Università degli Studi di Padova
Facoltà di Ingegneria
Corso di Laurea in Control System Engineering

Project presentation

P04 – UGV visual odometry

Marco Geremia
Riccardo Barbiero
Roberto Carta

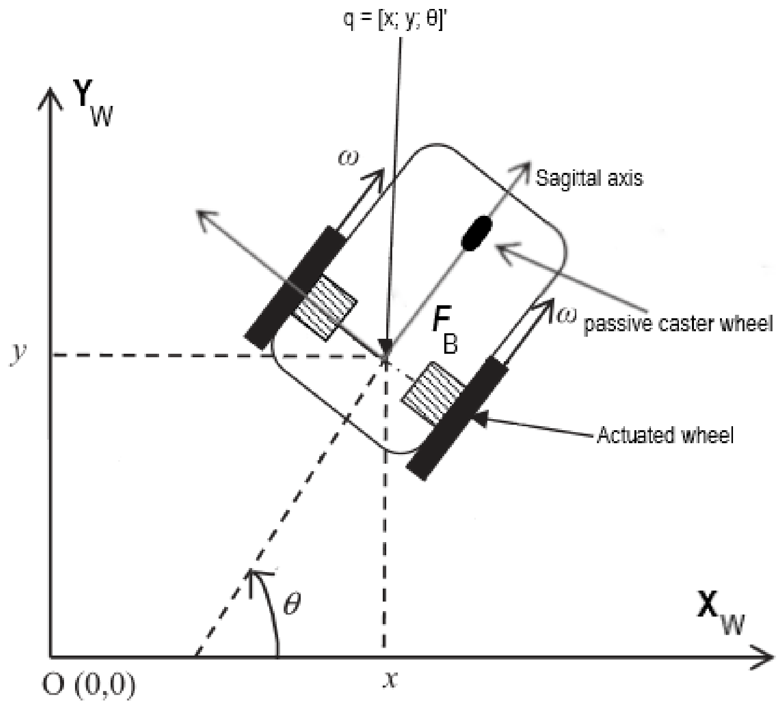
A.A. 2021/2022

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Aim of the project

- Control of the UGV for following a desired trajectory.
- A posteriori - offline reconstruction of the path of the UGV using photographic data exploiting fixed beacon features.
- Filtering of the resulting trajectory.

Tracking control: briefly recap of the UGV model



- State variable:

  - $$q = \begin{bmatrix} p \\ \theta \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, p \in R^2, \theta \in \$^1$$

- Kinematic equation:

  - $$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} c\theta \\ s\theta \\ 1 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} w = \begin{bmatrix} c\theta & 0 \\ s\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix}$$

Tracking control: theory approach

- Starting from a feasible trajectory to track $q(t)^{des} = \begin{bmatrix} x^{des} \\ y^{des} \\ \theta^{des} \end{bmatrix}$.
- Find the error vector $e_W(t) = \begin{bmatrix} x^{des} - x \\ y^{des} - y \\ \theta^{des} - \theta \end{bmatrix} \xrightarrow{R_z(\theta)} e_B(t)$.
- Dynamics of the error: $\begin{cases} \dot{e}_1 = v^{des}\cos(e_3) - v + e_2 w \\ \dot{e}_2 = v^{des} sen(e_3) - e_1 w \\ \dot{e}_3 = w^{des} - w \end{cases}$
- Change of input by using an invertible map

$$\begin{cases} v = v^{des}\cos(e_3) - u_1 \\ w = w^{des} - u_2 \end{cases} \leftrightarrow \begin{cases} u_1 = -v + v^{des}\cos(e_3) \\ u_2 = w^{des} - w \end{cases}$$

In order to be able to express the dynamics of the error in terms of u1, u2.

Tracking control: theory approach

$$\dot{e} = \begin{bmatrix} 0 & w^{des} & 0 \\ -w^{des} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} e + \begin{bmatrix} 0 \\ s(e_3) \\ 0 \end{bmatrix} v^{des} + \begin{bmatrix} 1 & -e_2 \\ 0 & e_1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\begin{cases} \sin(e_3) \cong e3 \\ e_2 \cong 0 \\ e_1 \cong 0 \end{cases}$$

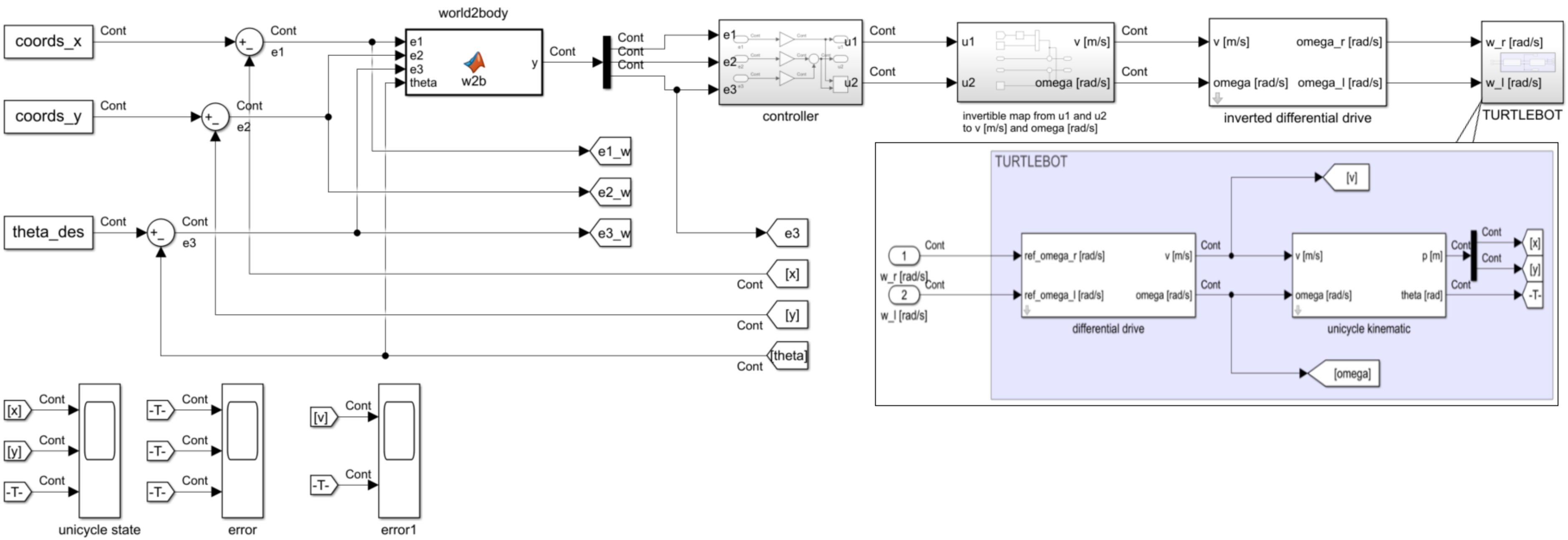
- After the linearization around the desired trajectory (e≈0) we obtain

$$\dot{e} = \begin{bmatrix} 0 & w^{des} & 0 \\ -w^{des} & 0 & v^{des} \\ 0 & 0 & 0 \end{bmatrix} e + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

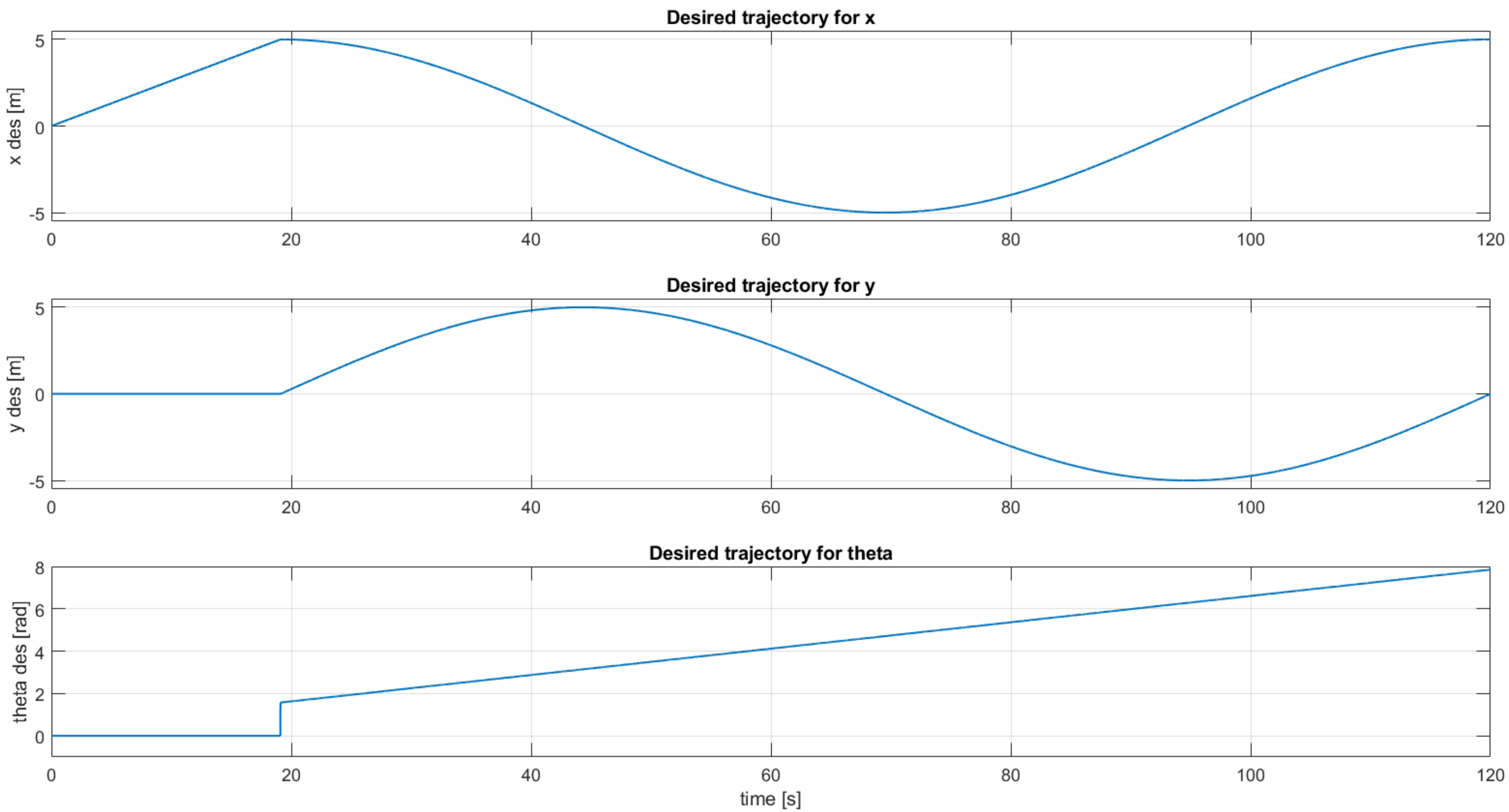
- Defining this state-feedback control law $\begin{cases} u_1 = -k_1 e_1 \\ u_2 = -k_2 e_2 - k_3 e_3 \end{cases}$ we obtain the desired error state matrix:

$$\dot{e} = \begin{bmatrix} -k_1 & w^{des} & 0 \\ -w^{des} & 0 & v^{des} \\ 0 & -k_2 & -k_3 \end{bmatrix} e$$

