Monocular Two-View Pose Estimation

Section 1 - Introduction

Estimating the relative pose between two images is a fundamental problem in computer vision. Given two images of the same scene, our goal is to recover the relative rotation matrix, and translation vector between the camera viewpoints.

This project focuses on the design and evaluation of a monocular two-view pose estimation pipeline. Our final system accepts two RGB images and camera intrinsics as input and outputs a relative rotation matrix $R \in SO(3)$ and translation vector $t \in \mathbb{R}^3$, recovered up to an unknown scale factor.

Over the course of the project, the pipeline was iteratively improved through a sequence of design decisions. These iterations included feature detector selection (ORB vs. SIFT), descriptor matching strategies, epipolar geometry estimation, camera calibration, and sensitivity analysis of image resolution.

In order to elevate our project's robustness further, we did video pose processing, ArUco markers pose estimation, camera calibration and more to debug and evaluate our code, which are not part of the final system.

Section 2 - Problem Definition and Assumptions

2.1 Problem Statement

Given two images I_1 and I_2 of a static scene captured by the same camera from two different viewpoints, estimate the relative camera Pose (R, t) such that:

$$x_2 \sim K(Rx_1 + t)$$

Where K is the camera intrinsics matrix, R is the relative rotation between two camera frames, and t is the relative translation vector, recovered up to an unknown scale.

2.2 Inputs and Outputs

Inputs:

- Two RGB images captured by the same camera
- Camera intrinsic matrix (K)

Outputs:

- Relative rotation matrix $R \in SO(3)$
- Relative translation vector $t \in \mathbb{R}^3$ (up to scale)

2.3 Scope and Limitations

This project deliberately excludes:

- Dense depth estimation
- Multi-view or temporal visual odometry
- Bundle adjustment
- Sensor fusion (e.g., IMU)

By constraining the problem to a two-view monocular setting, the project emphasizes geometric understanding, algorithmic robustness, and controlled evaluation of each pipeline component.

Section 3 - Initial Baseline and Early Experiments

At the beginning of the project, the goal was not to build a perfect solution, but to create a simple, working and efficient baseline that could estimate the relative motion between two images.

3.1 Feature Detection and Matching

The first implementation used **ORB (Oriented FAST and Rotated BRIEF)** as the feature detector and descriptor.

ORB was chosen because:

- It's fast and lightweight.
- Commonly used in real-time systems.
- It does not require floating-point descriptors

The initial pipeline followed these steps:

1. Detect ORB keypoints in both images.
2. Compute binary descriptors.
3. Match the descriptors using brute-force Hamming Distance.

Observations:

- In scenes full of texture, ORB produced a large number of matches
- A lot of pairs resulted in reasonable pose estimates.
- **However,** results **were very unstable:**
    - Many incorrect matches
    - High sensitivity to image resolution (changing the image resolution by 10% would ruin the pose estimation).
    - Large rotation errors in some image pairs.

This showed that while ORB is fast, **raw matching alone wasn't reliable enough.**

3.2 Fundamental Matrix Estimation

In order to introduce some geometric constraints, at the very first stages of the project, we estimated the Fundamental Matrix from matched feature points. Which was mainly used for:

1) Verifying geometric consistency
2) Visualizing epipolar lines
3) Debugging feature matching quality.

Limitations:

- The Fundamental Matrix doesn't use camera intrinsics
- It does not provide metric pose information
- Recovering reliable rotation and translation is difficult

This is where we moved our calculations to be based on the Essential Matrix.

Section 4 - Transition to the Essential Matrix and Pose Recovery

We had significant limitations with the Fundamental matrix that made it clear that a calibrated approach is required. However, at this stage **camera intrinsics were not yet available** and we didn't research **calibration yet.** Thus, we introduced an intrinsic estimation model that guesses the camera intrinsics given an image.

4.1 Approximating the Camera Intrinsics

Since we still didn't perform any camera calibration, and we needed a camera intrinsics matrix to estimate E, we approximated the camera intrinsics matrix K from the image resolution. The following assumptions were made:

- Focal Length is proportional to the image size.
- The principal point lies at the image center.

$$K = \begin{bmatrix} \max(w,h) & 0 & \frac{w}{2} \\ 0 & \max(w,h) & \frac{h}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

Where w and h are the image width and height.

While the estimation is not physically accurate, it's commonly used in early-stage experiments and proved to be sufficient for testing.

4.2 Essential Matrix Estimation with Approximate Intrinsics

We estimated the Essential Matrix using the approximated intrinsics using RANSAC. Even with rough intrinsics, this step improved our pipeline compared to the Fundamental matrix, and gave:

- Stronger geometric constraints on feature correspondences.
- More reliable outlier rejection
- Improved numerical stability

4.3 Recovering Rotation and Translation

Once the Essential Matrix was estimated, it was decomposed into rotation and translation using SVD decomposition by the built-in cv2.recoverPose function. This process gave us:

1) Relative Rotation Matrix R
2) Translation direction t, recovered up to an unknown scale.

