

# **CMPUT 412 Exercise #2**

Antonio Lech Martin-Ozimek

February 14, 2025



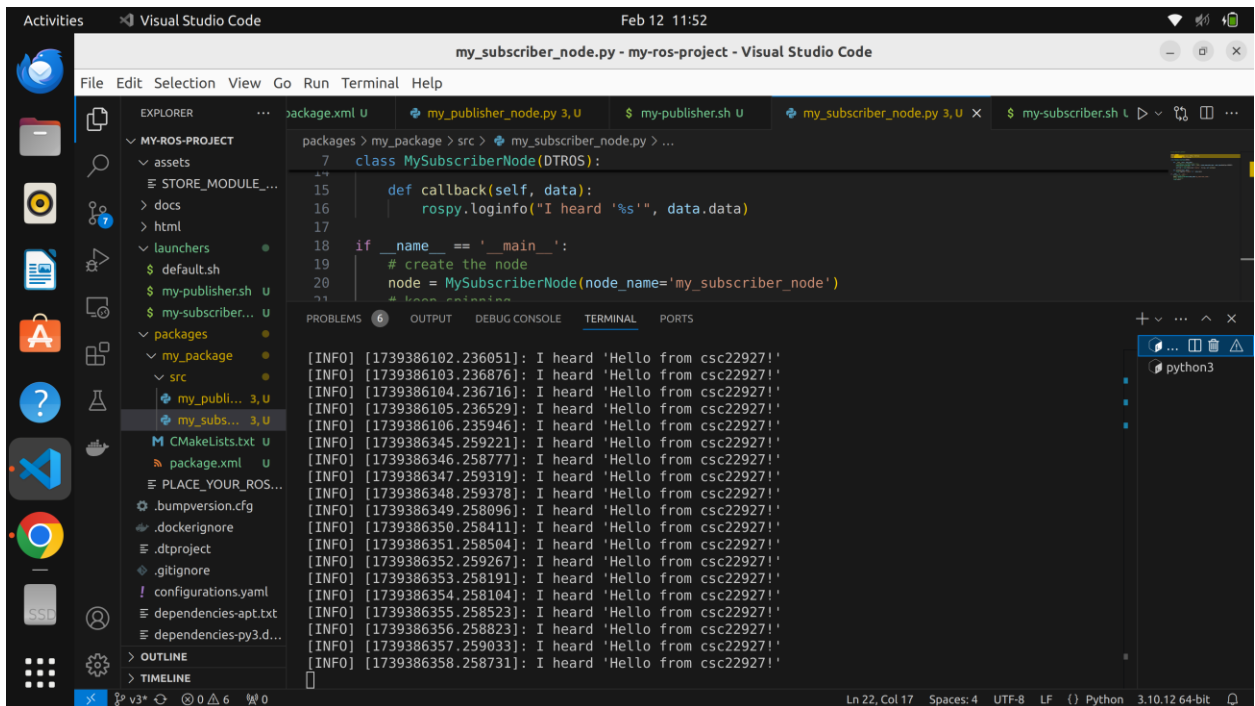
## **PART 1**

### **1. ROS Definitions & Learning**

- **Node** – A node acts as a separate process running in the ROS ecosystem that helps break up processing during the running of a robot/system. One node could be running image pre-processing while the other is running localization using the image data. The data collected by one node can be published to a topic so other nodes can access it.
- **Topic** – A topic is uni-directional MIMO pipe that any node can connect to in order to publish data. To connect a node must match the type defined by the node who created the topic using a message. Any other nodes can request data from the topic. Neither the publisher nor the subscriber is aware of each other, the topic is purely for receiving data from a node. In order to get a response from the publisher using multi-directional communication, one should use a service.
- **Service** – Services are similar to topics in that they are used for data transfer. However, in this case the service is defined using two messages, one for requests and one for replies. The node that is requesting the data must match the request message defined by the publishing node. If the request is correct the node at the other end will send a reply. It is a one time connection like a /GET request, not like a pipe or a continuous data interaction like a topic.
- **Message** – A message is what is used to communicate between nodes. It is a simple struct made up of primitive types to make it as simple as possible for communication. For a topic it is used to define how to pull data from the topic itself based on the definition from the publishing node. For a service, we define the reply type and the request type using messages.
- **Bag** – A bag is essentially a log file. It records all messages passed to a topic and serializes and stores them. The bags can also playback to a topic all the messages it has stored. So, it is a logfile that contains all the history of messages passed over a

topic. It can be used to create datasets, but it is also used to create visualizations of all the messages as a graph.

