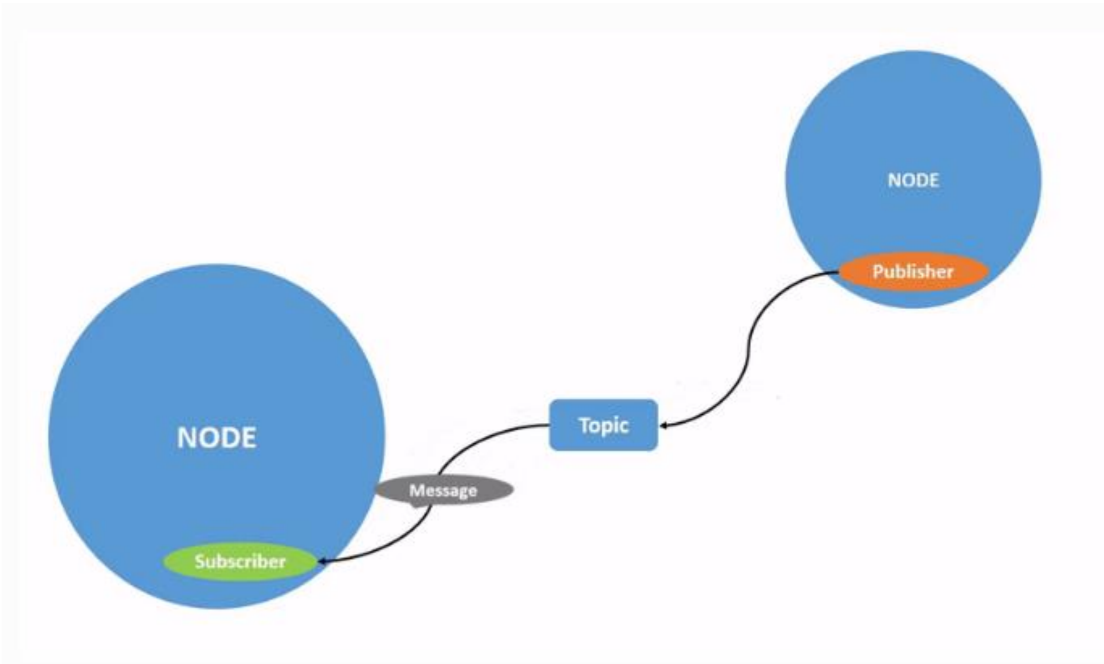
safety_node.py

How ROS2 topics work: https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html




```
self.publisher_ = self.create_publisher(AckermannDriveStamped, '/drive', 10)
```

"AckermannDriveStamped" provides multiple properties for Ackermann steering.

ackermann_msgs / msg / AckermannDriveStamped.msg

mstuettgen initial commit

Code Blame 5 lines (4 loc) · 120 Bytes

```
## Time stamped drive command for robots with Ackermann steering.
#  $Id$

std_msgs/Header header
AckermannDrive  drive
```

Create a publisher that publishes a message with data type 'AckermannDriveStamped,' to the topic '/drive', with a queue sized 10.

```
self.subscription_odom = self.create_subscription(
            Odometry,
            '/ego_racecar/odom',
            self.odom_callback,
            10)
```

A subscriber that gets data type Odometry for topic '/ego_racecar/odom' and runs a function 'odom_callback.'

```
self.subscription_scan = self.create_subscription(
            LaserScan,
            '/scan',
            self.scan_callback,
            10)
```

Similarly, make a subscriber for scan messages. Its data type is LaserScan with the /scan topic running self.scan_callback function, and it has a queue sized 10.

```
def odom_callback(self, odom_msg):
        self.speed = odom_msg.twist.twist.linear.x
```

"The twist in this message (odom_msg) corresponds to the robot's velocity in the child frame, normally the coordinate frame of the mobile base, along with an optional covariance for the certainty of that velocity estimate."

twist.twist.linear.x indicates the velocity to the x direction. odm_msg also has other properties such as twist.twist.linear.y and twist.twist.angular.z.

(https://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom)

```
def scan_callback(self, scan_msg):
        min_ttc = float('inf')
        for i, range in enumerate(scan_msg.ranges):
            if range > 0:
                angle = scan_msg.angle_min + i * scan_msg.angle_increment
```

Initialize min_ttc with infinity. It will update the min_ttc as the subscriber gets scan messages. LaserScan messages represent the distance from a LIDAR sensor to the nearest obstacle in a particular direction. Message definition can be found here (llink). So, if the range (distance) is larger than 0, calculate angle minimum angle + counting index * angle increment.

```
relative_speed = self.speed * np.cos(angle)
```

Relative speed can be calculated with its current speed * cosine of the angle. The speed is obtained from odom_callback. If the relative speed is larger than 0, it means the car is moving toward an obstacle. TTC is calculated by relative speed / range.

"A positive range rate means the range measurement is expanding, and a negative one means the range measurement is shrinking. Thus, it can be calculated in two different ways. First, it can be calculated by mapping the vehicle's current longitudinal velocity onto each scan beam's angle by using vx* cosθi. (F1Tenth tutorial link)" Update min_ttc if it is less than the current min_ttc.

```
if min_ttc < 1.0:
            brake_msg = AckermannDriveStamped()
            brake_msg.drive.speed = 0.0
            brake_msg.drive.acceleration = -5.0
            self.publisher_.publish(brake_msg)
```

If the time to collision is less than 1 seconds, create a AckermannDriverStamped message to brake. Stops the vehicle (speed = 0.0) and applies negative acceleration (braking). This braking message is published to the '/drive' topic.