

PSET 1 Review + Project Overview

Ishikaa Lunawat

CS231A

04/11/2025

A bit about me

Hi I'm Ishikaa!

- 2nd year MSEE student (graduating soon, yay!)
- Used to work with Jiajun and Chelsea at SVL
- Interests/Past work:
 - Depth Estimation
 - 3D Scene Reconstruction
 - Robot Grasping



Overview

- PSET 1 Review
- Project Overview

Problems Outline

- Q1: Projective Geometry
- Q2: Affine Camera Calibration
- Q3: Single View Geometry

Q1 - Projective Geometry

1 Projective Geometry Problems [20 points]

In this question, we will examine properties of projective transformations. We define a camera coordinate system, which is only rotated and translated from a world coordinate system (with a rigid transform).

- (a) **Prove that parallel lines in the world reference system are still parallel in the camera reference system** (note: this is an example of a rigid transformation). [4 points]
- (b) Now let us consider affine transformations, which are any transformations that preserve parallelism. Affine transformations include not only rotations and translations, but also scaling and shearing. Given some vector p , an affine transformation is defined as

$$A(p) = Mp + b$$


where M is an invertible matrix. **Prove that under any affine transformation, the ratio of parallel line segments is invariant, but the ratio of non-parallel line segments is not invariant.** [6 points]

- (c) Consider a unit square $pqrs$ in the world reference system where p, q, r , and s are points. **Will the same square in the camera reference system always have unit area? Prove or provide a counterexample. Similarly, will the unit square be preserved or not under an affine transformation?** [4 points]
- (d) You have explored whether these three properties (rotational and translational invariance, and ratio of parallel line segments) hold for affine transformations. Do these properties hold under any projective transformation? **Justify briefly in one or two sentences** (no proof needed). [6 points]

(a) Prove that parallel lines in the world reference system are still parallel in the camera reference system (note: this is an example of a rigid transformation). [4 points]

- Lines k and l are parallel
 - k_1 and k_2 are any two points on k
 - l_1 and l_2 are any two points on l
 - by definition of parallel lines:

$$(k_1 - k_2) \times (l_1 - l_2) = 0$$


$$(k_1 + p - k_2 - p) \times (l_1 + p - l_2 - p)$$

$$(Rk_1 - Rk_2) \times (Rl_1 - Rl_2)$$

- (b) Now let us consider affine transformations, which are any transformations that preserve parallelism. Affine transformations include not only rotations and translations, but also scaling and shearing. Given some vector p , an affine transformation is defined as

$$A(p) = Mp + b$$

where M is an invertible matrix. **Prove that under any affine transformation, the ratio of parallel line segments is invariant, but the ratio of non-parallel line segments is not invariant. [6 points]**

Consider any two parallel lines k and l . Take the segment between any two points k_1, k_2 on k and the segment between any two points l_1, l_2 on l . By definition of parallel segments,

$$k_1 - k_2 = k(l_1 - l_2)$$

for some real number k . Thus,

$$\|k_1 - k_2\| = k\|l_1 - l_2\|$$

(c) Consider a unit square $pqrs$ in the world reference system where p, q, r , and s are points. **Will the same square in the camera reference system always have unit area? Prove or provide a counterexample. Similarly, will the unit square be preserved or not under an affine transformation?** [4 points]

- Given a square $pqrs$,

$$\text{Area} = \|(q - p) \times (s - p)\| = \|q - p\| * \|s - p\| \sin \theta = 1$$

- Hint: projecting to camera frame is isometric (they preserve lengths, as well as angles between lines)
- How about affine transformation?

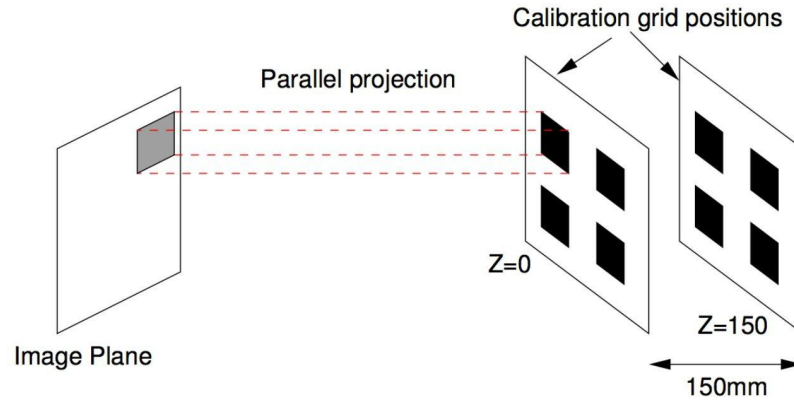
(d) You have explored whether these three properties (rotational and translational invariance, and ratio of parallel line segments) hold for affine transformations. Do these properties hold under any projective transformation? **Justify briefly in one or two sentences** (no proof needed). **[6 points]**

- Hint: Think about the difference between Projective Transformation and Affine Transformation. Briefly explain in 1 or 2 sentences.