- The typical workflow is to decide whether you want to use a topic or service. Once you have decided you need to set up the message type that you will be using for either one. This involves importing the correct message type to be published and setting a name for the topic or service. Along with the name you set a callback function which can be run when a request is made. The callback runs and returns the data request by the subscriber. In a service the published only publishes data when requested but in a topic, data is usually pushed even if there is no request made.



```
my_subscriber_node.py - my-ros-project - Visual Studio Code

class MySubscriberNode(DTROS):
    def callback(self, data):
        rospy.loginfo("I heard '%s'", data.data)

if __name__ == '__main__':
    # create the node
    node = MySubscriberNode(node_name='my_subscriber_node')
```

```
[INFO] [1739386102.236051]: I heard 'Hello from csc22927!'
[INFO] [1739386103.236876]: I heard 'Hello from csc22927!'
[INFO] [1739386104.236716]: I heard 'Hello from csc22927!'
[INFO] [1739386105.236529]: I heard 'Hello from csc22927!'
[INFO] [1739386106.235946]: I heard 'Hello from csc22927!'
[INFO] [1739386345.259221]: I heard 'Hello from csc22927!'
[INFO] [1739386346.258777]: I heard 'Hello from csc22927!'
[INFO] [1739386347.259319]: I heard 'Hello from csc22927!'
[INFO] [1739386348.259378]: I heard 'Hello from csc22927!'
[INFO] [1739386349.258096]: I heard 'Hello from csc22927!'
[INFO] [1739386350.258411]: I heard 'Hello from csc22927!'
[INFO] [1739386351.258504]: I heard 'Hello from csc22927!'
[INFO] [1739386352.259267]: I heard 'Hello from csc22927!'
[INFO] [1739386353.258191]: I heard 'Hello from csc22927!'
[INFO] [1739386354.258104]: I heard 'Hello from csc22927!'
[INFO] [1739386355.258523]: I heard 'Hello from csc22927!'
[INFO] [1739386356.258823]: I heard 'Hello from csc22927!'
[INFO] [1739386357.259033]: I heard 'Hello from csc22927!'
[INFO] [1739386358.258731]: I heard 'Hello from csc22927!'
```

2.