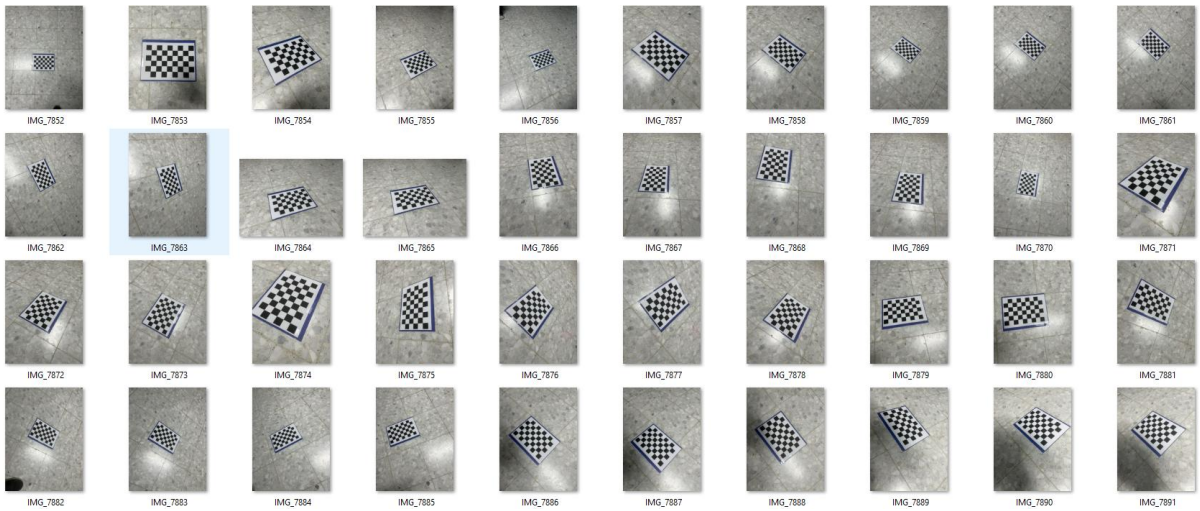
This was a major milestone in our project, as we now have a pipeline that consistently returned meaningful rotation matrix. However, pose accuracy was highly sensitive to image resolution and scaling. Which brought us to the next major improvement in the project: **Camera Calibration.**

Section 5 – Camera Calibration

The strong sensitivity to image resolution and scaling observed in the previous section indicated that the approximate intrinsics estimation model was insufficient. Thus, we introduced **explicit camera calibration** to stabilize the pipeline and obtain reliable pose estimates.

5.1 Calibration Procedure

Camera calibration was performed offline using a standard chessboard pattern and the OpenCV calibration pipeline.



The calibration process consisted of:

1) Capturing multiple images of the chessboard from different viewpoints.
2) Detecting 2D corner locations in each image.
3) Associating them with known 3D world coordinates.
4) Estimating the camera intrinsic matrix and lens distortion parameters (distortion was not used, as it was not required in our final project nor given as an input).

5.2 Integrating Calibration into the Pipeline

Once calibration was completed, the estimated intrinsic matrix replaced our guessed matrix used earlier. Special care was taken to ensure consistency between images and preprocessing and calibration, for instance:

- When images were resized, the intrinsic matrix was scaled accordingly
- The same calibrated intrinsics were used across all of the experiment.

While calibration led to improved results, we still needed to improve the quality of feature correspondences.

Section 6 - Improving Feature Matching Robustness

6.1 Lowe's Ratio Test

Instead of accepting the single nearest neighbor for each descriptor, the two closest matches were compared. A match was accepted only if the distance to the best match was sufficiently smaller than the distance to the second-best match. This helped us to eliminate ambiguous matches that may be found in:

1) Repetitive textures and patterns
2) Low texture regions
3) Visually similar features

6.2 Symmetric Mutual Matching

To further improve robustness, symmetric mutual matching was introduced. A feature is accepted only if:

1) A feature in image 1 matches a feature in image 2
2) The same feature in image 2 matches back to the original feature in image 1

Cons: Increase in computational cost, but more accurate correspondences.

This alone, combined with the calibration, improved our matching strategy and rotation estimates showed reduced variance.

Section 7 – Evaluation and Validation

After many observations, with sections 5 and 6, we observed that the rotation variance has decreased. **However,** this was insufficient to evaluate our pipeline. Thus, we needed a quantitative approach to compare our pipeline to.

7.1 – Ground Truth using ArUco Markers

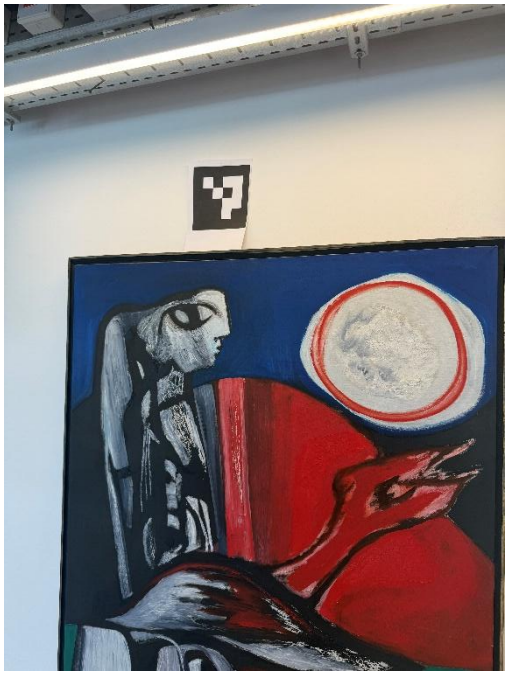
To obtain ground truth relative pose estimates, **ArUco markers** were used as an external reference system.

