Southampton Ocean Engineering Joint Institute

at Harbin Engineering University

Vision Measurement and Control for Robots

Lab-IV Experiment Report

A Monocular Visual Odometry Experiment for Camera Localization Using Smartphone Video

Class: 20222832

Name: 于洪洋

Student ID： 2022283219

Teacher's Name: 朱齐丹

Experiment Time: 23/11/2025

Abstract

The experiment investigates a simple feature-based monocular visual odometry (VO) pipeline applied to handheld smartphone video recorded around the main building of Harbin Engineering University. Using ORB keypoints and binary descriptors, consecutive keyframes are matched and the essential matrix is estimated with RANSAC to recover relative camera pose under an approximate pinhole model with FOV-based intrinsics. The incremental motions are integrated to obtain an up-to-scale 3-D trajectory, which is then smoothed with a sliding-window average, and complemented by per-step image-motion analysis and global statistics. On a 49-second 1080p sequence, the system successfully reconstructs a trajectory of 745 keyframes, with a total accumulated image motion of 1837.56 pixel-steps and a straightness index of 0.176, clearly reflecting the planned straight–turn–slanted path while exhibiting noticeable drift between start and end positions. The results qualitatively confirm the expected behaviour and limitations of monocular VO—scale ambiguity, drift and sensitivity to motion patterns—and motivate future extensions with camera calibration, additional sensors and loop-closure optimisation.

I. Experimental Background

Accurate localization is a core requirement for mobile robots and autonomous systems. The lecture notes introduce map-based localization and SLAM, in which a vSLAM pipeline typically contains input search, pose tracking, mapping and loop closing. Visual odometry (VO) is the pose-tracking component: it estimates the relative motion of a camera by analyzing consecutive images. In the monocular case, the recovered trajectory is only defined up to an unknown global scale and suffers from drift, but it requires only a single camera and no additional range sensors. This experiment implements a simple feature-based monocular VO method on smartphone video to realize camera localization in a relative coordinate frame.

II. Experimental Purpose

A. Brief Review of Principle

Feature-based monocular visual odometry relies on the calibrated pinhole camera model. A 3-D point $\mathbf{X}$ in homogeneous world coordinates is projected to an image point $\tilde{\mathbf{x}} = (u, v, 1)^{\top}$via

$$\lambda\tilde{x} = K[R \mid t]\tilde{X}$$

where K is the intrinsic matrix, $R$and $\mathbf{t}$ are the rotation and translation from world to camera, and λ is an unknown depth scale.

Given two consecutive frames and their calibrated image points $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}}'$, the relative motion satisfies the epipolar constraint

$$\tilde{x}'^{\top} E\tilde{x} = 0, \quad E = [t]_{\times} R,$$

where E is the essential matrix and $[\mathbf{t}]_{\times}$is the skew-symmetric matrix of $\mathbf{t}$. In practice, E is estimated robustly from many point correspondences using RANSAC; decomposing E yields the relative rotation $R$and translation direction $\mathbf{t}$ up to scale.

By chaining these incremental motions,

$$T_k = T_{k-1} T_{k-1 \to k},$$

the full camera trajectory $\{T_k\}$ is obtained in a common reference frame, albeit with unknown global scale and accumulated drift.

B. Objectives

The specific objectives of this experiment are:

1) To implement a basic feature-based monocular visual odometry pipeline on smartphone video, including feature extraction, feature matching, essential-matrix estimation and incremental pose integration.

2) To reconstruct and visualize the camera trajectory in a relative coordinate frame, and to extract per-step image-motion and global statistics (path length, displacement and "straightness").

3) To qualitatively and quantitatively examine the characteristic errors of monocular VO (scale ambiguity and drift), and relate the observations to the localization and vSLAM concepts introduced in the course.

III. Experimental Equipment

A. Image Device

The video was recorded using the **main wide camera of a Xiaomi 15** smartphone. The module employs a 50 MP "Light-Hunter 900" high-dynamic-range sensor with a large photosensitive area, f/1.62 aperture and a 23 mm (equivalent) focal length. In normal use it performs pixel binning to an effective 2.4 μm pixel size and supports optical image stabilization (OIS), quad phase-detection (QPD) autofocus and ALD lens coating. The experiment used the default photo/video mode, with manual adjustment of focus and exposure where necessary.

To reduce hand-shake and obtain smoother motion, the phone was mounted on a **DJI Osmo Mobile 7** hand-held gimbal and operated in landscape orientation.

B. Recording Specifications

The raw footage was captured at a resolution of **1920×1080 (1080p)** and **30 frames per second**. After trimming for stable segments, the final clip used for visual odometry has a duration of approximately 49 seconds.

C. Recording Path

The recording took place near the main building of Harbin Engineering University. The planned path consists of three segments: **an initial approximately straight walk**, **a left turn**, and **a second straight segment with a slight inclination relative to the first**. This provides both straight-line motion and a clear change of heading for evaluating the reconstructed trajectory.