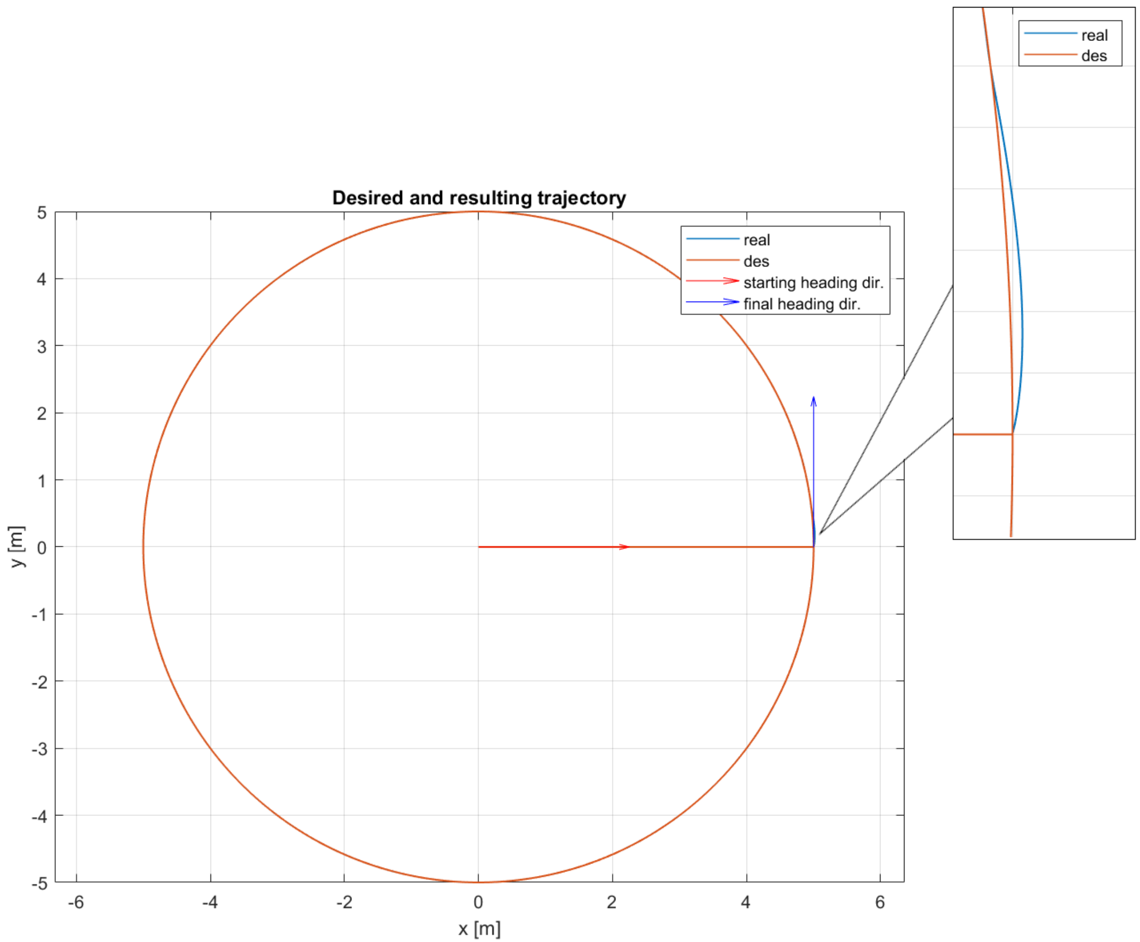
Tracking control: Simulink implementation