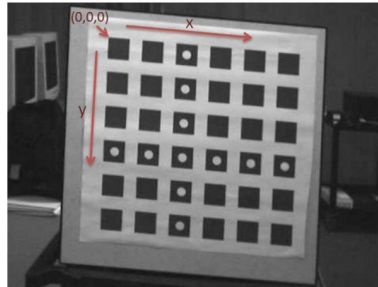
Problems Outline

- Q1: Projective Geometry
- Q2: Affine Camera Calibration
- Q3: Single View Geometry

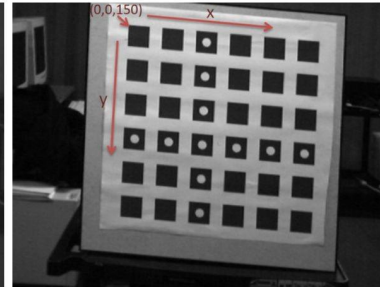
P2: Setup



(a) Image formation in an affine camera. Points are projected via parallel rays onto the image plane



(b) Image of calibration grid at $Z=0$



(c) Image of calibration grid at $Z=150$

P2: Affine Camera Model

- (a) Given correspondences for the calibrating grid, solve for the camera parameters using Eq. 2. Note that each measurement $(x_i, y_i) \leftrightarrow (X_i, Y_i, Z_i)$ yields two linear equations for the 8 unknown camera parameters. Given N corner measurements, we have $2N$ equations and 8 unknowns. Using the given corner correspondences as inputs, complete the method `compute_camera_matrix()`. You will construct a linear system of equations and solve for the camera parameters to minimize the least-squares error. After doing so, you will return the 3×4 affine camera matrix composed of these computed camera parameters. **Explain your approach and include the camera matrix that you compute in the written report.** [15 points for code + 5 for write-up]

Recall from Lecture 3

Perspective Camera Model

$$p_i = \begin{bmatrix} u_i \\ v_i \end{bmatrix} = \begin{bmatrix} \frac{\mathbf{m}_1 P_i}{\mathbf{m}_3 P_i} \\ \frac{\mathbf{m}_2 P_i}{\mathbf{m}_3 P_i} \end{bmatrix} = M P_i \quad \text{[Eq. 1]} \quad M = \begin{bmatrix} \mathbf{m}_1 \\ \mathbf{m}_2 \\ \mathbf{m}_3 \end{bmatrix}$$

in pixels

$$u_i = \frac{\mathbf{m}_1 P_i}{\mathbf{m}_3 P_i} \rightarrow u_i(\mathbf{m}_3 P_i) = \mathbf{m}_1 P_i \rightarrow u_i(\mathbf{m}_3 P_i) - \mathbf{m}_1 P_i = 0$$

$$v_i = \frac{\mathbf{m}_2 P_i}{\mathbf{m}_3 P_i} \rightarrow v_i(\mathbf{m}_3 P_i) = \mathbf{m}_2 P_i \rightarrow v_i(\mathbf{m}_3 P_i) - \mathbf{m}_2 P_i = 0$$

[Eqs. 2]

$$\left\{ \begin{array}{l} u_1(\mathbf{m}_3 P_1) - \mathbf{m}_1 P_1 = 0 \\ v_1(\mathbf{m}_3 P_1) - \mathbf{m}_2 P_1 = 0 \\ \vdots \\ u_i(\mathbf{m}_3 P_i) - \mathbf{m}_1 P_i = 0 \\ v_i(\mathbf{m}_3 P_i) - \mathbf{m}_2 P_i = 0 \\ \vdots \\ u_n(\mathbf{m}_3 P_n) - \mathbf{m}_1 P_n = 0 \\ v_n(\mathbf{m}_3 P_n) - \mathbf{m}_2 P_n = 0 \end{array} \right. \quad \text{[Eqs. 3]}$$