D. Host Computer and Software

MECHREVO Yaoshi 16 Ultra, with:
CPU: Intel® Core™ Ultra 9 275HX
GPU: NVIDIA GeForce RTX 5070 Ti Laptop GPU
Memory: 32 GB RAM
Software Stack: Python **3.12.7**; NVIDIA CUDA **12.8.97** driver; OpenCV (**cv2**) **4.12.0**; PyTorch **2.8.0+cu128**.

IV. Experiment Procedures

A. Video Acquisition

The smartphone was mounted on the DJI Osmo Mobile 7 gimbal and operated in landscape mode. A single continuous sequence was recorded along the planned path near the main building, at 1920×1080 resolution and 30 fps. After discarding unstable segments at the beginning and end, the clip was trimmed to a duration of approximately 49 s and then transferred to the host computer as new_walk.mp4. The file path was specified in the configuration of mono_vo.py (VIDEO_PATH).

Figure 1: Hand-held capture rig — Xiaomi 15 main wide camera mounted on a DJI Osmo Mobile 7 gimbal. The phone is used in landscape orientation at 1080p/30 fps, with focus and exposure locked where necessary, to record a forward walking sequence around the HEU main building for monocular visual odometry.

B. Pre-processing and Camera Model

On the host computer, the script mono_vo.py is executed to perform all subsequent processing. First, the programme opens the input video using cv2.VideoCapture and reads the first frame. To reduce computation, each frame is resized by a factor RESIZE_SCALE = 0.5, effectively converting 1080p input to approximately 960×540 resolution.

The camera intrinsics are then defined. If calibration files K.npy and dist.npy are available, they are loaded and used to undistort the frames. In this experiment, USE_CAMERA_CALIB is set to False, so the code instead estimates a pinhole camera matrix K from the resized image width and an assumed horizontal field of view of 60° using the function build_intrinsics. All frames are converted to grayscale before feature extraction.

Parameter	Value	Description
RESIZE_SCALE	0.5	Image down-sampling factor
MAX_FEATURES	2000	Max ORB keypoints per frame
MIN_MATCHES	100	Min. matches to accept a keyframe pair
RATIO_TEST	0.7	Lowe's ratio threshold
FRAME_STEP	2	Frame interval between keyframes
SMOOTHING_WINDOW	5	Window size for 1-D trajectory smoothing
STEP_SCALE	1.0	Nominal step length in VO integration

Table 1: Visual odometry configuration parameters — summary of key hyperparameters used in the monocular VO pipeline, including image down-sampling factor, ORB feature budget, matching thresholds, keyframe interval and trajectory smoothing settings.

C. Feature Extraction and Matching

For each processed frame, ORB keypoints and binary descriptors are extracted using an cv2.ORB_create detector with MAX_FEATURES = 2000. The first frame initialises the previous keypoint set $\{k_0\}$ and descriptors. During the main loop, only every FRAME_STEP = 2-nd frame is treated as a keyframe to limit computational load.

Between consecutive keyframes, feature matching is performed with a brute-force Hamming matcher in K-nearest neighbors mode (k=2). Lowe's ratio test with threshold RATIO_TEST = 0.7 rejects ambiguous matches, and only point pairs that pass this test are retained. If fewer than MIN_MATCHES = 100 correspondences remain, the current frame is discarded as a keyframe. Otherwise, the matched pixel coordinates $\{p_i, p_i'\}$ are used as input to motion estimation.

In addition, the script computes the Euclidean displacement of each matched pair on the image plane and records the median value as the "per-step image motion", which is later plotted as a function of keyframe index.

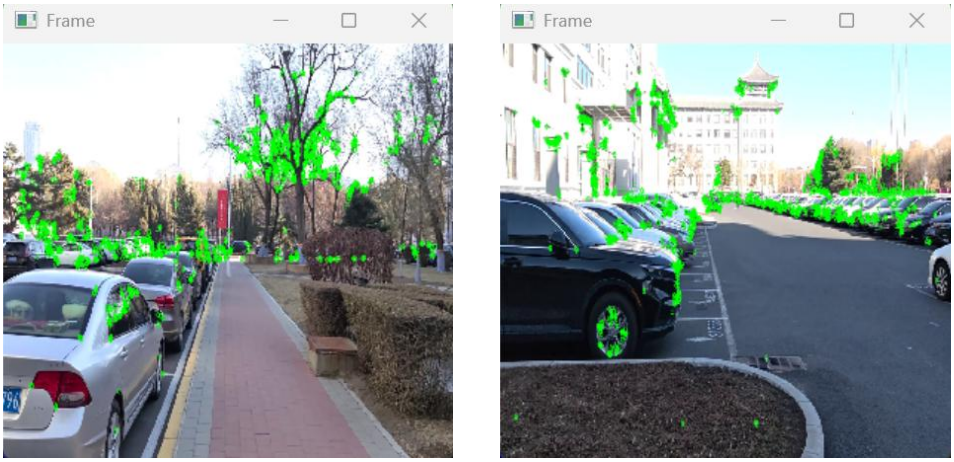

Figure 2: Example keyframe with ORB features and inter-frame matches. Green markers indicate the detected ORB keypoints (up to MAX = 2000) on a down-sampled video frame

