TECHNICAL REPORT MATA KULIAH ROBOTIKA DAN SISTEM CERDAS

HACKING “ROBOTIC PROGRAMMING” BOOK

Disusun oleh :

Nama	: Alvin Yoga Nugraha
Kelas	: TK-44-04
NIM	: 1103204159

JURUSAN TEKNIK KOMPUTER

FAKULTAS TEKNIK ELEKTRO

UNIVERSITAS TELKOM

PENDAHULUAN

Laporan teknis ini menggambarkan konsep dan implementasi yang dibahas dalam buku "A Systematic Approach to Learning Robot Programming with ROS". Laporan ini mencakup bab-bab kunci dan mencakup contoh kode untuk setiap bab yang menunjukkan aplikasi praktis menggunakan Sistem Operasi Robot (ROS). Dengan mengikuti contoh kode dan petunjuk yang disediakan, pembaca dapat memperoleh pengalaman praktis dan pemahaman tentang berbagai alat, node ROS, pesan, simulasi, transkrip, dan robot operasi sistem.

ANALISIS

Chapter 1 : Introduction to ROS : ROS tools and nodes

Memperkenalkan konsep dasar ROS dan membahas alat dan node ROS. ROS menyediakan kerangka kerja untuk menulis perangkat lunak untuk robot, dan node adalah unit dasar perangkat lunak dalam ROS.

```
#!/usr/bin/env python

import rospy

def my_node():
    # Inisialisasi node dengan nama "my_node"
    rospy.init_node('my_node', anonymous=True)

    # Loop utama node
    rate = rospy.Rate(10)  # 10 Hz
    while not rospy.is_shutdown():
        # Tulis log pesan
        rospy.loginfo("Hello, ROS!")

        # Tunggu selama 0.1 detik
        rate.sleep()

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

1. Import modul **rospy** yang diperlukan untuk bekerja dengan ROS menggunakan Python.
2. Buat fungsi **my_node** yang akan menjadi fungsi utama dari node kita.
3. Panggil **rospy.init_node()** untuk menginisialisasi node dengan nama "my_node". Parameter **anonymous=True** menunjukkan bahwa jika ada node lain dengan nama yang sama, node kita akan tetap berjalan tanpa mempengaruhi node lainnya.

4. Buat objek **rate** dengan kecepatan 10 Hz, yang akan digunakan untuk mengatur kecepatan loop utama node.
5. Dalam loop utama, gunakan **rospy.loginfo()** untuk menulis pesan ke log ROS.
6. Tunggu selama 0.1 detik menggunakan **rate.sleep()** untuk mencapai kecepatan yang diinginkan.
7. Gunakan **if __name__ == '__main__':** untuk memastikan bahwa kode hanya dijalankan jika file dieksekusi langsung, bukan diimpor sebagai modul.
8. Panggil **my_node()** di dalam blok **try-except** untuk memulai node kita.
9. Tangani **rospy.ROSInterruptException** agar program dapat dihentikan dengan benar jika ada sinyal **Ctrl+C** yang diterima.

Chapter 2 : Messages, Classes and Servers

Fokus pada pesan, kelas, dan server ROS. Pesan ROS digunakan untuk komunikasi antara node, dan tipe pesan kustom dapat didefinisikan menggunakan kelas pesan.

```
#!/usr/bin/env python

import rospy
from std_msgs.msg import String
from my_robot_msgs.srv import AddTwoInts

class ExampleNode:
    def __init__(self):
        rospy.init_node('example_node')
        rospy.Subscriber('input_topic', String, self.callback)
        self.result = 0
        self.add_two_ints_service = rospy.Service('add_two_ints', AddTwoInts,
self.handle_add_two_ints)

