State Estimation Lab

Lab 2

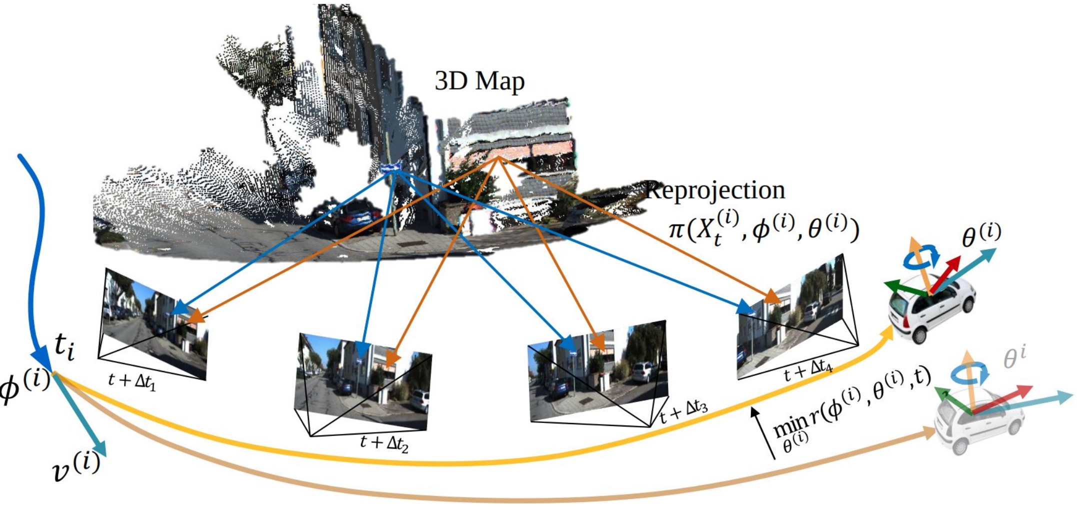
Overview

- **Visual Odometry Introduction**
- Lab
  - 2.1 Monocular Visual Odometry
  - 2.2 Monocular Visual Odometry by Control Input

What is Visual Odometry (VO) ?

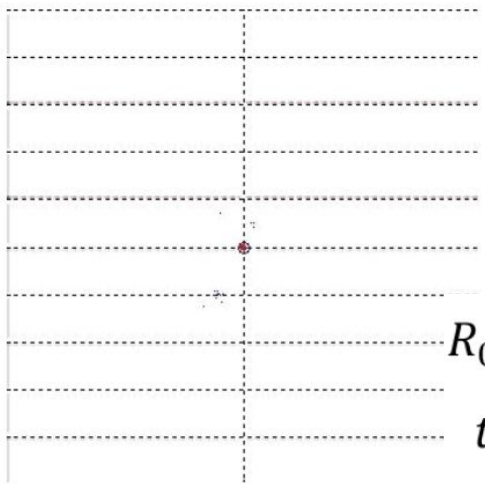
It estimates the movement trajectory of a robot using a sequence of camera images.

Input



A sequence of consecutive images.

Output



$R_0, R_1, \dots, R_i$

$t_0, t_1, \dots, t_i$

The robot (or camera) trajectory and its position/orientation at each time step.

