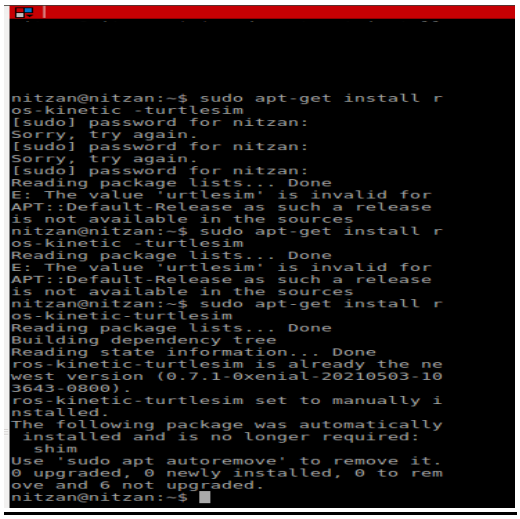
Ros Assignment

Members: Guy Farjon 201484037,
Amit Cohen 322330010 And Nitzan Levy 322572488

Note: we did the assignment with ros1.

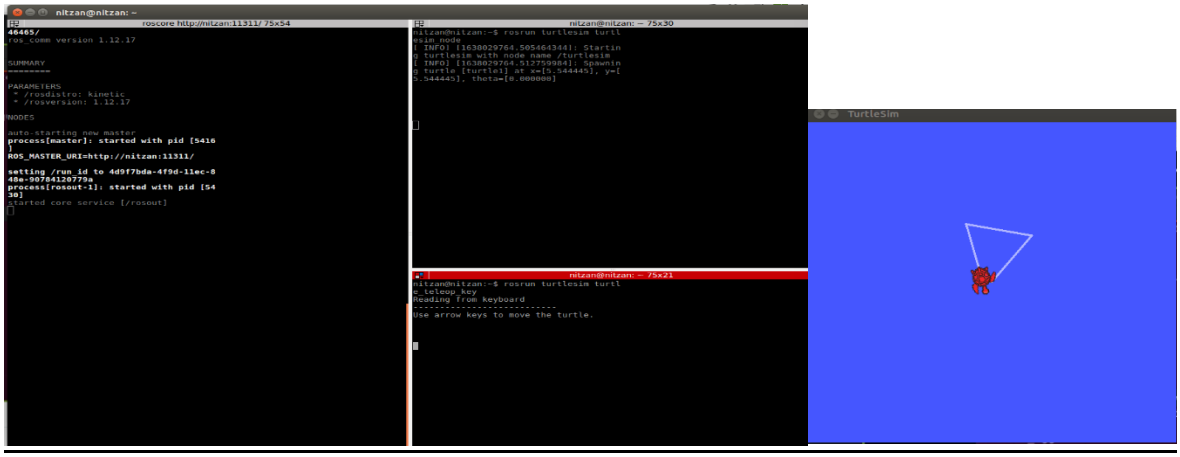
Task 1

This screenshot shows how to install the turtlesim package.



rosrun is the command for running a node in ros, Here we have run the turtlesim node

First, we need to run the roscore, which is the main node through with all the other nodes are communicating (in ros1). Then, we run the turtlesim node and the the turtlesim teleop node. Now we have a shell through which we can controll the turtle with our keyboard.



Task 2

We defined a pose callback which updates the x,y,yaw on every movement of the turtle. We did this in order to track how much distance the turtle has moved, when we have passed the distance to move, the movement stops. We chose a rate of 10ghz

(10 times in a sec) because if fits nicely and the robot is not passing to much distance after the threshold while not checking too often if it did.

```python
x=0
y=0
z=0
yaw=0

def poseCallback(pose_message):
    global x
    global y, z, yaw
    x= pose_message.x
    y= pose_message.y
    yaw = pose_message.theta

def move():
    velocity_message = Twist()
    x0=x
    y0=y
    velocity_message.linear.x = 1.0
    velocity_message.angular.z = np.random.rand() *3
    distance_moved = 0.0
    loop_rate = rospy.Rate(10)
    cmd_vel_topic='/turtle1/cmd_vel'
    velocity_publisher = rospy.Publisher(cmd_vel_topic, Twist, queue_size=10)

    while True :
            rospy.loginfo("Turtlesim moves forwards")
            velocity_publisher.publish(velocity_message)

            loop_rate.sleep()
            distance_moved = distance_moved+abs(0.5 * math.sqrt(((x-x0) ** 2) + ((y-y0) ** 2)))
            if  not (distance_moved<5.0):
                rospy.loginfo("reached")
                break

    velocity_message.linear.x =0
    velocity_publisher.publish(velocity_message)

if __name__ == '__main__':
    rospy.init_node('turtlesim_random_move')
    position_topic = "/turtle1/pose"
    pose_subscriber = rospy.Subscriber(position_topic, Pose, poseCallback)
    while True:
        time.sleep(3)
        print('move: ')
        move()
```

Task 3

Option 1:

a) Again, we defined the same pose callback for the same purposes as in the previous task.

In order to slow down gradually as we reach the goal, we defined Klinear and Kangular, to multiply with the velocities.

This node simply calculates euclidean distance from the starting position to the goal, computes the angle using arctangens, and sends velocity commands until the distance to the goal is smaller than a 0.01 threshold.

```python
x=0
y=0
z=0
yaw=0

def poseCallback(pose_message):
    global x
    global y, z, yaw
    x= pose_message.x
    y= pose_message.y
    yaw = pose_message.theta


def go_to_goal(x_goal, y_goal):
    global x
    global y, z, yaw
    cmd_vel_topic='/turtle1/cmd_vel'
    velocity_publisher = rospy.Publisher(cmd_vel_topic, Twist, queue_size=10)

    velocity_message = Twist()
    cmd_vel_topic='/turtle1/cmd_vel'

    while (True):
        K_linear = 0.5
        distance = abs(math.sqrt(((x_goal-x) ** 2) + ((y_goal-y) ** 2)))

        linear_speed = distance * K_linear

        K_angular = 4.0
        desired_angle_goal = math.atan2(y_goal-y, x_goal-x)
        angular_speed = (desired_angle_goal-yaw)*K_angular

        velocity_message.linear.x = linear_speed
        velocity_message.angular.z = angular_speed

        velocity_publisher.publish(velocity_message)


        if (distance <0.01):
            break


if __name__ == '__main__':
    try:
        rospy.init_node('turtlesim_motion_pose', anonymous=True)
        position_topic = "/turtle1/pose"
        pose_subscriber = rospy.Subscriber(position_topic, Pose, poseCallback)
        go_to_goal(np.random.rand()*11,np.random.rand()*11)

    except rospy.ROSInterruptException:
        rospy.loginfo("node terminated.")
```