Fig 1. ROS Subscriber running on Duckiebot

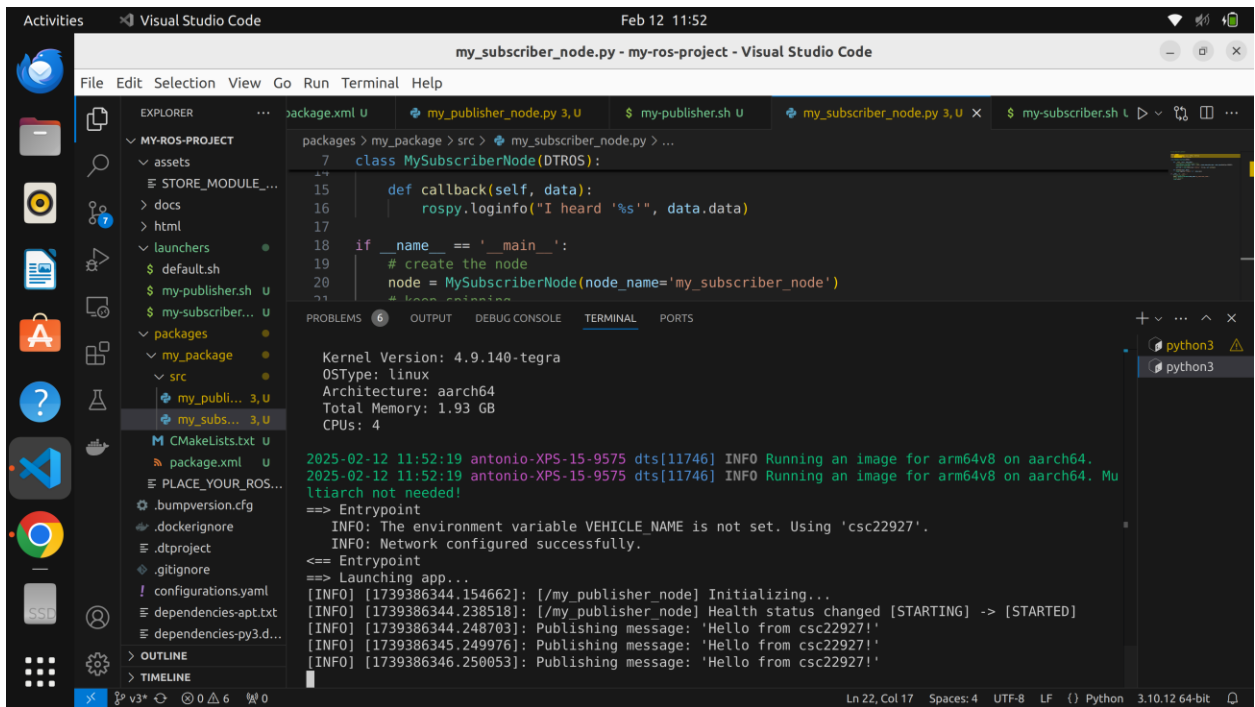


Fig. 2. ROS Publisher running on

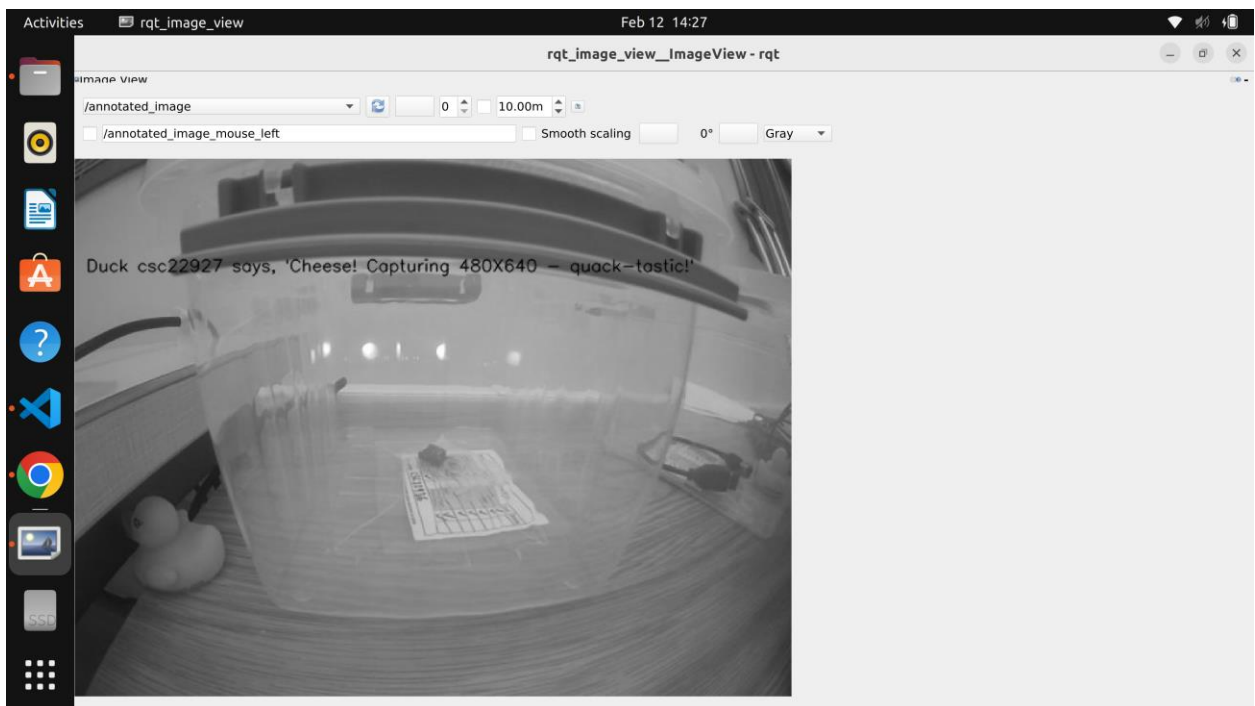


Fig. 3. Annotated Image sent using Image message type

- Recall that this did not work when trying to publish using `compressed_imgmsg` type. The problem is most likely that `rqt_image_view` expects only the regular `Image` type.

