NDT Evaluation #4175

✓ Answered by SakodaShintaro charan-rs asked this question in Q&A

Category

🙏 Q&A

Labels

component:localiza...

2 participants

charan-rs on Feb 14

Hello everyone,

I am at the point where NDT is running on my dataset but I am trying to evaluate NDT by varying its parameters and I am using EVO for plotting. If there are any other suggestions to evaluate NDT Localization, any suggestions would help.

How can we save diagnostics info as a .csv file? Or how can we get the NDT execution time information? I wanted to make a parametric study of NDT on my data.

Time duration for creating new ndt_ptr: 41.659000 [ms]
What is the difference between this time and the output from /localization/pose_estimator/exe_time_ms
And how can we plot `/localization/pose_estimator/exe_time_ms` using evo?

I also want to evaluate the score/transform probability w.r.t time. How can we use EVO to plot any topic w.r.t time?

↑ 1

✓ Answered by **SakodaShintaro** on Feb 14

Hello.
Sorry, I'm not familiar with EVO, so I'll only answer as much as I can.

- To output diag information regarding ndt to csv, use the following command.

View full answer ↓

4 comments · 12 replies

Oldest Newest Top

SakodaShintaro on Feb 14 Collaborator

Hello.
Sorry, I'm not familiar with EVO, so I'll only answer as much as I can.

- To output diag information regarding ndt to csv, use the following command.

```
ros2 topic echo /diagnostics diagnostic_msgs/msg/DiagnosticAr
--csv --qos-depth 5000 | grep ndt_scan_matcher > out.csv
```

- I usually save topics such as `/localization/pose_estimator/exe_time_ms` in a rosbag, and then load the rosbag with python to create graphs etc. Please execute the following command in advance in the terminal where you will run the python script.

```
source ~/autoware/install/setup.bash
```

```python
import rosbag2_py
from rclpy.serialization import deserialize_message
from rosidl_runtime_py.utilities import get_message
import argparse
import matplotlib.pyplot as plt
import numpy as np
from collections import defaultdict


def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('rosbag_path', type=str)
    return parser.parse_args()


if __name__ == "__main__":
    args = parse_args()
    rosbag_path = args.rosbag_path
    serialization_format = 'cdr'
    storage_options = rosbag2_py.StorageOptions(
        uri=rosbag_path, storage_id='sqlite3')
    converter_options = rosbag2_py.ConverterOptions(
        input_serialization_format=serialization_format,
        output_serialization_format=serialization_format)

    reader = rosbag2_py.SequentialReader()
    reader.open(storage_options, converter_options)

    topic_types = reader.get_all_topics_and_types()
    type_map = {
        topic_types[i].name: topic_types[i].type for i in range(len

    target_topic = "/localization/pose_estimator/exe_time_ms"
    storage_filter = rosbag2_py.StorageFilter(topics=[target_topic]
    reader.set_filter(storage_filter)

    name_to_appear_num = defaultdict(int)
    name_to_time_list = defaultdict(list)

    while reader.has_next():
        (topic, data, t) = reader.read_next()
        msg_type = get_message(type_map[topic])
        msg = deserialize_message(data, msg_type)
        print(msg)
```

- "Time duration for creating new ndt_ptr:" indicates the time it takes to rebuild the KDTree in NDT by dynamic map updates. It is not the time it takes to process NDT on each frame.

I hope this information helps you.

✓ Marked as answer ↑ 1 👍 1 1 reply

charan-rs on Feb 14 Author

Yes, this helps.. Thanks a lot :)

👍 1

Answer selected by **charan-rs**

charan-rs on Feb 16 Author

Hello,

I hope posting another question related to NDT evaluation here is not a problem.. :)