Recall from Lecture 3

$$\begin{cases} -u_1(\mathbf{m}_3^T P_1) + \mathbf{m}_1^T P_1 = 0 \\ -v_1(\mathbf{m}_3^T P_1) + \mathbf{m}_2^T P_1 = 0 \\ \vdots \\ -u_n(\mathbf{m}_3^T P_n) + \mathbf{m}_1^T P_n = 0 \\ -v_n(\mathbf{m}_3^T P_n) + \mathbf{m}_2^T P_n = 0 \end{cases} \quad \longrightarrow \quad \boxed{\mathbf{P} \mathbf{m} = 0} \quad [\text{Eq. 4}]$$

Homogenous linear system

$$\mathbf{P} \stackrel{\text{def}}{=} \begin{pmatrix} \mathbf{P}_1^T & \mathbf{0}^T & -u_1 \mathbf{P}_1^T \\ \mathbf{0}^T & \mathbf{P}_1^T & -v_1 \mathbf{P}_1^T \\ \vdots & \vdots & \vdots \\ \mathbf{P}_n^T & \mathbf{0}^T & -u_n \mathbf{P}_n^T \\ \mathbf{0}^T & \mathbf{P}_n^T & -v_n \mathbf{P}_n^T \end{pmatrix}_{2n \times 12}$$

$$\mathbf{m} \stackrel{\text{def}}{=} \begin{pmatrix} \mathbf{m}_1^T \\ \mathbf{m}_2^T \\ \mathbf{m}_3^T \end{pmatrix}_{12 \times 1}$$

P2: Affine Camera Model

Weak perspective model

$$\begin{bmatrix} x \\ y \\ \boxed{1} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ \boxed{0} & \boxed{0} & \boxed{0} & \boxed{1} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- **Linear**
- **8 Unknowns**

P2: Affine Camera Model

Pixel points, p Camera matrix, M 3D points, P

↓ ↓ ↓

$$x = \begin{bmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \\ 1 & \dots & 1 \end{bmatrix}, M = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ 0 & 0 & 0 & 1 \end{bmatrix}, X = \begin{bmatrix} X_1 & \dots & X_n \\ Y_1 & \dots & Y_n \\ Z_1 & \dots & Z_n \\ 1_1 & \dots & 1_n \end{bmatrix}$$

$$x = MX$$

(or)

$$p = M.P$$

P2: Affine Camera Model

numpy.linalg.lstsq

`linalg.lstsq(a, b, rcond='warn')`

[\[source\]](#)

Return the least-squares solution to a linear matrix equation.

Computes the vector x that approximately solves the equation $a @ x = b$. The equation may be under-, well-, or over-determined (i.e., the number of linearly independent rows of a can be less than, equal to, or greater than its number of linearly independent columns). If a is square and of full rank, then x (but for round-off error) is the “exact” solution of the equation. Else, x minimizes the Euclidean 2-norm $||b - ax||$. If there are multiple minimizing solutions, the one with the smallest 2-norm $||x||$ is returned.

Solve for M using `np.linalg.lstsq` or `np.linalg.pinv`

<https://numpy.org/doc/stable/reference/generated/numpy.linalg.lstsq.html>

<https://numpy.org/doc/stable/reference/generated/numpy.linalg.pinv.html>

Once you got the values for M , be careful about how to reconstruct the final camera matrix!

P2: Affine Camera Model

- (b) After finding the calibrated camera matrix, you will compute the RMS error between the given N image corner coordinates and N corresponding calculated corner locations in `rms_error()`. Recall that

$$\text{RMS}_{\text{total}} = \sqrt{\sum ((x - x')^2 + (y - y')^2) / N}$$

Compute the RMS error for the camera matrix that you found in part (a). **Include the RMS error that you compute in the written report. [10 points for code]**

Reproject the scene points onto the image plane and calculate the error. N is the number of points.

- (c) Could you calibrate the matrix with only one checkerboard image? **Explain briefly in one or two sentences. [5 points]**

Problems Outline

- Q1: Projective Geometry
- Q2: Affine Camera Calibration
- Q3: Single View Metrology

P3: Single View Metrology

In this question, we will estimate camera parameters from a single view and leverage the projective nature of cameras to find both the camera center and focal length from vanishing points present in the scene above.

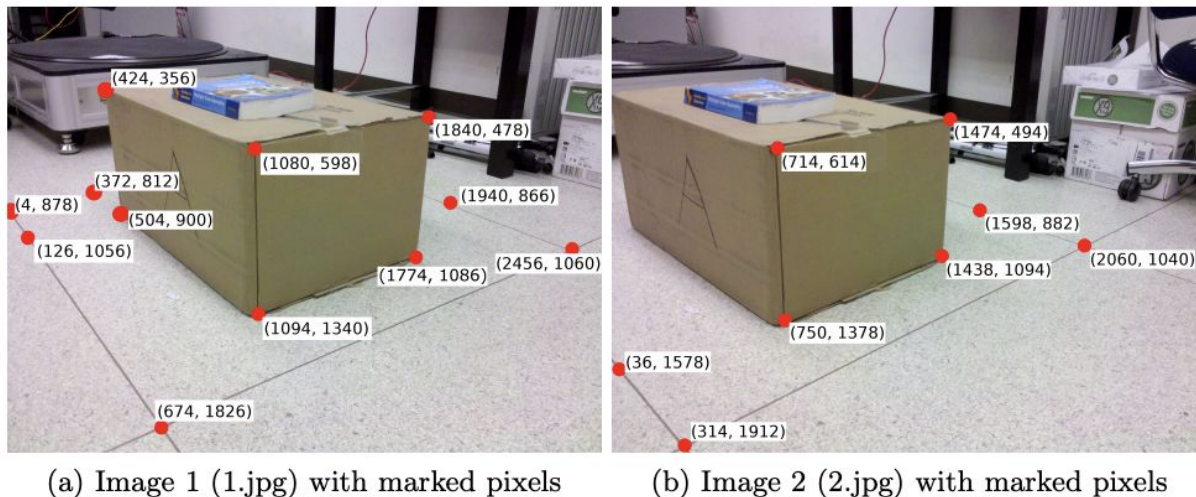


Figure 2: Marked pixels in images taken from different viewpoints.

P3: Single View Metrology

(a) In Figure 2, we have identified a set of pixels to compute vanishing points in each image. Please complete `compute_vanishing_point()`, which takes in these two pairs of points on parallel lines to find the vanishing point. You can assume that the camera has zero skew and square pixels, with no distortion. [5 points]

- Points in L_1 : $(x_1, y_1), (x_2, y_2) \rightarrow$ slope: $m_1 = (y_2 - y_1)/(x_2 - x_1)$
- Points in L_2 : $(x_3, y_3), (x_4, y_4) \rightarrow$ slope: $m_2 = (y_4 - y_3)/(x_4 - x_3)$
- Intersection of L_1 and L_2 : Vanishing Point

```
v1 = compute_vanishing_point(np.array([[1080, 598], [1840, 478], [1094, 1340], [1774, 1086]]))
v2 = compute_vanishing_point(np.array([[674, 1826], [4, 878], [2456, 1060], [1940, 866]]))
v3 = compute_vanishing_point(np.array([[1094, 1340], [1080, 598], [1774, 1086], [1840, 478]]))

v1b = compute_vanishing_point(np.array([[314, 1912], [2060, 1040], [750, 1378], [1438, 1094]]))
v2b = compute_vanishing_point(np.array([[314, 1912], [36, 1578], [2060, 1040], [1598, 882]]))
v3b = compute_vanishing_point(np.array([[750, 1378], [714, 614], [1438, 1094], [1474, 494]]))
```

P3: Single View Metrology

- (b) Using three vanishing points, we can compute the intrinsic camera matrix used to take the image. Do so in `compute_K_from_vanishing_points()`. **[10 points]**

Single view calibration - example

$$\omega = \begin{bmatrix} \omega_1 & \omega_2 & \omega_4 \\ \omega_2 & \omega_3 & \omega_5 \\ \omega_4 & \omega_5 & \omega_6 \end{bmatrix}$$

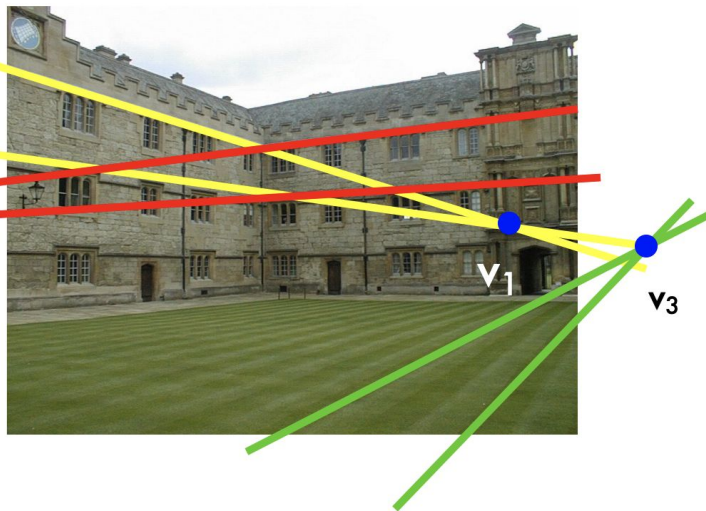
known up to scale

\mathbf{v}_2

- Square pixels $\rightarrow \omega_2 = 0$
- No skew $\rightarrow \omega_1 = \omega_3$

[Eqs. 31]

$$\begin{cases} \mathbf{v}_1^T \omega \mathbf{v}_2 = 0 \\ \mathbf{v}_1^T \omega \mathbf{v}_3 = 0 \\ \mathbf{v}_2^T \omega \mathbf{v}_3 = 0 \end{cases}$$



Single view calibration - example

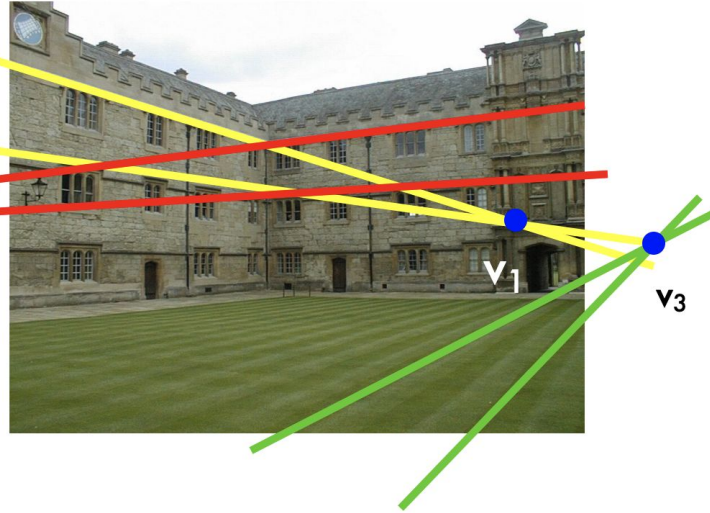
$$\omega = \begin{bmatrix} \omega_1 & 0 & \omega_4 \\ 0 & \omega_1 & \omega_5 \\ \omega_4 & \omega_5 & \omega_6 \end{bmatrix} \quad \text{known up to scale}$$

\mathbf{v}_2

- Square pixels $\rightarrow \omega_2 = 0$
- No skew $\rightarrow \omega_1 = \omega_3$

[Eqs. 31]

$$\begin{cases} \mathbf{v}_1^T \omega \mathbf{v}_2 = 0 \\ \mathbf{v}_1^T \omega \mathbf{v}_3 = 0 \\ \mathbf{v}_2^T \omega \mathbf{v}_3 = 0 \end{cases}$$



\rightarrow Compute ω !

Single view calibration - example

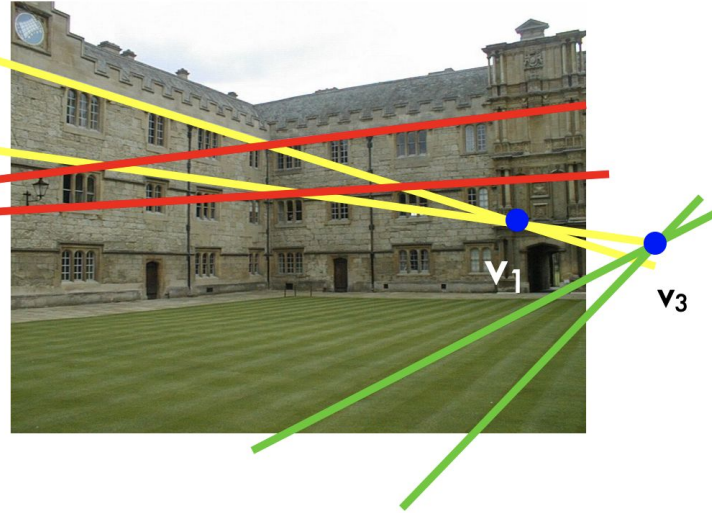
$$\omega = \begin{bmatrix} \omega_1 & 0 & \omega_4 \\ 0 & \omega_1 & \omega_5 \\ \omega_4 & \omega_5 & \omega_6 \end{bmatrix}$$

\mathbf{v}_2

- Square pixels
 - No skew
- \rightarrow
- $$\omega_2 = 0$$
- $$\omega_1 = \omega_3$$

