Lab 1

Exercise 0:

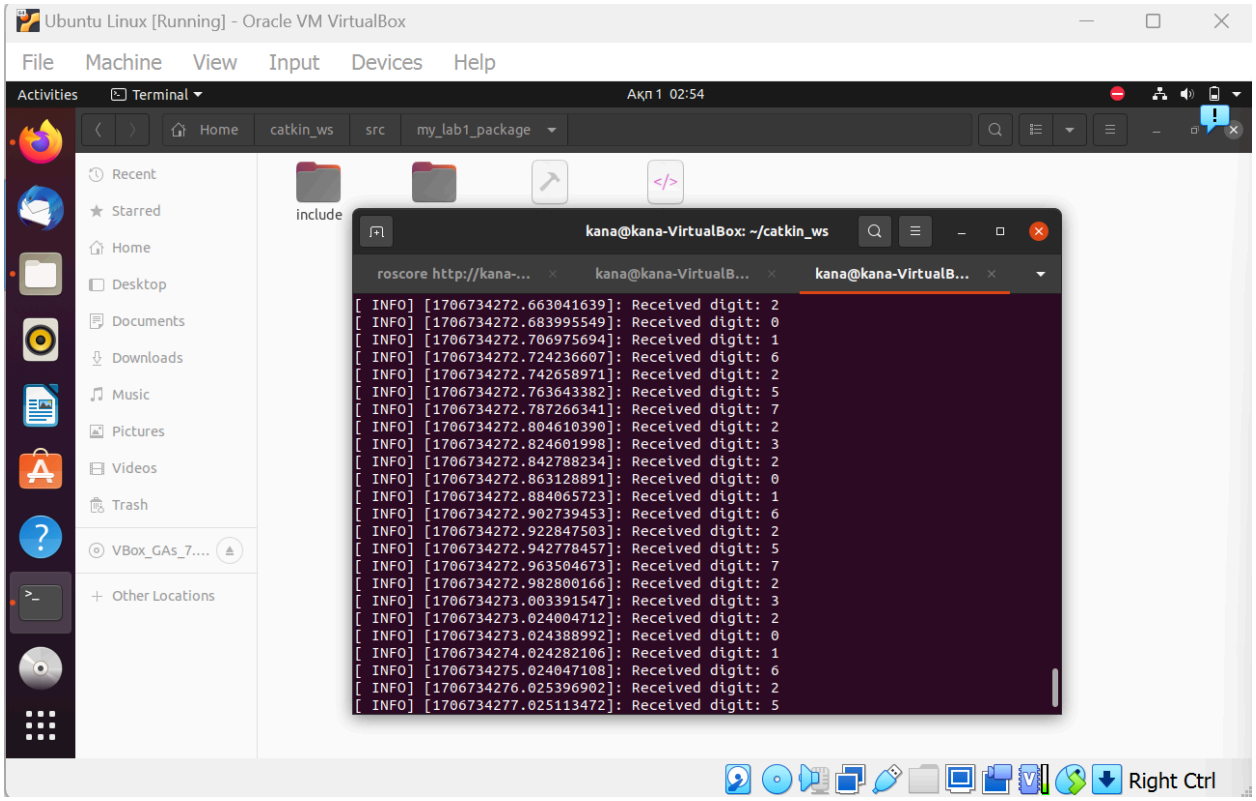
publisher_node.cpp:

```
#include "ros/ros.h"
#include "std_msgs/Int32.h"
#include <sstream>

class MyPublisher {
public:
    MyPublisher() {
        pub = n.advertise<std_msgs::Int32>("/Kantoreyeva_topic", 9);
        rate_1hz = new ros::Rate(1);
        rate_50hz = new ros::Rate(50);
    }

    void publishNumbers() {
        std::vector<int> idNumbers = {2,0,1,6,2,5,7,2,3};
        while (ros::ok()) {
            for (const auto &digit : idNumbers) {
                std_msgs::Int32 msg;
                msg.data = digit;
                pub.publish(msg);
                rate_1hz->sleep();
            }

            for (int i = 0; i < 50; ++i) {
                for (const auto &digit : idNumbers) {
                    std_msgs::Int32 msg;
                    msg.data = digit;
                    pub.publish(msg);
                    rate_50hz->sleep();
                }
            }
        }
    }

private:
    ros::NodeHandle n;
    ros::Publisher pub;
    ros::Rate *rate_1hz;
    ros::Rate *rate_50hz;
};

int main(int argc, char **argv) {
    ros::init(argc, argv, "my_publisher_node");
    MyPublisher myPublisher;
    myPublisher.publishNumbers();

    return 0;
}
```

subscriber_node.cpp:

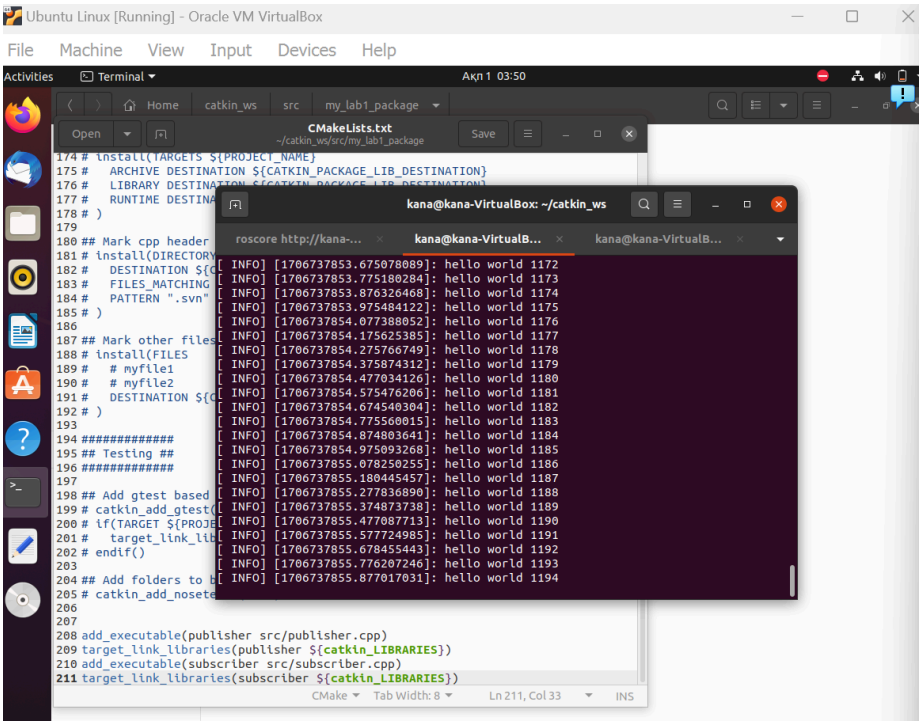
```
#include "ros/ros.h"
#include "std_msgs/Int32.h"

void callback(const std_msgs::Int32::ConstPtr &msg) {
    ROS_INFO("Received digit: %d", msg->data);
}

int main(int argc, char **argv) {
    ros::init(argc, argv, "my_subscriber_node");
    ros::NodeHandle n;
    ros::Subscriber sub = n.subscribe("/Kantoreyeva_topic", 9, callback);
    ros::spin();

    return 0;
}
```

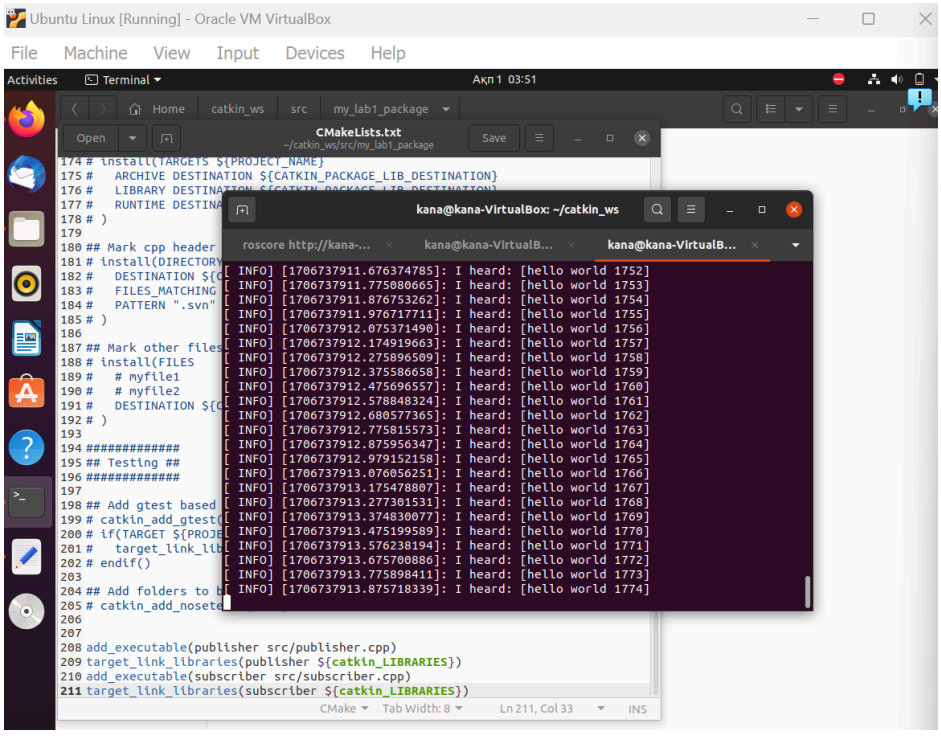
Screenshot:



Exercise 1:

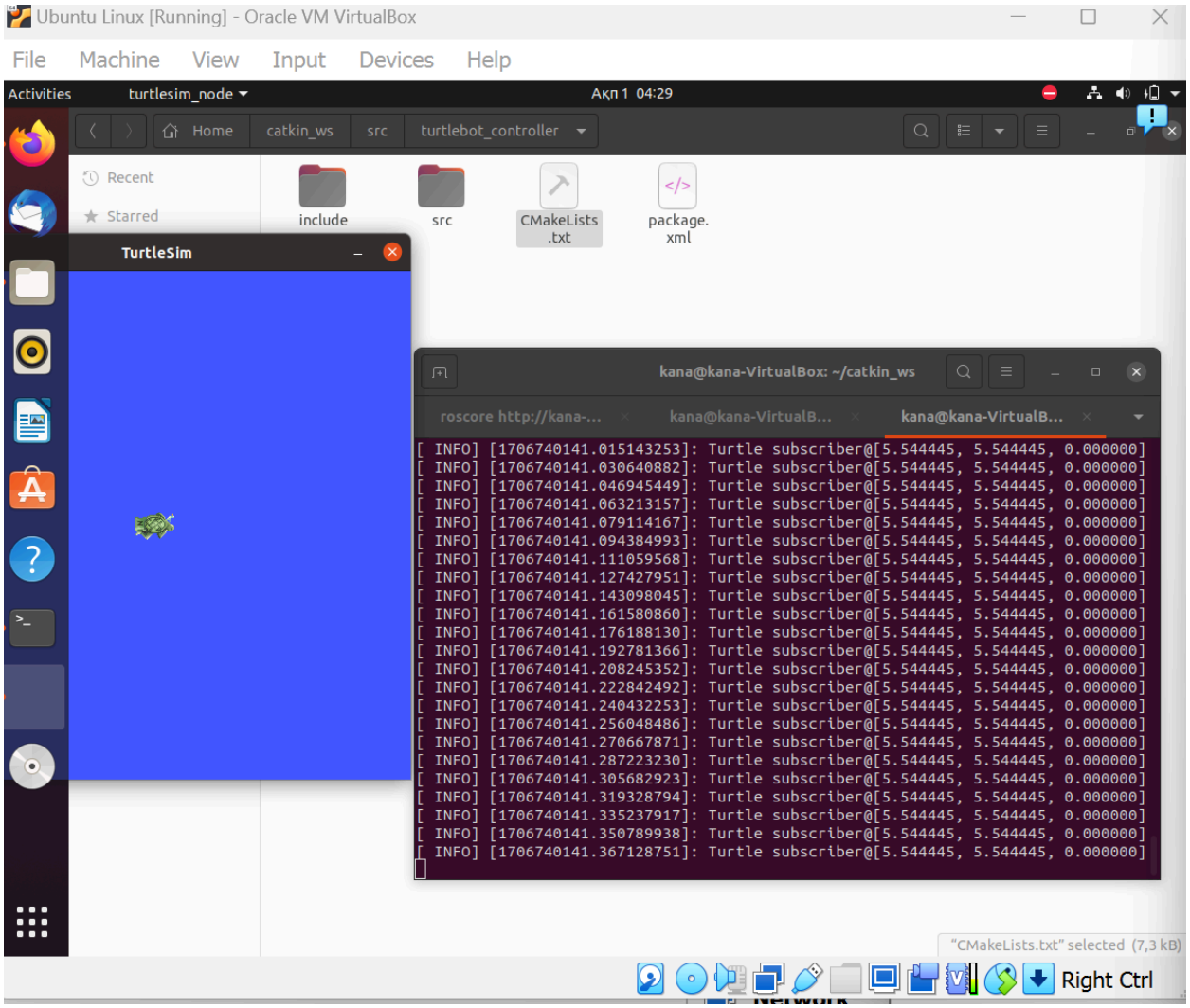
Screenshots:





codes were provided

Exercise 2:
Screenshots:



codes were provided

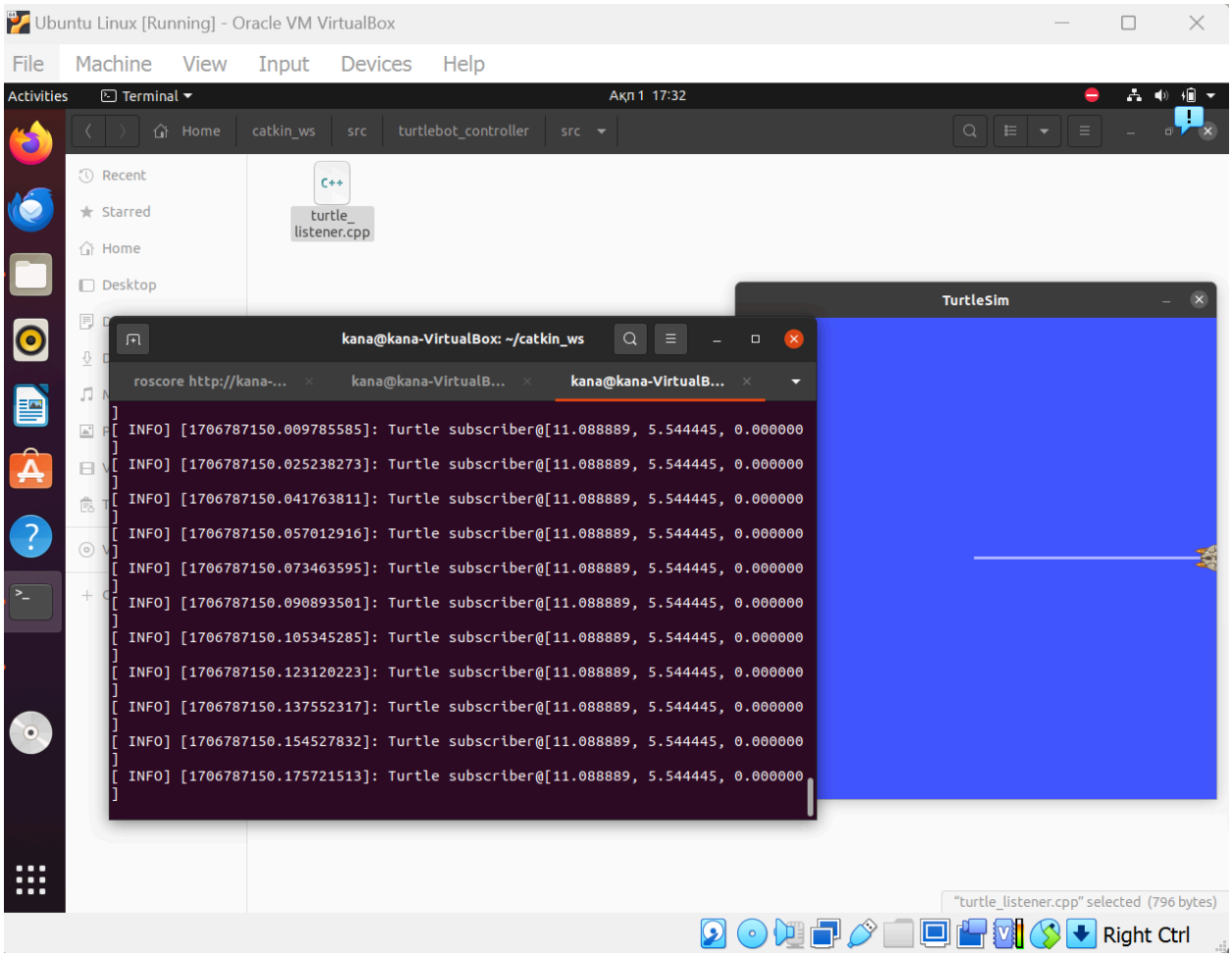
Exercise 3:

turtle_listener.cpp:

```
#include "ros/ros.h"
#include "turtlesim/Pose.h"
#include "geometry_msgs/Twist.h"

ros::Publisher pub;

void turtleCallback(const turtlesim::Pose::ConstPtr& msg)
{
   ROS_INFO("Turtle subscriber @ [%f, %f, %f]", msg->x, msg->y, msg->theta);
```

```
    geometry_msgs::Twist my_vel;
    my_vel.linear.x = 1.0;
    my_vel.angular.z = 1.0;
    pub.publish(my_vel);
}

int main(int argc, char** argv)
{
    // Initialize the node, setup the NodeHandle for handling the communication with the ROS system
    ros::init(argc, argv, "turtlebot_subscriber");
    ros::NodeHandle nh;

    pub = nh.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1);

    // Define the subscriber to turtle's position
    ros::Subscriber sub = nh.subscribe("turtle1/pose", 1, turtleCallback);

    ros::spin();
    return 0;
}
```

Screenshot:

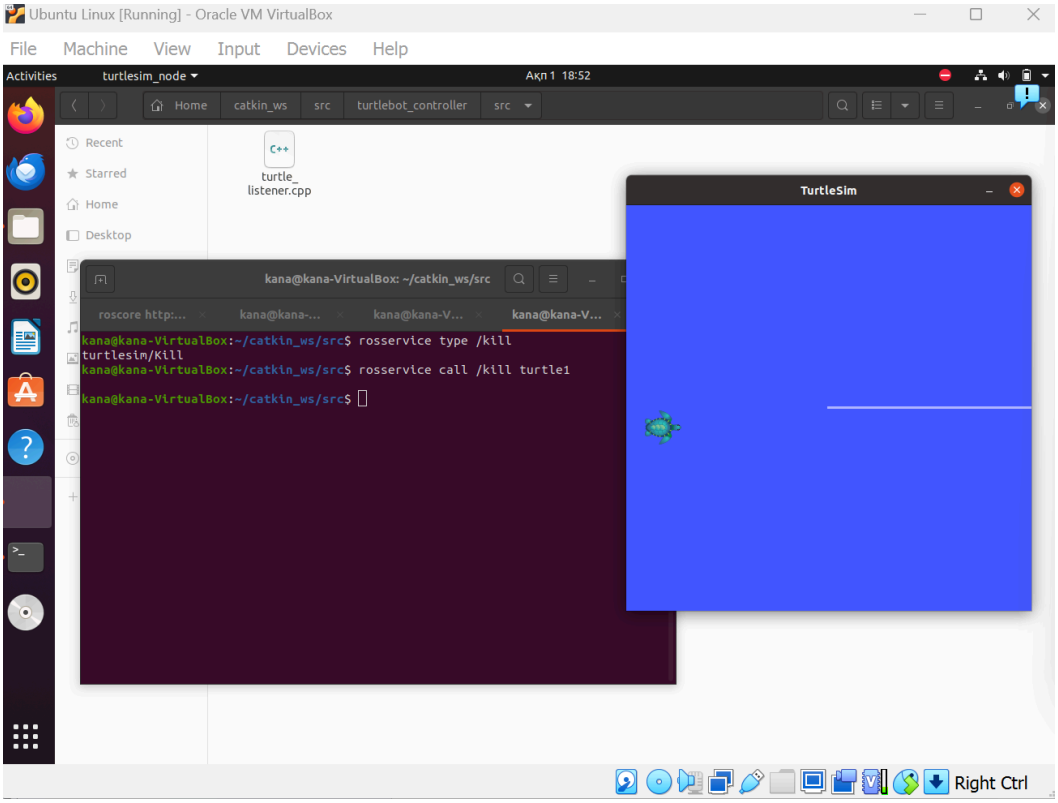


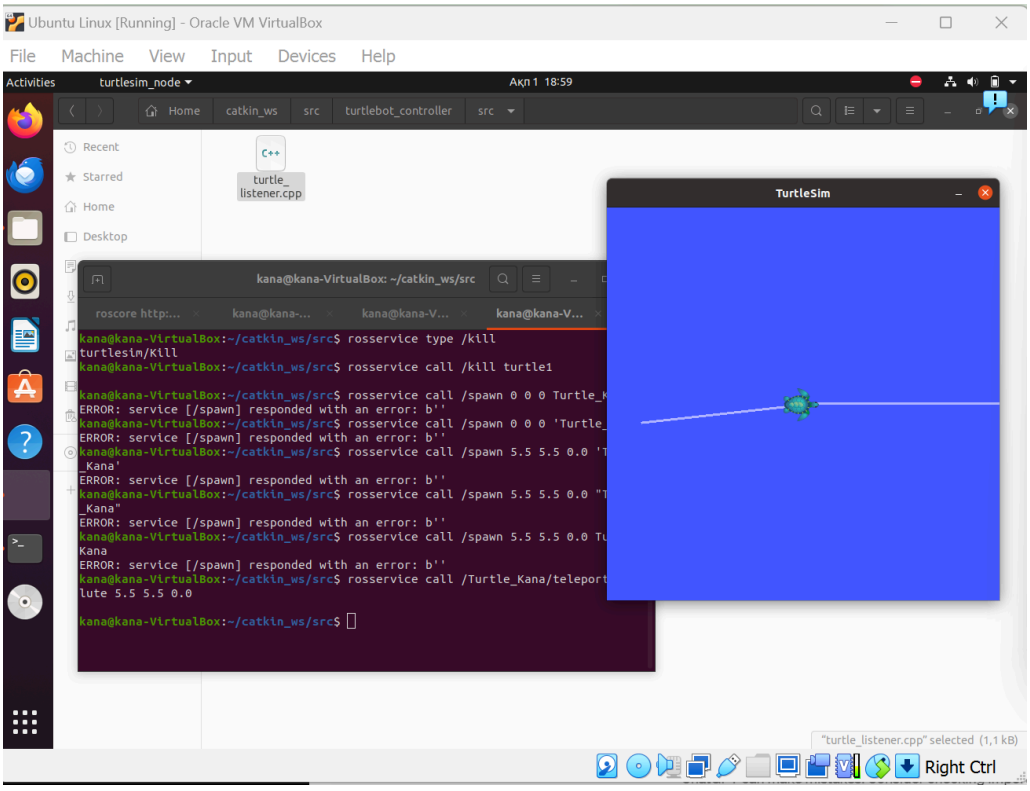
Exercise 4:

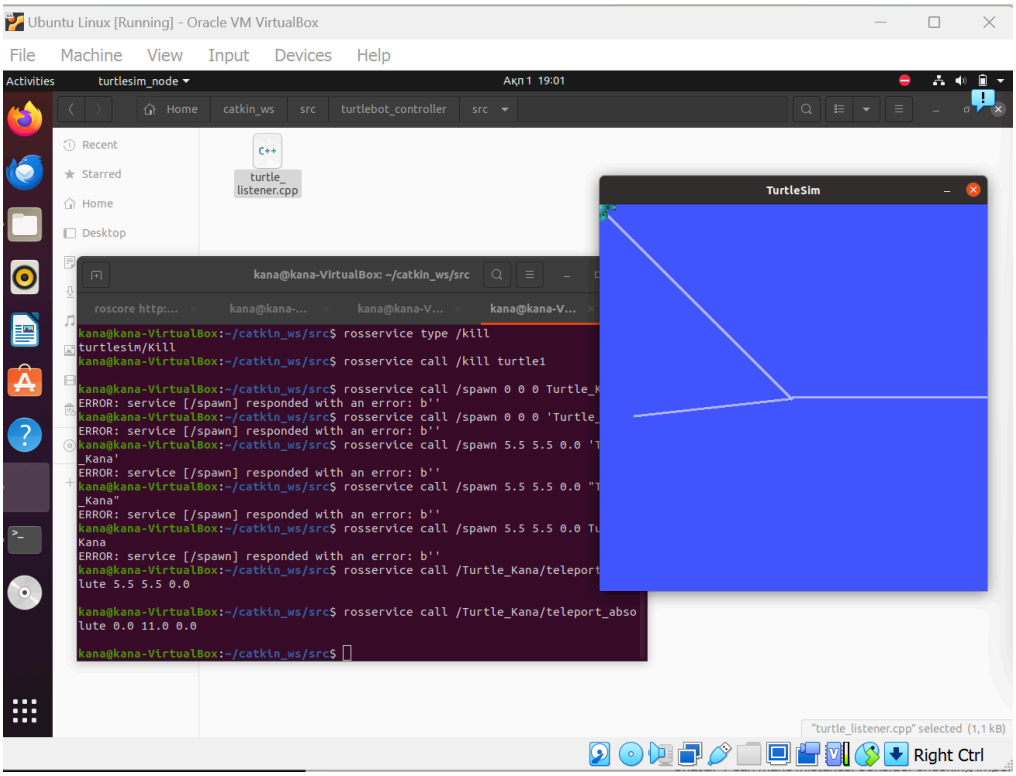
turtle_listener.cpp:

```
#include "ros/ros.h"
#include "turtlesim/Pose.h"
#include "geometry_msgs/Twist.h"
#include "turtlesim/Spawn.h"

ros::Publisher pub;
ros::ServiceClient client1;

void turtleCallback(const turtlesim::Pose::ConstPtr& msg)
{
  ROS_INFO("Turtle subscriber @ [%f, %f, %f]", msg->x, msg->y, msg->theta);

  geometry_msgs::Twist my_vel;
  my_vel.linear.x = 1.0;
  my_vel.angular.z = 1.0;
  pub.publish(my_vel);
}

int main(int argc, char** argv)
{
  // Initialize the node, setup the NodeHandle for handling the communication with the ROS system
  ros::init(argc, argv, "turtlebot_subscriber");
  ros::NodeHandle nh;

  pub = nh.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1);

  ros::Subscriber sub = nh.subscribe("turtle1/pose", 1, turtleCallback);

  client1 = nh.serviceClient<turtlesim::Spawn>("/spawn");

  turtlesim::Spawn srv1;
  srv1.request.x = 1.0;
  srv1.request.y = 5.0;
  srv1.request.theta = 0.0;
  srv1.request.name = "Turtle_Kana";

  client1.call(srv1);

  ros::spin();
  return 0;
}
```