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

    def handle_add_two_ints(self, req):
        self.result = req.a + req.b
        rospy.loginfo("Adding %d and %d", req.a, req.b)
        return self.result

    def run(self):
        rate = rospy.Rate(10)
        while not rospy.is_shutdown():
            rospy.loginfo("Result: %d", self.result)
            rate.sleep()

if __name__ == '__main__':
```

```
node = ExampleNode()
node.run()
```

1. Pastikan Anda memiliki file pesan (messages) dan layanan (services) yang diperlukan, sesuai dengan buku dan contoh yang diberikan. Dalam contoh di atas, kita mengimpor pesan String dari paket **std_msgs** dan layanan AddTwoInts dari paket **my_robot_msgs**. Pastikan file-file **.msg** dan **.srv** telah dibuat dan dikompilasi dengan benar menggunakan **rosmsg** dan **rossrv**.
2. Pastikan bahwa direktori yang berisi file pesan dan layanan tersebut telah ditambahkan ke **PYTHONPATH** dengan menjalankan perintah **source devel/setup.bash** dari direktori catkin workspace Anda.
3. Jalankan ROS master dengan perintah **roscore** di terminal.
4. Jalankan node Anda dengan menjalankan perintah **rosrun package_name example_node.py**, dengan mengganti **package_name** dengan nama paket ROS yang sesuai.
5. Di terminal lain, Anda dapat mempublikasikan pesan ke topik **input_topic** dengan menggunakan perintah **rostopic pub /input_topic std_msgs/String "data: 'Hello, ROS!'"**.
6. Anda juga dapat memanggil layanan **add_two_ints** dengan menggunakan perintah **rosservice call /add_two_ints "a: 3 b: 4"**.
7. Anda akan melihat pesan yang dicetak di terminal sesuai dengan fungsionalitas yang ditentukan dalam kode. Misalnya, pesan yang diterima dari topik **input_topic** akan dicetak dalam metode **callback**, dan hasil penjumlahan dari layanan **add_two_ints** akan dicetak di dalam metode **handle_add_two_ints**.

Pastikan Anda telah mengganti **package_name** dengan nama paket ROS yang sesuai dan telah menyesuaikan file pesan dan layanan yang digunakan dengan benar.

Semoga contoh kode ini membantu Anda memahami cara bekerja dengan pesan, kelas, dan server dalam ROS, sesuai dengan materi yang dibahas dalam Chapter 2 buku "A Systematic Approach to Learning Robot Programming with ROS".

Chapter 3 : Simulation in ROS

Menjelajahi simulasi di ROS. Simulasi memungkinkan pengujian dan pengembangan perilaku robot di lingkungan virtual sebelum diterapkan pada robot fisik.

```
#!/usr/bin/env python

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

class SimulationNode:
    def __init__(self):
```

```
        rospy.init_node('simulation_node')
        self.cmd_vel_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
        rospy.Subscriber('/scan', LaserScan, self.scan_callback)

    def scan_callback(self, scan):
        ranges = scan.ranges
        min_range = min(ranges)
        rospy.loginfo("Minimum range: %.2f", min_range)

        if min_range > 1.0:
            self.move_forward()
        else:
            self.stop()

    def move_forward(self):
        twist = Twist()
        twist.linear.x = 0.2
        twist.angular.z = 0.0
        self.cmd_vel_pub.publish(twist)

    def stop(self):
        twist = Twist()
        twist.linear.x = 0.0
        twist.angular.z = 0.0
        self.cmd_vel_pub.publish(twist)

    def run(self):
        rospy.spin()

if __name__ == '__main__':
    node = SimulationNode()
    node.run()
```

1. Pastikan Anda telah mengimpor pesan yang diperlukan dari paket **geometry_msgs** dan **sensor_msgs**. Dalam contoh di atas, kita menggunakan pesan **Twist** untuk mengontrol gerakan robot dan pesan **LaserScan** untuk membaca data sensor lidar.
2. Pastikan ROS master telah dijalankan dengan perintah **roscore** di terminal.
3. Jalankan simulator Gazebo dengan dunia dan robot yang sesuai. Misalnya, Anda dapat menjalankan simulasi TurtleBot 3 dengan perintah **roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch**.
4. Jalankan node Anda dengan menjalankan perintah **rosrun package_name simulation_node.py**, dengan mengganti **package_name** dengan nama paket ROS yang sesuai.
5. Node akan berlangganan topik **/scan** untuk membaca data sensor lidar dan melakukan tindakan berdasarkan jarak terendah yang terdeteksi. Jika jarak terendah lebih besar dari 1.0 meter, robot akan bergerak maju dengan kecepatan 0.2 m/s. Jika jarak terendah kurang dari atau sama dengan 1.0 meter, robot akan berhenti.

6. Anda dapat melihat log yang dicetak di terminal sesuai dengan logika yang ditentukan dalam kode. Misalnya, jarak terendah yang terdeteksi oleh lidar akan dicetak dalam metode **scan_callback**.

Chapter 4 : Coordinate Transforms in ROS

Membahas transformasi koordinat di ROS, yang penting untuk tugas navigasi dan manipulasi robot. ROS menyediakan alat untuk bekerja dengan sistem koordinat dan mentransformasikan antara mereka.

