Geometry and model fitting [40 points]

1. Line-point duality in projective geometry.
   (a) [5 points] Using a clear 3D figure showing a camera screen and center and two points $p_1, p_2$ on the screen, illustrate the fact that the line between $p_1$ and $p_2$ on the screen can be defined by its normal vector of homogeneous coordinates $l = p_1 \times p_2$. Hint: use the properties of the cross-product of two 3D vectors.

   (b) [5 points] Using a clear 3D figure showing a camera screen and center and two lines $l_1, l_2$ on the screen, illustrate the fact that the intersection point of the lines has homogeneous coordinates $p = l_1 \times l_2$. Hint: use the properties of the cross-product of two 3D vectors.

2. Least squares and SVD. When fitting a model (e.g. line through a set of points in homogeneous coordinates), we need to solve a system of form $Aw = 0$, where $A$ contains the measurements (e.g. position of the points) and $w$ contains the parameters of the model, that are unknown (e.g. line parameters). For instance in the case of the line defined by the parameters $w = [a \ b \ c]^T$, for all points on the line we have $ax + by + c = [x \ y \ 1]w = 0$:

   $$\begin{bmatrix}
   x_1 & y_1 & 1 \\
   \vdots & \vdots & \vdots \\
   x_n & y_n & 1
   \end{bmatrix}
   \begin{bmatrix}
   a \\
   b \\
   c
   \end{bmatrix} = 0_n$$

   In practice each row of $A$ contains information from a measurement and there are many measurements, therefore $A$ is $m \times n$ with $m > n$: $A$ is “skinny”, the system is over-determined and generally it is not possible to solve $Aw = 0$ exactly. Instead we can perform least-squares estimation, i.e. minimize $\|Aw\|^2$ for $w \in \mathbb{R}^n$.

   To avoid the trivial solution $w = 0$, we add the constraint $\|w\| = 1$, i.e. $w$ is on the sphere of radius 1. The constraint has a natural interpretation: for instance, when fitting a line, the line equation is defined up to a scaling, therefore we set $\|w\| = 1$ to avoid $w = 0$. 

1
2 Panoramic image stitching [70 points]

In this section, you will write a simple program to build a panoramic picture by stitching together several pictures of a same building facade. In general, off-the-shelf softwares detect interest points that appear in two different images (for instance by matching there SIFT descriptors) and use the 2D-2D matchings to rectify the images. You will implement a slightly different algorithm that uses lines to find vanishing points, and aligns the images based on the position of those vanishing points.

Preliminaries:

- You will use the three images `img_left.jpg`, `img_center.jpg`, `img_right.jpg` in the homework toolkit.
- You will extensively use the helper function `getClick()` we provide to retrieve the position of user clicks: `getClick(winname, img)` displays image `img` in window `winname` (creating it if necessary), and waits for the user to `double-click`, then returns a `Point` with the click position.

```c++
Mat A = ...;
SVD svd(A, SVD::FULL_UV);
```

Afterwards, `svd.u` is the matrix `U`, `svd.vt` is the matrix `V^T`, and `svd.w` contains the diagonal elements of `Σ`. (Compare this to `[U S V] = svd(A);` in MATLAB.) Complete the function `minimizeAx()` in hw4.cpp that finds the optimal `x (||x|| = 1)` minimizing `||Ax||`, using the OpenCV class SVD. You many NOT use SVD: : solveZ(). We will use this technique in question 2.

(a) **[2 points]** Write the Lagrangian of the constrained minimization and show that it is convex.
(b) **[2 points]** Set the derivative with respect to `x` to zero and show that the minimizer is an eigenvector of `A^T A`.
(c) **[2 points]** Prove that `A^T A` is symmetric and positive semi-definite (psd). Hint: Recall that a real symmetric matrix `M` is psd iff `x^T M x ≥ 0` for all `x`.
(d) **[2 points]** Show that all the eigenvalues of a symmetric real matrix `M` are real. Hint: Compute `e^H M e` for a given eigenvector `e` associated to eigenvalue `λ`, where we don’t know `a priori` if `e` has real coordinates, and `e^H` is the Hermitian transpose (transpose and take complex conjugate of coordinates).
(e) **[3 points]** Show that for two distinct eigenvalues `λ_1 ≠ λ_2` of a real symmetric matrix `M`, the corresponding eigenvectors `e_1, e_2` are orthogonal. Hint: Compute `e_1^H M e_2`.
(f) **[4 points]** Using the previous questions, prove that any real symmetric matrix `M` can be factored as follows:

\[ M = QΛQ^T, \quad Q^T Q = I \] (i.e. `Q` is orthonormal)

This fundamental result of linear algebra is known as the spectral theorem, and is also sometimes formulated as `M = \sum_i λ_i e_i e_i^T` where the `λ_i, e_i` respectively denote the eigenvalues and eigenvectors of `M`.