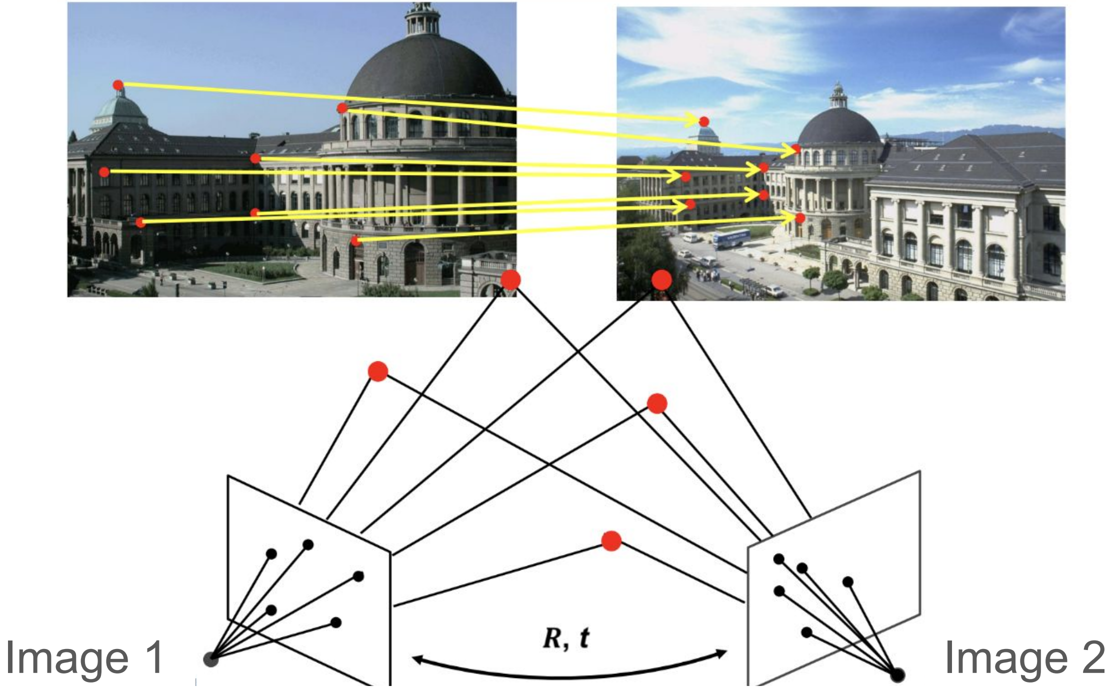
Wang, Xin, et al. "Continuous-time stereo visual odometry based on dynamics model." Asian Conference on Computer Vision. Cham: Springer International Publishing, 2018.

Assumption

- ➢ Sufficient illumination in the environment
- ➢ Dominance of static scene over moving objects
- ➢ Enough texture to allow apparent motion to be extracted
- ➢ Sufficient scene overlap between consecutive frames

Working Principle

1. Detect and match image feature points
2. Estimate the camera's rotation and translation (R, t)
3. Accumulate motions to get the full trajectory



How a Drone Sees Its Path

1. Initialize Subscribers, Feature Detector, Matcher, and Plot

- Subscribe to camera images (/image_raw) and velocity commands (/cmd_vel)(lab 2-2)
- Set up ORB feature detector and FLANN matcher
- Initialize trajectory plot and path publisher

```
self.orb = cv2.ORB_create(3000)
self.flann = cv2.FlannBasedMatcher(indexParams=..., searchParams=...)
self.create_subscription(Image, '/image_raw', self.image_callback, qos)
self.create_subscription(Twist, '/cmd_vel', self.cmd_callback, 10)
```

2. Detect ORB Feature Points in Consecutive Frames

Extract distinctive feature points (keypoints) and descriptors in the last and current images

```
kp1, des1 = self.orb.detectAndCompute(self.last_gray, None)
kp2, des2 = self.orb.detectAndCompute(gray, None)
```

3. Match Features Using FLANN and Ratio Test

- Find matches between descriptors from consecutive images
- Apply ratio test to keep only reliable matches

```
matches = self.flann.knnMatch(des1, des2, k=2)
good = [m1 for m1, m2 in matches if m1.distance < 0.8 * m2.distance]
```

4. Estimate Motion Between Frames (Essential Matrix, RecoverPose)

- Use good matches to estimate camera rotation (R) and translation direction (t)

```
E, _ = cv2.findEssentialMat(pts1, pts2, self.K, cv2.RANSAC, 0.999, 1.0)
_, R, t, _ = cv2.recoverPose(E, pts1, pts2, self.K)
```

