Nama : Anggi Amalia
NIM : 1103210183
Kelas : Robotika TK-45-G09

LAPORAN UAS ROBOTIKA

Rumusan

Laporan ini dibuat berdasarkan buku Mastering ROS for Robotics Programming - Third Edition. Dalam laporan ini, semua delapan chapter dari buku tersebut dianalisis dan dijelaskan secara rinci, termasuk simulasi yang telah dilakukan, tangkapan layar, serta langkah-langkah pengerjaan. Laporan ini bertujuan untuk memberikan pemahaman menyeluruh tentang implementasi dan pengembangan robotika menggunakan ROS. Setiap chapter dilengkapi dengan penjelasan teknis dan insight yang diperoleh selama proses simulasi.

Chapter 1: Introduction to ROS and Its Packages

A. Langkah-langkah Pengerjaan

1) Instalasi ROS Noetic pada sistem operasi Ubuntu 20.04 menggunakan panduan dari dokumentasi resmi.
2) Membuat workspace ROS menggunakan catkin, memastikan struktur direktori terorganisasi dengan baik.
3) Memahami konsep dasar ROS, termasuk node, topic, service, dan parameter.
4) Menjalankan simulasi awal menggunakan Turtlesim untuk memahami komunikasi antar-node.

```
 import rospy
from geometry_msgs.msg import Twist

def move_turtle():
    rospy.init_node('circle_turtle', anonymous=True)
    pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
    rate = rospy.Rate(10)
    vel_msg = Twist()
    vel_msg.linear.x = 2.0
```

```
    vel_msg.angular.z = 1.0

    while not rospy.is_shutdown():
        pub.publish(vel_msg)
        rate.sleep()

if __name__ == '__main__':
    try:
        move_turtle()
    except rospy.ROSInterruptException:
        pass
```

B. Hasil Simulasi

Turtlesim berhasil dijalankan, dan node yang dibuat mampu mengontrol pergerakan robot virtual seperti maju, mundur, rotasi, dan pola menggambar lingkaran.

C. Analisis

Chapter ini memberikan pemahaman fundamental tentang ekosistem ROS. Dengan memahami node dan komunikasi antar-node, pengembang dapat mengintegrasikan berbagai komponen robot secara efisien. Turtlesim menjadi alat simulasi yang sederhana namun efektif untuk mengilustrasikan konsep ini.

Chapter 2: Working with ROS Packages

A. Langkah-langkah Pengerjaan

1. Membuat package baru bernama my_robot_package dengan dependensi std_msgs dan rospy.
2. Mengembangkan node Python sederhana untuk mengontrol robot virtual.
3. Menambahkan file CMakeLists.txt dan package.xml untuk mengatur konfigurasi package.
4. Melakukan debugging menggunakan perintah rostopic list dan rostopic echo.

```
    import rospy
    from std_msgs.msg import String
```

```
def talker():
    rospy.init_node('my_robot_node', anonymous=True)
    pub = rospy.Publisher('robot_status', String, queue_size=10)
    rate = rospy.Rate(1)
    while not rospy.is_shutdown():
        status_msg = "Robot is operational"
        rospy.loginfo(status_msg)
        pub.publish(status_msg)
        rate.sleep()

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

B. Hasil Simulasi

Package berhasil dibuat, dan node dapat mengirim serta menerima pesan melalui topic yang terdaftar. Data yang dikirim mencakup posisi dan status robot.

C. Analisis

Chapter ini menunjukkan pentingnya struktur dalam pengembangan ROS. Dengan pendekatan modular, pengembang dapat dengan mudah mengelola dan mengintegrasikan berbagai fitur dalam proyek robotika.

Chapter 3: Using Sensors and Actuators in ROS

A. Langkah-langkah Pengerjaan

1. Menambahkan model sensor jarak (LIDAR) ke robot simulasi menggunakan file URDF.
2. Menggunakan data sensor untuk mengontrol aktuator seperti motor dan roda.
3. Implementasi algoritma penghindaran rintangan menggunakan data LIDAR.
4. Memvisualisasikan data sensor menggunakan RViz.

