MOBILE ROBOTS (01416511)

Mobile Robot Chapter 4 : Odometry

Odometry

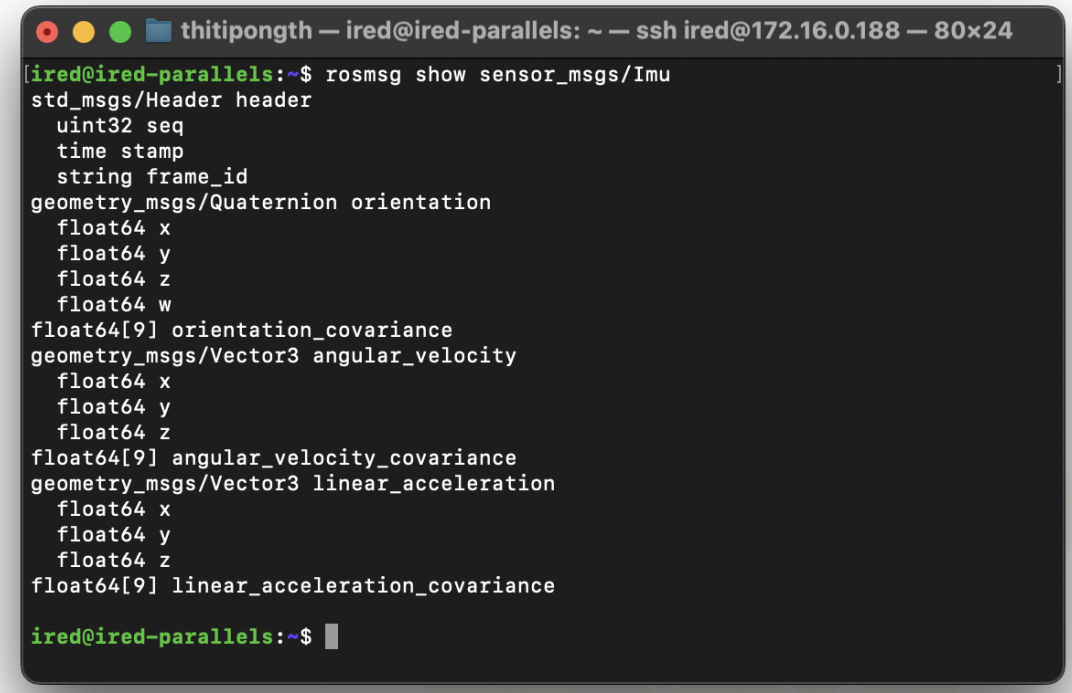
IMU Data

Odometry from IMU data involves using the information from the IMU's accelerometers and gyroscopes to estimate the robot's position and orientation over time.

Accelerometer: Measures linear acceleration along three axes (x, y, z). This data is crucial for estimating the robot's velocity and position.

Gyroscope: Measures angular velocity (rate of rotation) around three axes (roll, pitch, yaw). This data is essential for estimating the robot's orientation.

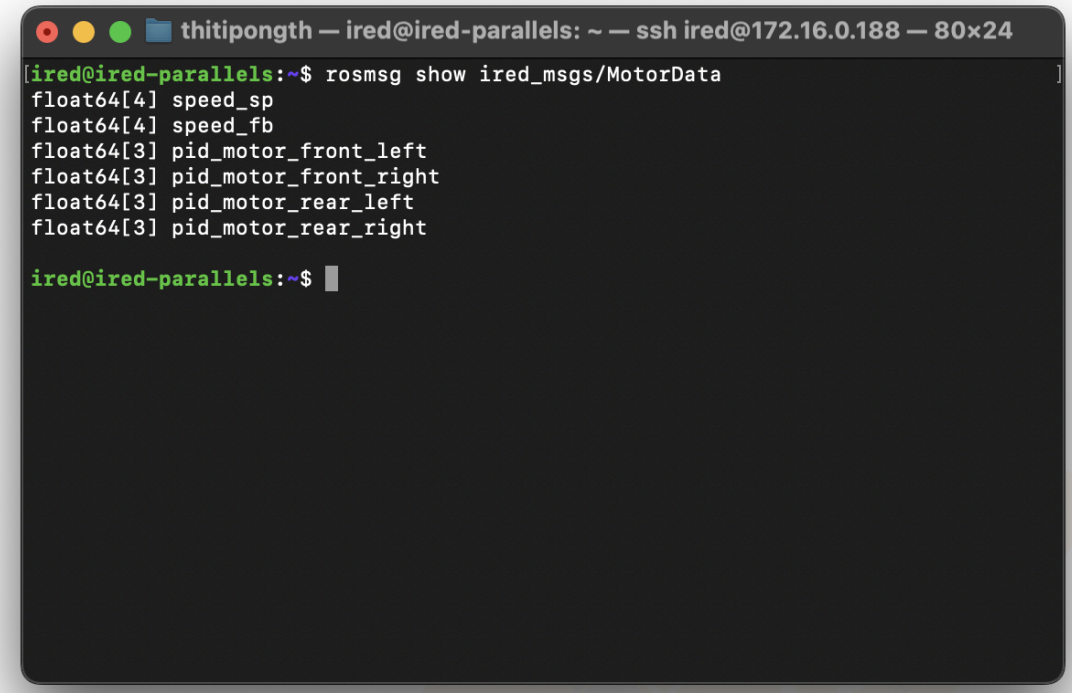
In ROS, we use IMU with the message type sensor_msgs/IMU to calculate odometry and for future sensor fusion. Accelerations should be in m/s² (not in g's), and rotational velocity should be in rad/sec. If the covariance of the measurement is known, it should be filled in. If all you know is the variance of each measurement (e.g., from the datasheet), just place those along the diagonal. A covariance matrix of all zeros will be interpreted as 'covariance unknown.' To use the data, a covariance will have to be assumed or obtained from another source. If you have no estimate for one of the data elements (e.g., your IMU doesn't produce an orientation estimate), set element 0 of the associated covariance matrix to -1. If you are interpreting this message, please check for a value of -1 in the first element of each covariance matrix and disregard the associated estimate.