5. Recover the Scale Using Velocity and Timestamps (Lab 2-2)

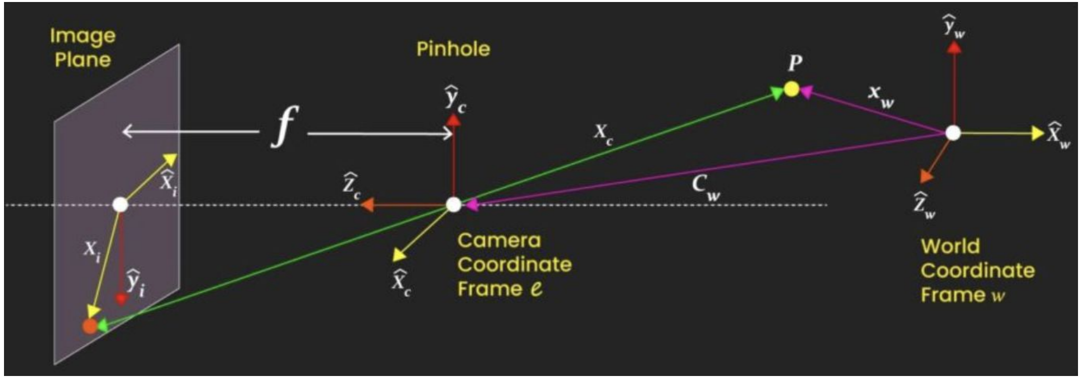
- Compute the real-world scale using /cmd_vel and the time difference
- Apply this scale to the translation vector

```
dt = img_stamp - self.prev_img_stamp
d_cmd = self.current_vel * dt
scale = d_cmd / np.linalg.norm(raw_t) if norm_t > 1e-6 else 1.0
t_scaled = raw_t * scale
```

6. Accumulate the Pose in Global Coordinates

- Update the current global pose by chaining relative transformations

```
T = np.eye(4)
T[:3,:3] = R
T[:3,3] = t_scaled
self.cur_pose = self.cur_pose @
np.linalg.inv(T)
```

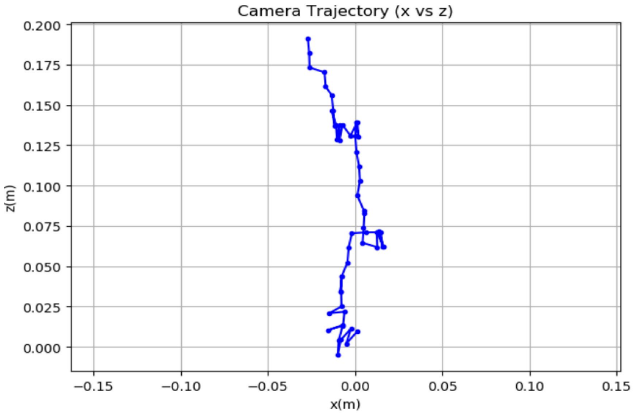


https://learnopencv.com/monocular-slam-in-python/

Overview

- Visual Odometry Introduction
- **Lab**
  - 2.1 Monocular Visual Odometry
  - 2.2 Monocular Visual Odometry by Control Input

Lab 2.1 - Monocular Visual Odometry



Detail: https://hackmd.io/@gantingyu/S1LKg6hkll

1. Install dependencies、Download **tello_vo** from Github
2. Fill in the blanks in ”~/tello_ros_ws/src/tello_vo/tello_vo/vo_node.py”
3. Build the workspace
4. Turn on the drone, connect to its wifi.
5. Do the following commands:

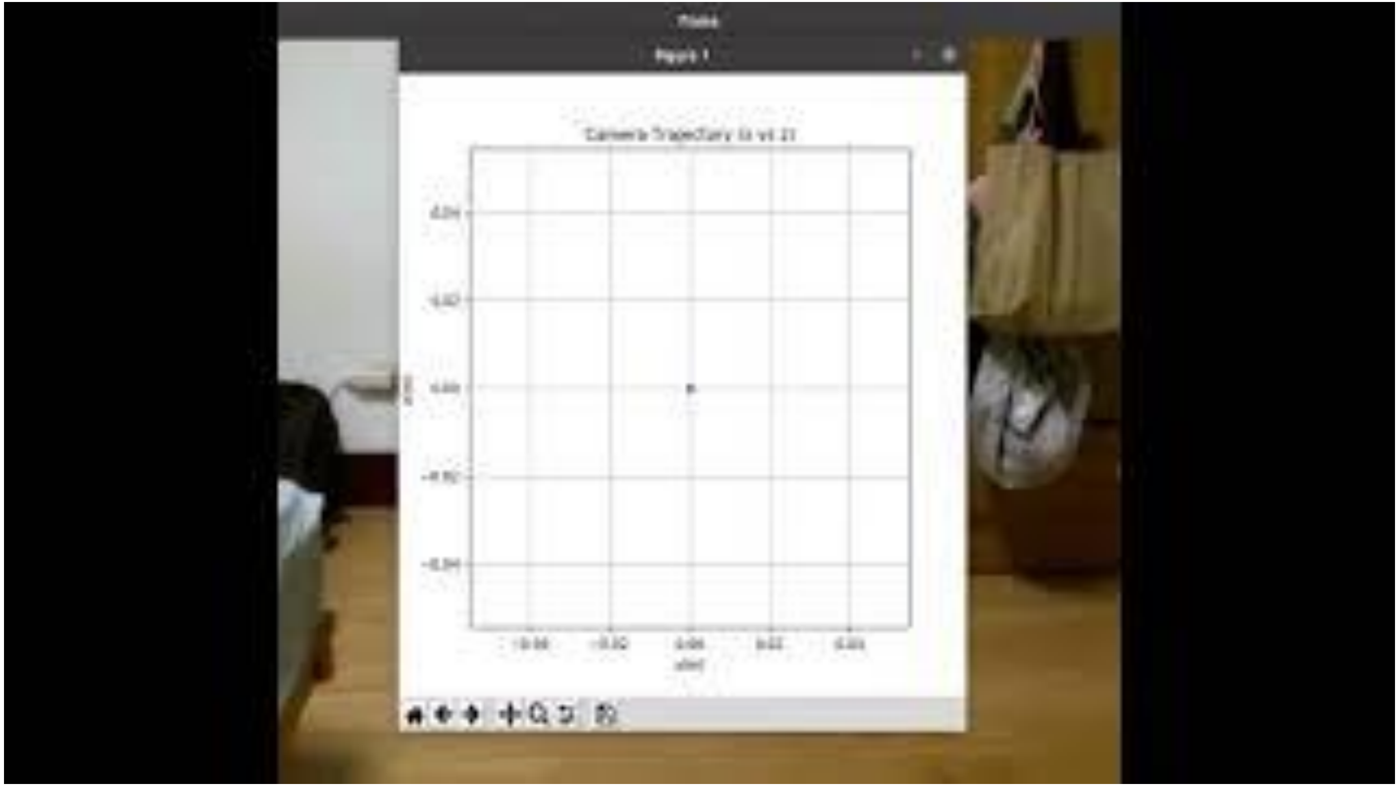
One Terminal:

```
cd ~/tello_ros_ws
source install/setup.bash
ros2 launch tello_driver teleop_launch.py
```

The Other Terminal:

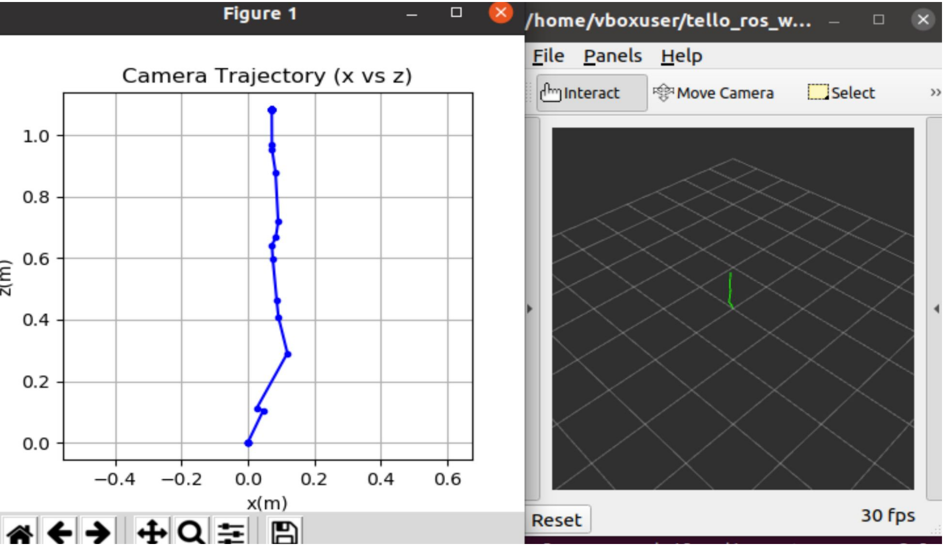
```
cd ~/tello_ros_ws
source install/setup.bash
ros2 run tello_vo vo_node
```

Lab 2.2 - Monocular Visual Odometry by Control Input

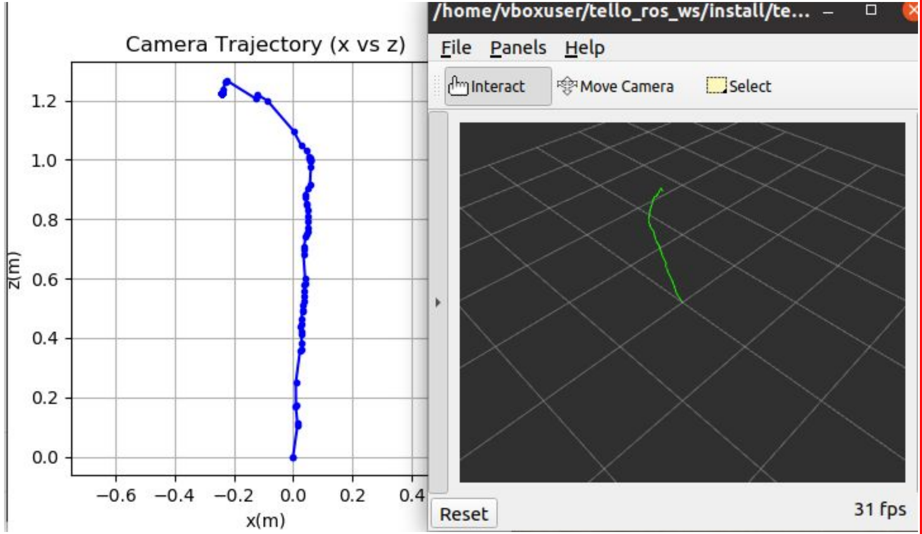


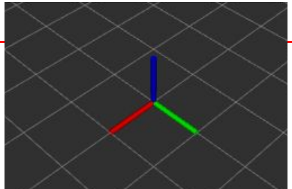
Goal: Look at the trajectory plot for moving forward and moving left.

Move Forward



Move Forward + Move Left





Red: x-axis
Green: y-axis
Blue: z-axis

Detail: https://hackmd.io/@gantingyu/S1LKg6hkll

1. Download **control.py** from E3, adjust the control instructions you want, and put into ~/tello_ros_ws/src.
2. Fill in the blanks in ”~/tello_ros_ws/src/tello_vo/tello_vo/vo_node_control.py”
3. Build the workspace
4. Turn on the drone, connect to its wifi.
5. Do the following commands:

One Terminal:

```
cd ~/tello_ros_ws
source install/setup.bash
ros2 launch tello_driver teleop_launch.py
```

Another Terminal:

```
cd ~/tello_ros_ws
source install/setup.bash
ros2 launch tello_vo vo_with_rviz.launch.py
```

The Other Terminal:

```
cd ~/tello_ros_ws
source install/setup.bash
python3 src/control.py
```