CSCE 452/752 - Project 3: Group 11

How the program works

Our launch file is used to launch the various nodes and execute the commands that keep our program together. It takes in the parameters bag_in and bag_out and uses them to execute ros2 commands to play the input bag and record all published data as well as that original scan data to the output bag.

1. **start.launch.py**

   The launch file we used to configure and launch the nodes and processes within our project. The main purpose of the launch file is to play and record ROS bag files, while also running scan_subscriber_node and count_and_track_node.

- DeclareLaunchArgument: This is used to declare two launch arguments, **rosbag_in** and **rosbag_out**, with default values and descriptions. These arguments are used to specify the input and output bag files, respectively.
- Two nodes are defined, **scan_subscriber_node** and **count_and_track_node**, which are part of the **project3** package. These nodes will be launched when the launch file is executed.
- **play_bag**: This process is used to play a ROS bag file. It uses the ros2 bag play command and takes the bag_in argument as input.
- **record_bag**: It is used to record a ROS bag file using the ros2 bag record command and takes the bag_out argument as the output file.
- **RegisterEventHandler**: This block registers an event handler for the play_bag process. When the play_bag process exits, it performs several actions, it emits the shutdown event to stop out code from executing.

2. **scan_subscriber_node.py**

This subscribes to laser scan data and processes it to calculate (x, y) coordinates, and publishes the processed data as a PointCloud message for the count_and_track node to use.

- ScanSubscriber class inherits from Node.
- The constructor initializes the node with the name "scan_subscriber."
  - It subscribes to **/scan**: It subscribes where LaserScan messages are published and has a callback method laser_scan_callback to process laser scan data.
  - It creates a publisher for sending processed laser point data to the /current_laser_points topic using PointCloud messages.
  - It initializes a list called current_laser_points to store the laser points

- **send_current_laser_point** method prepares and publishes PointCloud messages containing the current laser points.
  - It creates PointCloud message, sets header with timestamp, frame_id and current laser points.
  - It then publishes the PointCloud message and increments the timestamp.
- **laser_scan_callback** method is called when laser scan data is received on the /scan topic.
  - It extracts relevant information from the LaserScan message, including the angles and ranges.
  - It calculates (x, y) coordinates of each point in the laser frame based on angle and distance.
  - It creates PointCloud messages for each point and appends them to the current_laser_points list.
  - After processing all points, it calls send_current_laser_point.

3. **count_and_track_node.py**

   This node receives the data published by scan_subscriber_node on the and /current_laser_points topic and uses that data to calculate the current location of people and the total number of people seen. It publishes this information on two topics: /person_locations and /person_count. It performs clustering to group laser points into objects and tracks their motion over time to get this data. In the constructor, the node is initialized, publishers for **/person_locations** and **/person_count** are created, a timer for the timer_callback function is set up, and various variables for tracking people and their movements are initialized.

- The **current_laser_points_callback** function is called whenever new laser scan data is received on the /current_laser_points topic. It processes the laser scan data, clusters the points using **DBSCAN** (Density-Based Spatial Clustering of Applications with Noise), and detects moving objects. The results are stored in self.clustered_points and self.moving_objects.
- The **timer_callback** function is executed at a regular interval specified by timer_period. It publishes the current location of people (or moving objects) on the /person_locations topic and the total count of people on the /person_count topic. The moving objects are tracked based on their centroids.
- The **cluster_points** function uses DBSCAN to cluster the laser points into groups, where each group represents a potential object or person. The centroids of these clusters are returned as a list of PointCloud messages.
- The **detect_motion** function takes the centroids of current laser points and detects moving objects by comparing them with the centroids of the previous frame. It updates the self.tracked_people dictionary to track the positions of moving objects. If the moving

object has been moving for more frames than the number defined in self.frames_bef_person and the distance moved is between self.min_motion_threshold and self.max_motion_threshold it will save that centroid as a person.

- The **match_to_tracked_person** function attempts to match a centroid to a previously tracked person based on proximity. If a match is found, it updates the tracked person's position.
- The **remove_disappeared_people** function removes people from the tracking dictionary if they haven't been detected for too long. It uses a count to keep track of how long a person has disappeared for.

4. **Parameters**

The code uses several parameters that influence the behavior of the object tracking system.

- eps and min_samples for DBSCAN clustering:
  - eps defines the maximum distance between two samples to be considered as in the neighborhood of the other.
  - min_samples specifies the number of samples in a neighborhood for a point to be considered a core point.
  - The values of eps and min_samples (0.5 and 5, respectively) are initially set to work well with the specific data and object sizes the code was designed for. The selection of these values often involved experiments and observation of the clustering results on the provided nine examples.
- min_motion_threshold and max_motion_threshold:
  - They are used to define the minimum and maximum distances between the current centroid and the closest previous centroid for an object to be considered as moving.
  - These thresholds depend on the physical characteristics of the objects being tracked and the sensor's precision. These values were adjusted based on how fast the objects moved and the noise in the sensor data via experimentation and observation.
- frames_bef_person:
  - It specifies the number of consecutive frames an object needs to be moving to be considered a person. This parameter helps in distinguishing between stationary objects and people.
  - The value of frames_bef_person is set to 7, which means an object must move for 7 consecutive frames to be recognized as a person. This value was adjusted based on experimentation and observation.
- disappearance_count_threshold:

    - It is the count of frames after which a tracked person is considered disappeared if not detected.
    - The threshold value of 100 is a value we found after testing with the bag files.
- Thresholds for matching and tracking:
    - The code uses a maximum distance threshold of 1.0 to match the current centroid to a tracked person's last position. If the distance between the current centroid and the last position is less than or equal to this threshold, the centroids are considered a match.
    - These threshold values were selected based on experiments and observations.

Did the results meet your expectations? Why or why not?

To some extent the results did meet our expectations. Our system is able to track and count moving objects in the given examples fairly well. However, it turned out to be far more difficult to differentiate the noise in the samples from people moving. So our results aren't as clean as we hoped they would be.

Team Members:

Jonas Land
Ayushri Jain