D. Relative Pose Estimation and Trajectory Integration

For each pair of consecutive keyframes with sufficient matches, the essential matrix E is estimated using cv2.findEssentialMat with RANSAC (prob = 0.999, threshold = 1.0). This implements the epipolar constraint described in Section II-A. The function cv2.recoverPose then decomposes E into a relative rotation R_{rel} and translation direction t_{rel} between the two views.

Because monocular VO cannot observe absolute scale, the translation is normalized: the vector t_{rel} is rescaled to have length STEP_SCALE = 1.0, and an upper bound of 2 STEP_SCALE is applied to suppress occasional outliers. The global camera pose is updated recursively as

$$p_k = p_{k-1} + R_{global}\hat{t}_{rel}, \quad R_{global} \leftarrow R_{rel}R_{global},$$

where R_{global} is the cumulative orientation and p_k the position of the k-th keyframe in the chosen reference frame. The sequence $\{\mathrm{p}_k\}$ is stored as trajectory.txt for later analysis.

E. Trajectory Smoothing, Visualization and Statistics

After all frames have been processed, the list of 3-D positions is converted to arrays (x_k, y_k, z_k). To reduce frame-to-frame jitter, a simple one-dimensional sliding-window average with window size SMOOTHING_WINDOW = 5 is applied independently to the x and z components, producing a smoothed ground-plane trajectory.

The script then generates three figures using Matplotlib:

1) Main trajectory plot (trajectory.png): smoothed X–Z trajectory with the start and end points highlighted (green circle and red cross) and an arrow indicating the final motion direction.

2) Raw vs. smoothed trajectory (trajectory_raw_vs_smooth.png): overlaid plot comparing the unsmoothed and smoothed trajectories to illustrate the effect of filtering.

3) Per-step image motion curve (step_distance.png): median pixel displacement

versus keyframe index, reflecting changes in walking speed and turning.

In addition, scalar statistics are computed: the total image motion (sum of per-step medians), the Euclidean displacement between start and end positions, a straightness index (displacement divided by total motion), and the mean and standard deviation of per-step image motion. These values are printed to the console and written to vo_stats.txt for inclusion in the report.

Detailed code is provided in the Appendix A - Main Code.

V. Results

A. Global VO statistics

The implemented monocular VO pipeline successfully processed the entire 49-s video and produced a trajectory of 745 keyframes. The overall statistics, computed from the per-step image motion and the 3-D pose increments, are summarized in Table 2. The total accumulated image motion is 1837.56 pixel-steps, while the net displacement between the first and last estimated camera positions is 322.57 arbitrary units. This yields a straightness index of 0.176, indicating that the reconstructed path contains substantial turns and accumulated drift rather than forming a nearly straight segment. The mean per-step image motion is 2.47 pixels with a standard deviation of 5.15 pixels, reflecting a mixture of slow walking and occasional rapid camera rotations or translations.

Metric	Value	Interpretation
Number of keyframes	745	Number of poses used in the trajectory
Total image motion (sum steps)	1837.5590	Sum of median pixel displacements
Start–end displacement	322.5732	Euclidean distance between start and end poses
Straightness index	0.1755	Displacement / total image motion
Mean per-step image motion	2.4698	Average median pixel displacement per step
Std of per-step image motion	5.1455	Variation in per-step image motion

Table 2: Monocular visual odometry summary statistics — global indicators of the reconstructed trajectory, including the number of keyframes, accumulated image motion, start–end displacement, straightness index, and the mean and variability of per-step image motion.

Since no full calibration was available, the intrinsics were approximated from the resized image resolution $\mathbf{960 \times 540}$ and an assumed horizontal field-of-view of $\mathbf{60}^\circ$. Under a pinhole model with zero skew and centred principal point, the focal length in pixels is

$$f_x = f_y = \frac{w}{w \tan(\mathrm{FOV}/2)} = \frac{960}{2 \tan 30^\circ} \approx 831.38\text{px},$$

with principal point $(c_x, c_y) = (480,270)$.