ArUco Markers provide:

1) Known Marker Geometry
2) Robust pose estimation using a calibrated camera
3) Reliable relative pose measurements between views

ArUco was used **only** for **internal testing and evaluation purposes** and was not part of the final pose estimation pipeline.

The figure below shows an ArUco that we used to test pose estimation on a calibrated iPhone camera.



7.2 – Error Metrics:

To compare the estimated pose against ground truth, error metrics were defined in terms of rotation and translation. Since monocular pose estimation cannot recover absolute scale, our evaluation focuses on **rotation/angle accuracy.** The following metrics were used:

1) **Rotation Error (Degrees):** Angular difference between our estimated and ground truth rotation matrices.
2) **Euler Angle Error:** Difference in roll, pitch and yaw components.
3) **Translation direction consistency:** Qualitative comparison of translation direction.

7.3 – Aggregated Evaluation Metrics

In addition to the error measurements above that were done per pair, aggregated statistics were computed over all evaluated image pairs in order to better understand the overall pipeline performance.

Note: Aggregated evaluation was performed on sets of images captured from the same static scene. For each scene, more than 25 images were captures, and later a maximal subset of pairs (300+ pairs per scene) were created. Aggregated statistics were then computed over these subsets. Figures 8.1/8.2/8.3 show pure csv files and a folder containing images of a lab scene.

For each experiment configuration and for each maximal matched subset, the following statistics were reported:

1) Count
2) Mean
3) Median
4) Standard Deviation
5) Minimum
6) Maximum

For each of the following error metrics:

1) Rotation Error (Degrees)
2) Euler RMS Error (Degrees)
3) Translation Direction Error

For different values of:

1) Ratio Thresholds
2) Ransac Thresholds and Probability
3) Image down scaling methods and scales
4) Symmetry Matching, (additional methods, such as D1-gated matching, were tested but are not reported here).

Example Aggregated Evaluation Metrics for Lab Set:

Simple Lowe's Ratio Test with Down Sampling:

Metric	rot_err_deg	euler_rms_err_deg	t_dir_err_deg
Count	325	325	325
Mean	10.8836	6.7741	21.4086
Median	4.3301	2.6870	7.3806
Std Dev	18.9934	12.2895	25.3389
Min	0.2016	0.1166	0.1097
Max	178.9084	121.3573	89.8752

Simple Lowe's Ratio Test without Down Sampling: (Larger errors)

Metric	rot_err_deg	euler_rms_err_deg	t_dir_err_deg
Count	325	325	325
Mean	18.0171	12.0416	24.4462
Median	4.5764	2.9932	8.5357
Std Dev	35.6880	25.7373	27.2520
Min	0.1327	0.0769	0.2489
Max	177.2957	138.8761	89.8334

Adaptive Ratio tests with adaptive matching

Metric	rot_err_deg	euler_rms_err_deg	t_dir_err_deg
Count	325	325	325
Mean	3.8796	2.5079	8.0619
Median	2.6913	1.7140	4.0673
Std Dev	4.8106	3.1383	13.5748
Min	0.1224	0.0722	0.1528
Max	44.5725	26.3258	87.8199

We also did adaptive ratio tests with adaptive matching, which will be discussed in section 8.
(Adaptive matching: Choose symmetric matching when inliers are high, else one way matching).

Section 8 – Descriptor Choice and Resolution Sensitivity

With a quantitative evaluation framework, it became possible to analyze how different design choices affected pose accuracy. One factor that always emerged during evaluation is that our pipeline was really sensitive to image resolution and scaling, particularly, using ORB features.

8.1 Resolution Sensitivity with ORB

Using the ArUco evaluation allowed us to evaluate our ORB based pipeline. While ORB-based matching produced stable results at certain resolutions, evaluation revealed a strong dependency on the chosen image scale. For instance, scaling the image down from 3500 pixels to 900 pixels (max length of a single side), ORB gave good results where in a lot of the pose estimations the error was very low, sometimes the error was 6 degrees only.

We noticed, however, that when we change the image scale to 1000 instead of 900 pixels, or to 2500 pixels, ORB error rises from 6 degrees to 60 degrees.

8.2 Motivation for Evaluation SIFT

The resolution sensitivity observed with ORB motivated an investigation into alternative feature descriptors with stronger scale invariance.

SIFT (Scale-Invariant Feature Transform) was selected for comparison due to:

1) Explicit scale-space construction
2) Improved robustness to scale changes
3) Widespread use in geometric vision tasks.

8.3 ORB vs SIFT: Evaluation Based Comparison

The pipeline was re-evaluated using SIFT features while keeping all other components unchanged, including:

1) Camera Calibration
2) Matching Strategy (Except, that now we use L2 NORM instead of Hamming Distance)
3) Essential Matrix Estimation and Pose Recovery

This allowed for a fair comparison between ORB and SIFT under identical conditions.

Key Observations:

1) SIFT produced more stable rotation estimated across different image resolutions.
2) Pose accuracy varied more smoothly with scaling.
3) Extreme failure cases observed with ORB were **reduced**

However, SIFT comes at a huge cost:

1) SIFT was noticeably about 5 times slower than ORB
2) Descriptor computation and matching time increased

Below, are two figures from an excel sheet that includes pose estimation on 325 pairs. About 30 images were taken of the same static scene from different viewpoints, and 325 pairs were made.