(g) **[2 points]** We can now apply this result to `A^T A`. Prove that there is a matrix `Σ = \begin{bmatrix} σ_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & σ_n \end{bmatrix} \in \mathbb{R}^{n \times n}`

and an orthogonal matrix `V \in \mathbb{R}^{n \times n}` such that `A^T A V = VΣ^2`.

(h) **[4 points]** Let `U_{\text{mono}} = AVΣ^{-1}`. Show that `U^T U = I` and that the columns of `U` are eigenvectors of `AA^T`.

(i) **[2 points]** Show that `A_{\text{mono}} = U_{\text{mono}} Σ_{\text{mono}} V_{\text{mono}}^T` by completing the orthogonal basis in `U` (by Gram-Schmidt) and by adding `m - n` rows of zeros to \(\Sigma\). `U_{\text{mono}}` and `V_{\text{mono}}` are now two orthogonal matrices and we have `A = UΣV^T`, which is known as the SVD decomposition. The `σ_i`'s are named singular values and are the eigenvalues of `A^T A` (and `AA^T`), and `U, V` contains the corresponding eigenvectors, respectively for `AA^T` and `A^T A`.

(j) **[7 points]** In OpenCV, there is an SVD class which computes the singular value decomposition.
2.1 Image rectification  [30 points]

In this section you will simply rectify the pictures of the building facade such that the projections of parallel lines appear parallel in the image.

The program `image_rectification` (see `src/image_rectification.cpp`) provides the layout for doing image rectification. Once you complete all the programming parts of this section, you should have a fully working program which reads in `images/img_center.jpg`, rectifies it based on user input, and saves the result to `images/img_center_rectified.jpg`. You should only need to modify the file `src/hw4.cpp` by completing the functions therein, but feel free to hack the rest of the code if you want to eg improve the user interface. However, DO NOT modify any of the header files/function signatures as I will need them to stay intact for the purpose of testing your code when you submit.

1. [6 points] Complete the function `fitLine(im, n)` in `hw4.cpp` that displays the image `im`, prompts the user to click `n` aligned points, fits a line on them, and outputs the line coefficients as `Vec3d` (which is a $3 \times 1$ `Mat` of type `double`). Hint: you can use the OpenCV function `line()` to draw lines in an image if you want to draw a small cross where the user clicked.

   In practice, if there are more than two points, you need to solve a least-squares problem: leverage the `minimizeAx()` function to solve it.

2. [6 points] Complete the function `findIntersection(L)` that takes an arbitrary number of concurrent lines (for example projections of parallel lines) represented as rows of a $N \times 3$ matrix $L = [l_1, ..., l_N]^T$ and outputs a `Vec3d` of homogeneous coordinates of the intersection point.

   In practice if there are more than two lines, you need to solve a least-squares problem: you can leverage `minimizeAx()` again.

3. We are now interested in computing the projective transformation that would rectify the image of a facade: we want the projections of parallel vertical and horizontal lines to be parallel in the image. In this question, assume the following:

   - We clicked on a set of projections of horizontal lines in the image and obtained their equations and intersection using questions 1 and 2: we have the homogeneous coordinates of $v_x$ (`Vec3d`), projection of the horizontal vanishing point.
   - Similarly we clicked on projections of vertical lines and obtained $v_y$, homogeneous coordinates of the vertical vanishing point.
   - We now click on two points $x_{00}$ and $x_{11}$ that define the “area” covered by the facade: $x_{00}$ will be mapped to the bottom-left corner of the rectified image and $x_{11}$ to the upper-right one. Concretely $x_{00}$ is the image of $[0, 0, 1]$ and $x_{11}$ the image of $[1, 1, 1]$ by a projective transformation that goes from the rectified image to the input image.

   (a) [6 points] (Course question) Show that if $H = \begin{bmatrix} v_x & v_y & x_{00} \\ v_x & v_y & x_{11} \end{bmatrix}$, the projective transformation $A$ from the rectified image to the original image is the following:

      $$A = H \cdot \text{diag}(H^{-1}x_{11}),$$

      where $\text{diag}(v) = \begin{bmatrix} v_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & v_n \end{bmatrix}$ for a vector $v = [v_1 ... v_n] \in \mathbb{R}^n$.

   (b) [4 points] Complete the function `computeProjTransfo(v_x, v_y, x_{00}, x_{11})` that takes as input two vanishing points $v_x$ and $v_y$ and two points on a building facade $x_{00}$ and $x_{11}$ selected so that the building fits in the target picture, and outputs a $3 \times 3$ projective transformation as computed in question 3a. Tip: $A^{-1}b$ is shorthand for solving $Ax = b$ for $x$, which can be done in OpenCV with `solve(A, b, x)` (this uses the the LU decomposition by default. See the documentation for other options if interested). If you really need
the inverse of a \( \mathbf{Mat} \ A \), then you can get it with \( A.inv() \).