The intrinsic matrix used in the visual odometry pipeline is therefore

$$K = \begin{bmatrix} 831.38 & 0 & 480 \\ 0 & 831.38 & 270 \\ 0 & 0 & 1 \end{bmatrix}$$

B. Per-step image motion

The per-step median pixel displacement curve is plotted in Figure 3. For most of the sequence the median motion remains below about 2 pixels, corresponding to relatively gentle frame-to-frame motion when the camera moves slowly along the straight path. Around step indices 290–340 the curve exhibits a sharp burst, with peak values exceeding 25 pixels. This segment coincides with the left turn and the subsequent oblique straight line in the walking route, where both camera translation and yaw change are larger. After this high-motion interval, the displacement gradually falls back to the 1–3 pixel range and finally decays towards zero when the camera stops moving near the end of the sequence.

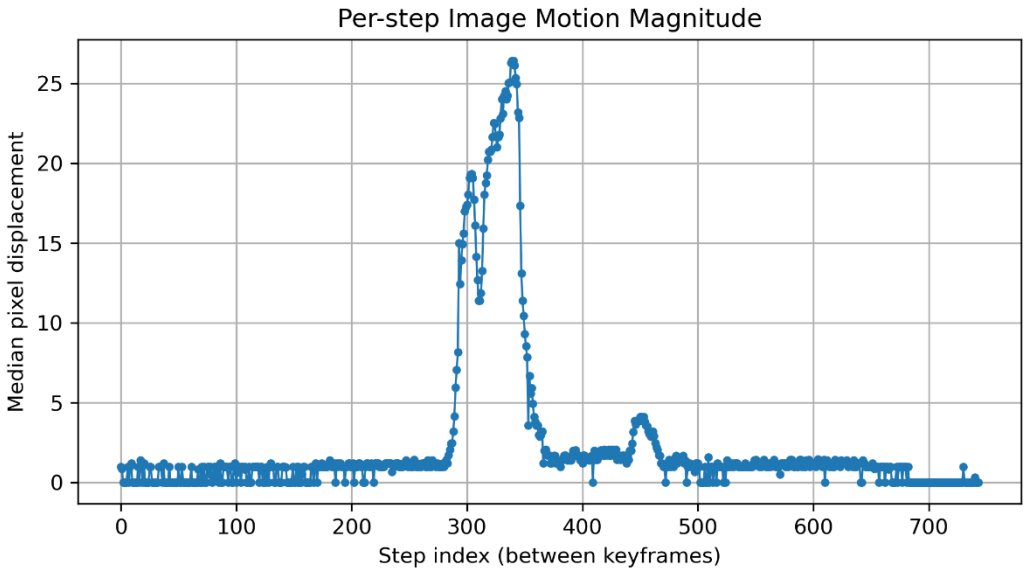

Figure 3: Per-step image motion magnitude along the keyframe sequence. Each marker shows the median Euclidean displacement (in pixels) between matched ORB features in two consecutive keyframes. A pronounced burst around step indices 290–340 corresponds to the left-turn and oblique-walk segment with faster motion, while the remainder of the sequence stays mostly below about 2 pixels and decays towards zero as the camera comes to rest.

C. Reconstructed trajectory

Figure 4 shows the up-to-scale VO trajectory projected onto the X–Z plane, using the smoothed coordinates. The overall path starts near the origin, first moves along a shallow segment, then follows a long curved descent before turning and rising back towards the final position. Qualitatively, this shape is consistent with the planned route of "straight segment → left turn → slanted straight line" around the main building. However, the start and end positions do not coincide, and their separation of 322.57

arbitrary units visualises the accumulated scale and orientation drift typical of monocular VO without loop closure.

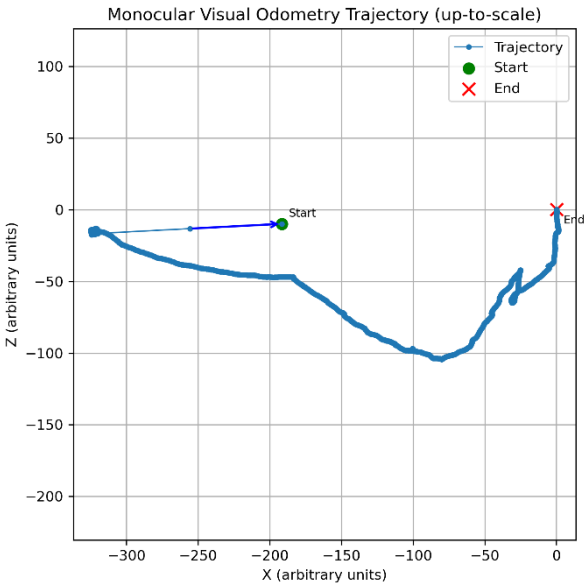

Figure 4: Up-to-scale monocular visual odometry trajectory in the X–Z plane. The smoothed path starts near the origin, follows a short almost straight segment, then traces a long curved descent before returning along a slanted straight section. The start (green) and end (red) markers do not coincide; their separation of 322.57 arbitrary units illustrates the accumulated scale and orientation drift of monocular VO without loop closure.