The figure below depicts the results of the ORB matching:

As we can see from the highlighted entries (which are the error entries), just by changing to SIFT, the error dropped significantly. In the first row, error fropped from 8.4 to 2.87, which is a ~75% drop.

img1	img2	aruco_yav	aruco_roll	aruco_pitc	core_yaw_	core_roll_	core_pitch	rot_err_de	euler_rms	t_dir_err_	used_core
202601211	202601211	-13.1591	5.728262	-47.5669	-4.43349	-5.00551	-49.4006	7.371237	8.056306	8.415538	0
202601211	202601211	1.147073	-1.67742	-7.10448	-2.13263	0.422885	-8.34364	3.853546	2.359609	3.096139	0
202601211	202601211	3.285091	3.092451	18.06398	4.328899	1.543067	26.9618	9.159361	5.249167	40.19444	0
202601211	202601211	-3.93716	-4.64853	-23.027	-5.95038	-3.1818	-16.8407	6.515685	3.850317	23.71913	0
202601211	202601211	-5.78187	1.542374	-32.5451	-6.89652	1.988496	-32.9494	1.032927	0.731421	3.203395	0
202601211	202601211	-0.36183	-1.65977	-3.31941	-0.6749	-2.01299	-4.88266	1.637788	0.942788	3.034997	0
202601211	202601211	-12.4788	1.825012	-56.7136	-19.4435	13.4992	-52.2713	8.483595	8.256879	9.984617	0
202601211	202601211	7.220419	3.913026	33.29886	8.891514	6.700233	39.639	6.723649	4.113326	0.928292	0
202601211	202601211	3.400807	1.510247	15.81817	3.566259	1.040121	18.69527	2.927834	1.685834	9.595406	0
202601211	202601211	2.544865	1.64851	11.09503	2.329418	7.117717	9.84245	5.652838	3.24179	40.25601	1
202601211	202601211	19.05787	12.61154	61.47502	23.80622	19.99095	64.02222	4.600529	5.275436	3.146159	0
202601211	202601211	-9.73369	4.870749	-35.9771	-3.87936	-1.76355	-3.77504	32.96048	19.28089	48.55652	0
202601211	202601211	-0.65043	-5.38152	-22.8845	-5.37106	-0.47572	-28.5486	7.634073	5.113181	3.646691	0
202601211	202601211	-4.45998	1.208335	-27.8082	-6.76957	23.07955	-4.09469	31.79705	18.67275	46.52634	1
202601211	202601211	-1.96789	-4.19545	-37.8751	-11.044	3.808649	-47.279	11.66452	8.848247	3.886148	0
202601211	202601211	-5.76493	2.086254	-28.6589	-6.76391	0.927769	-18.0612	10.75006	6.182004	11.36098	0
202601211	202601211	-3.45461	1.194966	-15.1561	-7.00298	7.268253	-21.3858	8.643263	5.424774	2.341211	0
202601211	202601211	3.280552	4.814008	14.72438	3.694395	1.836725	18.73017	5.078141	2.891472	10.41888	0
202601211	202601211	11.55772	6.161711	46.30409	-1.63792	1.71145	1.340487	46.4419	27.17631	45.28707	1
202601211	202601211	-0.83278	1.195564	-5.75789	-0.55393	0.057039	-5.11915	1.31217	0.770711	4.76091	0

Figure 8.1 - ORB Matching (First 20/325)

img1	img2	aruco_yav	aruco_roll	aruco_pitc	core_yaw_	core_roll_	core_pitch	rot_err_de	euler_rms	t_dir_err_	used_core
202601211	202601211	-13.1591	5.728262	-47.5669	-12.0094	5.115706	-50.834	3.362182	2.030691	2.876742	0
202601211	202601211	1.147073	-1.67742	-7.10448	-2.16912	2.297655	-7.86998	4.893281	3.021278	5.362839	0
202601211	202601211	3.285091	3.092451	18.06398	3.486969	0.493265	21.19738	4.118931	2.353344	21.69985	0
202601211	202601211	-3.93716	-4.64853	-23.027	-6.37606	0.443327	-22.4526	4.754029	3.276437	18.44863	0
202601211	202601211	-5.78187	1.542374	-32.5451	-7.08536	1.50712	-33.4697	1.614069	0.922892	3.517903	0
202601211	202601211	-0.36183	-1.65977	-3.31941	-0.98533	-0.72697	-3.96735	1.266792	0.748042	1.366674	0
202601211	202601211	-12.4788	1.825012	-56.7136	-19.6925	11.36727	-61.0829	6.646311	7.352619	6.493169	0
202601211	202601211	7.220419	3.913026	33.29886	8.268428	3.01233	33.98998	1.852871	0.892041	2.251525	0
202601211	202601211	3.400807	1.510247	15.81817	3.75154	1.02654	17.53841	1.847579	1.051379	8.171746	0
202601211	202601211	2.544865	1.64851	11.09503	2.604803	0.046224	11.90947	1.809002	1.038305	2.233646	0
202601211	202601211	19.05787	12.61154	61.47502	22.39731	14.83859	67.3617	6.109679	4.113578	2.875845	0
202601211	202601211	-9.73369	4.870749	-35.9771	-1.33699	-2.14205	-4.05686	33.08412	19.48148	50.85271	0
202601211	202601211	-0.65043	-5.38152	-22.8845	-4.9313	1.191783	-23.2059	6.292013	4.532748	6.232757	0
202601211	202601211	-4.45998	1.208335	-27.8082	-5.65313	1.01265	-27.0149	1.518599	0.834902	1.984708	0
202601211	202601211	-1.96789	-4.19545	-37.8751	-8.00008	2.695099	-44.0727	8.22401	6.384288	3.862507	0
202601211	202601211	-5.76493	2.086254	-28.6589	-3.3793	-2.02201	-31.1061	4.334304	3.08535	7.017714	0
202601211	202601211	-3.45461	1.194966	-15.1561	-3.37376	0.530604	-17.6236	2.550697	1.47608	2.091654	0
202601211	202601211	3.280552	4.814008	14.72438	3.104283	0.156876	17.91339	5.605595	3.260353	12.70356	0
202601211	202601211	11.55772	6.161711	46.30409	13.74377	7.744609	52.24556	6.109964	3.767644	1.39945	0
202601211	202601211	-0.83278	1.195564	-5.75789	-0.57211	0.022832	-4.75391	1.547636	0.903921	6.038815	0