```
#!/usr/bin/env python

import rospy
import tf2_ros
import geometry_msgs.msg

class TransformNode:
    def __init__(self):
        rospy.init_node('transform_node')
        self.tf_buffer = tf2_ros.Buffer()
        self.tf_listener = tf2_ros.TransformListener(self.tf_buffer)
        self.transformed_point_pub = rospy.Publisher('/transformed_point',
geometry_msgs.msg.PointStamped, queue_size=10)

    def transform_point(self, point):
        try:
            # Mencari transformasi dari 'base_link' ke 'map'
            transform = self.tf_buffer.lookup_transform('map', 'base_link', rospy.Time())

            # Menerapkan transformasi pada titik
            transformed_point = tf2_ros.do_transform_point(point, transform)

            # Publikasikan titik yang sudah ditransformasi
            self.transformed_point_pub.publish(transformed_point)

        except (tf2_ros.LookupException, tf2_ros.ConnectivityException,
tf2_ros.ExtrapolationException) as e:
            rospy.logwarn(str(e))

    def run(self):
        rate = rospy.Rate(1)
        while not rospy.is_shutdown():
            # Buat objek PointStamped dengan header dan koordinat yang sesuai
            point = geometry_msgs.msg.PointStamped()
            point.header.frame_id = 'base_link'
            point.header.stamp = rospy.Time.now()
            point.point.x = 1.0
            point.point.y = 0.0
            point.point.z = 0.0
```

```
        # Panggil fungsi untuk mentransformasi titik
        self.transform_point(point)

        rate.sleep()

if __name__ == '__main__':
    node = TransformNode()
    node.run()
```

1. Pastikan Anda telah mengimpor pesan yang diperlukan dari paket **geometry_msgs** dan **tf2_ros**. Dalam contoh di atas, kita menggunakan pesan **PointStamped** untuk mewakili titik dalam kerangka koordinat yang terkait dengan header.
2. Pastikan ROS master telah dijalankan dengan perintah **roscore** di terminal.
3. Pastikan ada transformasi yang valid antara kerangka 'base_link' dan 'map' dalam sistem koordinat ROS Anda. Misalnya, Anda dapat menggunakan alat seperti **static_transform_publisher** untuk mengirimkan transformasi tetap antara kedua kerangka tersebut.
4. Jalankan node Anda dengan menjalankan perintah **rosrun package_name transform_node.py**, dengan mengganti **package_name** dengan nama paket ROS yang sesuai.
5. Node akan mengirimkan titik dalam kerangka 'base_link' setiap detik. Kemudian, node akan mencoba mentransformasikan titik tersebut ke kerangka 'map' menggunakan **tf2_ros.Buffer** dan **tf2_ros.TransformListener**. Titik yang sudah ditransformasi akan dipublikasikan ke topik '/transformed_point'.
6. Anda dapat melihat log yang dicetak di terminal untuk melihat hasil dari transformasi koordinat.

Chapter 5 : Sensing and Visualization in ROS

Fokus pada sensor dan visualisasi di ROS. ROS menyediakan berbagai antarmuka sensor dan alat visualisasi untuk memahami dan menganalisis data sensor.