To illustrate the effect of temporal smoothing, Figure 5 compares the raw trajectory with the smoothed one. The raw X–Z trace contains noticeable frame-to-frame jitter and small zig-zags, especially in the high-motion region. After applying a simple sliding-window average, the smoothed curve preserves the global shape of the route while suppressing local noise, making the large-scale motion pattern easier to interpret in the absence of ground-truth pose measurements.

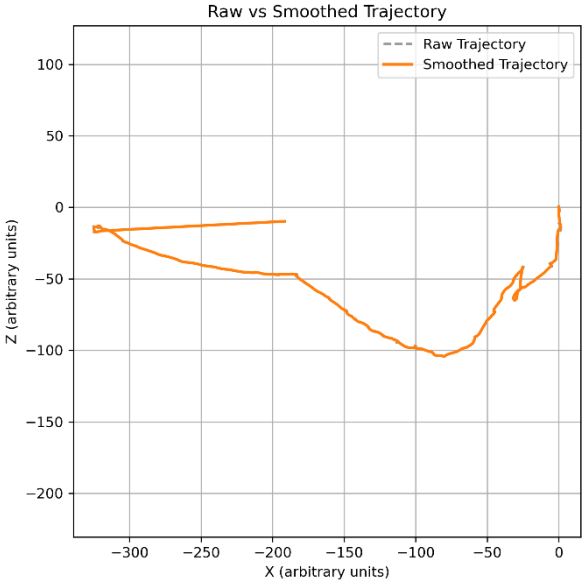

Figure 5: Raw and smoothed VO trajectories on the X–Z plane. The dashed grey curve shows

the original up-to-scale VO path with frame-to-frame jitter, while the solid orange curve shows the trajectory after sliding-window smoothing, which suppresses local noise while preserving the global route shape.

The primary programme outputs are shown in Appendix B – Main Results.

VI. Conclusion & Discussion

A. Conclusion

The experiment implemented a complete monocular visual odometry pipeline on a hand-held smartphone video recorded around the main building of Harbin Engineering University. Using ORB feature tracking and essential-matrix pose recovery with approximate intrinsics, the system reconstructed an up-to-scale 3-D trajectory with 745 keyframes. The smoothed path clearly reflects the long forward segment, the main left turn and the final inclined straight, while the straightness index $S \approx 0.18$confirms that the motion is strongly curved rather than near-linear. The per-step image-motion profile shows a sharp peak during the turning phase and smaller displacements during the slower, straighter sections, indicating that the estimated motion is at least qualitatively consistent with the actual walking behaviour.

B. Limitations and Future Work

Several limitations remain. First, the trajectory is only recovered up to an arbitrary global scale and uses FOV-based intrinsics rather than full calibration, so any metric interpretation of distance is unreliable. Second, drift accumulates over time because there is no loop-closure or global optimisation; this is visible in the mismatch between the start and end positions. Third, performance may degrade in low-texture or strongly dynamic regions, where ORB matching and essential-matrix estimation become unstable.

Future work could therefore: (i) perform offline camera calibration and incorporate the true intrinsics and distortion; (ii) fuse inertial measurements or wheel odometry to reduce scale ambiguity and drift; (iii) add keyframe selection and pose-graph optimisation with loop-closure constraints; and (iv) record sequences with external ground truth (e.g. motion-capture markers or high-precision GNSS/INS) to quantitatively evaluate trajectory accuracy.

References

[1] Scaramuzza D, Fraundorfer F. Visual odometry [tutorial][J]. IEEE robotics & automation magazine, 2011, 18(4): 80-92.

[2] Nistér D. An efficient solution to the five-point relative pose problem[J]. IEEE transactions on pattern analysis and machine intelligence, 2004, 26(6): 756-770.

[3] Hartley R. Multiple view geometry in computer vision[M]. Cambridge university press, 2003.

[4] Kaehler A, Bradski G. Learning OpenCV 3: computer vision in C++ with the OpenCV library[M]. " O'Reilly Media, Inc.", 2016.

Ethical, Legal and Copyright Statement

All figures, trajectories and numerical results in this report are generated by the author from a self-recorded handheld video taken in public areas around Harbin Engineering University. Unless otherwise stated, the copyright of the video, processed images and source code is held by the author (and, where applicable, Harbin Engineering University). These materials are provided for non-commercial academic study and teaching only; any commercial use is prohibited without prior written permission.

The footage unavoidably contains passers-by, vehicles and campus buildings. No face recognition, identity profiling or other attempts to identify specific individuals have been carried out. Any subsequent reuse of the data must not seek to identify people in the video, must respect their portrait and privacy rights, and must comply with the relevant laws and regulations of the jurisdiction of use.

The video, source code and all processed results used in this report have been uploaded to a GitHub repository (*https://github.com/Y-Flying9264/jei-mono-vo-lab*). Third parties may reuse them for non-commercial academic purposes provided that this report and the corresponding GitHub repository are properly cited.