Figure 8.2 - SIFT Matching (First 20/325)

The figure below shows **some** of the images taken that were later turned into 325 pairs:

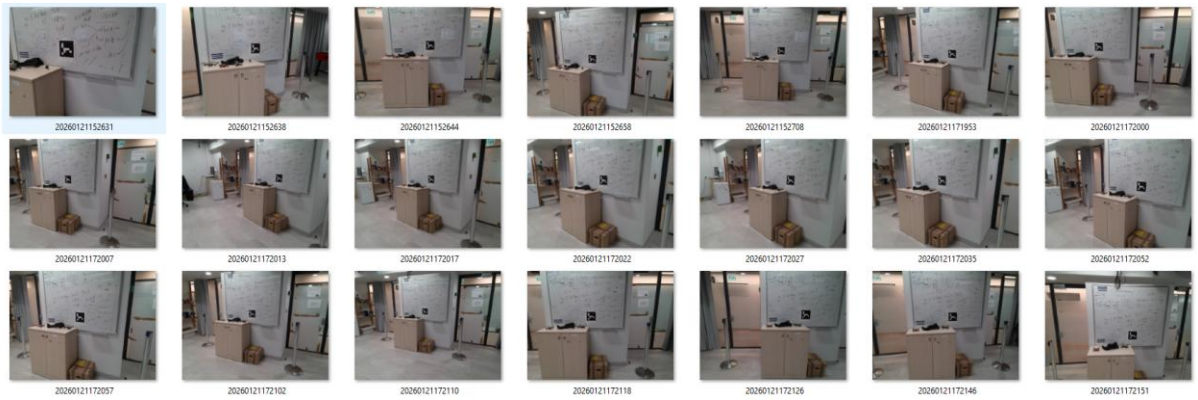


Figure 8.3 – Images took of the same scene, that were later paired and matches (maximal subset).

8.4 Trade-Offs and Design Considerations

The comparison highlighted a clear trade-off between speed and robustness:

- ORB offers high performance and efficiency but is sensitive to resolution changes, and produces a huge error.
- SIFT provides improved stability and accuracy at the expense of computational cost.

Section 9 – Debugging and Visualization Tools

Throughout the development of our code, extensive **debugging and visualization tools** were used to validate our results, run diagnostics and diagnose failures.

The following debugging and visaulizations were done:

1) Feature Detection and Matching Visualization
2) Inlier and Outlier Visualization
3) Epipolar Geometry Visualization
4) Pose and Orientation Visualization