[Eqs. 31]

$$\begin{cases} \mathbf{v}_1^T \omega \mathbf{v}_2 = 0 \\ \mathbf{v}_1^T \omega \mathbf{v}_3 = 0 \\ \mathbf{v}_2^T \omega \mathbf{v}_3 = 0 \end{cases}$$



Once ω is calculated, we get \mathbf{K} :

$$\omega = (\mathbf{K} \mathbf{K}^T)^{-1} \rightarrow \mathbf{K}$$

(Cholesky factorization; HZ pag 582)

P3: Single View Metrology

$$\text{[Eqs. 31]} \left\{ \begin{array}{l} \mathbf{v}_1^T \boldsymbol{\omega} \mathbf{v}_2 = 0 \\ \mathbf{v}_1^T \boldsymbol{\omega} \mathbf{v}_3 = 0 \\ \mathbf{v}_2^T \boldsymbol{\omega} \mathbf{v}_3 = 0 \end{array} \right.$$

$$\mathbf{A}\mathbf{w} = 0$$

A is a 3 x 4 matrix

w is a 4 x 1 vector

w is a null vector of A

- **Solve for w with SVD**
 - **A = UDV^T**
 - **Last column of V is the solution for w**
- **Use Cholesky Factorization to get K:**
 - **L = Cholesky(w)**
 - **K = (L^T)⁻¹** — remember to normalize K by last element of K

P3: Single View Metrology

- (c) Is it possible to compute the camera intrinsic matrix for any set of vanishing points? Similarly, is three vanishing points the minimum required to compute the intrinsic camera matrix? **Justify your answer. [5 points]**
- Hints: Orthogonality and vanishing points at infinity; #eqns / pairs created by vanishing points
- (d) The method used to obtain vanishing points is approximate and prone to noise. **Discuss what possible sources of noise could be, and select a pair of points from the image that are a good example of this source of error. [5 points]**
- Hints: How to estimate vanishing point better? Think # lines, etc

P3: Compute Angle Between Planes

- (e) This process gives the camera internal matrix under the specified constraints. For the remainder of the computations, use the following internal camera matrix:

$$K = \begin{bmatrix} 2448 & 0 & 1253 \\ 0 & 2438 & 986 \\ 0 & 0 & 1 \end{bmatrix}$$

Use the vanishing lines we provide in the starter code to verify numerically that the ground plane is orthogonal to the plane front face of the box in the image. Fill out the method `compute_angle_between_planes()` and **include a brief description of your solution and your computed angle in your report.** [8 points for code + 2 points for write-up]

P3: Compute Angle Between Planes

- Vanishing lines l_1 and l_2
- $l_1 = v_1 \times v_2$; $l_2 = v_3 \times v_4$
 - v_1 and v_2 : vanishing points corresponding to one plane
 - v_3 and v_4 : for the other plane

$$\cos\theta = \frac{l_1^T \omega^* l_2}{\sqrt{l_1^T \omega^* l_1} \sqrt{l_2^T \omega^* l_2}}$$

, where $\omega^* = \omega^{-1} = KK^T$

See course notes for more information

P4: Compute Rotation Matrix

- (f) Assume the camera rotates but no translation takes place. Assume the internal camera parameters remain unchanged. An Image 2 of the same scene is taken. Use vanishing points to estimate the rotation matrix between when the camera took Image 1 and Image 2. Fill out the method `compute_rotation_matrix_between_cameras()` and **submit a brief description of your approach and your results in the written report.** [8 points for code + 2 points for write-up]

P4: Compute Rotation Matrix

- Find corresponding vanishing points from both images (v_1, v_2, v_3) and (v_1', v_2', v_3')
- Calculate directions of vanishing points:

$$- v = K d \rightarrow \mathbf{d} = \frac{K^{-1} \mathbf{v}}{\|K^{-1} \mathbf{v}\|}$$

- $d_i' = R d_i$, where
 - d_i' = direction of the i^{th} vanishing point in second image
 - d_i = direction of the i^{th} vanishing point in first image

Project Overview

- Logistics
- Sharing with another class
- Proposal
- Milestone
- Presentation
- Report
- Class coverage & Do's-Dont's
- Advice

Project Logistics

- Overview [here](#)
- Teams of **1-3**: Number of people is taken into account when grading project
 - More members - More work
- Suggestions for project direction
 - Replicate an interesting paper
 - Compare different methods to a benchmark
 - Use a new approach to an existing problem
 - Implement an interesting system
 - Original research