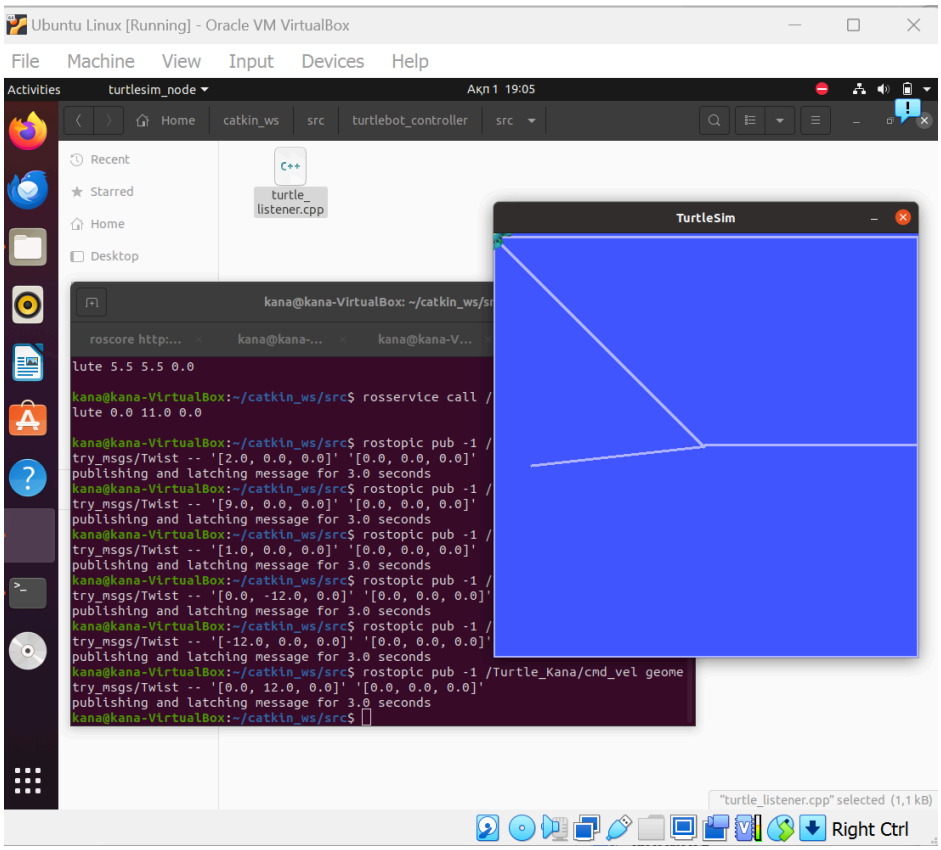
Exercise 5:
Screenshots:







Square trajectory:



Triangular trajectory: