Laboratory 6:
Rigid and Non-Rigid Structure from Motion

Computer Vision, Spring 2022

Overview

The goal of this session is to practice on rigid and non-rigid structure from motion. To this end, we will work on:

- Rigid structure from motion by factorization.
- Non-Rigid structure from motion by non-linear optimization and assuming a low-rank shape model.
- Non-Rigid structure from motion by factorization and assuming a low-rank trajectory model.

First of all, we must download the codes and supplementary materials from the link *Laboratory 6 Materials* on Aula Global of *Computer Vision*.

FAQs:

- **What do I need to send?** A report with your work, including missing codes, an explanation where you discuss your work, as well as your results and conclusions. Extra codes and analysis are also welcome.
- **How?** A single report is needed. Please, do not send partial reports. Once your work is finished, you should upload it to the *Laboratory 6 Submission* on the Aula Global, in a zip file. The submitted zip file has to contain your report and your source code (and all files needed to run it, such as images, other dependencies, etc). **Be clear and concise.** Your final score is not directly proportional to the number of pages in your report. The work in this session is individual.
- **When?** Anytime until 24 June 2022 (12:29 UTC+2). After that, nothing will be received.

  Within 24 hours prior to the deadline, no comments will be answered.

References

- Novel sensors and Reconstruction slides of the course on *Computer Vision*.
- Time-varying shapes slides of the course on *Computer Vision*.

1 Introduction

The main aim of this practice is to get familiar with the rigid and non-rigid structure from motion problems. This practice consists of three parts:

- In part #1 you will read and understand some provided functions that compute shape and motion from a measurement matrix by rigid factorization. You will implement a method to compute the SVD factorization.
- In part #2 you will read and understand some provided functions that compute non-rigid shape and motion from a measurement matrix by non-linear optimization. To improve efficiency and accuracy, you will implement a method to code a Jacobian pattern.
- In part #3 you will read and understand some provided functions that compute non-rigid shape and motion from a measurement matrix by non-rigid factorization in a trajectory space. You will implement a method to compute the SVD factorization.

1.1 Zero Part

In all methods we are going to use in this session, the goal is to jointly infer the 3D shape and the camera motion from a collection of 2D tracking data from a monocular video or a collection of pictures. To understand that, a zero part you need is to execute the Matlab script **visualize_2Dtracking.m**, where you will observe a monocular video (the frames are included in the "data-Dinosaur" folder) together with a set of 2D annotations. We will use the set of 2D annotations to compose our measurement matrix, i.e., our input data. Afterwards, we assume the 2D tracking data are ready.

1.2 First Part

In this task, the goal is to practice on structure from motion by assuming the shape is rigid. Particularly, we consider an orthographic camera, and will obtain a solution by rigid factorization. Go into the folder "sfm_shape_space" and look into the Matlab function **main_rigid.m**.

- Tasks [M - Mandatory, O - Optional]:

apply SVD enforcing rank = 3 ,bc object is rigid and photographic

M You need to understand what this function does in terms of functionality, input and output. Then you need to implement the missing instructions in the file **rigidfactorization_ortho.m**. The location of the code that needs to be implemented is indicated by the text "MISSING CODE HERE".

M Try to run your code in the provided example (a synthetic sequence with an ogre is provided to you), and report the 3D reconstruction error. You can see some visualizations about the 2D data, and the corresponding joint estimation. Use the option "rotate3D" to see the difference.

O You can collect your own rigid dataset by considering a rigid object. To obtain 2D tracking data, synthetically apply an orthographic projection and try to compute the 3D reconstruction from 2D data.

1.3 Second Part

Now, the goal is to practice on structure from motion by assuming the shape is non rigid. Again, we will consider an orthographic camera, but it will obtain a solution by non-linear optimization. Go into the folder "sfm_shape_space" and look into the Matlab function **main_nonrigid.m**. There, you can find some relevant functions:

- **nrsfm.m**: The main script to infer time-varying shape and motion from a measurement matrix.
- **JacobianPattern.m**: This file implements a function to encode the binary Jacobian pattern.
- **cost_RXT.m**: This file implements the cost function as a combination of the data term and the priors whether they are used.
- **model2vector.m**: This file implements a function that takes as input the matrices for all parameters (camera rotations, camera translations, shape basis, and weight coefficients) and transform this information into a vector. This function represents the inverse step of **vector2model.m**.
- **vector2model.m**: This file implements a function that takes as input a vector with all variables we have to estimate, and transform it into a set of model matrices. This function represents the inverse step of **model2vector.m**.

