

Problem 1

Question 1.1

When `listener_accumulator.py` is executed, we first enter the main function:

```
def main(args=None):  
    ''' Init ROS, launch node, spin, cleanup '''  
    roslpy.init(args=args)  
    listener_accumulator = ListenerAccumulator()  
    roslpy.spin(listener_accumulator)  
  
    # Node cleanup  
    listener_accumulator.destroy_node()  
    roslpy.shutdown()
```

`listener_accumulator` is instantiated as an object of class `ListenerAccumulator`, and upon this instantiation, we enter the `__init__` function of the class:

```
def __init__(self):  
    ''' Setup subscriber, publisher, timer, and array to hold accumulated messages '''  
    super().__init__('listener_accumulator')  
    self.subscriber = self.create_subscription(String, 'chatter', self.subscriber_callback, 10)  
    self.publisher = self.create_publisher(String, 'chatter_repeated', 10)  
    timer_period = 3 # seconds  
    self.timer = self.create_timer(timer_period, self.timer_callback)  
    self.accumulated_msgs = ""
```

We inherit from the `Node` class and give the node the name, 'listener_accumulator'. We create a subscription to the topic 'chatter' with a `subscriber_callback` function, and we create a publisher that publishes to the 'chatter_repeated' topic. Then, to create our timer, we set a `timer_period` of 3 seconds (1/3 Hz) as specified in the homework instructions, and create a timer with a `timer_callback`. Finally, we initialize an `accumulated_msgs` string that will store incoming messages.

In main, `spin` is called, blocking our program, consequently allowing the `listener_accumulator` node to not be destroyed.

Back in `ListenerAccumulator`, we listen on the topic 'chatter' for messages. Whenever a message is 'heard', we enter the `subscriber_callback` function, which will concatenate the new message to the `accumulated_msgs`:

```
def subscriber_callback(self, msg):  
    ''' Callback for subscriber '''  
    # Append message to the accumulated_msgs array, i.e., store message  
    self.accumulated_msgs = self.accumulated_msgs + msg.data
```

Since we set a timer for 3 seconds earlier, every 3 seconds, we will enter the `timer_callback` function, which publishes all of the strings we have accumulated in the last 3 seconds, and then resets the `accumulated_msgs` string to an empty string, allowing us to have a fresh storage for new messages:

```
def timer_callback(self):
    ''' Callback for timer '''
    # Publish the accumulated_msgs, then clear the accumulated_msgs
    msg = String()
    msg.data = self.accumulated_msgs
    self.publisher.publish(msg)
    self.get_logger().info(f'Published: {msg.data}')
    self.accumulated_msgs = ""
```

This process continues endlessly until a CTRL+C (SIGINT) signal is sent to the listener_accumulator.py process.

Problem 2

Question 2.1

```
14 # Homework 1 Question 2.1
15 def twist_to_speeds(speed_linear, speed_angular):
16     """
17     Calculates the speed of each wheel
18     given the current linear and rotational speed of the robot
19     """
20     # gives the code a range around zero,
21     # could be decreased dependent on the noise of the data recieved
22     close0 = 0.05
23
24     # if angular speed is around zero, both the speed of the left and right motors have to be equal
25     if -close0 < speed_angular < close0:
26         speed_left = speed_linear
27         speed_right = speed_left
28
29     # linear speed is the motor speed closest to zero,
30     # and angular speed tells which motor is greater
31     elif close0 < speed_angular < 1:
32         #in this case, assuming right motor has greater output
33         speed_left = speed_linear
34         #adding arbitrary ratio including angular speed
35         speed_right = speed_linear * (1+speed_angular)
36
37     #in this case, assuming left motor is greater, adding the same ratio to left motor
38     elif -1 < speed_angular < -close0:
39         speed_left = speed_linear * (1+speed_angular)
40         speed_right = speed_linear
41
42     #impossible to have angular speed of 1 with linear speed as
43     elif speed_angular == 1:
44         # that would require that one of the motors exceed their max output
45         speed_left = 0
46         speed_right = 1
47
48     # same story here
49     elif speed_angular == -1:
50         speed_left = 1
51         speed_right = 0
52     return speed_left, speed_right
53
```

