Lab3: Bayes Filter

Camera Sensor and Image Processing Node

The Turtlebot 3 robot model that we use is equipped with a monocular camera, which we can use to sense the environment. The `prob_rob_labs_ros_2` package that was provided to you has a simple vision processing node that we will use in this lab to determine the state of the door. Start the simulation, but this time we will use a command-line parameter to include the vision processor in the set of nodes to spawn:

```
ros2 launch prob_rob_labs turtlebot3_and_door_launch.py run_vision_processor:=true
```

The vision processing node subscribes to the video-signal topic produced by the camera and runs a Shi-Tomasi corner-detection algorithm available in a widely-used OpenCV library. If you are interested in learning more about this algorithm, read this paper (you can access it through IEEEXplore using institution login; Look up Columbia University and log in using your UNI):

https://ieeexplore.ieee.org/document/323794

If you find this topic interesting, consider taking computer vision or image processing courses.

To see the extracted features, type the following:

```
ros2 run rqt_image_view rqt_image_view /goodfeature/image_raw
```

A new window will open showing the first-person view of the robot camera along with the extracted features. Besides the annotated image (which is primarily used for humans to visualize the detection), the vision-processing node also publishes the list of coordinates of detected features. These are more suitable for machine processing. Let's look at the topic:

```
ros2 topic echo /goodfeature/corners
```

As you can tell, the published data is a list of (x, y) coordinates in the pixel space of detected features. This topic feeds into another node that calculates the mean x-coordinate of all detected features and publishes it to `/feature_mean` topic. To see the data coming from the topic type this:

```
ros2 topic echo /feature_mean
```

You can also plot the signal associated with the topic in real time like this:

```
ros2 run rqt_plot rqt_plot /feature_mean
```

You can also visualize the relationship between the nodes and topics. Type this and select to display nodes and topics and use the graph to answer the question in Assignment 1:

ros2 run rqt_graph rqt_graph

Assignment 1: How many nodes (including the simulation model) does the vision pipeline that produces /feature_mean topic consist of? List node names.

Now that you understand how the vision pipeline works, let's try a few experiments to understand the incoming data. Open the door (using the hinge-torque topic) and see how the feature-mean signal changes. The signal levels should be very distinguishable and it should be easy to define a threshold that can be used to tell whether the door is open or closed. We don't need probabilistic methods for that.

Assignment 2: Explain in your own words how the detection of the open door works using the feature-mean signal. Think about which features the vision algorithm picks up and what happens to the features as the door opens and closes.

Assignment 3: Open and close the door a few times and capture the feature-mean signal on the plot. Take a screenshot and use the plot to determine the threshold. Include the annotated plot in your lab report.

Assignment 4: Modify the node from the previous lab that instead of waiting for some arbitrary time for the door to presumably open, uses the feature-mean signal to check that the door is open before proceeding forward. Submit the code.

Hint: to solve the Assignment 4, create a subscriber with its own callback, store the incoming data into the class attribute and use it in your state machine to compare with the threshold to decide when to move.

Background Noise and Uncertainty

Stop the current simulation and start the new one by typing this:

```
ros2 launch prob_rob_labs turtlebot3_door_and_firetruck_launch.py \
      run_vision_processor:=true
```

Now we have a fire truck visible through the door and our feature detector picks up on it, which creates the noise. The thresholds are still distinguishable but there are noise spikes that can

cause confusion. Make sure you watch the signal long enough and that you open and close the door multiple times to spot the problem.

Assignment 5: Write the node that subscribes to the feature-mean topic and reports for a given time window and given threshold value how many times the signal was above and below the threshold. Use the measurements to estimate the four conditional probabilities: P(z=open | x = closed), P(z=closed | x = open), P(z=open | x=closed), and P(z=closed | x=closed). Submit the plot that justifies your selection of the threshold and mark example measurements that can create confusion.

Assignment 6: Extend the control node to use Bayesian inference to decide if the door is open. The node should estimate the probability of the door being open and compare it with a threshold. The threshold is a tradeoff between not bumping into the door and the motion delay. Good values are 0.999, 0.9999, or 0.99999, etc. Submit the code and calculated beliefs during the transition.

Full Bayes Filter

The filter you designed in the previous section uses Bayesian inference to deal with the noisy measurements, but the door handling is still deterministic. There is no prediction step. Consider the case when the door may not open reliably. We now need a full Bayes filter.

Stop the current simulation and start it with one additional parameter:

```
ros2 launch prob_rob_labs turtlebot3_door_and_firetruck_launch.py \
   run_vision_processor:=true run_door_opener:=true
```

This will start a new node called `/flaky_door_opener` that will create a new topic called /door_open. If you publish an empty type to this topic the door may or may not open at random:

ros2 topic pub -1 /door_open std_msgs/msg/Empty

The exact reason why the door would behave this way is not very important. You can assume that we are modeling a controller connected to the network and that the command message can be lost in transmission.

Assignment 7: Publish to the door-open topic multiple times and record if the door has opened or not. If the door opens, close it by publishing to the hinge-torque topic and continue to publish to the door-open topic. Do enough measurements to estimate the probability of door opening when commanded. Find the Flaky Door Opener code in the repository and deduce from the code what the actual probability is. Report both numbers.

Assignment 8: Design the controller that attempts to open the door using the door-opener topic, and estimates the probability of the door being opened using the Bayes filter. Use the probabilities you measured to design the prediction and measurement model. The robot should move if the estimated probability of the door being open is 0.99 or higher. Submit the code and demonstrate to the instructor that your estimator works.