## PART 2

Incremental Encoder – Does not have an absolute positioning, but rather calculates when connected to power how the encoder changes over time

Absolute Encoder – During setup we set a start position as absolute 0 so we know the position relative to the start when plugging in everything

### Calculating Position

Circumference Using Paper Wrapper = 21.0-21.1cm  $\approx$  21.05cm

Circumference Using Diameter Estimates = 20.43cm - 21.04cm or (6.5cm-6.7cm)  $\approx$  20.75 cm

Circumference Using Radius From Rosout Log =  $0.0318 \text{ m} * 2 * \pi = 19.98 \text{ cm}$

### 1. Straight Line Task

- Why is there a difference between the actual and desired location?
  - As far as I could tell, the difference between the predicted and the true distance was caused by the latency of reading from the node over the Wi-Fi. I assume there was latency, but I did not observe it necessarily. The largest latency was with the polling rate, I believe. There was a bit of a problem with the latency of reading from the wheel encoders. It caused an overshoot of about 2cm on both going forwards and backwards according to my script. This seemed about accurate when observing the real-life results.
- What speed did you use?
  - I used a speed of 0.5 for my resulting video but I also tested at 0.25 and 1.0. I didn't test at 0.75 because I really only cared about the extremes to see if going really fast or really slow had a difference on the results.
- What happens when you change the speed?
  - Increasing the speed had an interesting effect based on my implementation. Since the robot's velocity went to 0 immediately, the momentum lifted the back ball-bearing up into the air before the reversing began. When I increased the speed this problem increased. Another issue I noticed was that the overshoot increased when I increased the speed of the robot, even though my program only read it as being about 2cm over. Meaning that it is not accurate in that regard. For 0.5 and 0.25 the overshoot was roughly the same.

### 2. Rotation Task

- Did you notice any deviations? What caused them?

- So once again the deviations were caused by a combination of two things. The first is that the robot has momentum of its own so that when you stop its encoders, it still continues to move in its arc. The second is that the robot is polling from a topic which is computing the distance it is moving at a certain frequency. So, there is a delay between when it receives the information that it has moved far enough and when it has actually moved that far. Usually, it moves a bit past the upper bound on its movement. When the movement is large like the drive straight task, the overshoot is relatively negligible. However, that same overshoot on a small distance like the turning radius of the robot rotating on the spot, is proportionally larger and therefore more noticeable.

### **3. Node Shutdown Task**

- Does your program exit properly after you finish all the above tasks? Is it still running even if you shut down the program? If this is the case, think about how to resolve this issue.
  - No, my program exits correctly after every task is done running. This is because I have `on_shutdown` set to run when my move node stops running. And the node stops running when I complete my `run()` function. I also call the node shutdown function when `rospy.on_shutdown()` is triggered in case my code crashes. This causes the shutdown to be run twice.

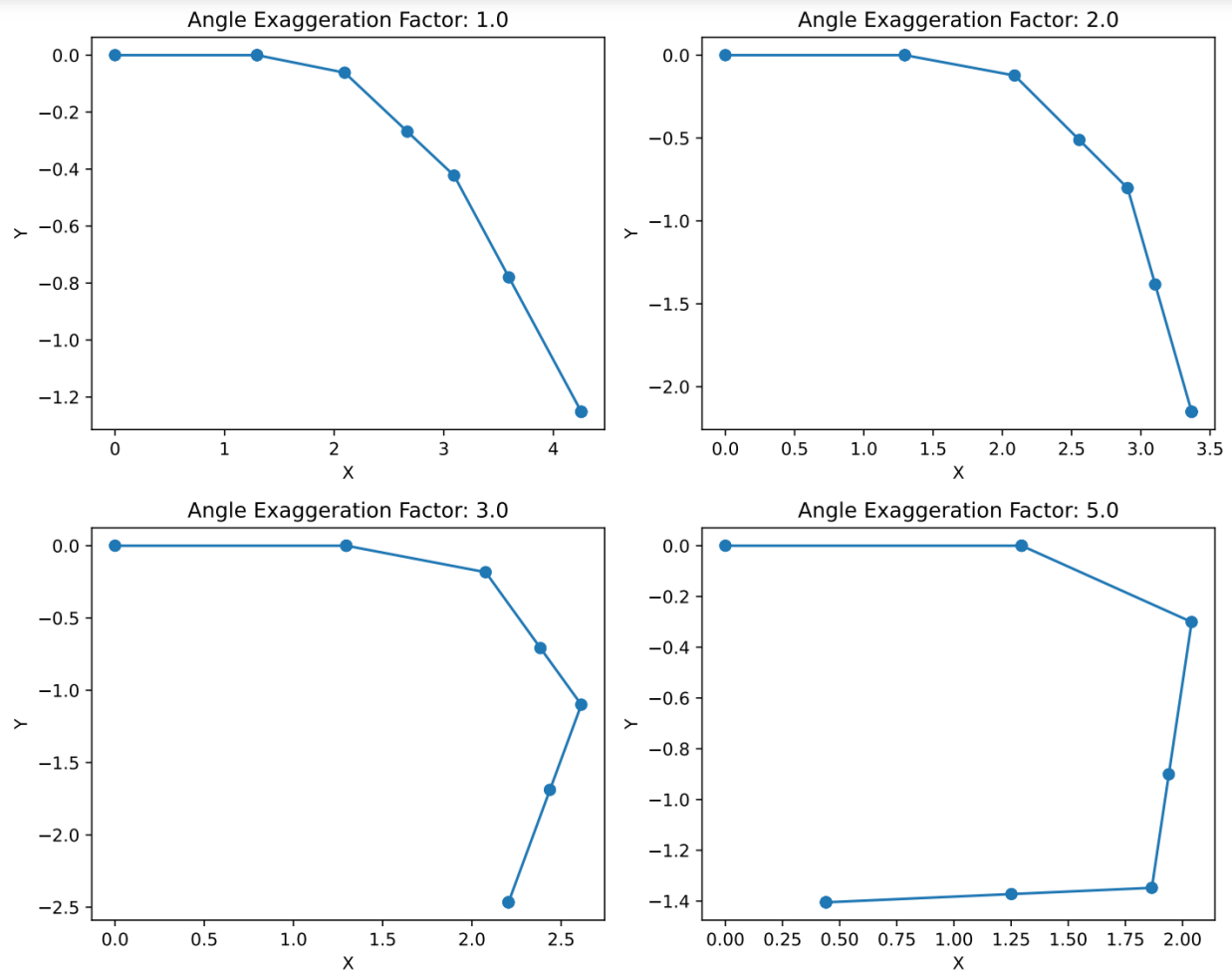
### **4. Plotting Task (ROS Bag)**

- This task I collected data from my `wheel_encoder` node in the form of velocities written to each wheel. Because those velocities are so uniform, the plot looks really uniformly spaced when compared to the real paths.

## **PART 3**

### **5. CREATING A SERVICE IN ROS**

- Please add a tutorial for this next time. I spent two hours working on this when it should have taken 5 minutes because of how simple it really is. I created a service message in my `/srv` folder in my package. I then added it to the `CMakeLists.txt` file so it could be generated correctly. Why doing it this way instead of just publishing directly to the topic controlling the LED's, I am not sure. But overall, now I know how to do it.
- For my program: RED is stage 1 where we are waiting, Green is the intermediate stage where we are drawing the D, and then we go back to Red once we return to the beginning



**Fig. 4.** ROSBag Plots showing trajectory with a theta exaggeration applied to show the path better

- The last plot shows best what happened but the first segment from point 1 to point 2 was the longest straight line in my code. Then we have the rotation on the spot by 90 degrees which is the transition between segment 1 and two. Then we have the curve that we took which was changing theta for longer meaning it was more exaggerated. Then we had a small straight segment before again curving for another transition between segments. And then finally we had another short straight segment to return to the beginning. All of it is explainable in the code.
- The total execution time from startup to shutdown including all the pauses was 27.44 seconds.

I found that the hardest part of this lab was tuning my parameters for my robot so it could actually complete the correctly. The original code did not work as I thought, or at least the curving and rotating were happening either too much or not enough. So I had to scale the effects of rotation to get my robot to work properly. Surprisingly after the initial trim of the wheels, the robot had no problems going straight really. In fact, the slight offset from the off center driving corrected for some other mistakes in the rotation of the robot. It ended up being a consistent error so it just would sometimes cancel out and fix itself or multiply and ruin everything. Interestingly, the over rotation I tuned for while turning clockwise, developed into under rotation when turning counterclockwise. The overall takeaway from this lab, is that using kinematics and absolute positioning or just estimated positioning is a really horrid way to control a robot.

## **BONUS**

1. I made my code so that I could reverse easily by passing in velocities directly. The same was for my rotation and curve code essentially. Therefore, no problems there. The largest problem was that counterclockwise rotation was short because I had overcorrected for the clockwise rotation. However, for the final shape (a square) it did not pose a problem since I just rotated clockwise the whole time. But it was interesting to see that the problem was not universal, which leads to me to believe that the over rotation is a hardware problem rather than a software problem.