Appendix A: Main Code

```
# -*- coding: utf-8 -*-
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt


# ================== CONFIG ==================

VIDEO_PATH = r"E:/Projects/zzzz_exp/new_walk.mp4"   # TODO:
change to your own path

USE_CAMERA_CALIB = False
CAM_K_PATH = "K.npy"
CAM_DIST_PATH = "dist.npy"

RESIZE_SCALE = 0.5        # 1080p -> 540p
MAX_FEATURES = 2000
MIN_MATCHES = 100
RATIO_TEST = 0.7
ASSUMED_FOV_DEG = 60.0

STEP_SCALE = 1.0
FRAME_STEP = 2

SHOW_FRAME = True

SAVE_TRAJ_FIG = True
TRAJ_FIG_NAME_MAIN = "trajectory.png"
TRAJ_FIG_NAME_RAW_SMOOTH = "trajectory_raw_vs_smooth.png"
TRAJ_FIG_NAME_STEP_DIST = "step_distance.png"

SMOOTHING_WINDOW = 5


# ================== UTILS ==================

def build_intrinsics(width, height, fov_deg=60.0):
    """Build pinhole intrinsics K from image size and assumed
horizontal FOV."""
    fov_rad = np.deg2rad(fov_deg)
```

```
    fx = width / (2.0 * np.tan(fov_rad / 2.0))
    fy = fx
    cx = width / 2.0
    cy = height / 2.0

    K = np.array([[fx, 0, cx],
                  [0, fy, cy],
                  [0,  0,  1]], dtype=np.float64)
    return K


def extract_orb_features(gray, orb):
    """Extract ORB keypoints and descriptors from a grayscale
frame."""
    keypoints, descriptors = orb.detectAndCompute(gray, None)
    return keypoints, descriptors


def match_features(des1, des2, kp1, kp2, ratio=0.7):
    """Match ORB descriptors with BFMatcher + KNN + Lowe
ratio test."""
    if des1 is None or des2 is None:
        return None, None

    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=False)
    knn_matches = bf.knnMatch(des1, des2, k=2)

    pts1 = []
    pts2 = []

    for m, n in knn_matches:
        if m.distance < ratio * n.distance:
            pts1.append(kp1[m.queryIdx].pt)
            pts2.append(kp2[m.trainIdx].pt)

    if len(pts1) < 8:
        return None, None

    pts1 = np.array(pts1, dtype=np.float32)
    pts2 = np.array(pts2, dtype=np.float32)

    return pts1, pts2
```

```python
def estimate_pose_from_essential(pts1, pts2, K):
    """Estimate essential matrix with RANSAC and recover
relative pose (R, t)."""
    E, mask = cv2.findEssentialMat(
        pts1, pts2,
        cameraMatrix=K,
        method=cv2.RANSAC,
        prob=0.999,
        threshold=1.0
    )
    if E is None:
        return None, None, None

    _, R, t, mask_pose = cv2.recoverPose(E, pts1, pts2, K)
    return R, t, mask_pose


def smooth_1d(x, window=5):
    """1-D moving average smoothing."""
    if window <= 1:
        return x
    kernel = np.ones(window) / window
    return np.convolve(x, kernel, mode='same')


# ================== MAIN VO PIPELINE ==================

def run_visual_odometry(video_path):
    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        print(f"[ERROR] Cannot open video: {video_path}")
        return

    ret, frame0 = cap.read()
    if not ret:
        print("[ERROR] Video is empty or failed to read the
first frame.")
        return

    original_height, original_width = frame0.shape[:2]

    proc_width = int(original_width * RESIZE_SCALE)
    proc_height = int(original_height * RESIZE_SCALE)
```

```
    frame0_resized = cv2.resize(frame0, (proc_width,
proc_height))

    if USE_CAMERA_CALIB:
        if not (os.path.exists(CAM_K_PATH) and
os.path.exists(CAM_DIST_PATH)):
            print("[WARN] K.npy or dist.npy not found, falling
back to approximate FOV intrinsics.")
            use_calib = False
        else:
            use_calib = True
    else:
        use_calib = False

    dist = None
    if use_calib:
        K = np.load(CAM_K_PATH)
        dist = np.load(CAM_DIST_PATH)
        frame0_proc = cv2.undistort(frame0_resized, K, dist)
        print("[INFO] Using calibrated intrinsics and
distortion.")
    else:
        K = build_intrinsics(proc_width, proc_height,
fov_deg=ASSUMED_FOV_DEG)
        frame0_proc = frame0_resized
        print("[INFO] Using approximate intrinsics from
assumed FOV.")

    print("[INFO] Camera intrinsic matrix K:")
    print(K)

    gray0 = cv2.cvtColor(frame0_proc, cv2.COLOR_BGR2GRAY)

    orb = cv2.ORB_create(nfeatures=MAX_FEATURES)
    kp_prev, des_prev = extract_orb_features(gray0, orb)

    R_global = np.eye(3, dtype=np.float64)
    pos = np.zeros((3, 1), dtype=np.float64)

    trajectory = [pos.copy()]
    step_dists = []
    frame_idx = 0

    if SHOW_FRAME:
```

```
        cv2.namedWindow("Frame", cv2.WINDOW_NORMAL)

    print("[INFO] Start processing video frames...")

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        frame_idx += 1
        if frame_idx % FRAME_STEP != 0:
            continue

        frame_resized = cv2.resize(frame, (proc_width,
proc_height))

        if use_calib:
            frame_proc = cv2.undistort(frame_resized, K, dist)
        else:
            frame_proc = frame_resized

        gray = cv2.cvtColor(frame_proc, cv2.COLOR_BGR2GRAY)

        kp, des = extract_orb_features(gray, orb)
        if des is None or des_prev is None or len(kp) < 10 or
len(kp_prev) < 10:
            kp_prev, des_prev = kp, des
            continue

        pts_prev, pts_curr = match_features(des_prev, des,
kp_prev, kp, ratio=RATIO_TEST)
        if pts_prev is None or pts_curr is None or
len(pts_prev) < MIN_MATCHES:
            kp_prev, des_prev = kp, des
            continue

        pixel_motion = np.linalg.norm(pts_curr - pts_prev,
axis=1)
        step_pixel_motion = float(np.median(pixel_motion))

        R_rel, t_rel, mask_pose =
estimate_pose_from_essential(pts_prev, pts_curr, K)
        if R_rel is None or t_rel is None:
            kp_prev, des_prev = kp, des
```

```
            continue

