Lab 2
Adding Suction Feedback
Goal: Implement Suction Feedback

• We want to subscribe to the information about Suction Feedback
  • This means we want to know about a digital or analog input
• There is a one other subscribe in our code, so let’s use that as a guide to help us

• Important commands
  • rostopic list
  • rostopic info <topic_name>
  • rostopic echo <topic_name>
  • rosmsg list
  • rosmsg info <message_name>
How does a Subscriber work?

```cpp
ros::Subscriber sub_position=nh.subscribe("ur3/position",1,position_callback);
```

• We can see that the function takes three arguments:
  • A topic name
  • A queue size
  • A callback function

• The callback function is called each time the topic is published

• We assign the function return to “sub-position,” but we don’t use this variable elsewhere in the code

The Callback Function

```c
void position_callback(const ece470_ur3_driver::positions::ConstPtr& msg)
{
    isReady=msg->isReady; // When isReady is True the robot arm has made it to its desired position
    pending=msg->pending; // pending is the opposite of isReady, pending is true until a new position is reached
    // ROS_INFO("Debug isRdy = %d, pending = %d",(int)isReady,(int)pending);
}
```

- This function takes **const ece470_ur3_driver::positions::ConstPtr& msg** as the argument.
- It then passes the value of data into global variables (isReady and pending)
- The data is stored in a data structure passed in as **msg**
How can we find this data?

- If we didn’t have this data how could we find it?
- Let’s explore using the commands from before
  - rostopic list
  - rostopic info <topic_name>
  - rostopic echo <topic_name>
  - rosmsg list
  - rosmsg info <message_name>
  - Note: show and info work the same
- This process is done while running ROS in another terminal
  - ros launch ece470_ur3_driver ece470_ur3_driver.launch
rostopic list

• This gives a list of all the topics
rostopic info /ur3/position

- This gives more information about a specific topic
- Note that we can see the data type used for the callback function
- This is also the name of the message
rosmsg list

• We can see the message here in `rosmsg list`
rosmsg info ece470_ur3_driver/positions

• This shows us all the members of the message data structure
• We can see
  • bool isReady
  • bool pending
rostopic echo /ur3/position

- echo, allows us to see the values of the topic
- Note that we can see the current values of isReady and pending
rostopic echo /ur3/position/isReady

• We can look at elements within the data structure as well
• rostopic echo /ur3/position/isReady -n 1
  • This allows us to echo only one instance of data instead of streaming it
Putting it all together

ros::Subscriber sub_position=nh.subscribe("ur3/position",1,position_callback);

• We know we want the values of `isReady` and `pending`  
• By searching the topics, we found the values in topic `/ur3/positions`  
• We can now create our subscriber function  
• We assign it to a convenient variable (`sub_position`)  
• Queue size = 1 (only keeps the most recent element in the queue)  
• We select an appropriate callback function name (`position_callback`)
The Callback function

```c
void position_callback(const ece470 Ur3_driver::positions::ConstPtr& msg)
{
    isReady=msg->isReady; // When isReady is True the robot arm has made it to its desired position
    // and is ready to be told to go to another point if desired.
    pending=msg->pending; // pending is the opposite of isReady, pending is true until a new position is reached
    ROS_INFO("Debug isRdy = %d, pending = %d",(int)isReady,(int)pending);
}
```

• We learned the data type and pass it as a constant pointer (`const ece470 Ur3_driver::positions::ConstPtr& msg`)
• We create global variables to receive the information update (`isReady` and `pending`)
• We extract the needed data from the data structure with:
  • `msg->isReady`
  • `msg->pending`
Questions to answer for suction feedback

• What is the topic?
• What is the data type?
• What is the name of the variable?
• Where is data we want in the data structure?
  • Note: There are two solutions to this question: An analog and a digital one.
• How does the data change?
Applying to suction feedback

• Create a subscriber function
• Create a callback function
• Implement it into your code and deal with the resulting error
• Remember that published data is only updated when a cycle occurs

ros::spinOnce();
loop_rate.sleep();