```
#!/usr/bin/env python

import rospy
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Point
from visualization_msgs.msg import Marker

class SensingVisualizationNode:
    def __init__(self):
        rospy.init_node('sensing_visualization_node')
        rospy.Subscriber('/scan', LaserScan, self.scan_callback)
        self.marker_pub = rospy.Publisher('/visualization_marker', Marker, queue_size=10)
```

```
    def scan_callback(self, scan):
        ranges = scan.ranges
        min_range = min(ranges)

        # Publish marker jika jarak terdekat lebih kecil dari 1.0 meter
        if min_range < 1.0:
            marker = Marker()
            marker.header.frame_id = scan.header.frame_id
            marker.header.stamp = rospy.Time.now()
            marker.type = Marker.SPHERE
            marker.action = Marker.ADD
            marker.pose.position.x = min_range
            marker.pose.position.y = 0.0
            marker.pose.position.z = 0.0
            marker.pose.orientation.x = 0.0
            marker.pose.orientation.y = 0.0
            marker.pose.orientation.z = 0.0
            marker.pose.orientation.w = 1.0
            marker.scale.x = 0.1
            marker.scale.y = 0.1
            marker.scale.z = 0.1
            marker.color.a = 1.0
            marker.color.r = 1.0
            marker.color.g = 0.0
            marker.color.b = 0.0

            self.marker_pub.publish(marker)

    def run(self):
        rospy.spin()

if __name__ == '__main__':
    node = SensingVisualizationNode()
    node.run()
```

1. Pastikan Anda telah mengimpor pesan yang diperlukan dari paket **sensor_msgs**, **geometry_msgs**, dan **visualization_msgs**. Dalam contoh di atas, kita menggunakan pesan **LaserScan** untuk membaca data sensor lidar, pesan **Point** untuk mewakili posisi dalam ruang tiga dimensi, dan pesan **Marker** untuk visualisasi dalam RViz.

2. Pastikan ROS master telah dijalankan dengan perintah **roscore** di terminal.

3. Jalankan simulator Gazebo dengan dunia dan robot yang sesuai. Misalnya, Anda dapat menjalankan simulasi TurtleBot 3 dengan perintah **roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch**.

4. Jalankan node Anda dengan menjalankan perintah **rosrun package_name sensing_visualization_node.py**, dengan mengganti **package_name** dengan nama paket ROS yang sesuai.

5. Node akan berlangganan topik **/scan** untuk membaca data sensor lidar dan mempublikasikan tanda visual dalam bentuk marker jika jarak terdekat yang terdeteksi lebih kecil dari 1.0 meter.
6. Anda dapat melihat tanda visual yang ditampilkan di RViz dengan menambahkan tampilan **Marker** dan mengatur frame_id yang sesuai.

Chapter 6 : Using Cameras in ROS

Menjelaskan penggunaan kamera di ROS. ROS menyediakan antarmuka untuk bekerja dengan sensor kamera dan memproses data citra.

```
#!/usr/bin/env python

import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2

class CameraNode:
    def __init__(self):
        rospy.init_node('camera_node')
        self.image_pub = rospy.Publisher('/camera/image', Image, queue_size=10)
        self.cv_bridge = CvBridge()

    def capture_image(self):
        # Membaca gambar dari kamera
        image = self.read_camera()

        # Mengonversi gambar OpenCV menjadi pesan Image ROS
        image_msg = self.cv_bridge.cv2_to_imgmsg(image, encoding="bgr8")

        # Mengatur waktu timbang pesan gambar
        image_msg.header.stamp = rospy.Time.now()

        # Memublikasikan pesan gambar
        self.image_pub.publish(image_msg)

    def read_camera(self):
        # Menggunakan kode OpenCV untuk membaca gambar dari kamera
        # Di sini, kita menggunakan gambar statis untuk tujuan contoh
        image = cv2.imread('path_to_image.jpg')

        return image

    def run(self):
        rate = rospy.Rate(1)
        while not rospy.is_shutdown():
            self.capture_image()
```