Figure 9.1: Keypoints detected in image 1 and 2

Figure 9.2 – Raw Matches

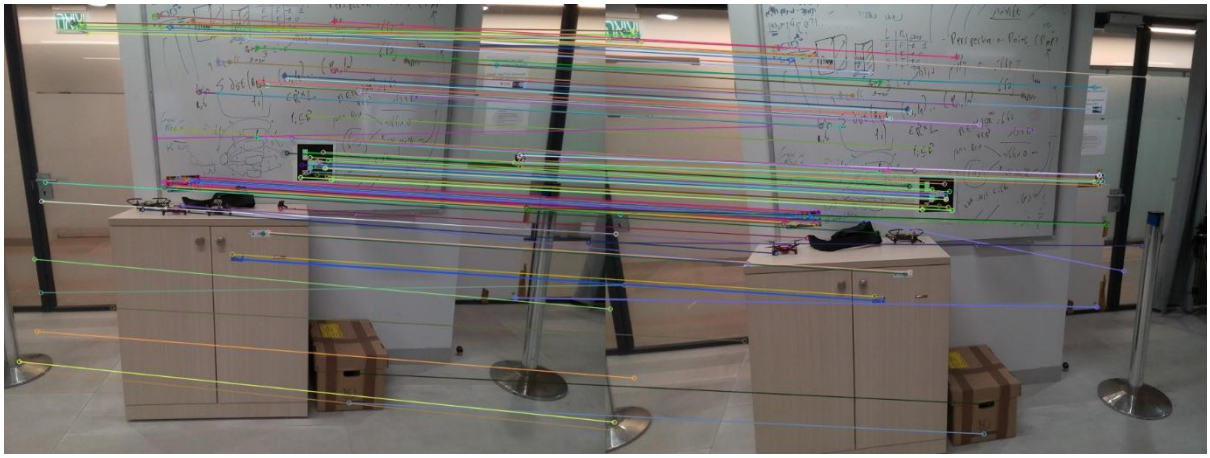
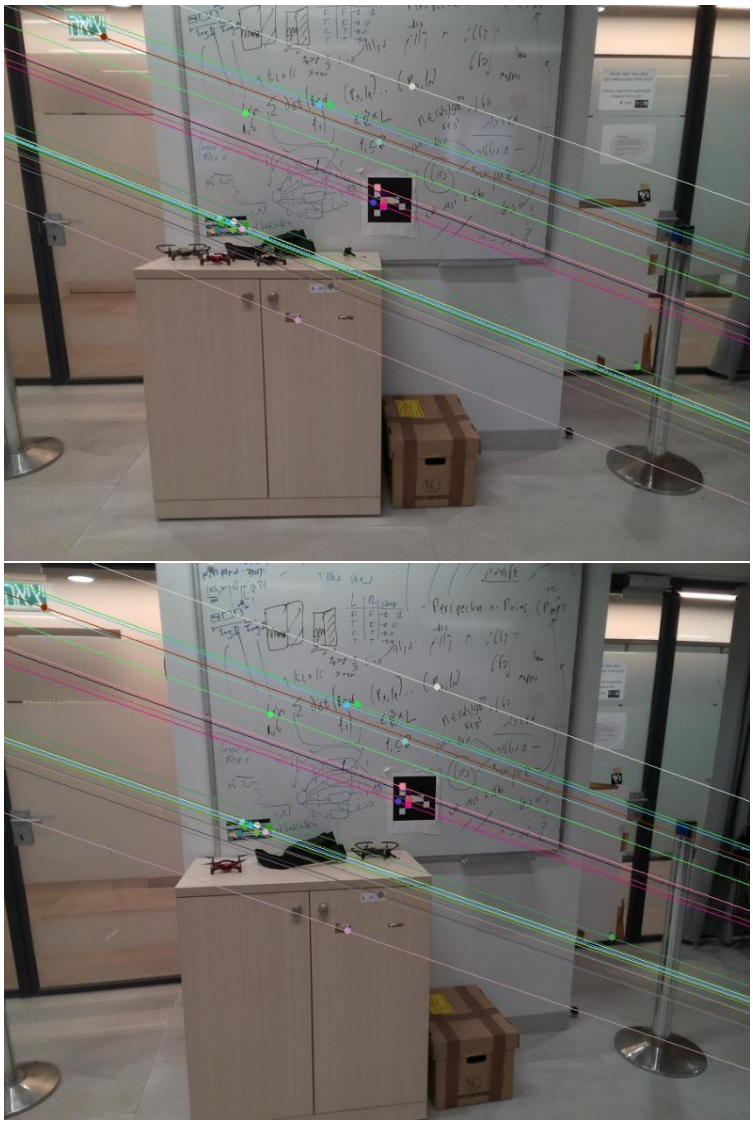


Figure 9.3 – Inlier Matches

Figure 9.4: Epipolar Lines in Image 1 and 2



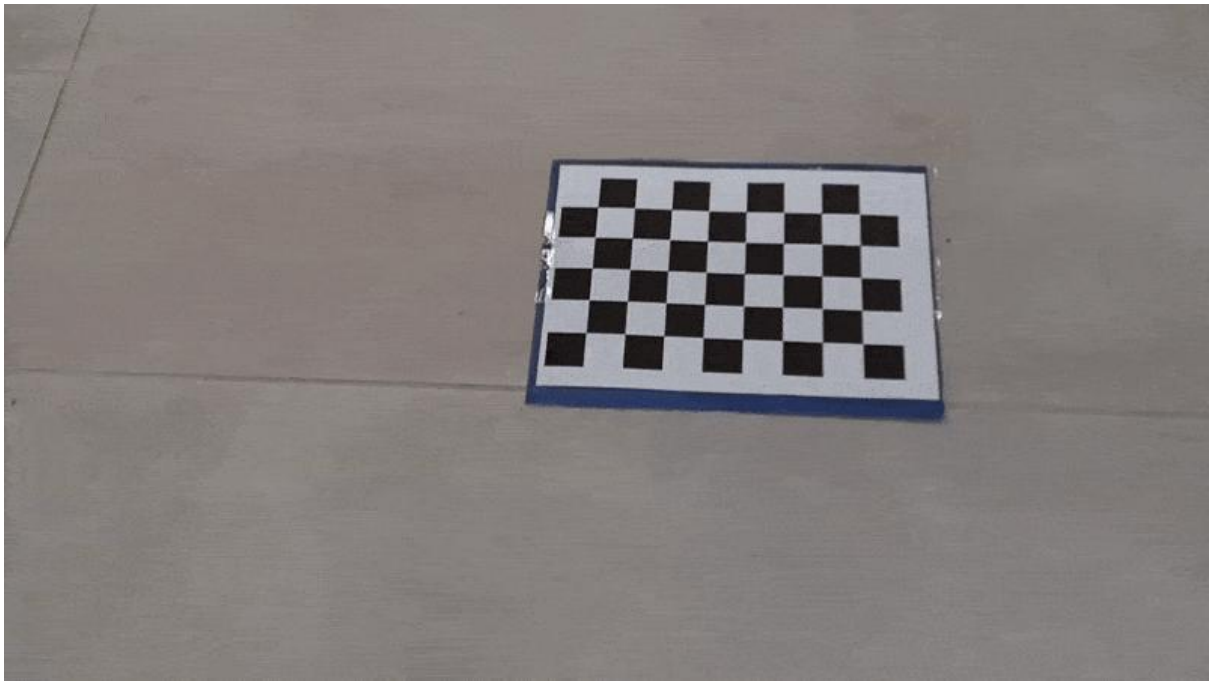
Section 10 – Video Experiments

In addition to the images, we tested our pipeline on short videos to analyze its behavior better. These experiments were not intended to produce a full VO system but rather serve as debugging tests.