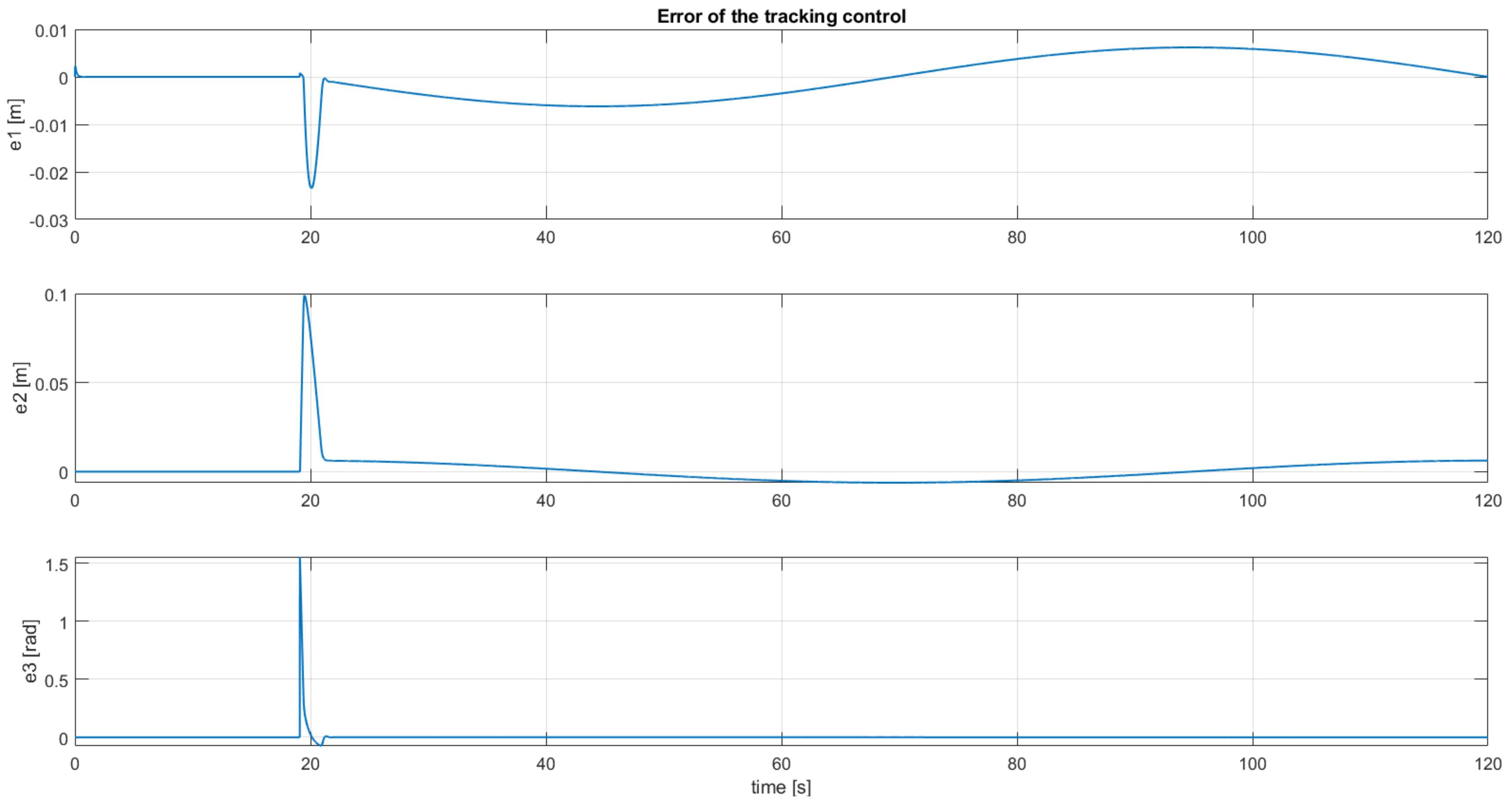
Desired trajectory



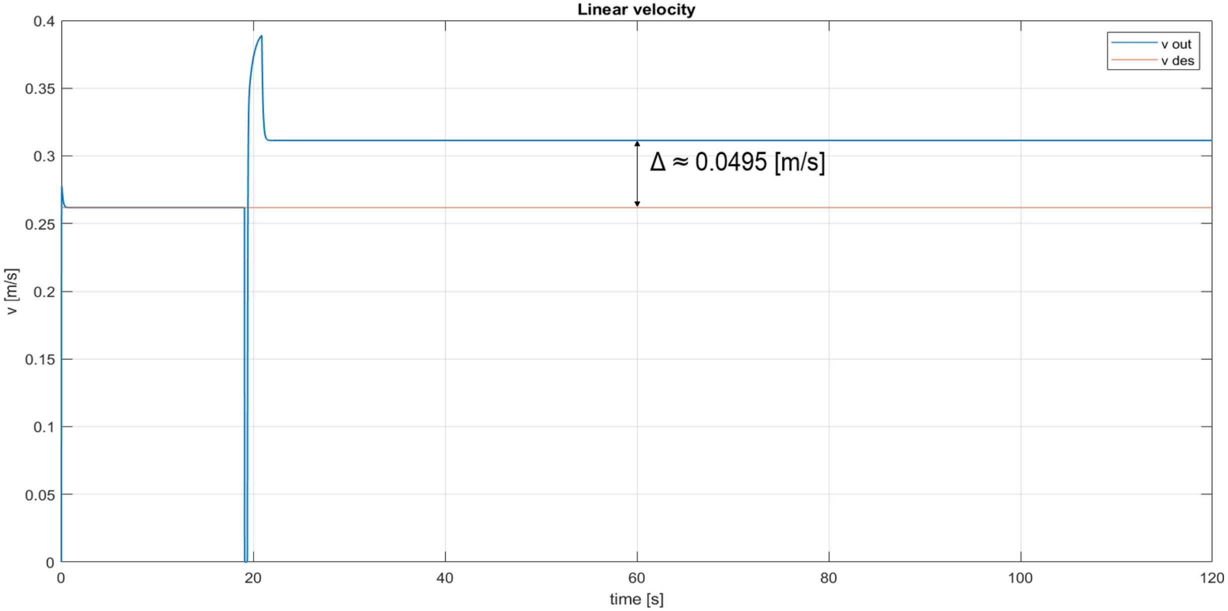
Resulting trajectory

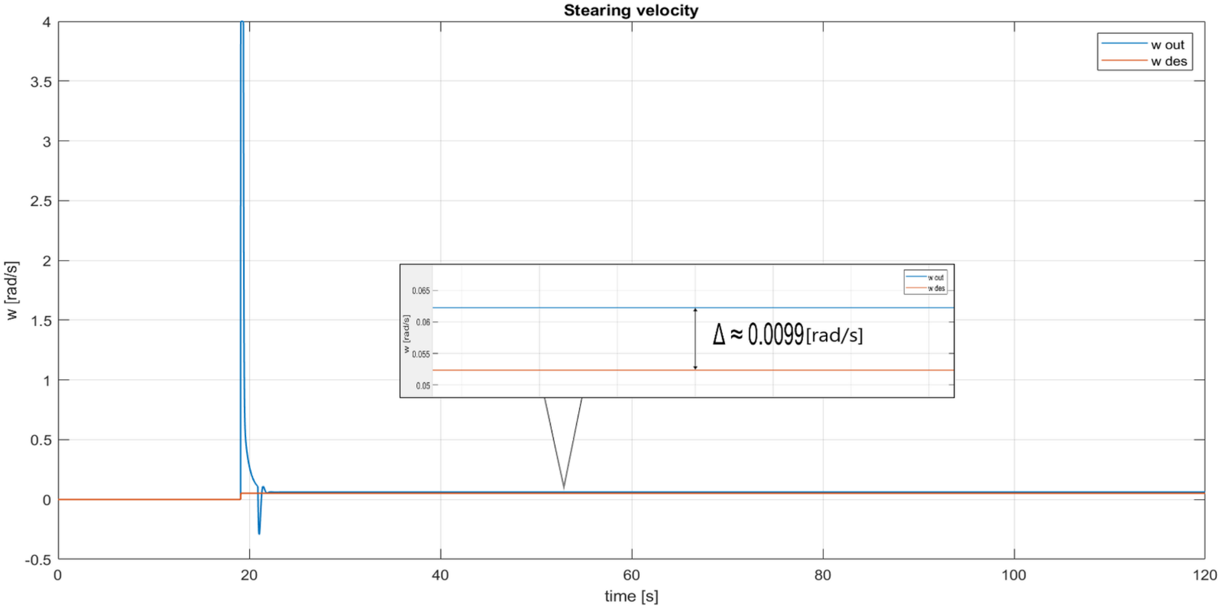


Error of the controller



Lin. vs ang. velocity





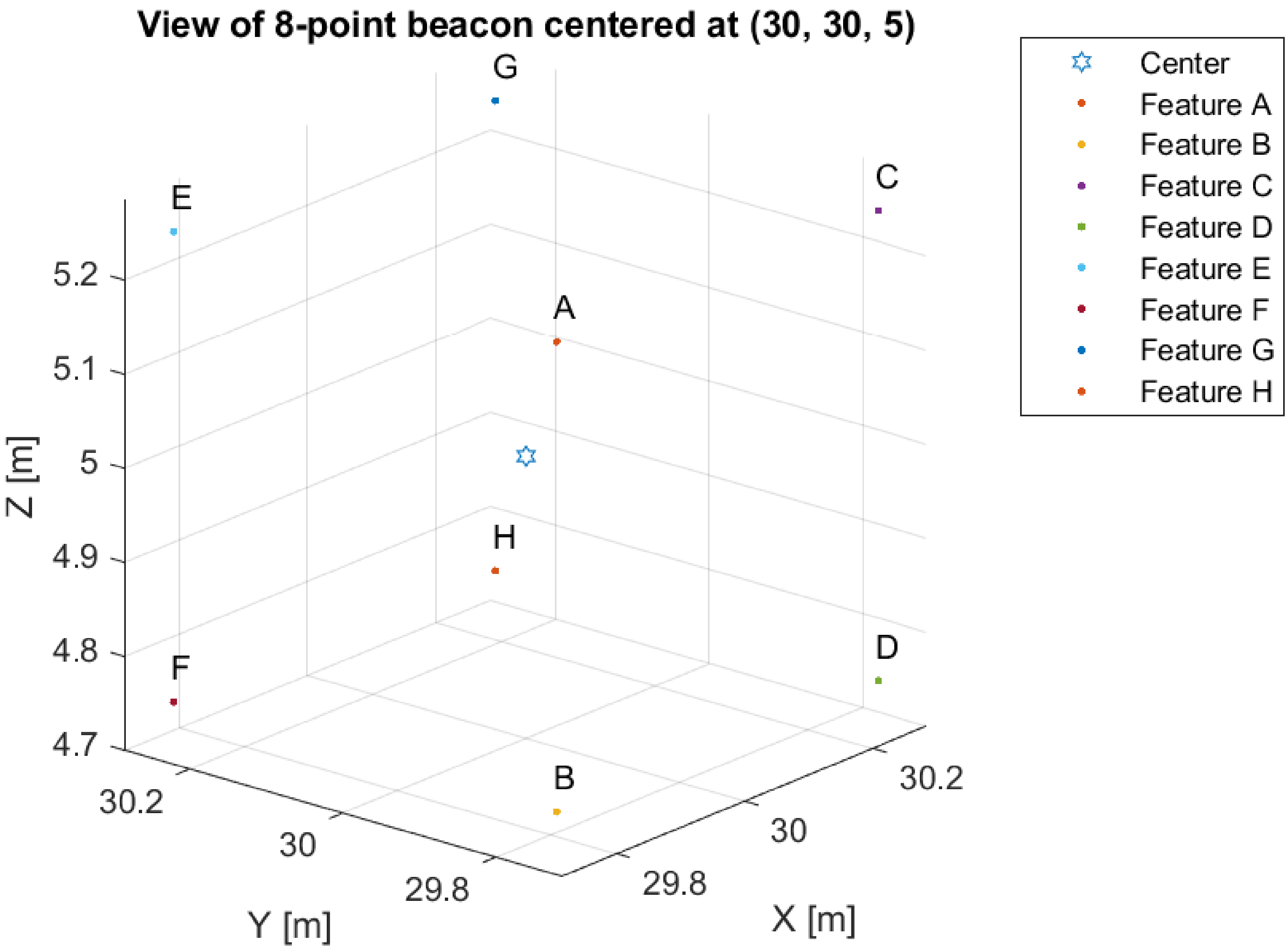
Visual Odometry Motivations

- Odometry based only on IMU is not so reliable
- When working on rough terrains wheels can slip
- IMU measure will increase even with no actual motion

Beacon Generation

- No computer vision task assumed
- Initial idea: spherical beacon
- Actual implementation: cuboidal beacon
- Important: at least 8 points are required!

Beacon 3D view



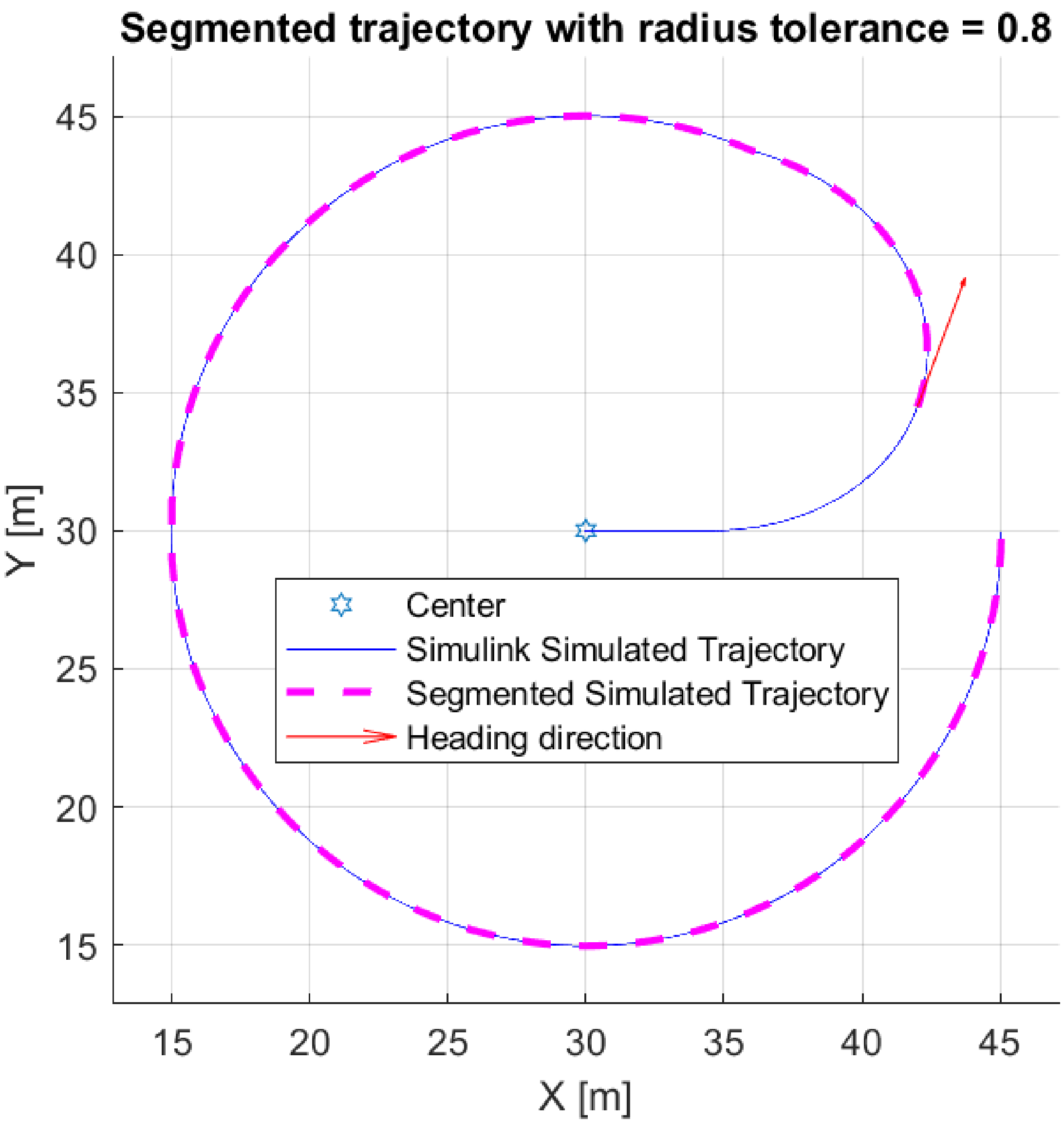
Robot trajectory assumptions

- Planar circular motion around beacon
- Trajectory data to be given (from simulation) as:

  $$[x(t), y(t), \vartheta(t)]$$

  where $x(t), y(t)$ are the cartesian coordinates, $\vartheta(t)$ is the sagittal angle w.r.t. x axis, referring to axis of the world reference frame
- Angular displacement of camera w.r.t. sagittal axis is supposed measured (*delta_cam_sag*) and used to calculate camera orientation in a real scenario
- **<u>Start and end positions are known</u>**

Robot trajectory assumptions



Camera frame assumptions

- The **camera frame z-axis pointing towards the center of the beacon**, and its origin is positioned at any time in the world frame at:

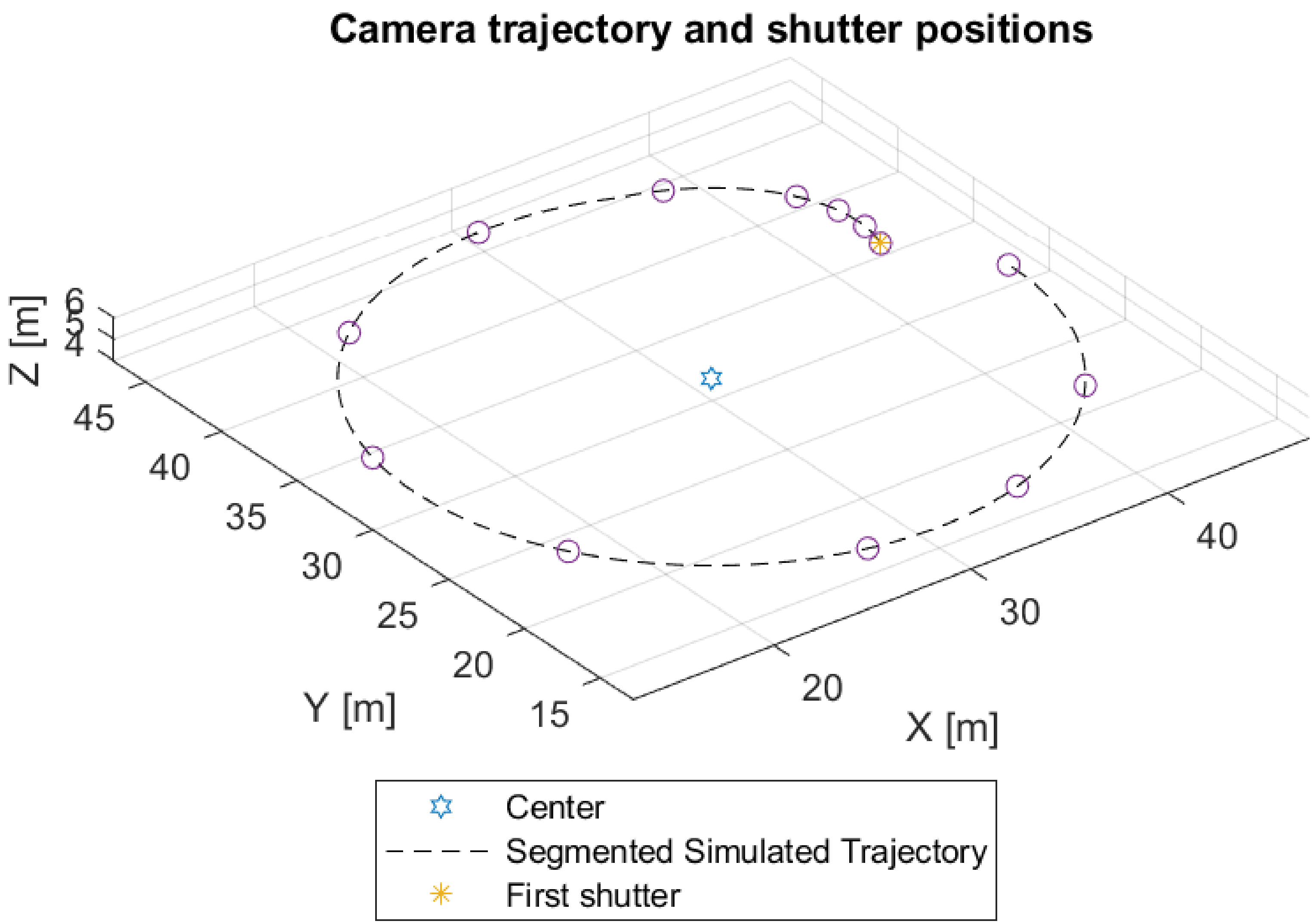
  $$[x(t), y(t), z_0]$$

  where z_0 is the height of the <u>center</u> of the beacon
- The **camera frame y-axis pointing towards the negative z-axis** in the world frame
- Only beacon center and $x(t), y(t)$ are needed to fully determine camera pose w.r.t. the world frame

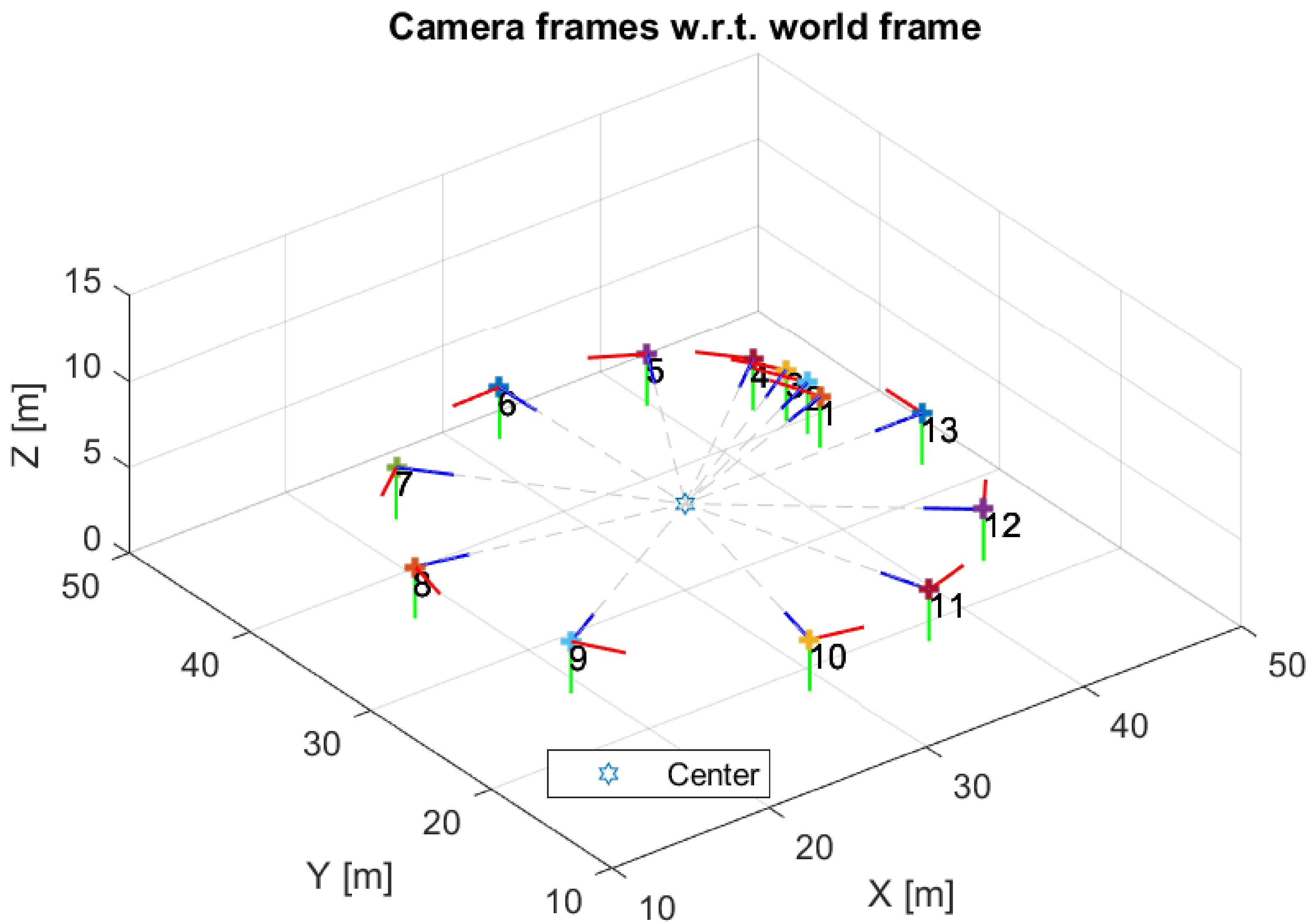
Image plane (photo) assumptions

- A fixed amount (*N*) of photos are taken at equally spaced (sample-wise) positions along the trajectory
- Computer vision tasks are supposed to have been already carried out
- Merely a projection of ordered 3D points onto a fixed plane on each camera frame
- Pinhole camera model with unitary focal length (normalized camera)

Camera frames origin trajectory



Camera frames poses sequence



Projection procedure

- Being R_{i0} the orientation of camera i and T_{i0} its position (world 2 body):

$$g = \begin{bmatrix} R_{i0} & R_{i0} \cdot T_{i0} \\ \bar{0} & 1 \end{bmatrix} \quad \mathrm{k} = \begin{bmatrix} f00 \\ 0f0 \\ 001 \end{bmatrix} \quad \mathrm{PI} = \begin{bmatrix} 1000 \\ 0100 \\ 0010 \end{bmatrix}$$

- Supposing Q is the 3D world frame point and $\hat{Q}$ the same point in the camera frame

$$\hat{Q} = k \cdot PI \cdot g \cdot Q$$

$$\widehat{q_x} = {}^{\hat{Q}_x}/_{\hat{Q}_z} \quad \widehat{q_y} = {}^{\hat{Q}_y}/_{\hat{Q}_z} \quad \hat{q} = [\widehat{q_x}; \widehat{q_y}]$$

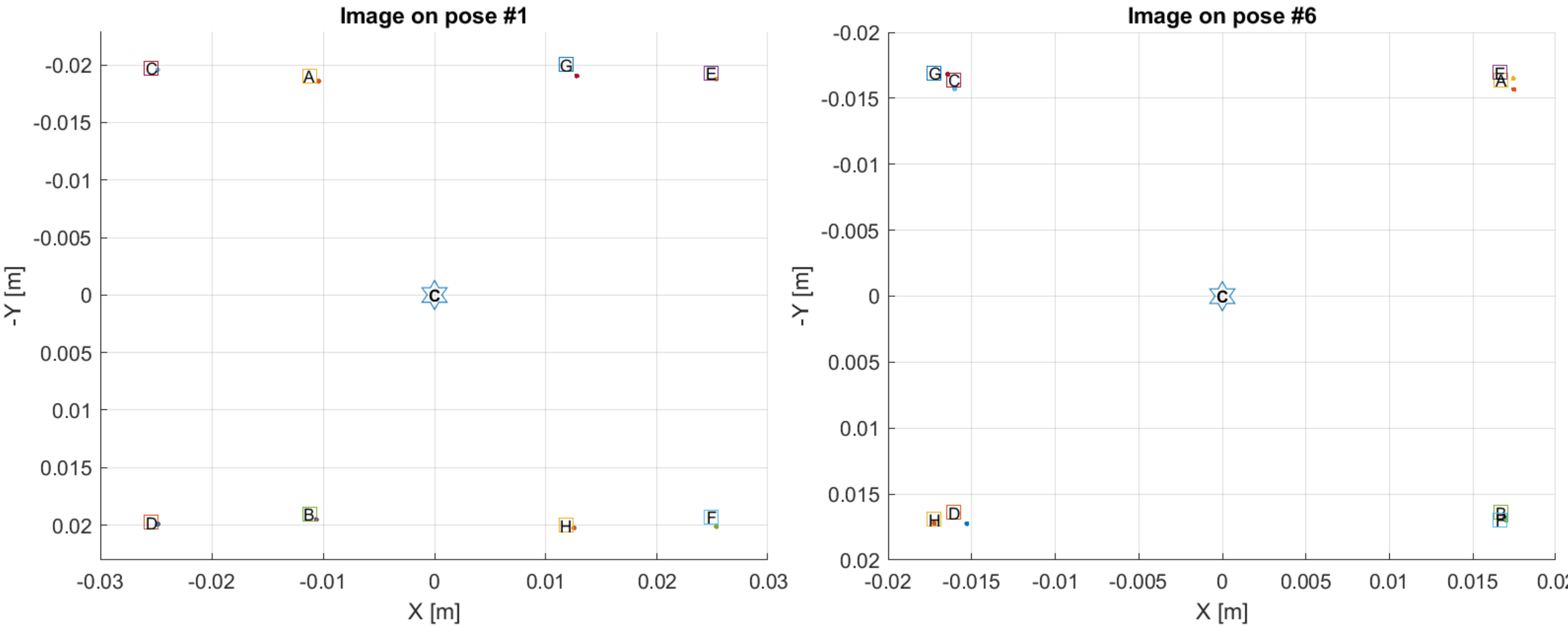
- $\hat{q}$ being the 2D coordinates on the image plane

Roadblock: unable to achieve correct projection

- Points projected into the camera frame do not respect at all the geometric property of the beacon
- Normalization into the image plane further exacerbates this problem
- Negotiated assumption:
  E matrix generated from data, and poisoned using uniform noise scaled with γ_E to simulate measurement, matching and estimation errors
- However…

Roadblock solution: correct projection achieved

- After careful debugging, projection went smoothly



- However…

Roadblock: unable to estimate E matrix from matched points

- Manually implementing appropriate algorithms (e.g. 8-points algorithm) is a daunting task
- Corresponding Matlab routines (*estimateFundamentalMatrix*) do not provide satisfactory results, if at all
- Negotiated assumption:
  E matrix generated from data, normalized to loose scale and poisoned using uniform noise scaled with γ_E to simulate measurement, matching and estimation errors (same as previous assumption)

E matrix generation from data procedure

- Let T_{ij} be the relative position of frame i w.r.t. frame j in the reference frame
- Let R_{ij} be the relative orientation of frame i w.r.t. frame j in the reference frame
- Then poison with uniform noise:

$$E_{ij} = [T_{ij}]_x \cdot Rij + \gamma_E \cdot rand(3,3)$$

- Normalize E_{ij} in order to lose the scale

Poses extraction from E matrix procedure

- Let $\hat{E} = \mathrm{U} \cdot \hat{S} \cdot \mathrm{V}^{\mathrm{T}}$
- Safety check on the determinant of U and V
- Reproject $\hat{E}$ by averaging the non null singular values to obtain $S = \begin{bmatrix} \frac{\sigma_1+\sigma_2}{2} & 0 & 0 \\ 0 & \frac{\sigma_1+\sigma_2}{2} & 0 \\ 0 & 0 & 0 \end{bmatrix}$
- Then extract:

$$\mathrm{T}_1, T_2 = UR_z\left(\pm\frac{\pi}{2}\right)\mathrm{S\ UT}$$

$$R_1, R_2 = \mathrm{U}R_z\left(\pm\frac{\pi}{2}\right)V^T$$

Pose discrimination via triangulation

- We have 4 possible configurations from the solution to the epipolar constraint:
  $(R1, T1)$ $(R1, T2)$ $(R2, T1)$ $(R2, T2)$
- Triangulation of each pair of feature points into the 3D space shall give a positive z coordinate (in both camera frames) for only one (correct) configuration
- Introduce a voting system to account for triangulation error

Pose discrimination via triangulation procedure

- Due to projection error and camera noise, closed form triangulation is not feasible.
- We have to solve a minimization problem
- We resort to the *triangulate* Matlab function, requiring the matched points and correspondent projection matrices for pose *i* and *j*
- Triangulation is carried out for all points for each of the 4 possible poses; each positive z point counts as a vote for that pose. The pose with more votes is chosen as the correct one.

Roadblock: discrimination via triangulation doesn't work

- The poses pairs $(T^{d}{}_{ij}, R^{d}{}_{ij})$ chosen by the previous algorithm do not reflect the expected ones, at least for what concerns the baseline orientation
- Negotiated assumption:
  Because of the circular motion, two subsequent position must not stray away from the rotation center; therefore, of the two possible $T_{ij,1}, T_{ij,2}$ only the one staying closer to the center is the correct one.
- However…

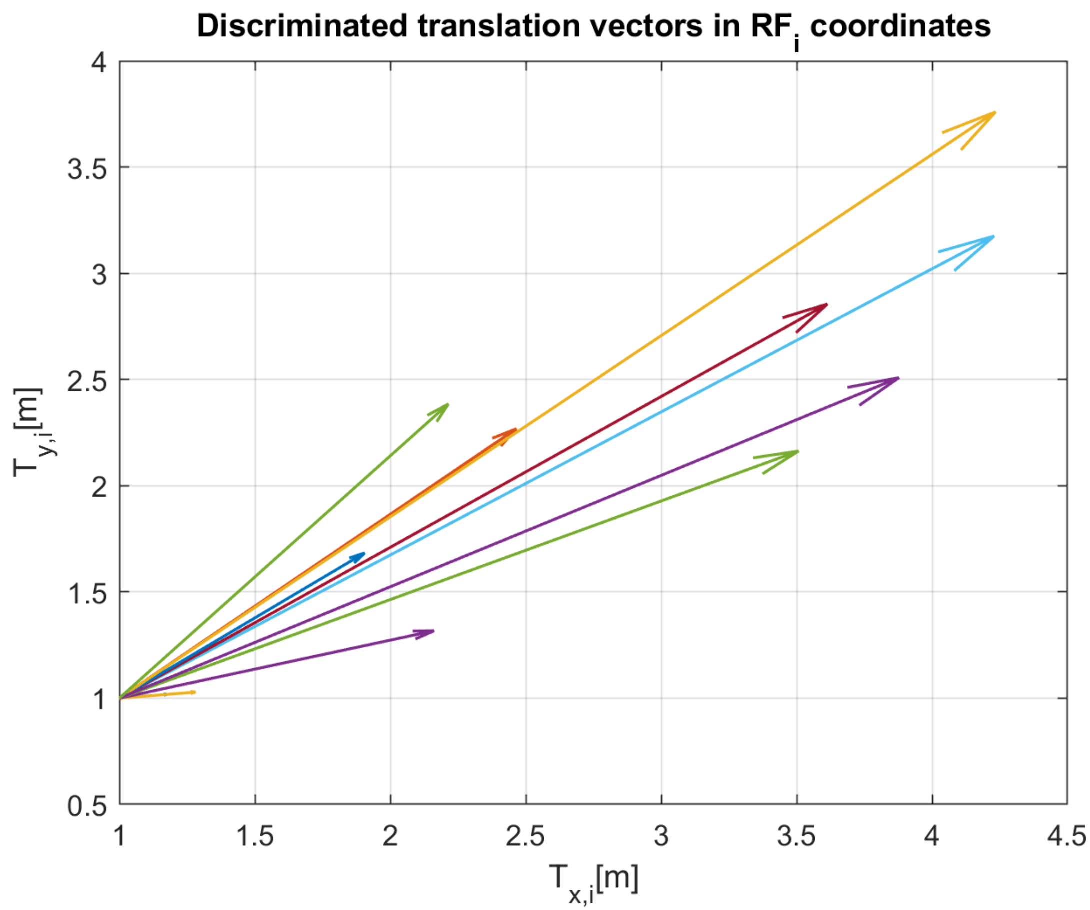
TRIANGULATION ACTUALLY WORKED(1)

- The **problem** was that we built the E_{ij} matrix from relative R_{ij} referred to i reference frame and T_{ij} referred to World Reference frame
- This is conceptually wrong, but it worked
- To adjust the formulation, the E matrices are computed from R_{ij} and T_{ij} both written in RF_i coordinates

TRIANGULATION ACTUALLY WORKED(2)

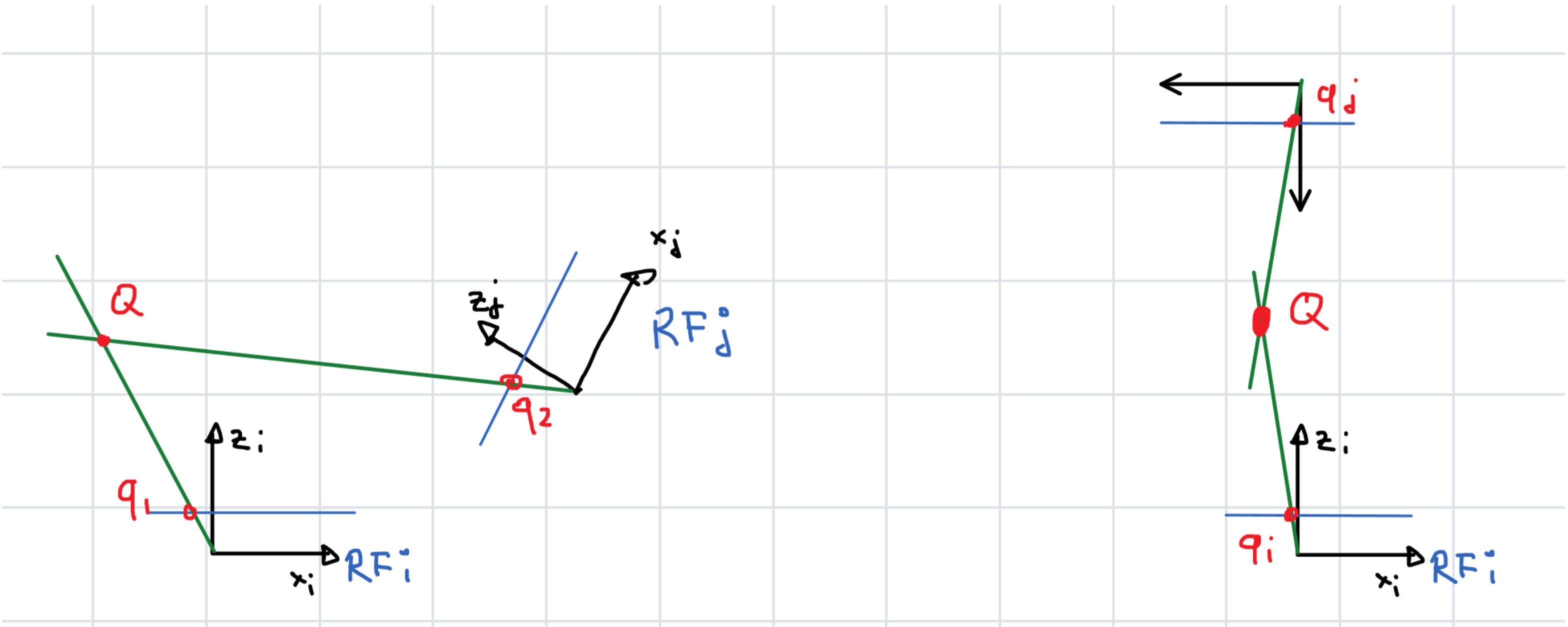
- From image plane 2D points, use *triangulate* routine to reconstruct 3D points in Reference Frames i and j