```
            rate.sleep()

if __name__ == '__main__':
    node = CameraNode()
    node.run()
```

1. Pastikan Anda telah mengimpor pesan yang diperlukan dari paket **sensor_msgs** dan modul **CvBridge** dari paket **cv_bridge**. Dalam contoh di atas, kita menggunakan pesan **Image** untuk mengirim gambar dari kamera dan modul **CvBridge** untuk mengonversi gambar OpenCV menjadi pesan Image ROS.
2. Pastikan ROS master telah dijalankan dengan perintah **roscore** di terminal.
3. Pastikan kamera yang sesuai telah terhubung atau disimulasikan dalam sistem ROS Anda.
4. Gantilah **path_to_image.jpg** dengan jalur lengkap ke gambar yang ingin Anda gunakan untuk tujuan contoh. Pastikan gambar tersebut dapat diakses oleh kode.
5. Jalankan node Anda dengan menjalankan perintah **rosrun package_name camera_node.py**, dengan mengganti **package_name** dengan nama paket ROS yang sesuai.
6. Node akan mengambil gambar dari kamera (atau gambar statis dalam contoh ini) menggunakan fungsi **read_camera()**. Gambar tersebut kemudian dikonversi menjadi pesan Image ROS menggunakan **cv_bridge** dan dipublikasikan ke topik **/camera/image** dengan menggunakan **image_pub**.
7. Anda dapat melihat gambar yang dipublikasikan di topik menggunakan alat visualisasi ROS seperti RViz atau dengan memeriksa topik menggunakan perintah **rostopic echo /camera/image**.

Chapter 7 : Depth Imaging and Point Clouds

Membahas pencitraan kedalaman dan awan titik. ROS menyediakan alat untuk bekerja dengan kamera kedalaman dan memproses data awan titik.

```
#!/usr/bin/env python

import rospy
from sensor_msgs.msg import Image, PointCloud2
from cv_bridge import CvBridge
from sensor_msgs import point_cloud2
import cv2

class DepthPointCloudNode:
    def __init__(self):
        rospy.init_node('depth_pointcloud_node')
        self.depth_pub = rospy.Publisher('/camera/depth', Image, queue_size=10)
        self.pointcloud_pub = rospy.Publisher('/camera/pointcloud', PointCloud2,
queue_size=10)
```

```
        self.cv_bridge = CvBridge()

    def capture_depth_pointcloud(self):
        # Membaca citra kedalaman (depth image) dari sensor
        depth_image = self.read_depth_image()

        # Mengonversi citra kedalaman menjadi pesan Image ROS
        depth_msg = self.cv_bridge.cv2_to_imgmsg(depth_image, encoding="passthrough")

        # Mengatur waktu timbang pesan citra kedalaman
        depth_msg.header.stamp = rospy.Time.now()

        # Memublikasikan pesan citra kedalaman
        self.depth_pub.publish(depth_msg)

        # Menghasilkan awan titik (point cloud) dari citra kedalaman
        pointcloud = self.generate_pointcloud(depth_image)

        # Mengatur waktu timbang pesan awan titik
        pointcloud.header.stamp = rospy.Time.now()

        # Memublikasikan pesan awan titik
        self.pointcloud_pub.publish(pointcloud)

    def read_depth_image(self):
        # Menggunakan kode OpenCV untuk membaca citra kedalaman dari sensor
        # Di sini, kita menggunakan citra kedalaman statis untuk tujuan contoh
        depth_image = cv2.imread('path_to_depth_image.png', cv2.IMREAD_ANYDEPTH)

        return depth_image

    def generate_pointcloud(self, depth_image):
        # Mengonversi citra kedalaman menjadi awan titik (point cloud)
        # Di sini, kita menggunakan titik (x, y, z) yang dihasilkan dari citra kedalaman
        points = []
        for v in range(depth_image.shape[0]):
            for u in range(depth_image.shape[1]):
                z = depth_image[v, u] / 1000.0  # Skala faktor kedalaman jika diperlukan
                x = (u - cx) * z / fx
                y = (v - cy) * z / fy
                points.append([x, y, z])