        step_norm = np.linalg.norm(t_rel)
        if step_norm < 1e-6:
            kp_prev, des_prev = kp, des
            continue

        t_step = (STEP_SCALE / step_norm) * t_rel

        max_step = 2.0 * STEP_SCALE
        if np.linalg.norm(t_step) > max_step:
            t_step = t_step * (max_step /
np.linalg.norm(t_step))

        pos = pos + R_global @ t_step
        R_global = R_rel @ R_global

        trajectory.append(pos.copy())
        step_dists.append(step_pixel_motion)

        if SHOW_FRAME:
            vis_frame = frame_proc.copy()
            cv2.drawKeypoints(frame_proc, kp, vis_frame,
color=(0, 255, 0))
            cv2.imshow("Frame", vis_frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        kp_prev, des_prev = kp, des

    cap.release()
    if SHOW_FRAME:
        cv2.destroyAllWindows()

    if len(trajectory) < 2:
        print("[WARN] Too few valid trajectory points, cannot
plot.")
        return

    traj_array = np.hstack(trajectory).T
    xs = traj_array[:, 0]
    ys = traj_array[:, 1]
    zs = traj_array[:, 2]
```

```
    xs_smooth = smooth_1d(xs, window=SMOOTHING_WINDOW)
    zs_smooth = smooth_1d(zs, window=SMOOTHING_WINDOW)

    root_dir = os.path.dirname(os.path.abspath(__file__))

    # Main trajectory (smoothed, X-Z plane)
    plt.figure(figsize=(6, 6))
    plt.plot(xs_smooth, zs_smooth, marker='.', linewidth=1,
label="Trajectory")

    plt.scatter(xs_smooth[-1], zs_smooth[-1], c='g',
marker='o', s=60, label='Start')
    plt.scatter(xs_smooth[0],  zs_smooth[0],  c='r',
marker='x', s=80, label='End')

    if len(xs_smooth) >= 2:
        plt.annotate(
            '',
            xy=(xs_smooth[-1], zs_smooth[-1]),
            xytext=(xs_smooth[-2], zs_smooth[-2]),
            arrowprops=dict(arrowstyle='->', lw=1.5,
color='blue')
        )

    plt.annotate('Start',
                 xy=(xs_smooth[-1], zs_smooth[-1]),
                 xytext=(5, 5),
                 textcoords='offset points',
                 fontsize=8)
    plt.annotate('End',
                 xy=(xs_smooth[0], zs_smooth[0]),
                 xytext=(5, -10),
                 textcoords='offset points',
                 fontsize=8)

    plt.xlabel("X (arbitrary units)")
    plt.ylabel("Z (arbitrary units)")
    plt.title("Monocular Visual Odometry Trajectory (up-to-
scale)")
    plt.axis('equal')
    plt.grid(True)
    plt.legend(loc='best')
    plt.tight_layout()
```

```
    if SAVE_TRAJ_FIG:
        fig_path_main = os.path.join(root_dir,
TRAJ_FIG_NAME_MAIN)
        plt.savefig(fig_path_main, dpi=300)
        print(f"[INFO] Main trajectory figure saved to:
{fig_path_main}")

    # Raw vs smoothed trajectory
    plt.figure(figsize=(6, 6))
    plt.plot(xs, zs, color='0.6', linestyle='--',
linewidth=1.5, label='Raw Trajectory')
    plt.plot(xs_smooth, zs_smooth, color='C1', linewidth=2.0,
label='Smoothed Trajectory')
    plt.xlabel("X (arbitrary units)")
    plt.ylabel("Z (arbitrary units)")
    plt.title("Raw vs Smoothed Trajectory")
    plt.axis('equal')
    plt.grid(True)
    plt.legend(loc='best')
    plt.tight_layout()

    if SAVE_TRAJ_FIG:
        fig_path_raw_smooth = os.path.join(root_dir,
TRAJ_FIG_NAME_RAW_SMOOTH)
        plt.savefig(fig_path_raw_smooth, dpi=300)
        print(f"[INFO] Raw vs smoothed trajectory figure
saved to: {fig_path_raw_smooth}")

    # Per-step image motion
    step_dists = np.array(step_dists)
    plt.figure(figsize=(7, 4))
    plt.plot(step_dists, marker='.', linewidth=1)
    plt.xlabel("Step index (between keyframes)")
    plt.ylabel("Median pixel displacement")
    plt.title("Per-step Image Motion Magnitude")
    plt.grid(True)
    plt.tight_layout()

    if SAVE_TRAJ_FIG:
        fig_path_step = os.path.join(root_dir,
TRAJ_FIG_NAME_STEP_DIST)
        plt.savefig(fig_path_step, dpi=300)
        print(f"[INFO] Per-step image motion figure saved to:
{fig_path_step}")
```

```
    plt.show()

    # Numerical stats
    total_motion = float(step_dists.sum())
    final_disp = float(np.linalg.norm(traj_array[-1] -
traj_array[0]))
    straightness = float(final_disp / (total_motion + 1e-8))
    avg_step = float(step_dists.mean()) if len(step_dists) >
0 else 0.0
    std_step = float(step_dists.std()) if len(step_dists) > 0
else 0.0

    print("\n========== VO Statistics (arbitrary scale units)
==========")
    print(f"Number of keyframes (trajectory points):
{len(trajectory)}")
    print(f"Total image motion (sum of steps):
{total_motion:.4f}")
    print(f"Displacement from start to end:
{final_disp:.4f}")
    print(f"Straightness index:
{straightness:.4f}")
    print(f"Mean per-step image motion:
{avg_step:.4f}")
    print(f"Std of per-step image motion:
{std_step:.4f}")

print("=================================================
=====\n")

    np.savetxt("trajectory.txt", traj_array, fmt="%.6f")
    print("[INFO] Trajectory data saved to trajectory.txt")

    stats_path = os.path.join(root_dir, "vo_stats.txt")
    with open(stats_path, "w", encoding="utf-8") as f:
        f.write("Monocular Visual Odometry Statistics (up-to-
scale)\n")
        f.write(f"Number of keyframes: {len(trajectory)}\n")
        f.write(f"Total image motion (sum of steps):
{total_motion:.6f}\n")
        f.write(f"Displacement from start to end:
{final_disp:.6f}\n")
        f.write(f"Straightness index: {straightness:.6f}\n")
```

```
        f.write(f"Average per-step image motion:
{avg_step:.6f}\n")
        f.write(f"Std of per-step image motion:
{std_step:.6f}\n")

    print(f"[INFO] VO statistics saved to: {stats_path}")
    print("[INFO] Processing finished.")


if __name__ == "__main__":
    run_visual_odometry(VIDEO_PATH)
```