- Check with which pose (T_{ij}, R_{ij}) the z-coordinates of the 3D points are positive



Issue of triangulation

- Issue: while 3D reconstruction worked very well with consecutive poses, with opposite poses it failed, due also to noise



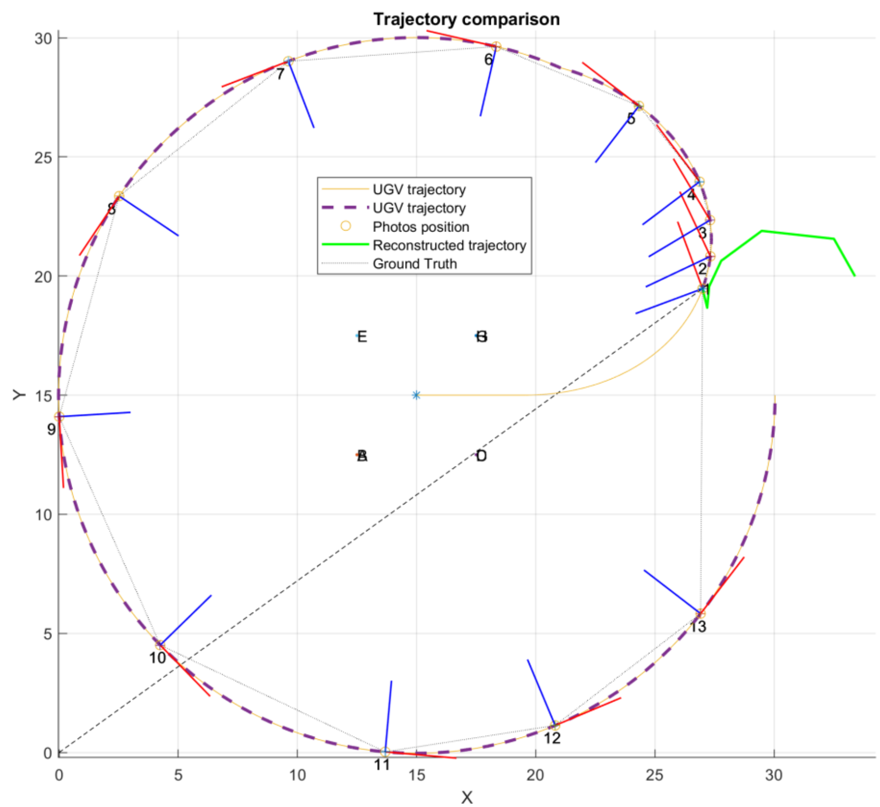
Reconstruction Problem

- Open problem in reconstructing the absolute pose w.r.t. World Reference frame of the UGV, a-posteriori
- Tought strategy: permutation of **Camera Frame** axes w.r.t. **World Frame** axes
- Reconstruction of absolute rotation $\boldsymbol{R_i}$ with cascade of consecutive matrix, by post-multiplying

$$R_i = R_1 R_{12} \dots R_{(i-1)i}$$

Why actually not used?

- Main Issue: Didn't reconstruct the trajectory
- <u>Noticed too late, no more time</u>



Roadblock: discrimination via distance from center is too biased

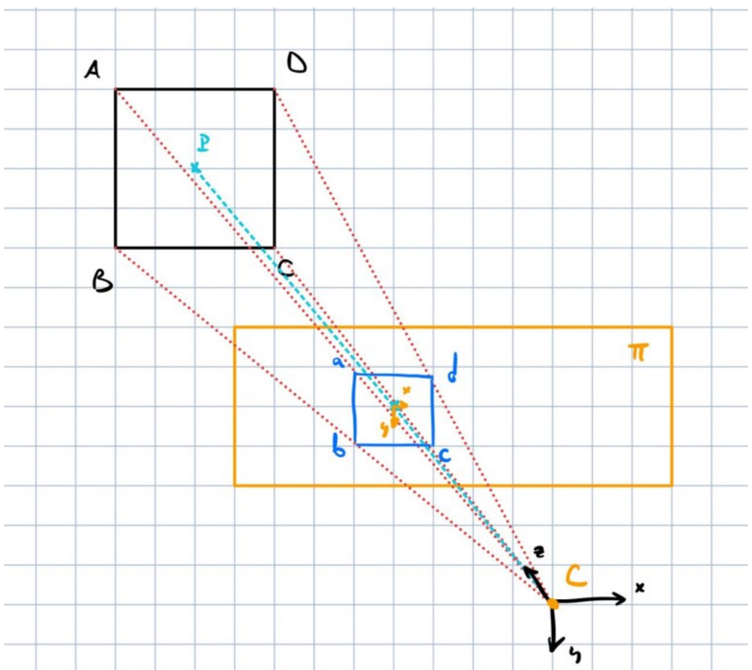
- This solution only works for ideal circular motion, but has some instability if there is some perturbation in the trajectory (as is the case of the simulated one)
- Assumption negotiated:
  Poses discrimination should be carried out by manual comparison (norm-wise) to the ground truth (T_{ij}, R_{ij}) computed directly from camera frame trajectory data.

Scale reconstruction

- The translation vectors extracted from E are to defined up to a scaling factor
- Dimensions of the beacon in the world frame are supposed to be known
- Camera-beacon distance can be easily estimated from each photo using the fact that the beacon height has the same inclination in all poses
- Scale factor for T_{ij} versor can be computed using basic geometric considerations (Carnot's theorem)

Scale reconstruction procedure

- Side length in 3D is known
- Side length in photos can be easily measured



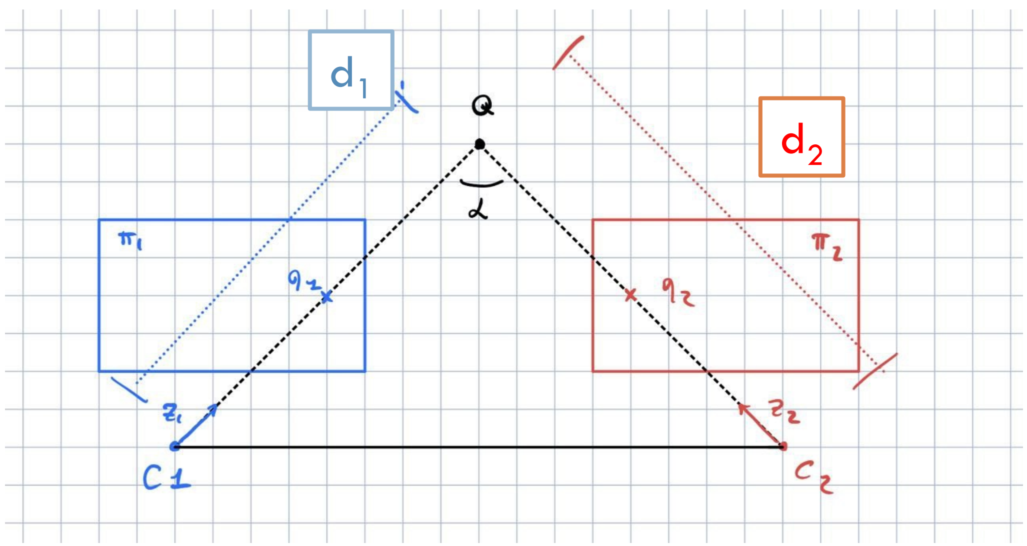
- $\frac{z_{AB}}{f} = \frac{\overline{AB}}{\overline{ab}}$ $(f = 1)$
- $d_k \approx \frac{\sum_{ij} zij}{n\_computed\_sides}$
- In the assumption of circular motion:

$$d_k \approx r_0$$

Scale reconstruction procedure

- Exploiting cosine theorem to compute baseline distance
- d_i and d_j are estimated using previous procedure
- α comes from the estimated R_{ij} matrix

$$|C_i C_j| = \sqrt{{d_1}^2 + {d_2}^2 - 2\, d_1 \cdot d_2 \, \cos\alpha}$$



Basic filtering of relative poses and positions

- Two *NxN* tables are computed (R_{ij} *and* T_{ij})
- Symmetry of tables (average angles)
- Odometer approach
- Clockwise and counter-clockwise average

Filtering proposed procedure

- Main Problem of Visual Odometry: Drift
- Filtering articulated essentially in 2 steps:

1. Filtering of the relative Rotation Matrices (angles)
2. Filtering of the Translation Vectors

Filtering of Rotation Matrices

- Average value among 2 rotation angles referred to two opposite poses

$$\hat{\alpha}_{ij} = \frac{|\alpha_{ij}| + |\alpha_{ji}|}{2}$$

- Achieved accuracy of 5 deg w.r.t. ground truth values

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	2.0195	1.6543	1.1510	1.5521	0.3049	0.9234	1.2140	2.9478	1.7821	0.3089	0.9823	1.6472
2	2.0195	0	1.2323	1.1770	1.1585	0.1683	0.3885	0.6903	3.6337	2.5217	0.4092	0.8792	1.2680
3	1.6543	1.2323	0	1.3413	1.4855	0.5049	2.1029	1.9465	2.1376	1.8504	0.8905	0.3412	0.4769
4	1.1510	1.1770	1.3413	0	1.3857	1.9021	0.6401	1.1135	1.8366	2.9481	0.7821	0.5121	0.7736
5	1.5521	1.1585	1.4855	1.3857	0	2.4712	1.4251	2.7620	2.8920	1.5778	1.5773	0.0372	0.4061
6	0.3049	0.1683	0.5049	1.9021	2.4712	0	2.8629	1.2668	0.9080	1.4806	1.3281	1.9946	1.0603
7	0.9234	0.3885	2.1029	0.6401	1.4251	2.8629	0	1.7403	0.4140	1.2998	2.1362	2.6767	2.1490
8	1.2140	0.6903	1.9465	1.1135	2.7620	1.2668	1.7403	0	0.0615	0.4539	0.8709	2.3083	1.3281
9	2.9478	3.6337	2.1376	1.8366	2.8920	0.9080	0.4140	0.0615	0	1.1127	0.8229	0.0671	2.7890
10	1.7821	2.5217	1.8504	2.9481	1.5778	1.4806	1.2998	0.4539	1.1127	0	3.0178	2.5201	1.8588
11	0.3089	0.4092	0.8905	0.7821	1.5773	1.3281	2.1362	0.8709	0.8229	3.0178	0	1.2893	0.0131
12	0.9823	0.8792	0.3412	0.5121	0.0372	1.9946	2.6767	2.3083	0.0671	2.5201	1.2893	0	0.7331
13	1.6472	1.2680	0.4769	0.7736	0.4061	1.0603	2.1490	1.3281	2.7890	1.8588	0.0131	0.7331	0