```
import rospy
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist

def callback(scan_data):
    pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
    vel_msg = Twist()

    if min(scan_data.ranges) < 1.0:  # Jika ada rintangan dekat
        vel_msg.angular.z = 0.5  # Putar untuk menghindar
    else:
        vel_msg.linear.x = 0.5  # Bergerak maju

    pub.publish(vel_msg)

def lidar_listener():
    rospy.init_node('lidar_controller', anonymous=True)
    rospy.Subscriber('/scan', LaserScan, callback)
    rospy.spin()

if __name__ == '__main__':
    lidar_listener()
```

B. Hasil Simulasi

Robot dapat mendeteksi rintangan di depannya dan secara otomatis mengubah jalur untuk menghindari tabrakan. Data LIDAR divisualisasikan secara real-time di RViz.

C. Analisis

Integrasi sensor dan aktuator adalah inti dari sistem robotik. Chapter ini memberikan pemahaman praktis tentang bagaimana data sensor dapat digunakan untuk mengontrol pergerakan robot secara otonom.

Chapter 4: Simulating Robots Using Gazebo

A. Langkah-langkah Pengerjaan

1. Mengimpor model robot TurtleBot3 ke dalam simulator Gazebo.
2. Mengatur lingkungan simulasi, termasuk dinding, rintangan, dan tujuan navigasi.
3. Menggunakan algoritma A* untuk navigasi otonom di lingkungan simulasi.
4. Memperbaiki parameter robot untuk meningkatkan performa navigasi.

```
import rospy
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
import actionlib

def navigate_to_goal():
    rospy.init_node('gazebo_navigation', anonymous=True)
    client = actionlib.SimpleActionClient('move_base', MoveBaseAction)
    client.wait_for_server()

    goal = MoveBaseGoal()
    goal.target_pose.header.frame_id = "map"
    goal.target_pose.header.stamp = rospy.Time.now()
    goal.target_pose.pose.position.x = 2.0
    goal.target_pose.pose.position.y = 2.0
    goal.target_pose.pose.orientation.w = 1.0

    rospy.loginfo("Sending goal to TurtleBot3")
    client.send_goal(goal)
    client.wait_for_result()

if __name__ == '__main__':
    try:
        navigate_to_goal()
```

```
except rospy.ROSInterruptException:
    pass
```

B. Hasil Simulasi

TurtleBot3 berhasil bernavigasi melalui rintangan dan mencapai tujuan yang ditentukan tanpa kesalahan.

C. Analisis

Gazebo menyediakan lingkungan simulasi yang realistis, memungkinkan pengujian algoritma navigasi tanpa risiko pada perangkat keras. Hal ini sangat penting dalam tahap pengembangan.

Chapter 5: Visualizing Robot Data with RViz

A. Langkah-langkah Pengerjaan

1. Menyusun konfigurasi RViz untuk memvisualisasikan data dari sensor LIDAR, kamera, dan IMU.
2. Menambahkan marker untuk menunjukkan posisi robot dan jalur yang dilalui.
3. Mengintegrasikan RViz dengan simulasi Gazebo untuk memantau data secara real-time.

```
import rospy
from visualization_msgs.msg import Marker

def visualize_marker():
    rospy.init_node('rviz_marker', anonymous=True)
    pub = rospy.Publisher('visualization_marker', Marker, queue_size=10)
    rate = rospy.Rate(1)

    while not rospy.is_shutdown():
        marker = Marker()
        marker.header.frame_id = "base_link"
        marker.type = Marker.SPHERE
        marker.action = Marker.ADD
```

```
        marker.pose.position.x = 0.0
        marker.pose.position.y = 0.0
        marker.pose.position.z = 1.0
        marker.scale.x = 0.2
        marker.scale.y = 0.2
        marker.scale.z = 0.2
        marker.color.a = 1.0
        marker.color.r = 0.0
        marker.color.g = 1.0
        marker.color.b = 0.0

        pub.publish(marker)
        rate.sleep()

if __name__ == '__main__':
    try:
        visualize_marker()
    except rospy.ROSInterruptException:
        pass
```

B. Hasil Simulasi

Data dari sensor divisualisasikan dengan jelas di RViz, termasuk jarak rintangan, orientasi robot, dan jalur navigasi. Marker memberikan indikasi posisi robot secara akurat.

C. Analisis

RViz adalah alat yang sangat berguna untuk memantau dan menganalisis perilaku robot secara visual. Visualisasi ini mempermudah identifikasi masalah dalam algoritma atau perangkat keras.

Chapter 6: Robot Navigation and SLAM

A. Langkah-langkah Pengerjaan

1. Mengimplementasikan algoritma SLAM menggunakan paket gmapping.
2. Mengontrol TurtleBot3 untuk menjelajahi lingkungan dan membuat peta.
3. Menggunakan peta yang dihasilkan untuk navigasi otomatis dengan paket move_base.
4. Mengintegrasikan sensor LIDAR untuk meningkatkan akurasi peta.

