MVG Assignment 5
Optimization

Instructions

- June 17 2021, 23:59
- Submission link: `https://www.dropbox.com/request/P27wtbUD0TGzB1C7TKGP`
- You should submit a zip file containing: A pdf file with all complete solutions and the code.
- The report should be written individually, however you are encouraged to work together.
- We highly recommend typing your solutions in Latex.
- Notice that in the end of this exercise you are given a list of useful matlab commands for all the computer exercises.

(a) (b) (c)

Figure 1: Drinking Fountain Somewhere in Zurich

Theoretical exercises

1. For a constrained convex optimization problem, show that any local minimizer is a global minimizer.
2. Show by the epigraph properties that the function $f(x_1, x_2) = x_1 x_2$ with $\mathrm{dom} f = R_+^2$ is a quasiconcave function.

MVG with SOCP

- In this part we will solve the triangulation problem with second order cone prgramming, as in "Multiple-View Geometry Under the L_∞-Norm" by Fredrik Kahl and Richard Hartley: `https://ieeexplore.ieee.org/document/4385722`.
- This exercise will be in Python.
  You will need to use numpy, matplotlib, plotly and cvxpy: `https://www.cvxpy.org/index.html`.
  You may need to install Visual Studio in order to run cvxpy.
  The packages are most easily installed with anaconda.
- **Important:** You are given the outline of the implementation, but you are free to change or rewrite all the code including the functions' declarations.

Computer Exercise 1 - Triangulation with DLT

You are given the file "Drinking Fountain Somewhere In Zurich.npz" that contains the following variables:

- P: ndarray of shape [n_cam, 3, 4], the cameras
- X: ndarray of shape [4, n_points], the ground truth 3D points

- x: ndarray of shape [n_cam, 3, n_points], the image points
- K: ndarray of shape [n_cam, 3, 3], the calibration matrices
- $visible\_points$: boolean matrix of shape [n_cam, n_points], what cameras see what points

a.

Implement the function 'utils.reprojection_errors(P, X, x, visible_points)'. This function receives the cameras, 3D points and image points, projects X onto P and computes the distance to the real image points. Formally, the function returns the variable "errors" of shape [n_cam, n_points]:

$$errors_{ij} = \left\| x_{ij} - \frac{P_i X_j}{P_i^3 X_j} \right\|$$

If point j is not visible in camera i, $errors_{ij} = np.nan$.

b.

Compute the reprojection errors of the ground truth 3D points, and **plot** a histogram of all the errors using matplotlib or plotly. (You should plot a histogram of a 1-d array and not a matrix, and the length of the array is the number of "True" entries of $visible\_points$.)

c.

Implement the function 'utils.normalize_cam_points(P,x,N)' which normalizes both the cameras and the image points with the matrices in N.

d.

Implement the function 'triangulation.DLT_triangulation(Ps, x, visible_indices)' that takes the cameras and the image points and returns predicted 3D points using DLT.
You can use lecture 3 slide 16, and the code of assignment 2.
Loop over the number of 3D points, and for each one take the cameras that see it and the matching image points. Build the matrix M and use numpy.linalg.svd. Notice that numpy returns $[U, S, V^H]$ unlike Matlab so you need to transpose V.

e.

Triangulate the image points in two scenarios:

1. Unnormalized cameras and points

2. Cameras and points are normalized with K^{-1} (np.linalg.inv)

Compare the mean reprojection error you get when: normalizing, not normalizing, taking the ground truth points. (Make sure to compute the reprojection errors with the original unnormalized P and x.)
For the report: The histogram, three mean reprojection errors and a short explanation.

Computer Exercise 2 -Triangulation with cones

In this question we will solve triangulation with a different approach - second order cone programming.
Denote the j'th row of the i'th camera matrix P_i^j, and the projection of the 3D point to the i'th camera as $(x_i^1, x_i^2)^T$.
As seen in the lecture, the residuals of the triangulation problem are expressed by:

$$r_i(X) = \left\| \left(x_i^1 - \frac{P_i^1 X}{P_i^3 X}, x_i^2 - \frac{P_i^2 X}{P_i^3 X} \right) \right\|_2$$

Where X is in homogenous coordinates.

a.

First **show** that the constraint $r_i(X) \leq \gamma$ forms a second order cone, which is a convex set, by finding expressions for $A_i \in \mathbb{R}^{d \times 4}, c_i \in \mathbb{R}^{4 \times 1}$ s.t:

$$r_i(X) \leq \gamma \iff \|A_i X\|_2 \leq \gamma c_i^T X$$

b.

Explain why the following function is quasi-convex:

$$f(X) = \max_i r_i(X)$$

c.

Implement the function 'triangulation.max_single_point_errors(P, x, Xi)' that returns the function $f(X)$ from section **b**. Given one 3D point this function computes the maximal reprojection error over all the cameras that see it.
Fill in the function 'triangulation.get_triangulation_parameters(P, x, gamma)' that given the cameras P and the projected points x returns A and c as in section **a**.
Fill in the function 'triangulation.solve_SOCP(A,c)' that given the lists A, c uses CVXPY to solve the second order cone feasibility problem. (Use CVXPY's tutorials and examples! Note that this is a feasibility problem.)

This function should return a feasible point X_pred if it exists.
Notice that the "solve()" function of CVXPY raises an exception when it is unable to solve the problem.

d.

Implement the function 'triangulation.SOCP_triangulate(u, P, max_iter, low, high, tol)', which computes a single 3D point given an array of cameras that see it and the corresponding image points. You are given the following algorithm which is similar to bisection - the algorithm that was taught in class:

Algorithm 1: Triangulation Bisection

Input: $high, tol, max\_iter$;
Output: $best\_U$;
$low \leftarrow 0$;
$\gamma \leftarrow high$;
while $high - low > tol$ ***and*** $curr\_iter < max\_iter$ **do**
  $U \leftarrow$Solve_Optimization();
  if $solvable$ ***and*** $max\_error(U) \leq \gamma$ **then**
    $high \leftarrow max\_error(U)$;
    $best\_U \leftarrow U$
  else
    $low \leftarrow \gamma$;
    $high \leftarrow 2 \cdot high$;
  end
  $\gamma = (high + low)/2$;
end

These are only general outlines of the code.
You need to compute the variables needed for optimization, solve the feasibility problem, compute the maximal residual of the predicted 3D point over all the cameras and update high, low, best_U, γ accordingly.
You can use the regular version of bisection from the lecture if you prefer - this version is slightly different.
Remember to handle the exception of the "solve()" function by treating it as "unsolvable" and getting to the "else" part of the algorithm.

e.

Triangulate all the points using 'SOCP_triangulate' with and without normalization with K^{-1}. **Plot** a bar chart showing the mean reprojection error with and without normalization, with DLT and SOCP, and ground truth points. (5 bars).
Is this what you expected?

f.

Use the *provided* function 'utils.plot_cameras(P, K, X, title)' in order to **plot** the points with the worst mean error, and the points with the best mean error. Can you see the difference in the resolution of the points and fine details?