You are not allowed to use any OpenCV functions like \( \text{findHomography()} \) or any other geometric image transformations.

Remark: Such a projective transformation can be also computed if you specify by hand which four “corner-points” in your facade will map to preselected corners of a rectangle in the target image.

4. [8 points] Complete the function \( \text{rectifyImage}(A, \text{im}, N) \) that takes as input the projective transformation \( A \) computed by \( \text{computeProjTransfo() \&\&} \) and an image \( \text{im} \), and produces a frontoparallel picture of the image of size \( N \times N \). Note: we are not trying to get the right scale or aspect ratio here, but simply rectify the image.

You are not allowed to use the OpenCV functions \( \text{transform()}, \text{warpPerspective()} \) or anything related to those. To return a clean output, your algorithm should go through positions in the rectified image and fill the values based on the value in the non-rectified image (going through positions in the original image and filling their pre-images in the rectified image doesn’t guarantee that the rectified image will be filled correctly). Show rectified images for the three images in the homework kit. Output images should be of size 200 \( \times \) 200.

2.2 Projective transformation between two images

The facades in the three images lie on the same building in the real world. This means that there exists a projective transformation between corresponding points in the images. In this section, you will compute the homographies between pairs of images using two methods.

We want to find the projective transformations \( A_{\text{left}} \) and \( A_{\text{right}} \) such that \( p_{\text{left}} \sim A_{\text{left}} p_{\text{center}} \) and \( p_{\text{right}} \sim A_{\text{right}} p_{\text{center}} \). You will state your answer so that \( A_{\text{left}}(3,3) = 1 \) and \( A_{\text{right}}(3,3) = 1 \). We will consider two methods:

For this and the following sections, there are only a couple functions to implement in \( \text{src/hw4.cpp} \). Feel free to extend \( \text{image_rectification.cpp} \) to show off the functionality of those functions, or create a new program altogether. (You almost surely want to do this for the sake of seeing your output and debugging your code.)

1. [8 points] Using vanishing points and lines: For each pair of images, choose \( x_{00} \) and \( x_{11} \) in each image to be corners of rectangles that match across the two images (e.g. the same balcony visible in both images), compute two projective transformations (using the functions above) and compose them the right way.

2. [8 points] Using matching points: Another classical way to compute the homography between two images is to use 2D-2D correspondences. The usual way to do this is to detect interest points and match their SIFT descriptors. We will do a less sophisticated version where the user clicks a point \( p_1 \) in one image and clicks its corresponding point \( p_2 \) in the other image.

(a) [4 points] Show that if \( p' = [x' \ y' \ 1] \) is the image of \( p = [x \ y \ 1] \) i.e. \( p' \sim Ap \) and the \( A_{ij} \)'s are the coefficients of \( A \), we have:

\[
\begin{bmatrix}
A_{11} \\
A_{12} \\
A_{13} \\
A_{21} \\
A_{22} \\
A_{23} \\
A_{31} \\
A_{32} \\
A_{33}
\end{bmatrix}
= \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\]

(b) [2 points] We can write the two equations above for \( N \) pairs of matching points, yielding the system \( Mw = 0 \) where \( M \) is \( 2N \times 9 \) and \( w = [A_{11} \ldots A_{33}]^T \). What is the minimum number \( N \) of pairs to determine \( A \)? What is the condition on the points?

(c) [6 points] In practice the coordinates of the points are not always precise (low precision if we click, wrong matchings if we do automatic SIFT matching) and we use many pairs, therefore we perform least squares by minimizing \( \|Mw\| \). Complete the function \( \text{fitHomography}(X1, X2) \), where the inputs are 2-channel
A double Mat of height \( N \) and width 1 (i.e., `Mat(N, 1, CV_64FC2)`); the \( i^{th} \) element of \( X_1 \) contains a Vec2d representing a point in the first image and the \( i^{th} \) element of \( X_2 \) contains its matching point in the second image. *Leverage the `minimizeAx()` function to solve the least squares.*

(d) **[4 points]** Complete the function `findCorrespAndFitHomography()` that displays a pair of images, lets the user click a point in the first image and its matching point in the second image, reiterates the process to obtain several pairs and computes the transformation between the two images using the clicked pairs.

### 2.3 Image stitching

Similarly to the previous section, you will need to either extend an existing program or create a new program to accomplish the tasks in the following problems.

1. **[10 points]** Produce one image as in figure 1 (by overlaying individual images – no blending is required) which contains the center image and the warped versions of the left and right images, which means it is the same as have been taken from the viewpoint of the second image. Show the resulting image. Show results for both ways of determining the homographies (using vanishing points and using pairs of matching points).

2. **[6 points]** Repeat the same procedure by taking the *rectified* version of the center image instead of the original one. Show the resulting image.

![Figure 1: Images stitched together.](image-url)