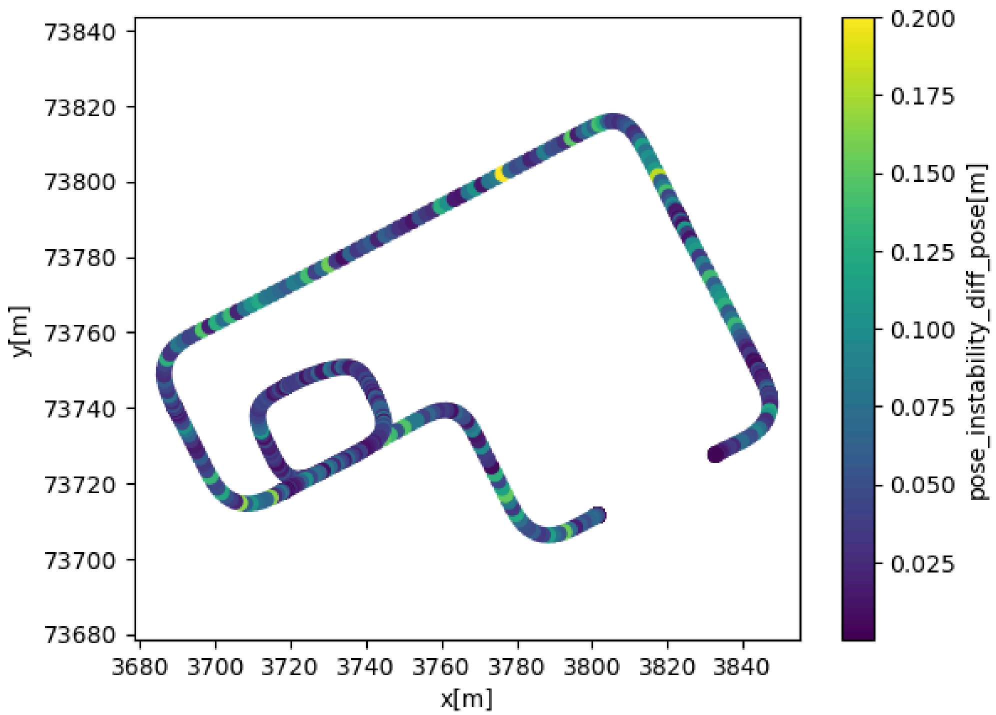
So, right now I am getting result plots with respect to time (for example with execution times, pose errors). I want to get more information from my graphs. I want to plot the errors, score or execution times along the trajectory of the vehicle to understand at what point in the map, the information varies. Can any one suggest or give me views on how this can be carried out?

↑ 1 👀 1 0 replies

SakodaShintaro on Feb 18 Collaborator

We are also looking at ways to visualize localization results.

Today I created a script and would like to share it with you.
Although I am targeting EKF rather than NDT, I hope this will be helpful.



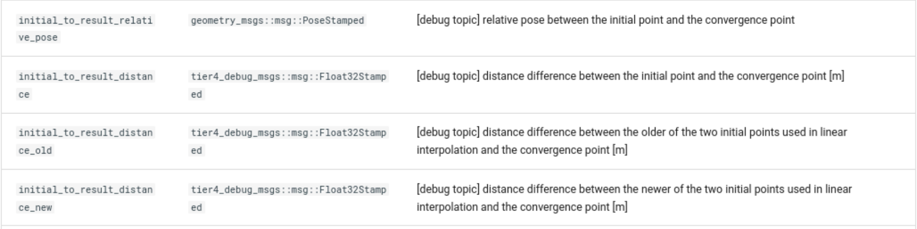
▶ A script to plot

We hope to be able to properly organize and publish such evaluation tools in the future.
Stay tuned!

↑ 1 👍 1 1 reply

charan-rs on Feb 19 Author

Thank you so much.. This is exactly what I was looking for..



`initial_to_result_relative_pose`	`geometry_msgs::msg::PoseStamped`	[debug topic] relative pose between the initial point and the convergence point
`initial_to_result_distance`	`tier4_debug_msgs::msg::Float32Stamped`	[debug topic] distance difference between the initial point and the convergence point [m]
`initial_to_result_distance_old`	`tier4_debug_msgs::msg::Float32Stamped`	[debug topic] distance difference between the older of the two initial points used in linear interpolation and the convergence point [m]
`initial_to_result_distance_new`	`tier4_debug_msgs::msg::Float32Stamped`	[debug topic] distance difference between the newer of the two initial points used in linear interpolation and the convergence point [m]

I am a little bit confused on how I should modify it for NDT... In the picture.. I am quite confused about the topics.. So is there an equivalent topic for NDT like `/localization/pose_twist_fusion_filter/pose_instability_detector/debug/diff_pose`
and for `/localization/pose_twist_fusion_filter/kinematic_state` can we use `/localization/kinematic_state`?

SakodaShintaro on Feb 19 Collaborator

Pose to display the trajectory

In the previous script, I used `/localization/pose_twist_fusion_filter/kinematic_state`, but I think `/localization/pose_estimator/pose` or `/localization/pose_estimator/pose_with_covariance` would also work. I think that's the appropriate way to look at NDT.

Topic for coloring

In the previous script, the trajectory was colored by the norm of `/localization/pose_twist_fusion_filter/pose_instability_detector/debug/diff_pose`, but I think anything is fine here.

The norm of `/localization/pose_estimator/initial_to_result_relative_pose` can also be used to evaluate NDT instability, and coloring by `/localization/pose_estimator/iteration_num` and `/localization/pose_estimator/exe_time_ms` seems like a good idea.

If you simply want to see the quality of NDT, you may find something by coloring with `/localization/pose_estimator/nearest_voxel_transformation_likelihood`.

↑ 1

10 replies

Show 5 previous replies

SakodaShintaro on Feb 21 Collaborator

Sorry, in my environment I was using the interpolate_pose function shown below in a separate file.
https://github.com/orgs/autowarefoundation/discussions/4175#discussioncomment-8511567

Please copy and paste or create a file in the same way.

👍 1

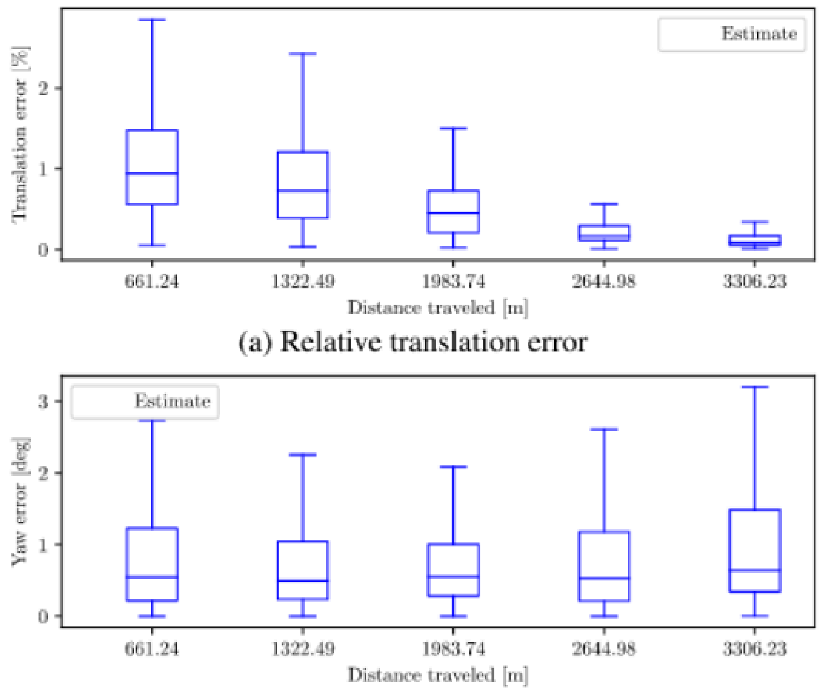
charan-rs on Feb 21 Author

Yes it works fine now. Thank you so much :)

charan-rs on Feb 23 Author

Hello..

So i wanted to evaluate NDT and reproduce the plots as in the figure.



(a) Relative translation error

(b) Relative yaw error

The code below almost does the job but I am not able to use the same units for visualizing the data. It would be really nice if someone can help me get the exact box plots as in the figure because i think its a very nice plot to evaluate NDT along the trajectory.