Appendix B: Main Results.

```
# For ease of open-sourcing, all original Chinese content in
the final code has been translated into equivalent English.


E:\ProgramData\anaconda3\python.exe
E:\Projects\zzzz_exp\mono.py
[信息] 使用假 FOV 估计内参 K。
[信息] 相机内参 K：
[[831.38438763   0.         480.        ]
 [  0.         831.38438763 270.        ]
 [  0.           0.           1.        ]]
[信息] 开始处理视频帧...
[信息] 轨迹主图已保存到: E:\Projects\zzzz_exp\trajectory.png
[信息] 原始 vs 平滑 轨迹图已保存到:
E:\Projects\zzzz_exp\trajectory_raw_vs_smooth.png
[信息] 图像步长曲线图已保存到:
E:\Projects\zzzz_exp\step_distance.png

========== VO 统计结果（任意尺度单位） ==========
关键帧数（轨迹点数）: 745
总图像运动量（步距之和）: 1837.5590
起点到终点的位移:         322.5732
轨迹直线性指标:           0.1755
单步图像运动量平均值:     2.4698
单步图像运动量标准差:     5.1455
==============================================

[信息] 轨迹数据已保存到 trajectory.txt
[信息] VO 统计结果已保存到: E:\Projects\zzzz_exp\vo_stats.txt
[信息] 处理完成。

进程已结束，退出代码为 0
```