10.1 Video Input and Calibration Considerations

Videos, just like images, have their own camera mode and resolution. Thus, a **separate camra calibration** was required for **video input**.

Below is a **GIF** of the video we took to calibrate our camera for video input. Please **click** on it to open the **GIF**.



10.2 Observed Behavior on Video Sequences

When applying the two-view pose estimation pipeline frame-to-frame on video input, results were mostly unstable and noisy.

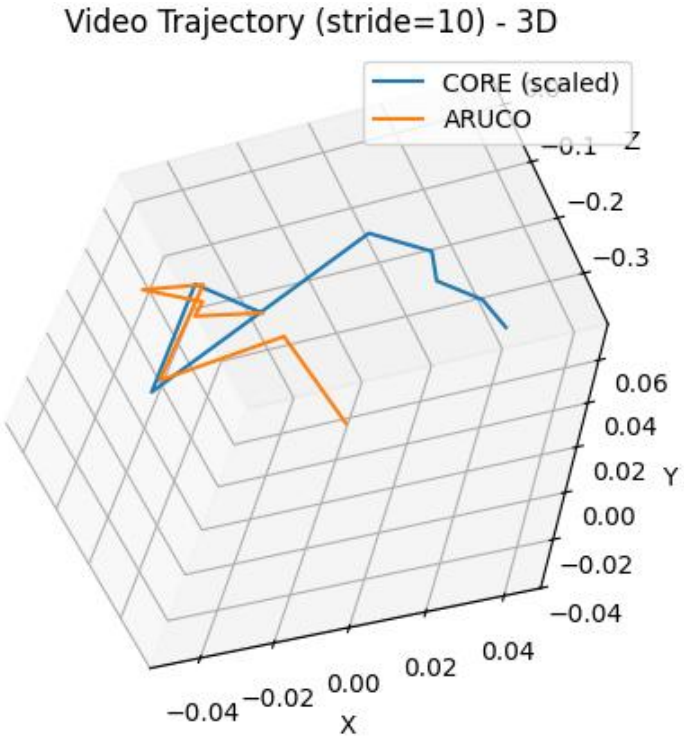
Issues that were present, included:

1) Large frame-to-frame pose variations
2) Sensitivity to small viewpoint changes
3) Occasional pose flips and inconsistent rotation estimates

However, these behaviors were totally expected. As our original pipeline wasn't designed for complete video slam, but rather two image input.

10.3 Comparison with ArUco-Based Ground Truth

To better understand the failures, we also used ArUco as a reference after also adapting it to take frame by frame.



Here's also another run for another video, where the results were outputted to a .csv file, where we can see the error between each and every frame. (ArUco vs Core):

step_idx	frame_idx	frame_idx	used_strid	core_ok	aruco_ok	core_step	aruco_ste	rot_err_de	tdir_err_d	aruco_t_n	core_t_no	scale_m_p
0	0	2	2	1	1	0.48832	0.49178	0.18706	70.9584	0.00479	1	0.00479
1	2	4	2	1	1	0.79547	0.78735	0.3719	78.7356	0.01606	1	0.01606
2	4	5	1	1	1	0.55562	0.95646	0.48834	72.1586	0.02317	1	0.02317
3	5	6	1	1	1	1.38269	1.42152	0.31218	32.2121	0.01038	1	0.01038
4	6	9	3	1	1	1.44575	0.94848	0.61573	49.6956	0.02556	1	0.02556
5	9	10	1	1	1	0.27666	0.48696	0.24827	68.2585	0.00911	1	0.00911
6	10	13	3	1	1	1.31158	1.92083	0.64787	82.8409	0.03535	1	0.03535
7	13	14	1	1	1	0.23572	0.09399	0.20701	21.487	0.01438	1	0.01438
8	14	15	1	1	1	0.81975	0.52921	0.29953	77.3918	0.01598	1	0.01598
9	15	17	2	1	1	1.10905	1.1529	0.16518	42.3529	0.01209	1	0.01209
10	17	18	1	1	1	0.54362	0.93679	0.40861	81.3438	0.02175	1	0.02175
11	18	19	1	1	1	0.78391	0.66719	0.1368	54.7227	0.00961	1	0.00961
12	19	20	1	1	1	0.8233	1.4224	0.63825	28.0098	0.04068	1	0.04068
13	20	21	1	1	1	6.38033	4.98264	1.53271	4.39688	0.18648	1	0.18648
14	21	22	1	1	1	1.14716	1.08654	0.153	3.93487	0.06389	1	0.06389
15	22	25	3	1	1	1.68354	1.3015	0.50807	8.96126	0.09074	1	0.09074
16	25	28	3	1	1	3.37871	3.0751	0.67772	6.95942	0.22452	1	0.22452
17	28	29	1	1	1	1.33509	3.42286	4.67182	44.6415	0.28554	1	0.28554
18	29	30	1	1	1	1.46241	1.10528	0.57745	11.4951	0.1023	1	0.1023
19	30	31	1	1	1	0.34056	0.93079	1.24786	42.252	0.08305	1	0.08305
20	31	32	1	1	1	1.00186	0.8507	0.41724	14.8383	0.07721	1	0.07721
21	32	33	1	1	1	3.27675	2.09385	2.40647	11.9664	0.44178	1	0.44178
22	33	34	1	1	1	0.07566	22.0175	21.9477	14.5063	1.39022	1	1.39022
23	34	36	2	1	1	0.04877	24.1682	24.1781	31.6802	1.38548	1	1.38548
24	36	37	1	1	1	0.06135	1.07252	1.11795	47.2812	0.03774	1	0.03774
25	37	38	1	1	1	0.1494	19.732	19.6197	7.00266	1.40425	1	1.40425
26	38	40	2	1	1	0.44767	24.4795	24.0496	85.4983	1.49181	1	1.49181
27	40	41	1	1	1	0.19322	22.7909	22.975	76.0602	1.4378	1	1.4378