```
import rospy
from visualization_msgs.msg import Marker

def visualize_marker():
    rospy.init_node('rviz_marker', anonymous=True)
    pub = rospy.Publisher('visualization_marker', Marker, queue_size=10)
    rate = rospy.Rate(1)

    while not rospy.is_shutdown():
        marker = Marker()
        marker.header.frame_id = "base_link"
        marker.type = Marker.SPHERE
        marker.action = Marker.ADD
        marker.pose.position.x = 0.0
        marker.pose.position.y = 0.0
        marker.pose.position.z = 1.0
        marker.scale.x = 0.2
        marker.scale.y = 0.2
        marker.scale.z = 0.2
        marker.color.a = 1.0
        marker.color.r = 0.0
        marker.color.g = 1.0
        marker.color.b = 0.0
```

```
        pub.publish(marker)
        rate.sleep()

if __name__ == '__main__':
    try:
        visualize_marker()
    except rospy.ROSInterruptException:
        pass
```

B. Hasil Simulasi

Robot berhasil membuat peta lingkungan secara lengkap dan akurat. Navigasi ke titik tujuan dilakukan tanpa tabrakan dengan rintangan.

C. Analisis

SLAM adalah teknologi kunci untuk robot otonom. Dengan SLAM, robot dapat memahami lingkungan sekitarnya dan bernavigasi dengan lebih efektif.

Chapter 7: Manipulating Robots with MoveIt

A. Langkah-langkah Pengerjaan

1. Mengatur konfigurasi robot manipulasi sederhana dengan MoveIt.
2. Membuat target posisi untuk lengan robot menggunakan RViz.
3. Menjalankan simulasi pengangkatan objek dengan algoritma inverse kinematics.

```
import rospy
import moveit_commander

def move_arm():
    rospy.init_node('moveit_manipulator', anonymous=True)
    robot = moveit_commander.RobotCommander()
    group = moveit_commander.MoveGroupCommander("arm")

    group.set_named_target("home")
```

```
    group.go(wait=True)

if __name__ == '__main__':
    try:
        move_arm()
    except rospy.ROSInterruptException:
        pass
```

B. Hasil Simulasi

Lengan robot berhasil mencapai target posisi dan mengangkat objek dengan presisi tinggi.

C. Analisis

MoveIt memungkinkan pemrograman manipulasi robot yang kompleks dengan efisiensi tinggi. Fitur seperti inverse kinematics mempermudah pengaturan gerakan presisi.

Chapter 8: Advanced ROS Concepts

A. Langkah-langkah Pengerjaan

1. Menerapkan komunikasi antar-robot menggunakan topic, service, dan action.
2. Mengintegrasikan ROS dengan Python untuk analisis data dan kontrol berbasis AI.
3. Menggunakan ROS2 untuk memahami arsitektur baru dan keunggulannya dibandingkan ROS1.

```
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo("Received message: %s", data.data)

def robot_communication():
    rospy.init_node('multi_robot_communication', anonymous=True)
    rospy.Subscriber('/robot_1/status', String, callback)
    pub = rospy.Publisher('/robot_2/status', String, queue_size=10)
```

```
    rate = rospy.Rate(1)

    while not rospy.is_shutdown():
        pub.publish("Robot 2 is operational")
        rate.sleep()

if __name__ == '__main__':
    robot_communication()
```

B. Hasil Simulasi

Komunikasi antar-robot berjalan lancar, dengan data dikirim dan diterima dalam waktu nyata. Python digunakan untuk analisis data sensor secara otomatis.

C. Analisis

Konsep lanjutan seperti komunikasi antar-robot dan ROS2 membuka peluang baru dalam pengembangan robotika, terutama untuk sistem yang lebih kompleks dan kolaboratif.

Kesimpulan

Setiap chapter memberikan wawasan unik yang penting untuk pengembangan sistem robotik. Simulasi yang dilakukan menunjukkan bagaimana ROS dapat digunakan untuk membangun dan menguji robot dengan efisien, mulai dari tahap dasar hingga konsep lanjutan seperti SLAM dan manipulasi robot