Distributed approach for filtering Translation vectors T_{ij}

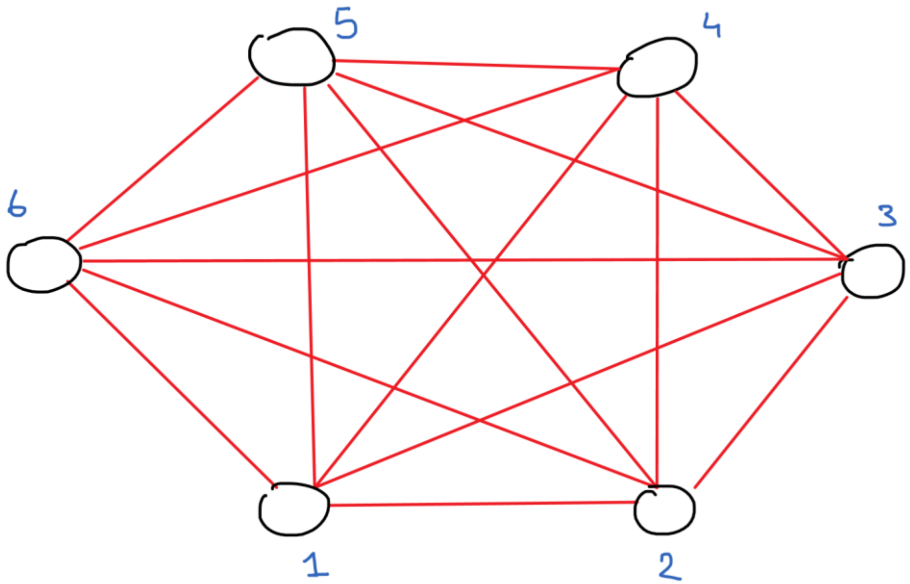
- Suppose that every node of the network represents the pose in which we take a photo
- Every node needs informations from any other node
- Distrubuted estimation:

1. Consensus algorithm
2. Analytic solution, by minimizing cost function

$$\varphi(\mathrm{t}) = \sum_{i \to j} \left\| R_i^T (T_j - T_i) - \lambda_{ij} \tilde{t}_{ij} \right\|^2$$

Why not implemented?

- The number of Spanning Trees is too high
- Example with 6 nodes which communicate each others:



$$L = \begin{bmatrix} 5 & -1 & -1 & -1 & -1 & -1 \\ -1 & 5 & -1 & -1 & -1 & -1 \\ -1 & -1 & 5 & -1 & -1 & -1 \\ -1 & -1 & -1 & 5 & -1 & -1 \\ -1 & -1 & -1 & -1 & 5 & -1 \\ -1 & -1 & -1 & -1 & -1 & 5 \end{bmatrix}$$

$$\sigma(L) = \{0,6,6,6,6,6\}$$

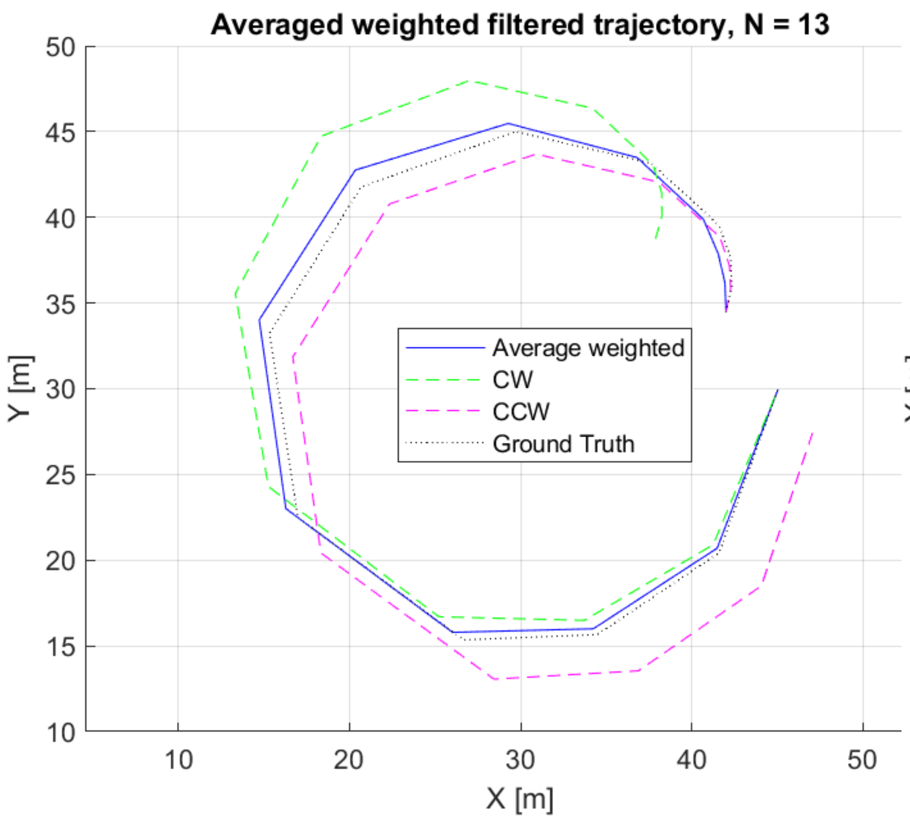
$$\#ST = \frac{6 \cdot 6 \cdot 6 \cdot 6 \cdot 6}{6} = 1296$$

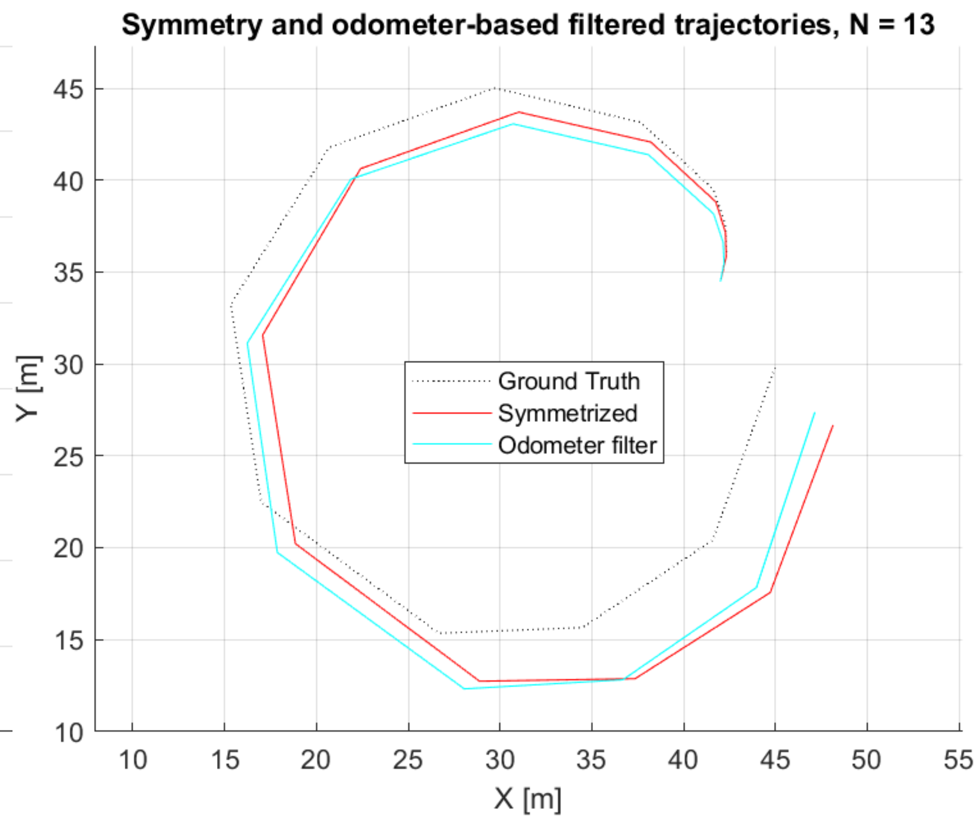
- **Unfeasible** due to too high computation
- No more time

Final Filtering Method

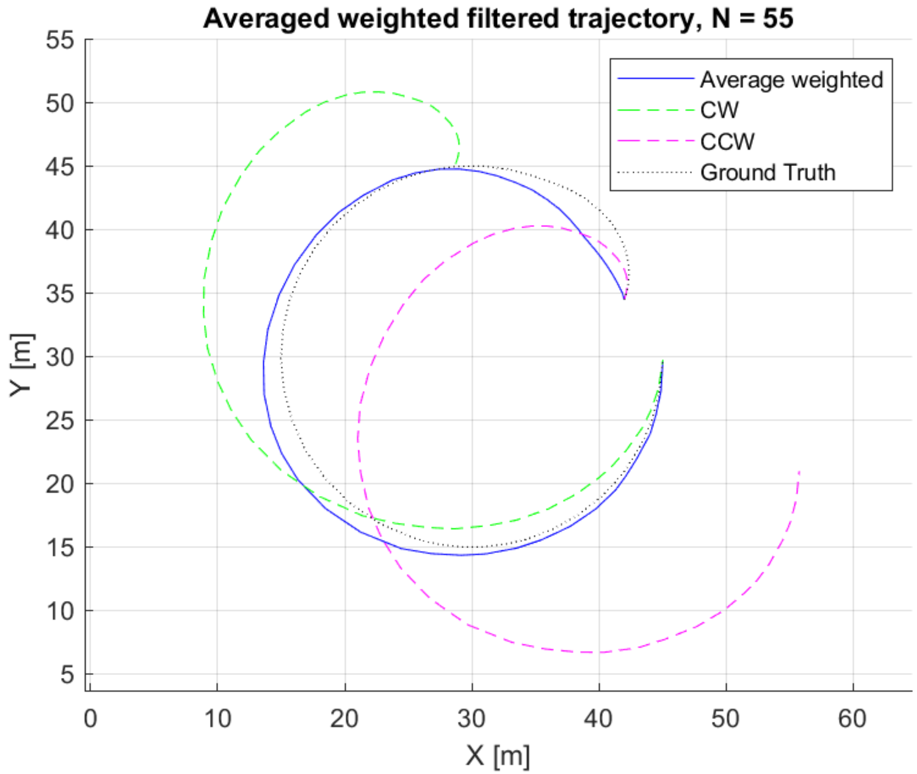
- Take clockwise and counter-clockwise reconstrcuted trajectories, with filtered R and raw T
- Compute the **weighted average between the two trajectiories**