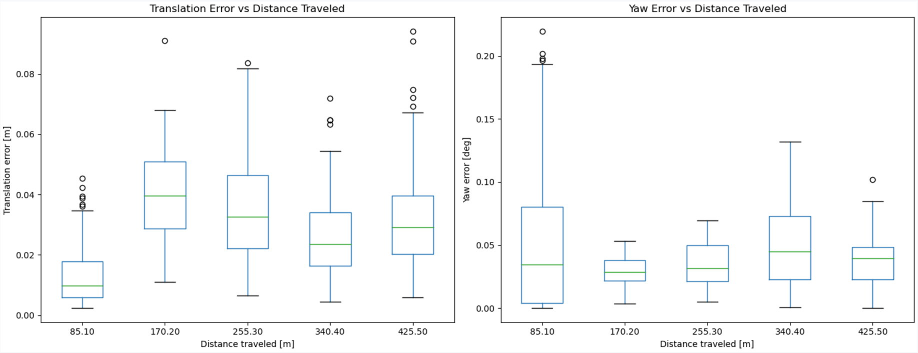
```
import numpy as np
import matplotlib.pyplot as plt
import tf_transformations
import rosbag2_py
import argparse
import pathlib
from rosidl_runtime_py.utilities import get_message
from rclpy.serialization import deserialize_message
import pandas as pd
from scipy.spatial.transform import Rotation, Slerp

def euler_from_quaternion(quaternion):
    return
tf_transformations.euler_from_quaternion([quaternion.qx,
quaternion.qy, quaternion.qz, quaternion.qw])

def interpolate_pose(df_pose: pd.DataFrame,
target_timestamp: pd.Series) -> pd.DataFrame:
    POSITIONS_KEY = ['x', 'y', 'z']
    ORIENTATIONS_KEY = ['qw', 'qx', 'qy', 'qz']
    target_index = 0
    df_index = 0
    data_dict = {
        'x': [],
        'y': [],
        'z': [],
        'qx': [],
        'qy': [],
        'qz': [],
        'qw': [],
        'timestamp': [],
    }
    while df_index < len(df_pose) - 1 and target_index <
len(target_timestamp):
        curr_time = df_pose.iloc[df_index]['timestamp']
        next_time = df_pose.iloc[df_index + 1]['timestamp']
        target_time = target_timestamp[target_index]

        # Find a df_index that includes target_time
        if not (curr_time <= target_time <= next_time):
            df_index += 1
            continue

        curr_weight = (next_time - target_time) /
(next_time - curr_time)
        next_weight = 1.0 - curr_weight

        curr_position = df_pose.iloc[df_index]
[POSITIONS_KEY]
        next_position = df_pose.iloc[df_index + 1]
[POSITIONS_KEY]
        target_position = curr_position * curr_weight +
next_position * next_weight

        curr_orientation = df_pose.iloc[df_index]
[ORIENTATIONS_KEY]
        next_orientation = df_pose.iloc[df_index + 1]
[ORIENTATIONS_KEY]
        curr_r = Rotation.from_quat(curr_orientation)
        next_r = Rotation.from_quat(next_orientation)
        slerp = Slerp([curr_time, next_time],
                      Rotation.concatenate([curr_r,
next_r]))
        target_orientation = slerp([target_time]).as_quat()
[0]


data_dict['timestamp'].append(target_timestamp[target_index])
        data_dict['x'].append(target_position[0])
        data_dict['y'].append(target_position[1])
        data_dict['z'].append(target_position[2])
        data_dict['qw'].append(target_orientation[0])
        data_dict['qx'].append(target_orientation[1])
        data_dict['qy'].append(target_orientation[2])
        data_dict['qz'].append(target_orientation[3])
        target_index += 1
    result_df = pd.DataFrame(data_dict)
    return result_df

def parse_args():
    parser = argparse.ArgumentParser(description="ROS2 Bag
file analyzer for NDT and GNSS pose data.")
    parser.add_argument('rosbag_path', type=pathlib.Path,
help="Path to the ROS2 bag file")
    return parser.parse_args()

if __name__ == '__main__':
    args = parse_args()
    rosbag_path = args.rosbag_path
    serialization_format = 'cdr'
    storage_options = rosbag2_py.StorageOptions(
    uri=str(rosbag_path), storage_id='sqlite3')
    converter_options =
rosbag2_py.ConverterOptions(input_serialization_format=seriali
output_serialization_format=serialization_format)

    reader = rosbag2_py.SequentialReader()
    reader.open(storage_options, converter_options)

    topic_types = reader.get_all_topics_and_types()
    type_map = {topic_types[i].name: topic_types[i].type
for i in range(len(topic_types))}

    target_topics = ["/localization/pose_estimator/pose",
"/sensing/gnss/pose"]
    storage_filter =
rosbag2_py.StorageFilter(topics=target_topics)
    reader.set_filter(storage_filter)
    ndt_data = list()
    gnss_data = list()

    while reader.has_next():
        (topic, data, t) = reader.read_next()
        msg_type = get_message(type_map[topic])
        msg = deserialize_message(data, msg_type)
        timestamp_header = int(msg.header.stamp.sec) + \
            int(msg.header.stamp.nanosec) * 1e-9