        fields = [point_cloud2.PointField(name='x', offset=0, datatype=7, count=1),
                  point_cloud2.PointField(name='y', offset=4, datatype=7, count=1),
                  point_cloud2.PointField(name='z', offset=8, datatype=7, count=1)]
        header = point_cloud2.create_cloud_header(frame_id='camera_frame',
stamp=rospy.Time.now())
        pointcloud = point_cloud2.create_cloud_xyz32(header, points, fields)

        return pointcloud
```

```
    def run(self):
        rate = rospy.Rate(1)
        while not rospy.is_shutdown():
            self.capture_depth_pointcloud()
            rate.sleep()

if __name__ == '__main__':
    node = DepthPointCloudNode()
    node.run()
```

2. Pastikan Anda telah mengimpor pesan yang diperlukan dari paket **sensor_msgs** dan modul **CvBridge** dari paket **cv_bridge**. Dalam contoh di atas, kita menggunakan pesan **Image** untuk mengirim citra kedalaman, pesan **PointCloud2** untuk mengirim awan titik, dan modul **CvBridge** untuk mengonversi gambar OpenCV menjadi pesan Image ROS.
3. Pastikan ROS master telah dijalankan dengan perintah **roscore** di terminal.
4. Pastikan kamera dan sensor kedalaman yang sesuai telah terhubung atau disimulasikan dalam sistem ROS Anda.
5. Gantilah **path_to_depth_image.png** dengan jalur lengkap ke citra kedalaman yang ingin Anda gunakan untuk tujuan contoh. Pastikan citra tersebut dapat diakses oleh kode.
6. Sesuaikan rumus dan parameter dalam metode **generate_pointcloud()** sesuai dengan kamera dan sensor kedalaman yang Anda gunakan. Anda perlu mengganti **cx**, **cy**, **fx**, dan **fy** dengan nilai-nilai yang sesuai dari spesifikasi kamera Anda.
7. Jalankan node Anda dengan menjalankan perintah **rosrun package_name depth_pointcloud_node.py**, dengan mengganti **package_name** dengan nama paket ROS yang sesuai.
8. Node akan membaca citra kedalaman (atau citra kedalaman statis dalam contoh ini) menggunakan fungsi **read_depth_image()**. Citra tersebut kemudian dikonversi menjadi pesan Image ROS menggunakan **cv_bridge** dan dipublikasikan ke topik **/camera/depth** menggunakan **depth_pub**. Selanjutnya, node akan menghasilkan awan titik (point cloud) dari citra kedalaman menggunakan **generate_pointcloud()** dan mempublikasikannya ke topik **/camera/pointcloud** menggunakan **pointcloud_pub**.
9. Anda dapat melihat citra kedalaman dan awan titik yang dipublikasikan di topik menggunakan alat visualisasi ROS seperti RViz atau dengan memeriksa topik menggunakan perintah **rostopic echo /camera/depth** dan **rostopic echo /camera/pointcloud**.

Chapter 8 : Point Cloud Processing

Berfokus pada pemrosesan awan titik. ROS menyediakan perpustakaan dan algoritma untuk memanipulasi dan menganalisis data awan titik.

```
#!/usr/bin/env python

import rospy
from sensor_msgs.msg import PointCloud2
from sensor_msgs import point_cloud2

class PointCloudProcessingNode:
    def __init__(self):
        rospy.init_node('pointcloud_processing_node')
        rospy.Subscriber('/camera/pointcloud', PointCloud2, self.pointcloud_callback)

    def pointcloud_callback(self, pointcloud_msg):
        # Mengonversi pesan awan titik menjadi daftar titik 3D
        points = []
        for point in point_cloud2.read_points(pointcloud_msg, field_names=("x", "y", "z"),
skip_nans=True):
            points.append(point)

        # Memproses daftar titik
        processed_points = self.process_pointcloud(points)

        # Menghasilkan awan titik yang diproses
        processed_pointcloud = self.generate_processed_pointcloud(processed_points,
pointcloud_msg)

