Simple Visual Odometry

EECE 7150 - Hanumant Singh

# 1. Introduction

## 1.1. Goal of the project

The goal of this mini-project is to implement a simple, monocular, visual odometry (VO) pipeline

with the most essential features:

* Initialization of 3D landmarks
* Keypoint tracking between two frames
* Pose estimation using established 2D ↔3D correspondences
* Triangulation of new landmarks

## 1.2. Datasets

We will use the a subset of the KITTI dataset, dataset 00 which can be downloaded from http://rpg.ifi.uzh.ch/docs/teaching/2016/kitti00.zip

## 1.3. Grading

Good luck!!!

## 1.4. Helpful libraries and functions

You can use OpenCV or any other library to help with implementation of subroutines as long as you don’t use a pre-written Visual Odometry or Visual slam package.

## 1.5. Submission

For your submissions, along with the code and instructions of how to run the code, include a video showing your solution in action. For Ubuntu, we recommend the screen capture program kazam.

# 2. Overview of the proposed pipeline

We first give a global overview of the proposed pipeline and the different components involved. We

will then go into more details separately for each component.

## 2.1. Notation

We denote the set of all N frames in a dataset by: {Ii ≡ I(ti)}, i ∈ [1, N ] . We denote the pose of the camera at time ti by TiWC .

## 2.2. Overview

The proposed pipeline is composed of two main components:

* An initialization module that extracts an initial set of 2D ↔ 3D correspondences from the first frames of the sequence and bootstraps the initial camera poses and landmarks.
* A continuous VO module that processes each frame Ii, estimates the current pose of the camera TiWC using the existing set of landmarks, and regularly triangulates new landmarks.

# 3. Initialization

We can use two-view geometry to estimate the relative pose between two frames, and triangulate a point cloud of landmarks as follows:

1. Manually select two frames Ii0 and Ii1 at the beginning of the dataset. (To make sure there is enough parallax in the features don’t use consecutive images. For Kitti dataset frame 1 and frame 3 gives reasonable results.)
2. Establish keypoint correspondences between these two frames using a Feature Detector
3. Estimate the relative pose between the frames:
   1. Since we are using a calibrated camera we can estimate the Essential matrix (OpenCV’s findEssentialMat) which uses Nister’s 5 point algorithm to compute E from atleast 5 sets of correspondences.
   2. Then decompose E to get the relative Pose. OpenCV’s recoverPose should help with this.
4. Given 2 poses, triangulate a point cloud of 3D landmarks. OpenCV’s triangulatePoints can help with this.
5. Initialize the continuous VO pipeline with the inlier keypoints and their associated landmarks.

# 4. Continuous operation

The continuous VO pipeline is the core component of the proposed VO implementation. It’s responsibilities are:

1. Associate keypoints in the current frame to previously triangulated landmarks.
2. Based on this, estimate the current camera pose.
3. Triangulate new landmarks using keypoints not associated to previously triangulated landmarks.



*Figure 1: Recommended data flow / function design for continuous operation. This allows continuous operation without the need to keep a global data structure with the full history of previous observations*

We recommend to implement this in a Markov way using the data flow shown in Figure 1. Formally, we define Si the state of the current frame, whose contents are specified further below. Then, we can define a function for processing incoming frames, updating Si and returning the pose TiWC as follows:

The key idea in this design is that the function inputs solely depend on the output of the previous

function call (and the new frame to process), i.e. it has the Markov property. That means we don't

need to build a data structure to maintain the history of the past frames, all that is needed is

contained in the state Si.

## 4.1. Associate keypoints to existing landmarks

This step can be summarized as follows:

1. Given :

* current image Ii
* previous images Ii-1
* keypoints in the previous image ( Pi={pik},k∈ [1, K], K being the number of keypoints )
* 3D landmarks denoted by Xi ={x(p)∀p ∈ Pi} ie. x(pik) is the 3D landmark associated to pik.

1. Create 2D feature ↔3D Landmark correspondences in current image by:

* Detect in features Pi in current image
* Match them to previously detected and 3D registered features Pi-1
* Set landmark x(pia) = x(pi-1b) if the matching is valid and a RANSAC inlier

## 4.2. Estimate Current Pose

From the 2D feature ↔3D Landmark correspondences estimate the current pose Tiwcof the camera using the PNP (Perspective N-point) algorithm eg. OpenCV’s solvePnPRansac. This will give you the pose along with RANSAC inliers.

## 4.3. Triangulate new landmarks

As landmarks go out of view of the camera, we need to continuously add new landmarks. Given matched features in current image and in the previous image which do not have associated landmarks, triangulate the associated landmark as done during initialization.

This is the naive way to add landmarks. An improvement would to be being selective about adding landmarks based on their parallax, distance, orientation etc.

The most comprehensive way to add candidate landmarks would be to keep a track of:

* The initial frame number when a candidate keypoint was observed
* Track length of candidate keypoint from initial observation to current frame
* Camera pose history for all the frames

And then once we find that the track length of a keypoint exceeds a certain length and/or meets a certain bearing angle threshold, triangulate it using the widest available baseline (current observation and initial observation) and the associate camera poses.

In order to achieve the above while keeping the Markov property in mind, we get the state to be propagated as:   
Si = (Pi,Xi,Ci,Ti) where Ci represents candidate keypoints, their first observation and track length and Ti represents history of camera poses.

# 5. GENERAL HINTS

In general, proceed step by step and verify your intermediate results visually.

For example, make sure that matching, localization and landmark propagation work properly before triangulating new landmarks.