        if topic == '/localization/pose_estimator/pose':
            pose = msg.pose
            ndt_data.append({
                'timestamp': timestamp_header,
                'x': pose.position.x,
                'y': pose.position.y,
                'z': pose.position.z,
                'qw': pose.orientation.w,
                'qx': pose.orientation.x,
                'qy': pose.orientation.y,
                'qz': pose.orientation.z,
            })
        elif topic == '/sensing/gnss/pose':
            pose = msg.pose
            gnss_data.append({
                'timestamp': timestamp_header,
                'x': pose.position.x,
                'y': pose.position.y,
                'z': pose.position.z,
                'qw': pose.orientation.w,
                'qx': pose.orientation.x,
                'qy': pose.orientation.y,
                'qz': pose.orientation.z,
            })
        else:
            assert False, f"Unknown topic: {topic}"

    # Processing data
    df_ndt_pose = pd.DataFrame(ndt_data)
    df_gnss_pose = pd.DataFrame(gnss_data)

    # Synchronize timestamps (interpolating GNSS data to
match NDT timestamps)
    df_gnss_interpolated = interpolate_pose(df_gnss_pose,
df_ndt_pose['timestamp'])

    min_length = min(len(ndt_data), len(gnss_data))
    translation_error = []
    yaw_error = []
    distance_traveled = [0]

    for i in range(1, min_length):
        ndt_pos = df_ndt_pose.iloc[i][['x', 'y', 'z']]
        gnss_pos = df_gnss_interpolated.iloc[i][['x', 'y',
'z']]
```

```
        translation_error.append(np.linalg.norm(ndt_pos -
gnss_pos))

        ndt_yaw =
euler_from_quaternion(df_ndt_pose.iloc[i])[2]
        gnss_yaw =
euler_from_quaternion(df_gnss_interpolated.iloc[i])[2]
        yaw_error.append(abs(ndt_yaw - gnss_yaw))

        prev_ndt_pos = df_ndt_pose.iloc[i-1][['x', 'y',
'z']]
        distance_traveled.append(distance_traveled[-1] +
np.linalg.norm(ndt_pos - prev_ndt_pos))

    num_subdivisions = 5
    max_distance = max(distance_traveled)
    distance_interval = max_distance / num_subdivisions

    # Categorize the distance traveled into subdivisions
    distance_categories = np.digitize(distance_traveled,
np.arange(0, max_distance, distance_interval))

    # Ensure that the arrays are of the same length
    min_length = min(len(distance_categories),
len(translation_error), len(yaw_error))
    distance_categories = distance_categories[:min_length]
    translation_error = translation_error[:min_length]
    yaw_error = yaw_error[:min_length]

    # Now create the DataFrames
    df_translation_error = pd.DataFrame({
        'DistanceCategory': distance_categories,
        'TranslationError': translation_error
    })

    df_yaw_error = pd.DataFrame({
        'DistanceCategory': distance_categories,
        'YawError': yaw_error
    })

    # Create a single figure for both subplots
    fig, axs = plt.subplots(1, 2, figsize=(15, 6))  # 1
row, 2 columns

    # Translation Error Box Plot
    df_translation_error.boxplot(column='TranslationError',
by='DistanceCategory', grid=False, ax=axs[0])
    axs[0].set_title('Translation Error vs Distance
Traveled')
    axs[0].set_xlabel('Distance Traveled Category')
    axs[0].set_ylabel('Translation Error')

    # Yaw Error Box Plot
    df_yaw_error.boxplot(column='YawError',
by='DistanceCategory', grid=False, ax=axs[1])
    axs[1].set_title('Yaw Error vs Distance Traveled')
    axs[1].set_xlabel('Distance Traveled Category')
    axs[1].set_ylabel('Yaw Error')

    # Adjust layout
    plt.tight_layout()
    plt.show()
```

👀 1

SakodaShintaro on Feb 26 Collaborator

I don't know what is the percentage of translation error.
So I plot simply error [m].
How about this?



I am implementing evaluation scripts at
autowarefoundation/autoware_tools#10
Please see plot.py in the PR.

👍 1

charan-rs on Feb 26 Author edited ▾

Yes.. This works too.. Thank you so much :)
You mean *plot_box.py* right?