        # Memublikasikan pesan awan titik yang diproses
        self.publish_processed_pointcloud(processed_pointcloud)

    def process_pointcloud(self, points):
        # Implementasikan logika pemrosesan awan titik di sini
        # Di sini, kita hanya mengalikan semua koordinat titik dengan 2 untuk tujuan contoh
        processed_points = []
        for point in points:
            processed_points.append((point[0] * 2, point[1] * 2, point[2] * 2))

        return processed_points

    def generate_processed_pointcloud(self, processed_points, original_pointcloud):
        # Membuat pesan awan titik yang diproses dengan menggunakan header dari awan
titik asli
        fields = original_pointcloud.fields
        header = original_pointcloud.header
        processed_pointcloud = point_cloud2.create_cloud_xyz32(header, processed_points,
fields)

        return processed_pointcloud

    def publish_processed_pointcloud(self, processed_pointcloud):
        # Menerbitkan pesan awan titik yang diproses
        pub = rospy.Publisher('/camera/processed_pointcloud', PointCloud2, queue_size=10)
```

```
        processed_pointcloud.header.stamp = rospy.Time.now()
        pub.publish(processed_pointcloud)

    def run(self):
        rospy.spin()

if __name__ == '__main__':
    node = PointCloudProcessingNode()
    node.run()
```

1. Pastikan Anda telah mengimpor pesan yang diperlukan dari paket **sensor_msgs** dan modul **point_cloud2** dari paket **sensor_msgs**. Dalam contoh di atas, kita menggunakan pesan **PointCloud2** untuk menerima dan mempublikasikan awan titik, dan modul **point_cloud2** untuk membaca dan menulis titik-titik dalam pesan awan titik.
2. Pastikan ROS master telah dijalankan dengan perintah **roscore** di terminal.
3. Pastikan awan titik (point cloud) yang sesuai telah dipublikasikan di topik **/camera/pointcloud**.
4. Implementasikan logika pemrosesan awan titik di metode **process_pointcloud()**. Dalam contoh ini, kita hanya mengalikan semua koordinat titik dengan 2 untuk tujuan contoh. Anda dapat menggantinya dengan logika pemrosesan yang sesuai dengan kebutuhan Anda.
5. Sesuaikan header dan bidang pesan awan titik yang diproses dalam metode **generate_processed_pointcloud()**. Di sini, kita menggunakan header dan bidang dari pesan awan titik asli.
6. Menggunakan **rospy.Publisher**, buatlah objek penerbit (publisher) baru di metode **publish_processed_pointcloud()**. Pastikan untuk mengganti **/camera/processed_pointcloud** dengan topik yang sesuai untuk awan titik yang diproses.
7. Jalankan node Anda dengan menjalankan perintah **rosrun package_name pointcloud_processing_node.py**, dengan mengganti **package_name** dengan nama paket ROS yang sesuai.
8. Node akan menerima pesan awan titik dari topik **/camera/pointcloud** menggunakan metode **pointcloud_callback()**. Pesan awan titik tersebut kemudian akan diubah menjadi daftar titik 3D menggunakan **point_cloud2.read_points()**. Daftar titik tersebut akan diproses menggunakan metode **process_pointcloud()**, dan awan titik yang diproses akan dihasilkan menggunakan **generate_processed_pointcloud()**. Selanjutnya, pesan awan titik yang diproses akan dipublikasikan di topik **/camera/processed_pointcloud** menggunakan **publish_processed_pointcloud()**.
9. Anda dapat memeriksa pesan awan titik yang diproses menggunakan alat visualisasi ROS seperti RViz atau dengan memeriksa topik menggunakan perintah **rostopic echo /camera/processed_pointcloud**

KESIMPULAN

Setiap bab buku "A Systematic Approach to Learning Robot Programming with ROS" memberikan gambaran konsep dan implementasi yang dibahas di dalamnya. Contoh kode untuk setiap bab menunjukkan aplikasi praktis yang menggunakan ROS, seperti alat dan node ROS, pesan dan server, simulasi, transformasi koordinat, sensor dan visualisasi, penggunaan kamera, pencitraan kedalaman dan awan titik, dan pemrosesan awan titik. Pembaca dapat memperoleh pengalaman praktis dan pemahaman yang lebih mendalam tentang pemrograman robot dengan ROS dengan mengikuti contoh kode dan petunjuk yang disediakan.