Aaron Anderson, Andrew Cao, Jilin Zheng

ME416 Homework 1

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 '''
    rclpy.init(args=args)
    listener_accumulator = ListenerAccumulator()
    rclpy.spin(listener_accumulator)

    # Node cleanup
    listener_accumulator.destroy_node()
    rclpy.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: twist_to_speed()

```
# Homework 1 Question 2.1
def twist_to_speeds(speed_linear, speed_angular):
    """
    Calculates the speed of each wheel
    given the currect linear and rotational speed of the robot
    """
    # gives the code a range around zero,
    # could be decreased dependent on the noise of the data recieveds
    close0 = 0.05

    # if angular speed is around zero, both the speed of the left and right motors have to be equal
    if -close0 < speed_angular < close0:
        speed_left = speed_linear
        speed_right = speed_left

    # linear speed is the motor speed closest to zero,
    # and angular speed tells which motor is greater
    elif close0 < speed_angular < 1:
        #in this case, assuming right motor has greater output
        speed_left = speed_linear
        #adding arbtiuary ratio including angular speed
        speed_right = speed_linear * (1+speed_angular)

    #in this case, assuming left motor is greater, adding the same ratio to left motor
    elif -1 < speed_angular < -close0:
        speed_left = speed_linear * (1+speed_angular)
        speed_right = speed_linear

    #impossible to have angular speed of 1 with linear speed as
    elif speed_angular == 1:
        # that would require that one of the motors exceed their max output
        speed_left = 0
        speed_right = 1

    # same story here
    elif speed_angular == -1:
        speed_left = 1
        speed_right = 0
    return speed_left, speed_right
```

Question 2.2: geometry_msgs/msg/Twist

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
```

```
acao@roslab:~$ ros2 interface show geometry_msgs/msg/Twist
# This expresses velocity in free space broken into its linear and angular parts.

Vector3  linear
Vector3  angular
```

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: me416_msgs/msg/MotorSpeedsStamped

```
std_msgs/Header header
float64 left 0
float64 right 0
```

```
acao@roslab:~$ ros2 interface show me416_msgs/msg/MotorSpeedsStamped
std_msgs/Header header
float64 left 0
float64 right 0
```

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:

```python
import robot_model
import me416_utilities
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist
from me416_msgs.msg import MotorSpeedsStamped


motor_left = me416_utilities.MotorSpeedLeft()
motor_right = me416_utilities.MotorSpeedRight()


class MotorCommand(Node):
    ''' Inherited Node class that subscribes and publishes concatenated messages with a delay '''
    def __init__(self):
        ''' Setup subscriber to robot_twist and publisher to motor_speeds'''
        super().__init__('motor_command')
        self.subscriber = self.create_subscription(Twist, 'robot_twist', self.subscriber_callback, 10)
        self.publisher = self.create_publisher(MotorSpeedsStamped, 'motor_speeds', 10)
        self.msg = MotorSpeedsStamped()
```

```python
def main(args=None):
    ''' Init ROS, launch node, spin, cleanup '''
    rclpy.init(args=args)
    motor_command = MotorCommand()
    rclpy.spin(motor_command)

    # Node cleanup
    motor_command.destroy_node()
    rclpy.shutdown()


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

Question 2.5: robot_twist() & Question 2.6: MotorSpeedsStamped

```python
    def subscriber_callback(self, twist_msg):
        ''' Callback for subscriber '''
        # Question 2.5
        left_speed, right_speed = robot_model.twist_to_speeds(twist_msg.linear.x, twist_msg.angular.z)
        motor_left.set_speed(left_speed)
        motor_right.set_speed(right_speed)

        # Question 2.6
        msg = MotorSpeedsStamped()
        msg.header.stamp = self.get_clock().now().to_msg()
        msg.left = left_speed
        msg.right = right_speed
        self.publisher.publish(msg)
```