

# vo\_\_benchmark

May 10, 2024

Notebook for two-view reconstruction without inertial data.

Author: Andre Schreiber

```
[1]: import symforce
symforce.set_epsilon_to_symbol()

import cv2
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
from scipy.spatial.transform import Rotation as R
import time

import vo
# pose_metrics requires evo (see top of pose_metrics.py for install
  ↳ instructions)
import pose_metrics
import utils

# Note: this notebook requires pandas in addition to all of Prof. Bretl's
  ↳ dependencies
import pandas as pd
```

## 0.0.1 Read data

```
[2]: # Specify the dataset (should be 'kitti' or 'euroc')
chosen_dataset = 'euroc'
# chosen_dataset = 'kitti'

assert(chosen_dataset in ['kitti', 'euroc'])
```

## 0.0.2 Provide settings

```
[3]: # When matching (max threshold for ratio test)
if chosen_dataset == 'euroc':
    matching_threshold = 0.5
else:
    matching_threshold = 0.3

# When deciding if triangulated points are invalid
max_reprojection_err = 0.75

# Temporary folder for evo metrics
temporary_folder = Path('./temp')
temporary_folder.mkdir(parents=True, exist_ok=True)

[4]: if chosen_dataset == 'euroc': # Note: euroc takes a bit longer to load.
    # Use EuRoC MAV

    mav_video_folder = Path('./data/mav0')

    # Read MAV data
    dataset_info = utils.read_data_mav(mav_video_folder)
    print("Read dataset with keys: {}".format(sorted(list(dataset_info.
↪keys()))))

    # Extract relevant data
    cam0_K = dataset_info['cam0_K']
    cam0_distortion = dataset_info['cam0_distortion']
    visual_inertial_data = dataset_info['visual_inertial_data']

    sigma_acc_wn = dataset_info['imu_accelerometer_noise_density']
    sigma_gyr_wn = dataset_info['imu_gyroscope_noise_density']
    sigma_acc_rw = dataset_info['imu_accelerometer_noise_density']
    sigma_gyr_rw = dataset_info['imu_gyroscope_random_walk']

    dt = 1/200 # IMU frequency

    # Get extrinsics
    T_inB_ofC = dataset_info['cam0_extrinsics']
    T_inC_ofB = np.block([[T_inB_ofC[:3,:3].T, (-T_inB_ofC[:3,:3].T @
↪T_inB_ofC[:3,-1])[:3,np.newaxis]], [np.zeros(3), 1]])

    # Collate
    acc_meas, gyr_meas = utils.imu_collate(dataset_info['visual_inertial_data'])
    R_inR_ofB, v_inR_ofB, p_inR_ofB, b_a, b_w = utils.
↪groundtruth_collate(dataset_info['visual_inertial_data'], True)
```

```

    # As EuRoC's ground-truth (MoCap) is not aligned with gravity (i.e., in
    ↪ world frame), we identify the orientation of MoCap frame in world frame
    gravity = np.array([0., 0., -9.81])

    g_inB = - np.mean(acc_meas[:10], axis=0)
    g_inW = gravity

    def align_vectors(g_inB, g_inW):
        # Normalize input vectors
        g_inB_unit = g_inB / np.linalg.norm(g_inB)
        g_inW_unit = g_inW / np.linalg.norm(g_inW)

        # Compute the axis of rotation
        v = np.cross(g_inB_unit, g_inW_unit)

        # Compute the angle of rotation
        cos_theta = np.dot(g_inB_unit, g_inW_unit)
        sin_theta = np.linalg.norm(v)
        theta = np.arctan2(sin_theta, cos_theta)

        v /= np.linalg.norm(v)

        # Compute the rotation matrix
        Rot = R.from_rotvec(theta*v)
        return Rot

    # Compute rotation matrix
    R_inW_ofB = align_vectors(g_inB, g_inW)
    print("Rotation Matrix:\n", R_inW_ofB.as_matrix())

    R_inW_ofB.apply(g_inB)

    R_inW_ofR = R_inW_ofB * R_inR_ofB[:10].mean().inv()
    R_inW_ofB = R_inW_ofR * R_inR_ofB
    v_inW_ofB = R_inW_ofR.apply(v_inR_ofB)
    p_inW_ofB = R_inW_ofR.apply(p_inR_ofB)

else:
    # Use KITTI

    kitti_base_path = './data/kitti'
    kitti_date = '2011_09_26'
    kitti_drive = '0022'

    # Read KITTI data
    dataset_info = utils.read_data_kitti('./data/kitti', '2011_09_26', '0022')

```

```

# Extract relevant data
cam0_K = dataset_info['cam0_K']
cam0_distortion = dataset_info['cam0_distortion']
visual_inertial_data = dataset_info['visual_inertial_data']

T_inC_ofB = dataset_info['cam0_extrinsics']
R_inB_of_C = T_inC_ofB[:3, :3].T
t_inB_of_C = R_inB_of_C @ T_inC_ofB[:3, 3]
T_inB_ofC = np.block([[R_inB_of_C, t_inB_of_C[:3, np.newaxis]], [np.zeros(3),
↪1]])

R_inC_of_B = T_inC_ofB[:3, :3]
t_inC_of_B = T_inC_ofB[:3, 3]

# Collate
acc_meas, gyr_meas = utils.imu_collate(visual_inertial_data)
R_inW_ofB, v_inW_ofB, p_inW_ofB, b_a, b_w = utils.
↪groundtruth_collate(visual_inertial_data, False)

```

Read dataset with keys: ['cam0\_K', 'cam0\_distortion', 'cam0\_extrinsics', 'imu\_accelerometer\_noise\_density', 'imu\_accelerometer\_random\_walk', 'imu\_extrinsics', 'imu\_gyroscope\_noise\_density', 'imu\_gyroscope\_random\_walk', 'visual\_inertial\_data']

Rotation Matrix:

```

[[-0.41384846  0.03948562 -0.90948905]
 [ 0.03948562  0.99889726  0.02539999]
 [ 0.90948905 -0.02539999 -0.41495121]]

```

### 0.0.3 Create random generator

```
[5]: rng = utils.create_rng(42)
```

seeding RNG with 42

### 0.0.4 Create image keypoint feature extractor

```
[6]: feature_extractor = cv2.SIFT_create() # could also do ORB_create() for ORB
↪features
```

### 0.0.5 Two view reconstruction

Get initial solution

```
[7]: if chosen_dataset == 'euroc': # Note: euroc takes a bit longer to load.
      # Use EuRoC MAV
      chosen_index = 500
      advance = 100
    else:
      chosen_index = 50
      advance = 5

    # Get first index closest to chosen index
    first_frame_idx = utils.get_index_of_next_image(visual_inertial_data,
      ↪ chosen_index)
    # Get second index
    second_frame_idx = utils.get_index_of_next_image(visual_inertial_data,
      ↪ first_frame_idx+advance)

    # Create two views
    views = [
        vo.create_view_data(utils.
      ↪ read_image(visual_inertial_data[first_frame_idx]['image_file']),
        first_frame_idx, feature_extractor, cam0_K,
      ↪ cam0_distortion),
        vo.create_view_data(utils.
      ↪ read_image(visual_inertial_data[second_frame_idx]['image_file']),
        second_frame_idx, feature_extractor, cam0_K,
      ↪ cam0_distortion)
    ]

    # Perform two-view reconstruction
    tic = time.time()
    tracks = vo.vo_2view(views, matching_threshold, cam0_K, rng, use_opencv=False)
    toc = time.time()

    analytical_guess = toc - tic
    print(f"Analytical guess: {analytical_guess:.2f} [s]")
```

```
found 118 good matches
found 87 inliers
Analytical guess: 2.40 [s]
```

```
[8]: vo.show_reproj_results(views, tracks, cam0_K, cam0_distortion,
      ↪ print_raw_reproj=True, show_reproj_histogram=True)
    vo.visualize_predictions(views, tracks, cam0_K, cam0_distortion)
```

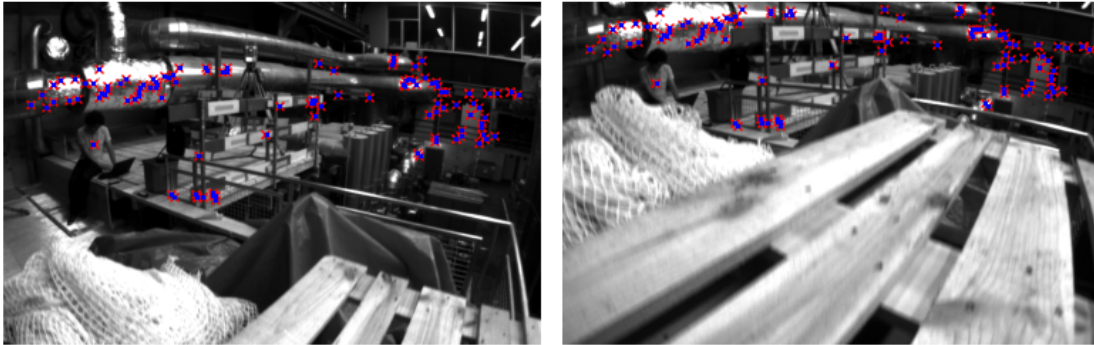
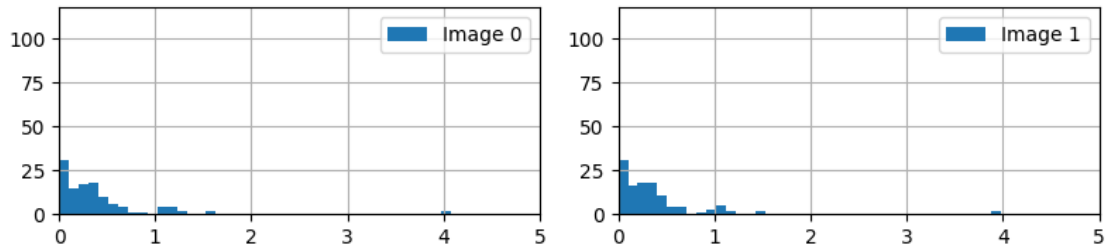
#### REPROJECTION ERRORS

```
Image 0 ( 118 points) : (mean, std, max, min) = (0.4617, 0.7235, 5.1047,
0.0069)
```

Image (raw reprojection) 0 ( 118 points) : (mean, std, max, min) = (0.3597, 0.4605, 2.9180, 0.0043)

Image 1 ( 118 points) : (mean, std, max, min) = (0.4405, 0.7032, 5.0185, 0.0064)

Image (raw reprojection) 1 ( 118 points) : (mean, std, max, min) = (0.2953, 0.3940, 2.6551, 0.0177)



Get post-optimization solution

Run below to keep the initial views and tracks

```
[9]: views_ini = views.copy()
     tracks_ini = tracks.copy()
```

```
[10]: tic = time.time()

     views, tracks, initial_values, results = vo.vo_nonlinear_optimize(views_ini,
     ↪ tracks_ini, cam0_K, max_reprojection_err)

     toc = time.time()
     nonlinear = toc - tic
     print(f"{nonlinear:.2f} [s]")
```

```
[2024-05-10 16:48:26.860] [info] LM<sym::Optimize> [iter    0] lambda:
1.000e+00, error prev/linear/new: 3.183e+01/0.000e+00/2.783e+01, rel reduction:
```

1.25665e-01  
[2024-05-10 16:48:26.876] [info] LM<sym::Optimize> [iter 1] lambda:  
1.000e-01, error prev/linear/new: 2.783e+01/0.000e+00/2.316e+01, rel reduction:  
1.67699e-01  
[2024-05-10 16:48:26.891] [info] LM<sym::Optimize> [iter 2] lambda:  
1.000e-02, error prev/linear/new: 2.316e+01/0.000e+00/2.101e+01, rel reduction:  
9.28426e-02  
[2024-05-10 16:48:26.906] [info] LM<sym::Optimize> [iter 3] lambda:  
1.000e-03, error prev/linear/new: 2.101e+01/0.000e+00/1.941e+01, rel reduction:  
7.60402e-02  
[2024-05-10 16:48:26.921] [info] LM<sym::Optimize> [iter 4] lambda:  
1.000e-04, error prev/linear/new: 1.941e+01/0.000e+00/1.798e+01, rel reduction:  
7.40683e-02  
[2024-05-10 16:48:26.937] [info] LM<sym::Optimize> [iter 5] lambda:  
1.000e-05, error prev/linear/new: 1.798e+01/0.000e+00/1.793e+01, rel reduction:  
2.67787e-03  
[2024-05-10 16:48:26.952] [info] LM<sym::Optimize> [iter 6] lambda:  
1.000e-06, error prev/linear/new: 1.793e+01/0.000e+00/3.512e+01, rel reduction:  
-9.58912e-01  
[2024-05-10 16:48:26.967] [info] LM<sym::Optimize> [iter 7] lambda:  
5.000e-06, error prev/linear/new: 1.793e+01/0.000e+00/1.633e+01, rel reduction:  
8.89506e-02  
[2024-05-10 16:48:26.982] [info] LM<sym::Optimize> [iter 8] lambda:  
5.000e-07, error prev/linear/new: 1.633e+01/0.000e+00/1.984e+01, rel reduction:  
-2.15031e-01  
[2024-05-10 16:48:26.997] [info] LM<sym::Optimize> [iter 9] lambda:  
2.500e-06, error prev/linear/new: 1.633e+01/0.000e+00/1.521e+01, rel reduction:  
6.86078e-02  
[2024-05-10 16:48:27.012] [info] LM<sym::Optimize> [iter 10] lambda:  
2.500e-07, error prev/linear/new: 1.521e+01/0.000e+00/1.593e+01, rel reduction:  
-4.69011e-02  
[2024-05-10 16:48:27.027] [info] LM<sym::Optimize> [iter 11] lambda:  
1.250e-06, error prev/linear/new: 1.521e+01/0.000e+00/1.414e+01, rel reduction:  
7.07830e-02  
[2024-05-10 16:48:27.042] [info] LM<sym::Optimize> [iter 12] lambda:  
1.250e-07, error prev/linear/new: 1.414e+01/0.000e+00/1.443e+01, rel reduction:  
-2.08936e-02  
[2024-05-10 16:48:27.057] [info] LM<sym::Optimize> [iter 13] lambda:  
6.250e-07, error prev/linear/new: 1.414e+01/0.000e+00/1.411e+01, rel reduction:  
1.56368e-03  
[2024-05-10 16:48:27.073] [info] LM<sym::Optimize> [iter 14] lambda:  
6.250e-08, error prev/linear/new: 1.411e+01/0.000e+00/1.419e+01, rel reduction:  
-5.59922e-03  
[2024-05-10 16:48:27.088] [info] LM<sym::Optimize> [iter 15] lambda:  
3.125e-07, error prev/linear/new: 1.411e+01/0.000e+00/1.406e+01, rel reduction:  
3.87824e-03  
[2024-05-10 16:48:27.103] [info] LM<sym::Optimize> [iter 16] lambda:  
3.125e-08, error prev/linear/new: 1.406e+01/0.000e+00/1.410e+01, rel reduction:

```

-2.63327e-03
[2024-05-10 16:48:27.118] [info] LM<sym::Optimize> [iter 17] lambda:
1.563e-07, error prev/linear/new: 1.406e+01/0.000e+00/1.403e+01, rel reduction:
1.86869e-03
[2024-05-10 16:48:27.133] [info] LM<sym::Optimize> [iter 18] lambda:
1.563e-08, error prev/linear/new: 1.403e+01/0.000e+00/1.406e+01, rel reduction:
-1.98572e-03
[2024-05-10 16:48:27.148] [info] LM<sym::Optimize> [iter 19] lambda:
7.813e-08, error prev/linear/new: 1.403e+01/0.000e+00/1.401e+01, rel reduction:
1.55591e-03
[2024-05-10 16:48:27.170] [info] LM<sym::Optimize> [iter 20] lambda:
7.813e-09, error prev/linear/new: 1.401e+01/0.000e+00/1.408e+01, rel reduction:
-4.81495e-03
[2024-05-10 16:48:27.185] [info] LM<sym::Optimize> [iter 21] lambda:
3.906e-08, error prev/linear/new: 1.401e+01/0.000e+00/1.398e+01, rel reduction:
1.83644e-03
[2024-05-10 16:48:27.200] [info] LM<sym::Optimize> [iter 22] lambda:
3.906e-09, error prev/linear/new: 1.398e+01/0.000e+00/1.407e+01, rel reduction:
-6.42715e-03
[2024-05-10 16:48:27.215] [info] LM<sym::Optimize> [iter 23] lambda:
1.953e-08, error prev/linear/new: 1.398e+01/0.000e+00/1.396e+01, rel reduction:
1.54352e-03
[2024-05-10 16:48:27.230] [info] LM<sym::Optimize> [iter 24] lambda:
1.953e-09, error prev/linear/new: 1.396e+01/0.000e+00/1.398e+01, rel reduction:
-1.34267e-03
[2024-05-10 16:48:27.246] [info] LM<sym::Optimize> [iter 25] lambda:
9.766e-09, error prev/linear/new: 1.396e+01/0.000e+00/1.395e+01, rel reduction:
1.03787e-03
[2024-05-10 16:48:27.260] [info] LM<sym::Optimize> [iter 26] lambda:
9.766e-10, error prev/linear/new: 1.395e+01/0.000e+00/1.393e+01, rel reduction:
9.94319e-04
[2024-05-10 16:48:27.276] [info] LM<sym::Optimize> [iter 27] lambda:
9.766e-11, error prev/linear/new: 1.393e+01/0.000e+00/1.393e+01, rel reduction:
2.82039e-04
[2024-05-10 16:48:27.292] [info] LM<sym::Optimize> [iter 28] lambda:
9.766e-12, error prev/linear/new: 1.393e+01/0.000e+00/1.393e+01, rel reduction:
8.11316e-05
1.02 [s]

```

```

[11]: vo.show_reproj_results(views, tracks, cam0_K, cam0_distortion,
    ↪ print_raw_reproj=True, show_reproj_histogram=True)
vo.visualize_predictions(views, tracks, cam0_K, cam0_distortion)

```

#### REPROJECTION ERRORS

```

Image 0 ( 112 points) : (mean, std, max, min) = (0.1312, 0.1177, 0.5575,
0.0002)
Image (raw reprojection) 0 ( 112 points) : (mean, std, max, min) = (0.1114,

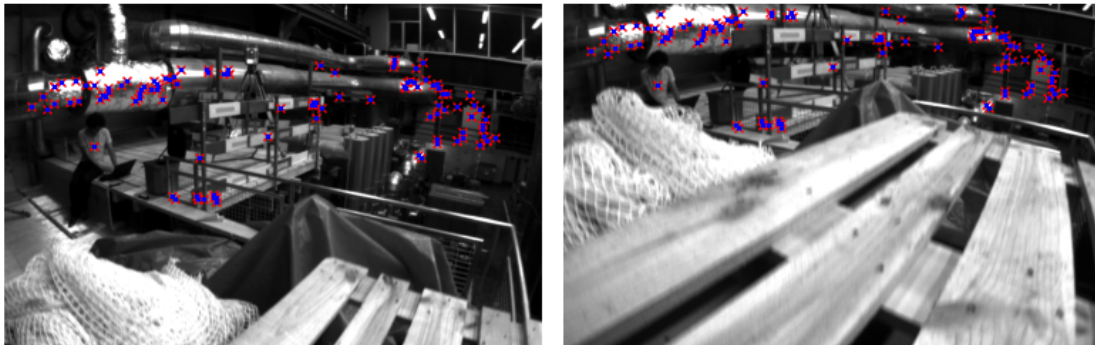
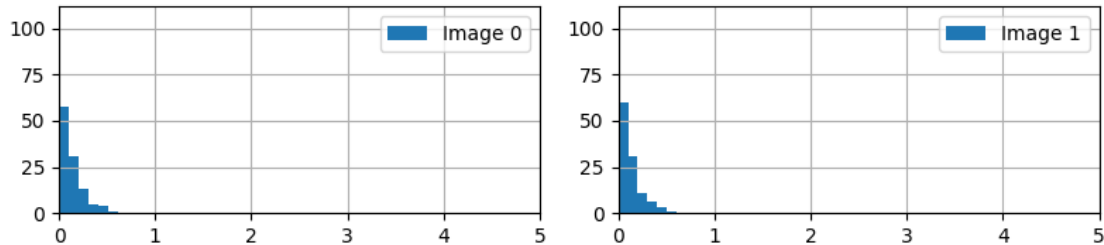
```



0.1009, 0.4932, 0.0012)

Image 1 ( 112 points) : (mean, std, max, min) = (0.1236, 0.1111, 0.5484, 0.0002)

Image (raw reprojection) 1 ( 112 points) : (mean, std, max, min) = (0.1385, 0.0948, 0.3845, 0.0048)



```
[12]: print(f"\nTiming\n{'='*100}")
      print(f"Analytical guess: {analytical_guess:.2f} [s]")
      print(f"Non linear (V0): {nonlinear:.2f} [s]")
```

Timing

```
=====
=====
Analytical guess: 2.40 [s]
Non linear (V0): 1.02 [s]
```

```
[13]: # This highlights the scale ambiguity seen in two-view reconstruction
      print("Metric pose difference norm (gt) = {:.3f}".format(np.linalg.norm(
          # Note: while this is in world frame, the frames do not have any scaling,
          # so measuring displacement this way still should provide the correct
          ↪ distance.
          np.linalg.norm(p_inW_ofB[first_frame_idx] - p_inW_ofB[second_frame_idx])
      )))
```

```

print("Metric pose difference norm (ini) = {:.3f}".format(np.linalg.
    ↪norm(results.initial_values['T_inB1_ofA'].t)))
print("Metric pose difference norm (sf) = {:.3f}".format(np.linalg.norm(results.
    ↪optimized_values['T_inB1_ofA'].t)))

```

Metric pose difference norm (gt) = 0.323  
 Metric pose difference norm (ini) = 1.000  
 Metric pose difference norm (sf) = 1.000

Relative change in position ( $\Delta p$ ) and rotation ( $\Delta R$ ) between frames (**NOT ERROR w.r.t Ground Truth**)

```

[14]: # First frame results
frame0_key = 'T_inB0_ofA'
# Symforce - Initial values
R_inB0_ofA_ini = results.initial_values[frame0_key].R.to_rotation_matrix()
p_inB0_ofA_ini = results.initial_values[frame0_key].t

# Symforce - Optimized values
R_inB0_ofA_sf = results.optimized_values[frame0_key].R.to_rotation_matrix()
p_inB0_ofA_sf = results.optimized_values[frame0_key].t

# ground truth
R_inW_ofB0_gt = R_inW_ofB[first_frame_idx].as_matrix()
p_inW_ofB0_gt = p_inW_ofB[first_frame_idx]
v_inW_ofB0_gt = v_inW_ofB[first_frame_idx]

# Second frame results
frame1_key = 'T_inB1_ofA'
# Symforce - Initial values
R_inB1_ofA_ini = results.initial_values[frame1_key].R.to_rotation_matrix()
p_inB1_ofA_ini = results.initial_values[frame1_key].t

# Symforce - Optimized values
R_inB1_ofA_sf = results.optimized_values[frame1_key].R.to_rotation_matrix()
p_inB1_ofA_sf = results.optimized_values[frame1_key].t

# Ground Truth
R_inW_ofB1_gt = R_inW_ofB[second_frame_idx].as_matrix()
p_inW_ofB1_gt = p_inW_ofB[second_frame_idx]
v_inW_ofB1_gt = v_inW_ofB[second_frame_idx]

print(f"RELATIVE CHANGE BETWEEN FRAMES (NOT ERROR)\n")
print(f'dp\n{"="*50}')
print('(Analytical guess) dp: {:.2f} [m]'.format( np.linalg.norm(
    ↪p_inB0_ofA_ini - p_inB1_ofA_ini)))

```

```

print('(Non linear - VO) dp: {:.2f} [m]'.format( np.linalg.norm( p_inB0_ofA_sf -
↳ p_inB0_ofA_sf)))
print('(Ground Truth)      dp: {:.2f} [m]'.format( np.linalg.norm( p_inW_ofB0_gt -
↳ p_inW_ofB1_gt)))

print(f'\ndR\n{"="*50}')
print('(Analytical guess) dR scalar: {:.5f} [deg]'.format(pose_metrics.
↳ rotational_error( R_inB0_ofA_ini, R_inB1_ofA_ini)))
print('(Non linear - VO) dR scalar: {:.5f} [deg]'.format(pose_metrics.
↳ rotational_error( R_inB0_ofA_sf, R_inB1_ofA_sf)))
print('(Ground Truth)      dR scalar: {:.5f} [deg]'.format(pose_metrics.
↳ rotational_error( R_inW_ofB0_gt, R_inW_ofB1_gt)))

```

RELATIVE CHANGE BETWEEN FRAMES (NOT ERROR)

dp

```

=====
(Analytical guess) dp: 1.00 [m]
(Non linear - VO) dp: 0.00 [m]
(Ground Truth)      dp: 0.32 [m]

```

dR

```

=====
(Analytical guess) dR scalar: 6.88214 [deg]
(Non linear - VO) dR scalar: 8.08283 [deg]
(Ground Truth)      dR scalar: 7.82089 [deg]

```