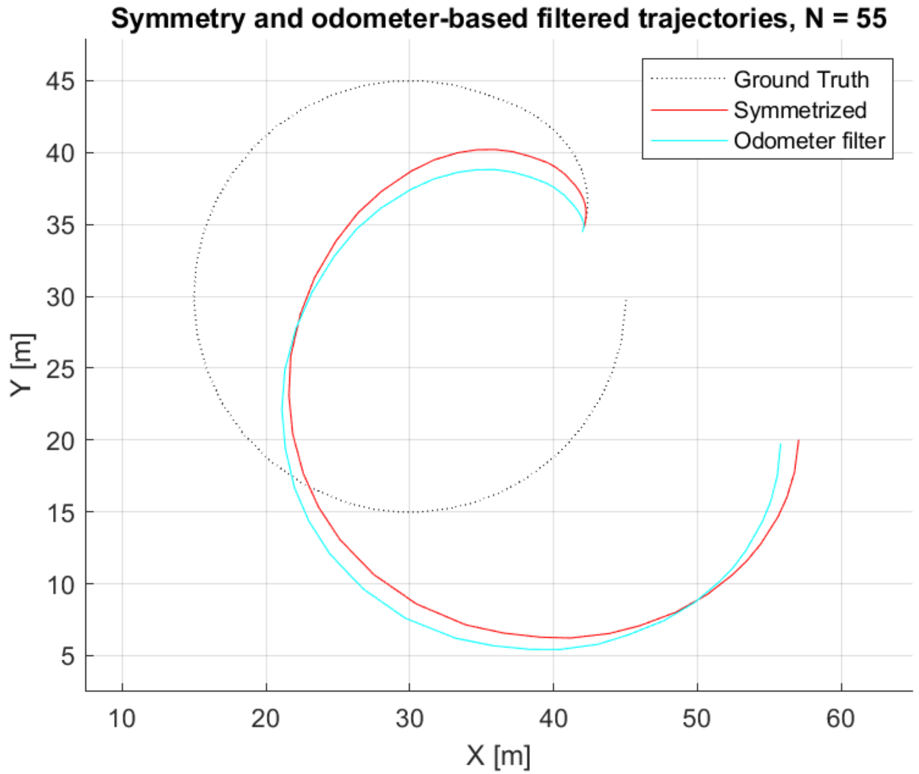
Filtered trajectories comparison N = 13



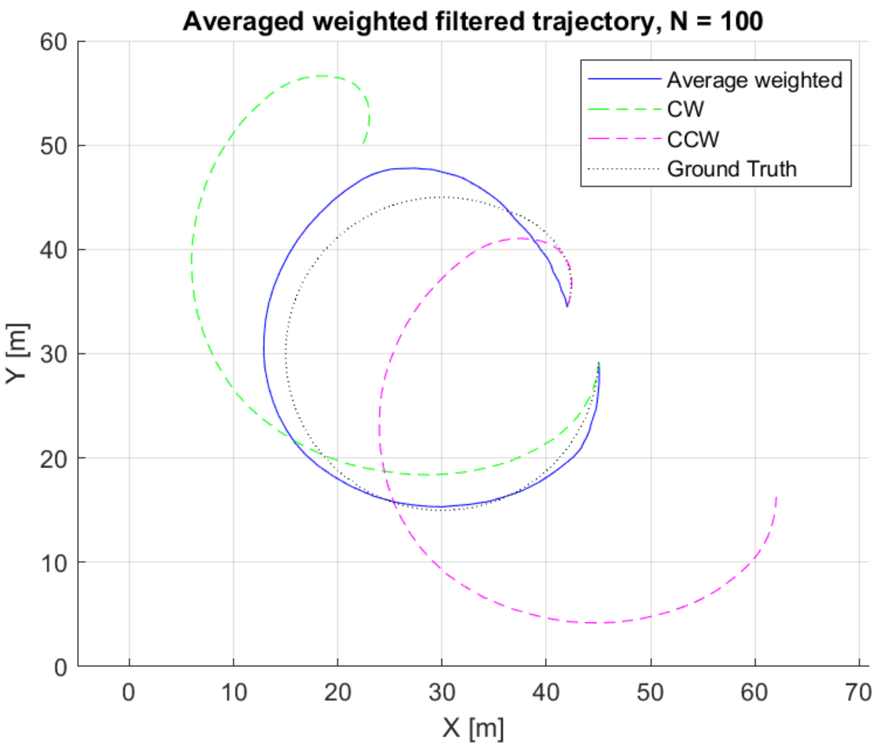


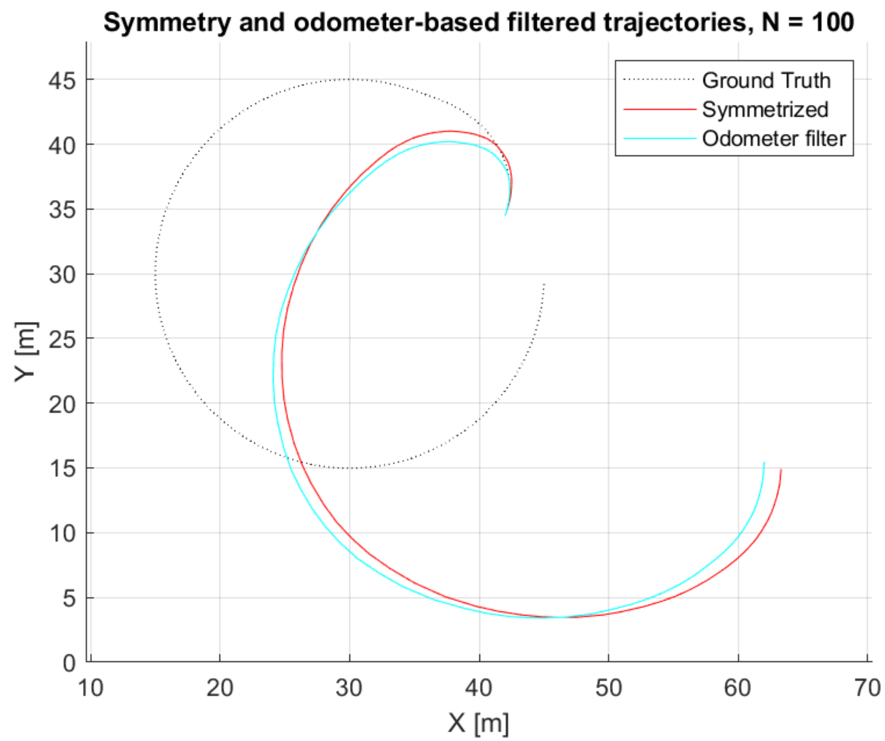
Filtered trajectories comparison N = 55



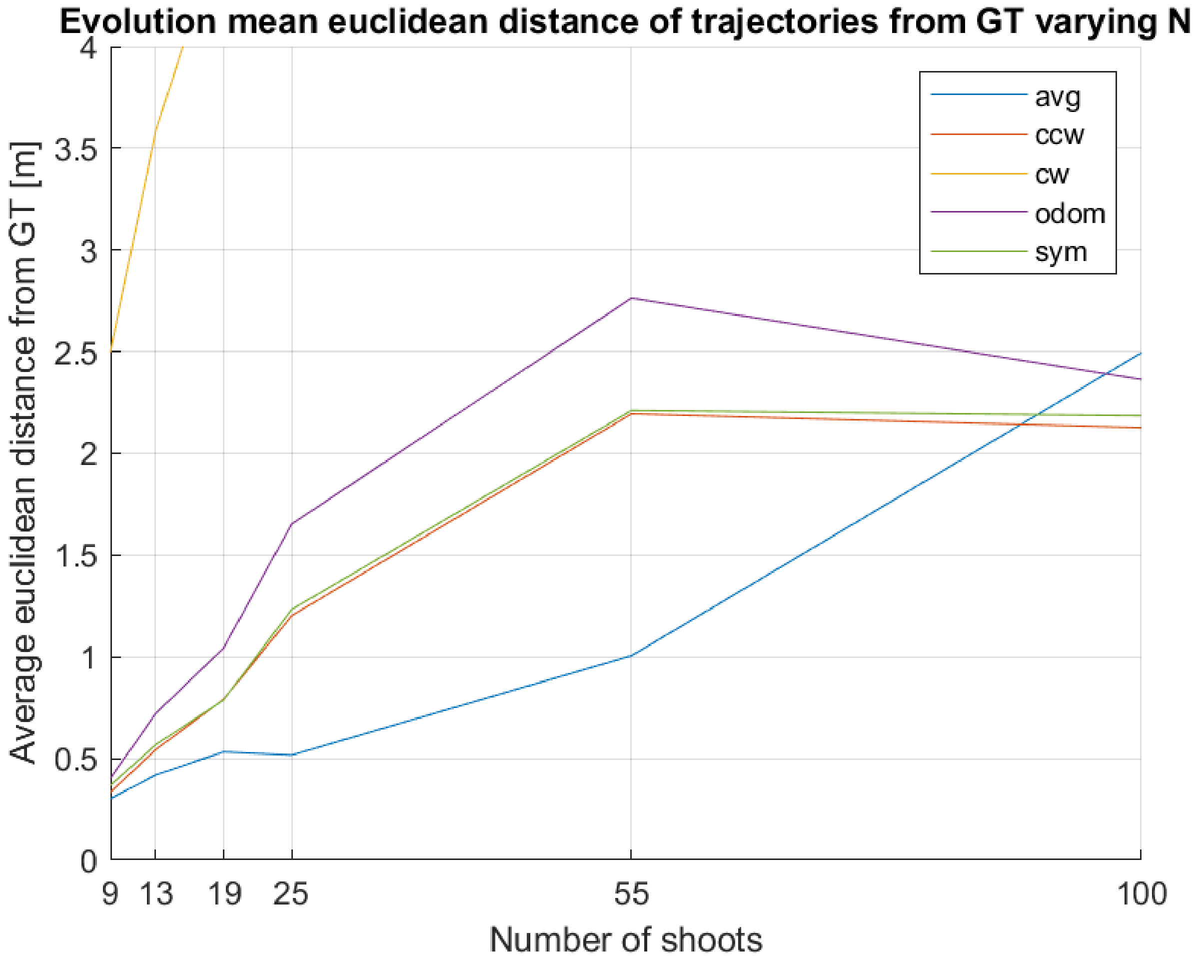


Filtered trajectories comparison
N = 100





Estimation error evolution



- Error stacks with increasing N!