Sharing a Project with Another Class

- Sharing projects is generally allowed
- Specify in reports
- Must be approved by both our staff and the other course staff
- Project must be profound enough that you can clarify which parts of the project were done for which class
 - Each part must be substantial enough to hold as a single project
 - Technical parts and experiments should be sufficient and different
 - For example, if you want to use CNN for flower classification, you can include some other components related to this course (e.g. geometry, ...)
- Will need a separate write-up for each class

Project Proposal

- Maximum of 2 pages
- Submit the report as a PDF document through Gradescope
- Include the following:
 - Title and authors
 - Sec. Introduction: Problem you want to solve and why
 - Sec. Technical Approach: How do you propose to solve it?
 - Sec. Milestones (dates and sub-goals)
 - References
- You will be assigned a project mentor

Project Milestone Report

- Maximum of 4 pages
- Submit the report as a PDF document through Gradescope
- Include the following:
 - Title and authors
 - Sec. Introduction: Problem you want to solve and why
 - Sec. Technical Approach: How do you propose to solve it?
 - Sec. Milestones achieved so far
 - Sec. Remaining Milestones (dates and sub-goals)
 - References

Project Presentations

- Short presentation with time for a brief Q&A
- Include the following:
 - Problem Motivation/Description
 - Technical Approach
 - Results
 - Maybe demo (+)!

Project Final Report

- Length of 6-8 pages
- Submit the report as a PDF document through Gradescope
- Submit your code
- Include the following:
 - Title and authors
 - Abstract
 - Sec. Introduction
 - Sec. Previous work
 - Sec. Technical Approach
 - Sec. Experiments
 - Sec. Conclusions
 - References

Class Coverage

Camera models and calibration

Single camera and how we model it

Single view metrology

Estimating geometry from a single view

Epipolar Geometry (Stereo Vision)

Estimating geometry from two viewpoints

Structure from Motion

Using motion/several viewpoints to estimate structure

Volumetric Stereo

Using multiple views to map 3D points

Class Coverage

Representations and Representation Learning

Extracting features from 2D images for downstream applications

Monocular Depth Estimation & Feature Tracking

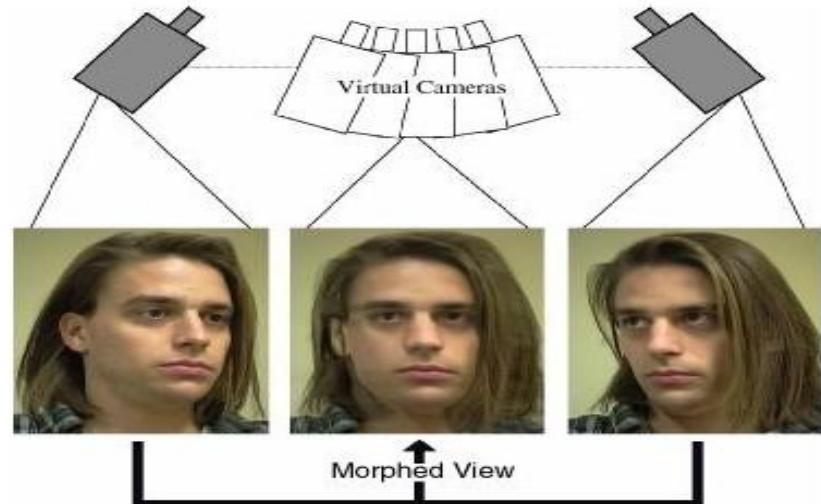
Estimating depth in images, tracking of pixels in videos

Optical and Scene Flow

Neural Radiance Fields

View Morphing

Image morphing techniques can generate compelling 2D transitions between images.



View Morphing



Automatic Photo Pop-Up



A fully automatic method for creating a 3D model from a single photograph
Hoiem, D., Efros, A. A., and Herbert, M, "Automatic Photo Pop-Up", SIGGRAPH 2005.

Mobile Devices

Can you take an existing vision algorithm and adapt it to a mobile device to make it more useful?