Encoder

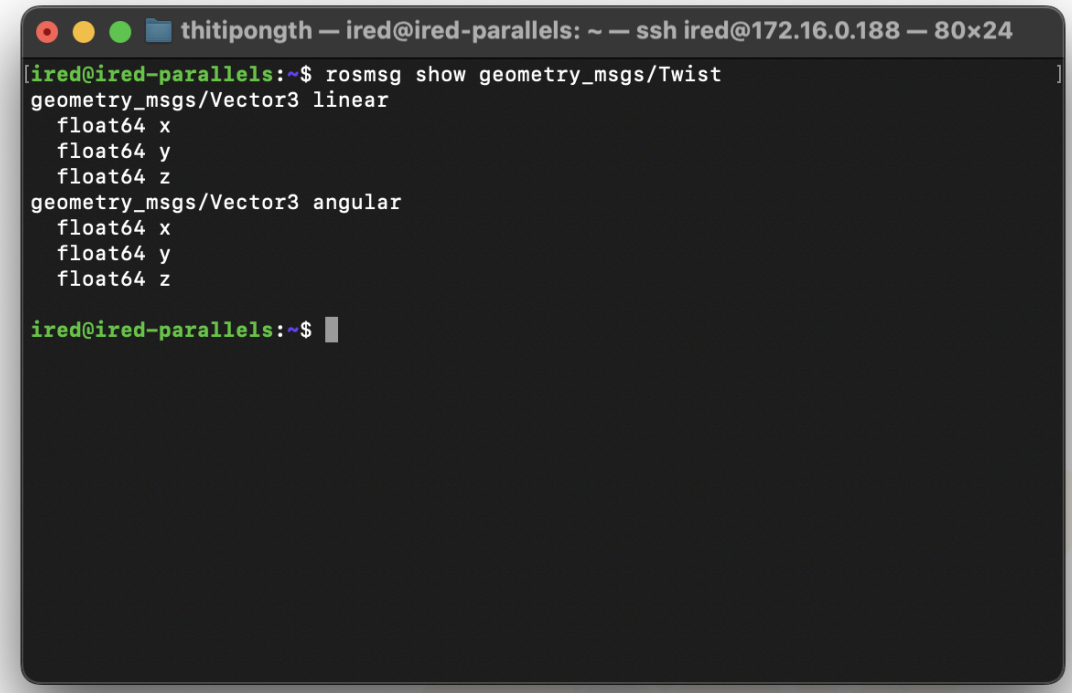
Odometry from encoders can be calculated using the data provided by the encoders attached to the robot's motors. The data, published under the message type ***ired_msgs/MotorData***, contains speed feedback from the motors. This speed feedback is used to determine the robot's linear velocities in the x and y directions, as well as the angular velocity around the z-axis, through forward kinematics. Forward kinematics translates the rotational speeds of the motors into these velocity components. Once you have these velocity measurements, you can compute the robot's odometry, which provides its position and orientation over time. This odometry information is then published to the ***/odom*** topic, allowing the robot to track its movement and position for tasks such as navigation and localization.

As shown in this picture, the message type ***ired_msgs/MotorData*** will display the data.