Question 2.2

[geometry_msgs/msg/Twist Message](#)

File: `geometry_msgs/msg/Twist.msg`

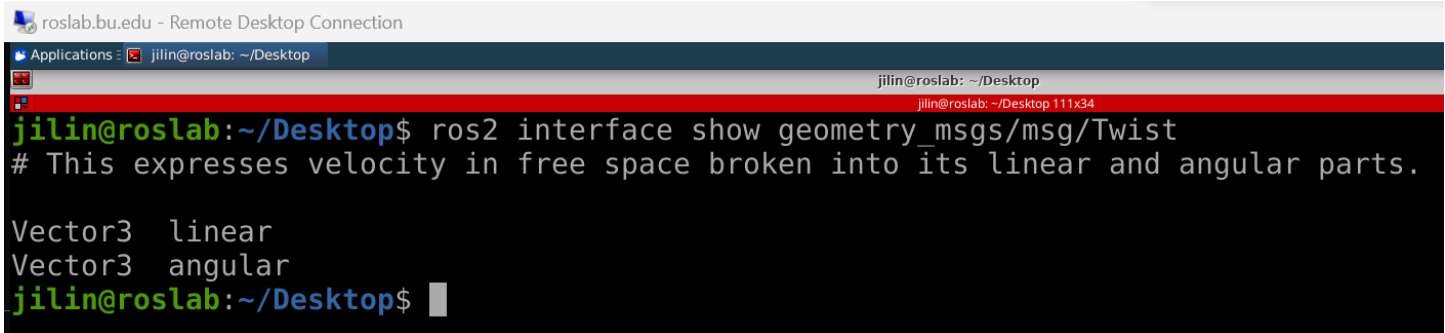
Raw Message Definition

```
# This expresses velocity in free space broken into its linear and angular parts.  
  
Vector3 linear  
Vector3 angular
```

Compact Message Definition

```
geometry_msgs/msg/Vector3 linear  
geometry_msgs/msg/Vector3 angular
```