We can see from the results above, that on that specific video run, the rotation error was almost always negligible except for some outliers. However, the translation error is more dominant and that's because most scenes had parallax.

10.4 Insights Gained from Video Experiments

Despite the poor performance obtained, the video experiment proved to be valuable for guiding us for further improvements, which will be discussed in **section 11.**

In particular, video input helped reveal:

1) Sensitivity to matching thresholds
2) The importance of adaptive parameter selection
3) The need for stricter match filtering under motion

These insights motivated us to refine our project better and focus more on parameter tuning.

Section 11 – Improvements and Beyond

While the final pipeline achieved stable results compared to our ground truth, several additional improvements were explored during development, and even further enhancements were done beyond the current scope.

11.1 Adaptive Ratio Test for Descriptor Matching

One limitation observed during evaluation was the sensitivity of the fixed matching thresholds (Lowe's Ratio) in different images and scenes. To address this, adapting matching strategies were implemented.

The adapting matching strategy implemented:

```python
def adaptive_ratio_test(knn_matches):
    distances = []
    for pair in knn_matches:
        if len(pair) >= 2:
            distances.append(pair[0].distance)

    if not distances:
        return 0.75

    median_dist = np.median(distances)

    if median_dist < 100:
        ratio = 0.55
    elif median_dist < 200:
        ratio = 0.70
    elif median_dist < 300:
        ratio = 0.80
    elif median_dist < 400:
        ratio = 0.85
    else:
        ratio = 0.90

    std_dist = np.std(distances)
    cv = std_dist / (median_dist + 1e-6)

    if cv > 0.7:
        ratio += 0.05
    elif cv < 0.2:
        ratio -= 0.05

    return float(np.clip(ratio, 0.5, 0.95))
```

The ratio threshold now is adjusted dynamically according to:

1) Median descriptor distance
2) Descriptor distance dispersion

This allows the matching process to:

1) Be strict in the high quality scenes with quality matches.
2) Remain permissive and lenient in challenging scenes with low-texture or parallax.

The adaptive ratio testing reduced catastrophic matching failures and improved the results significantly.

11.2 Image Scaling Strategy Revisited

Early experiments showed the simple width-based or height-based image resizing led to severe instabilities, as most of the downscaling algorithms that were tested, a simple scale change can lead to catastrophic results. However, to minimize the issue, the final pipeline adopted an area-based image downscaling. Instead of having a fixed width to scale according to, now we scale down by targeting a fixed number of pixels.

This design choice:

1) Preserved aspect ratios
2) Is camera agnostic
3) Produces more consistent behavior than width-based scaling.

11.3 Adaptive Symmetric Matching Selection System

Symmetric (mutual) matching improves the correspondence reliability but has a very noticeable disadvantage. It reduces the match count in difficult casing, leading to few inliers to work with. Thus, to balance robustness and also the availability of matches, the pipeline now dynamically selects between:

- Symmetric matching when sufficient mutual correspondences are available.
- One-way matching (normal matching) when mutual matches fall below a minimum threshold.

11.4 Handling Multiple Essential Candidate Matrices and Robustness

When multiple Essential Matrix candidates are returned, each candidate is evaluated independtly through the pose recovery function. The final pose is selected based on inlier consistency after recovery.

While this selection criterion is not guaranteed to identify and pick the optimal solution in all cases, it proved to be effective in practice and a lot of cases.

In addition, RANSAC was dismissed, and MAGSAC was selected as a more robust estimator.

11.5 Minimum Inlier Logic and Confidence Awareness

To avoid any unnecessary failures, the pipeline introduces a minimum inlier logic and a variable for confidence awareness:

1) Below a theoretical minimum: which is 5 inliers, the pipeline will not generate any pose.
2) Low Inlier count: The pipeline will return a result with a low confidence warning.
3) Sufficient Inliers: Normal operation.

Section 12 – Conclusion

The project presented a systematic exploration of monocular two view pose estimation, that was developed through an iterative and evaluative engineering process. Starting from a simple baseline, where certain matrices were “guessed” to a final pipeline that follows a rigid robust structure.

We encountered a lot of obstacles in this project, from sensitivity to camera resolutions, instability in feature matching , limitation of robust camera intrinsics, and much more. However, all of these issues were addressed and led to important design decisions, which were mentioned throughout this report.

While the final pipeline produces stable and meaningful pose estimates, monocular pose estimation remain inherently ambiguous and sensitive to scene geometry. Between decision of optimizing for speediness and accuracy, we took measures to optimize speed and accuracy. Although there exist more complex approaches to the project, efficiency was a huge factor.