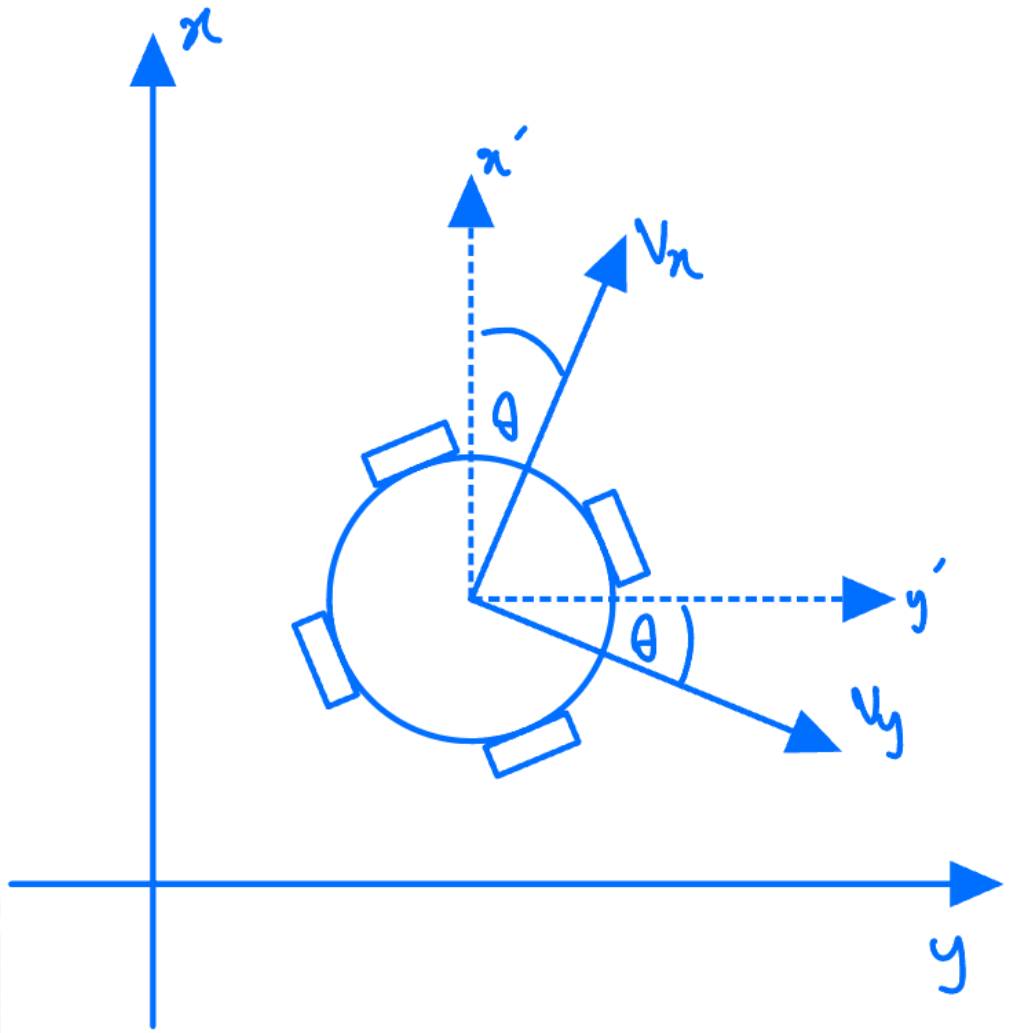
Command Velocity

To calculate the odometry of a robot using teleop_key commands, you first need to understand how these commands translate into physical movements. When you use teleop_key, you are sending velocity commands to the robot, which include both linear velocities (forward and backward movement) and angular velocities (rotation). These commands are published to the /cmd_vel topic and received by the robot's control system, which then adjusts the speeds of the motors to move the robot according to the input. Encoders attached to the motors measure the actual wheel rotations and provide feedback about the robot's movement.



How to calculate the odometry of the robot

To calculate 2D odometry using ***linear.x, linear.y, and angular.z***, you need to integrate these velocity commands over time to estimate the robot's position and orientation. The ***linear.x*** value represents the forward velocity, ***linear.y*** the sideways velocity, and angular.z the rate of rotation around the vertical axis. Start by calculating the time step dt between the current and previous measurements. Use ***angular.z*** to update the robot's orientation (theta), integrating it over time to find the change in heading. Then, compute the changes in position based on the robot's current orientation: linear.x affects forward movement while linear.y adjusts for sideways movement, both modified by the robot's heading. By applying these changes, you update the robot's position (x, y). Finally, create and publish an odometry message that includes the updated position and orientation, along with the velocities. This process provides an estimate of the robot's trajectory and helps in tracking its movement in a 2D plane.