- Tasks [M - Mandatory, O - Optional]:

  M You need to understand what all the functions do in terms of functionality, input and output. It is worth noting that you need the function **rigidfactorization_ortho.m** you previously implemented to initialize the optimization.

  M You need to implement the missing instructions in the file **JacobianPattern.m**. The location of the code that needs to be implemented is indicated by the text "MISSING CODE HERE". Particularly, between others, you need to add 2 entries for every translation vector and 4 more for rotations (quaternions are used). The order of the variables can be followed in the code. Additionally, the functions **model2vector.m** and **vector2model.m** can help you to understand the order you should use. Once you finish the Jacobian pattern for the data term, you must complete the temporal smoothness priors on deformation and motion. Visualize your binary pattern with the function **spy(J)**.

  M Try to run your code in the provided examples (press a R for running a real experiment on **face_real.mat**, or a S for a synthetic one on **face_mocap.mat**). For both cases, report the 2D re-projection error before/after applying the non-linear optimization, and the corresponding binary pattern you obtain in both experiments by using **spy(J)**. In addition, for the synthetic case, you should report the 3D reconstruction error before/after applying the non-linear optimization algorithm.

  M Try to run your code in the synthetic case with different K values (for example, you may consider $K = \{2, 3, 4\}$), while imposing or not the use of temporal smoothness priors. To active them, set "1" on the variables *test_options.priors.camera_prior* and *test_options.priors.coeff_prior*, respectively. Again, report the corresponding binary patterns after activating the priors.

  O You can collect your own non-rigid dataset by considering a deformable object. To obtain 2D tracking data, synthetically apply an orthographic projection and try to compute the 3D reconstruction solely from 2D data.

solve using toy example
k=2
frames = 6
points = 3
Full visibility
priors = 1

num_eq = n_frames*n_points*2 = 76

n_frames-1
n_frames-1
=
n_priors*(n_frames-1)

num_unknowns =
4*n_frames+
2*n_frames+
k*n_frames+
3*n_points*k = 66

consider order in the equations
and order in the nodes

in class: - in lab
x_i - x_1
y_1 - b_1
x_2 - x_2
y_2 - b_2

1.4 Third Part

Now, you are going to practice on non-rigid structure from motion by assuming a trajectory linear subspace. Again, we will consider an orthographic camera and will use a factorization approach. Go into the folder "sfm_trajectory_space" and look into the Matlab function **main_trajectory.m**. There, you can find also some relevant functions:

- **metricUpgrade.m**: This file implements a function to solve the metric upgrade step in a trajectory subspace.
- **recoverR.m**: This file implements a function to obtain a matrix of size [2F x 3F] from the rotations per frame in a [2F x 3] matrix.
- **DCT_basis.m**: This file implements a function to generate a DCT trajectory basis.
- Tasks [M - Mandatory, O - Optional]:

  M You need to understand what all the functions do in terms of functionality, input and output.

  M You need to implement the missing instructions in the file **nrsfm_trajectory.m**. The location of the code that needs to be implemented is indicated by the text "MISSING CODE HERE".

  M Try to run your code in the provided examples (*drink*, *yoga*, *stretch*, *pickup*, *dance* and *dinosaur* sequences) by pressing the corresponding letter. For all sequences, a 3D ground truth is available to establish a comparison. For every sequence, you need to report the 3D error as a function of the rank K. Can you find the optimal value for every sequence? (the values $K = \{2, \ldots, 14\}$ can be considered). Additionally, for the previous sequences excepting *dance* and *dinosaur*, you also have the rotation ground truth, and this error should be also reported together with the 3D error for different values of K.

  M For the *dinosaur* sequence, try to represent the 2D points you obtain after applying your estimation along with the 2D input data. Basically, you have to include your solution for the optimal K value in the Matlab script **visualize_2Dtracking.m**.

1.5 Fourth Part

To finish, you should consider the learning-based work *Neural Dense Non-Rigid Structure from Motion with latent space constraints*. Basically, you should briefly explain the main loss the authors are considering for training, and how the neural model is exploited to encode the deformation.