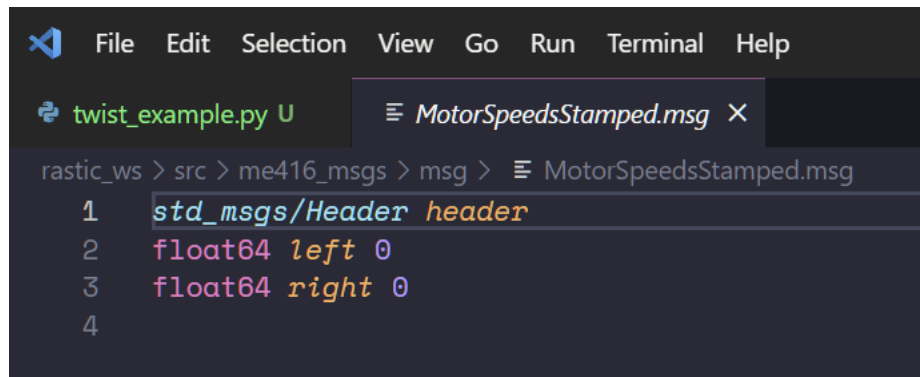
autogenerated on Oct 09 2020 00:02:33



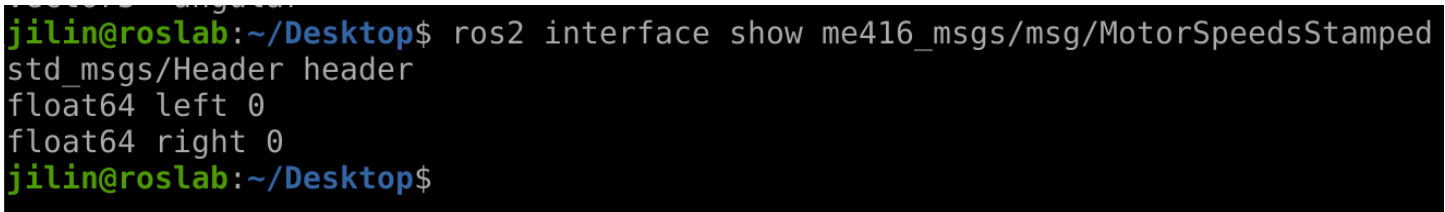
```
rosrab.bu.edu - Remote Desktop Connection  
Applications: jilin@rosrab: ~/Desktop  
jilin@rosrab: ~/Desktop  
jilin@rosrab: ~/Desktop 111x34  
jilin@rosrab:~/Desktop$ ros2 interface show geometry_msgs/msg/Twist  
# This expresses velocity in free space broken into its linear and angular parts.  
  
Vector3 linear  
Vector3 angular  
jilin@rosrab:~/Desktop$
```

`geometry_msgs` is a package that has the message type `Twist`. `Twist` has two fields, `linear` and `angular`, both of message type `Vector3`. `Vector3`, like `Twist`, also has multiple fields; they are `x`, `y`, and `z`, all of the fundamental type `float64`, i.e. 64-bit floating-point numbers.

Question 2.3



```
File Edit Selection View Go Run Terminal Help  
twist_example.py U MotorSpeedsStamped.msg X  
rastic_ws > src > me416_msgs > msg > MotorSpeedsStamped.msg  
1 std_msgs/Header header  
2 float64 left 0  
3 float64 right 0  
4
```



```
jilin@rosrab:~/Desktop$ ros2 interface show me416_msgs/msg/MotorSpeedsStamped  
std_msgs/Header header  
float64 left 0  
float64 right 0  
jilin@rosrab:~/Desktop$
```

We are looking at the contents of the message type `MotorSpeedsStamped` in the `me416_msgs` package. `MotorSpeedsStamped` has three fields: `header`, of message type `Header` found in the `std_msgs` package, and `left` and `right`, of fundamental type `float64`, initialized to be 0.

Question 2.4

```
4
5 import robot_model
6 import me416_utilities
7 import rclpy
8 from rclpy.node import Node
9 from geometry_msgs.msg import Twist
10 from me416_msgs.msg import MotorSpeedsStamped
11
12
13 motor_left = me416_utilities.MotorSpeedLeft()
14 motor_right = me416_utilities.MotorSpeedRight()
15
16
17 class MotorCommand(Node):
18     ''' Inherited Node class that subscribes and publishes concatenated messages with a delay '''
19     def __init__(self):
20         ''' Setup subscriber to robot_twist and publisher to motor_speeds'''
21         super().__init__('motor_command')
22         self.subscriber = self.create_subscription(Twist, 'robot_twist', self.subscriber_callback, 10)
23         self.publisher = self.create_publisher(MotorSpeedsStamped, 'motor_speeds', 10)
24         self.msg = MotorSpeedsStamped()
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42 def main(args=None):
43     ''' Init ROS, launch node, spin, cleanup '''
44     rclpy.init(args=args)
45     motor_command = MotorCommand()
46     rclpy.spin(motor_command)
47
48     # Node cleanup
49     motor_command.destroy_node()
50     rclpy.shutdown()
51
52
53 if __name__ == '__main__':
54     main()
55
```

Question 2.5 & 2.6

```
27 def subscriber_callback(self, twist_msg):
28     ''' Callback for subscriber '''
29     # Question 2.5
30     left_speed, right_speed = robot_model.twist_to_speeds(twist_msg.linear.x, twist_msg.angular.z)
31     motor_left.set_speed(left_speed)
32     motor_right.set_speed(right_speed)
33
34     # Question 2.6
35     msg = MotorSpeedsStamped()
36     msg.header.stamp = self.get_clock().now().to_msg()
37     msg.left = left_speed
38     msg.right = right_speed
39     self.publisher.publish(msg)
40
```