The Vx(linear.x) value represents the forward velocity, Vy(linear.y) represents the sideways velocity, and theta (orientation) can be calculated from the angular velocity(angular.z).

- **Time Step Calculation:**
  Measures the time between consecutive velocity commands to compute changes over time.
  dt = current_time - last_time
- **Orientation Update**:
  The change in orientation is integrated from the angular velocity, then adjusted to keep it within the standard range.
  delta_theta = angular.z * dt
  theta += delta_theta
- **Position Update**:
  Position updates account for the robot's current heading, converting linear and sideways velocities into changes in the robot's global position
  delta_x = (linear.x * cos(theta) - linear.y * sin(theta)) * dt
  delta_y = (linear.x * sin(theta) + linear.y * cos(theta)) * dt

x += delta_x
y += delta_y

Example Python Code

In this class, we will demonstrate an example of Python code to calculate the odometry of a robot using IMU data.

First, we need to create a ROS package named ***ired_odom***.

```
$ cd ~/ired_ws/src
$ catkin_create_pkg ired_odom roscpp rospy std_msgs
$ cd ~/ired_ws
$ catkin_make
$ source ~/.bashrc
```

Create a folder named ***scripts*** and add a file named ***odom_imu.py*** inside the scripts folder.

```
$ roscd ired_odom
$ mkdir scripts
$ touch odom_imu.py
$ chmod +x odom_imu.py
```

Here is an example code for calculating odometry from IMU data

```
#!/usr/bin/env python3
import rospy
from nav_msgs.msg import Odometry
from sensor_msgs.msg import Imu
import tf.transformations
from math import atan2, sin, cos

imu_data_ = Imu()
odom_imu_ = Odometry()
x = 0.0
y = 0.0
theta = 0.0
Vx = 0.0
Vy = 0.0

def IMUDataCallback(msg):
    global imu_data_
    imu_data_ = msg

def OdomIMU(dt):
    global imu_data_, odom_imu_, x, y, theta, Vx, Vy

    acc_x_ = imu_data_.linear_acceleration.x
    acc_y_ = imu_data_.linear_acceleration.y
    angular_z_ = imu_data_.angular_velocity.z
```

```
    delta_theta = angular_z_ * dt
    theta += delta_theta
    theta = atan2(sin(theta), cos(theta))  # Normalize to [-π, π]

    Vx += acc_x_ * dt
    Vy += acc_y_ * dt

    delta_x = (Vx * cos(theta) - Vy * sin(theta)) * dt
    delta_y = (Vx * sin(theta) + Vy * cos(theta)) * dt

    x += delta_x
    y += delta_y

    odom_imu_.pose.pose.position.x = x
    odom_imu_.pose.pose.position.y = y
    odom_imu_.pose.pose.position.z = 0.0

    quaternion = tf.transformations.quaternion_from_euler(0, 0, theta)
    odom_imu_.pose.pose.orientation.x = quaternion[0]
    odom_imu_.pose.pose.orientation.y = quaternion[1]
    odom_imu_.pose.pose.orientation.z = quaternion[2]
    odom_imu_.pose.pose.orientation.w = quaternion[3]

    odom_imu_.twist.twist.linear.x = Vx
    odom_imu_.twist.twist.linear.y = Vy
    odom_imu_.twist.twist.angular.z = angular_z_

def main():
    global odom_imu_
    rospy.init_node('odom_imu_node')
    rate = rospy.Rate(10)

    rospy.loginfo("Wait for the IMU data...")
    rospy.wait_for_message("/ired/imu/data", Imu)
    rospy.Subscriber("/ired/imu/data", Imu, IMUDataCallback)
    odom_imu_pub_ = rospy.Publisher("/odom/imu", Odometry, queue_size=10)
    rospy.loginfo('ROS publisher on /odom/imu [nav_msgs/Odometry]')
    rospy.loginfo('ROS subscriber on /ired/imu/data [sensor_msgs/Imu]')

    odom_imu_.header.frame_id = 'odom'
    odom_imu_.child_frame_id = 'base_footprint'

    last_time = rospy.Time.now()

    while not rospy.is_shutdown():
        current_time = rospy.Time.now()
        dt = (current_time - last_time).to_sec()
        last_time = current_time
        OdomIMU(dt)

        odom_imu_.header.stamp = current_time
```

```
        odom_imu_pub_.publish(odom_imu_)
        rate.sleep()

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

Assignment

Group Assignment: Create a new file named ***odom_fake.py*** to calculate the odometry of the robot. The file should subscribe to the topic ***/cmd_vel*** with the message type ***geometry_msgs/Twist*** and publish the odometry data to the topic ***/odom/cmd_vel*** with the message type ***nav_msgs/Odometry***. ***Submit the file to Classroom via the group representative.***