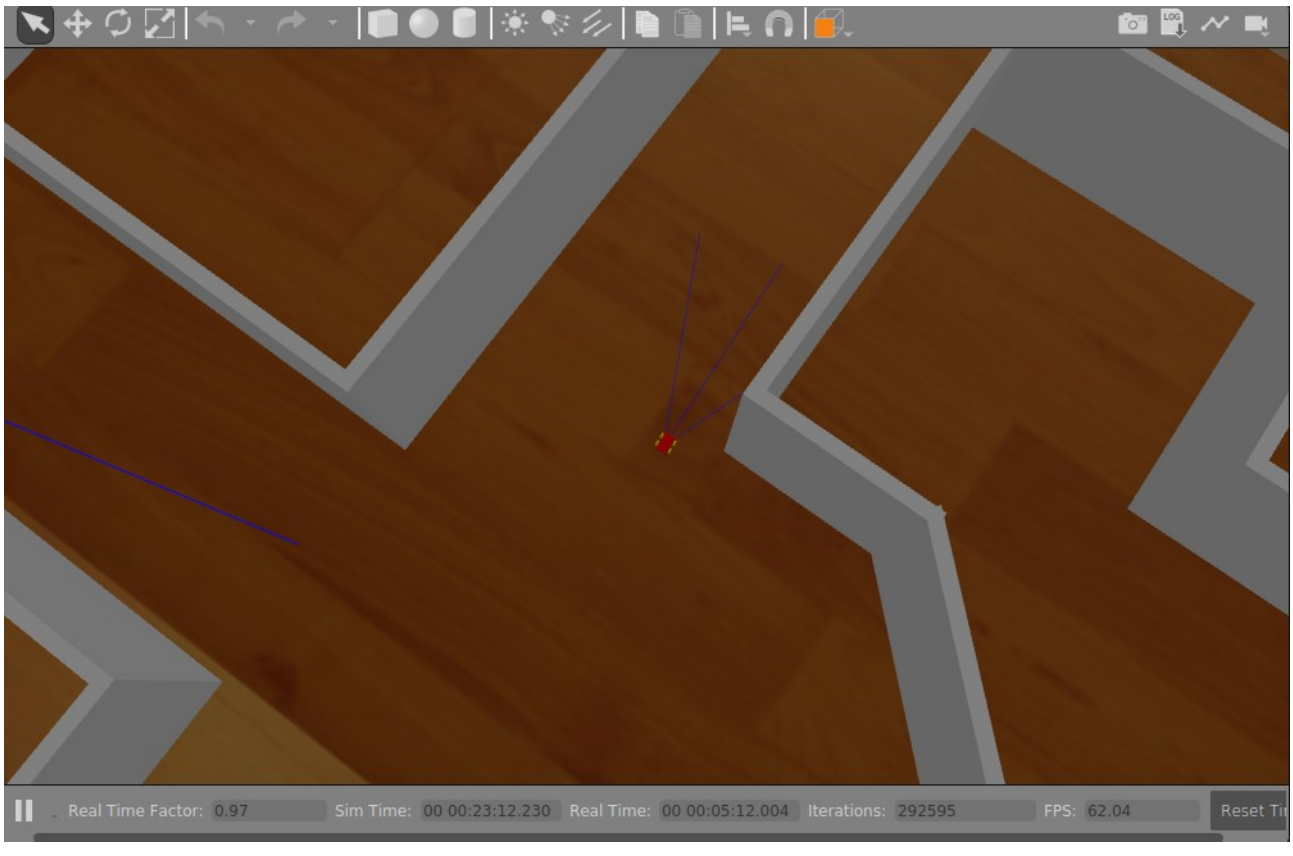

Рис. 12. Перемещение модели робота в Gazebo simulation

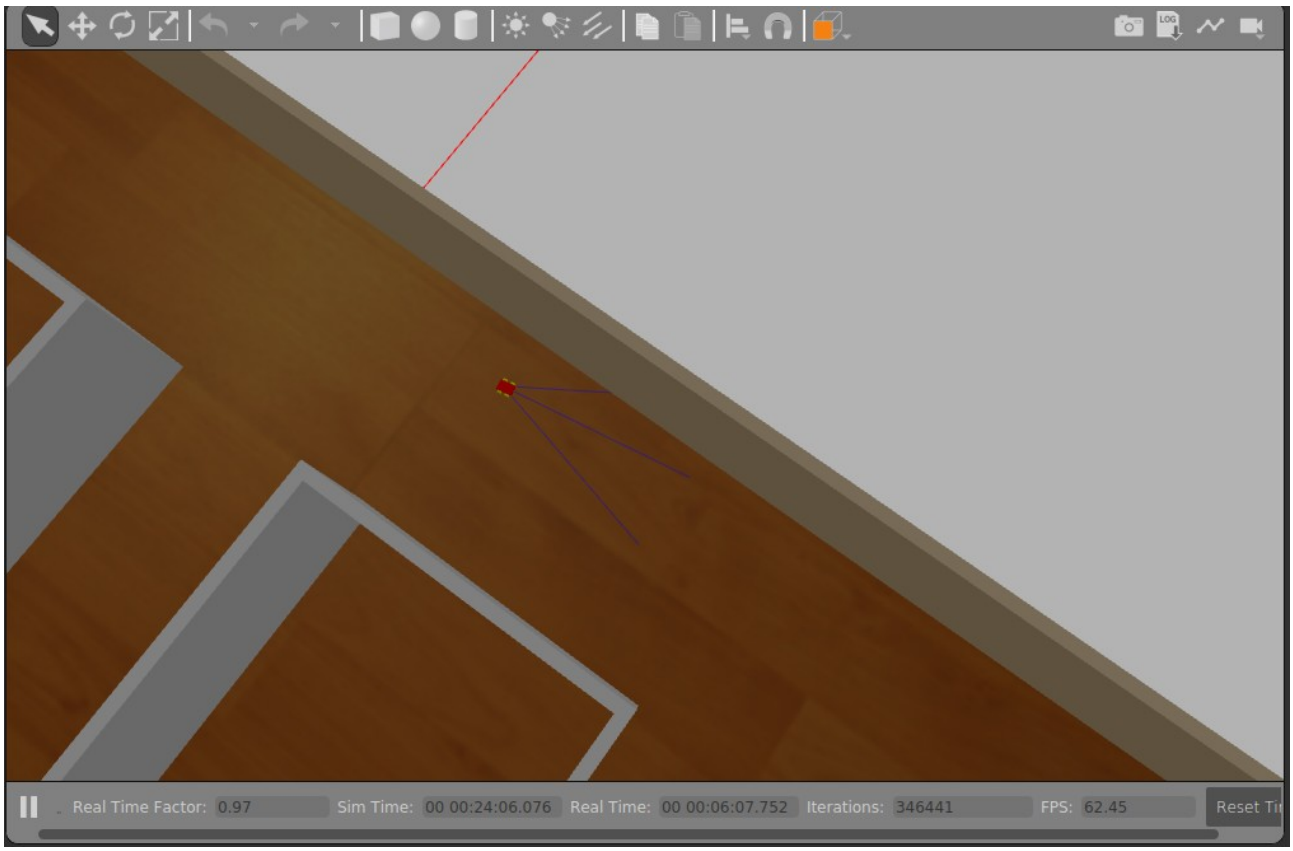

Рис. 13. Перемещение модели робота в Gazebo simulation

Вывод:

В ходе данной лабораторной работе была изучена операционная система для роботов — ROS. Был спроектирован мобильный четырехколесный робот с помощью описания в формате URDF и смоделирована симуляция данного робота в среде Gazebo и rviz. Также, были изучены основы работы с узлами в ROS, рассмотрены понятие Listener и Publisher для прослушивания и публикации данных между узлами. С помощью языка программирования Python было реализовано программное обеспечение для автономного управления робота по инфракрасным датчикам и протестировано в среде Gazebo, таким образом, что робот в данной работе успешно обходит препятствия, исходя из данных с датчиков.