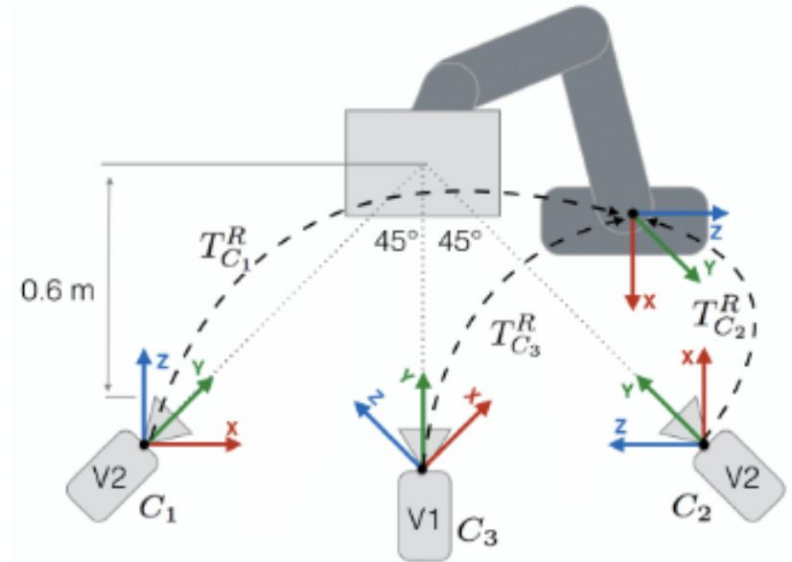


Monocular Depth estimation

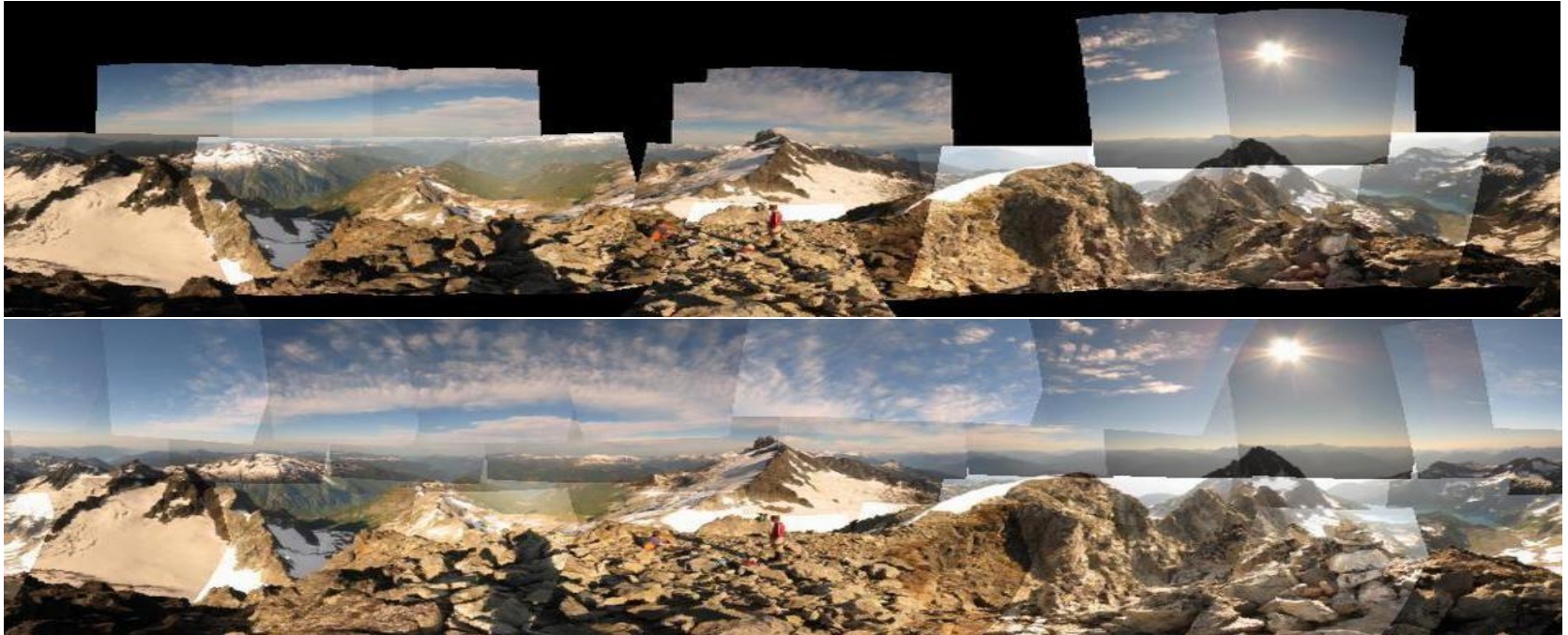


Figure 10. Snapcode of publicly available community lens

Camera Calibration for Robots



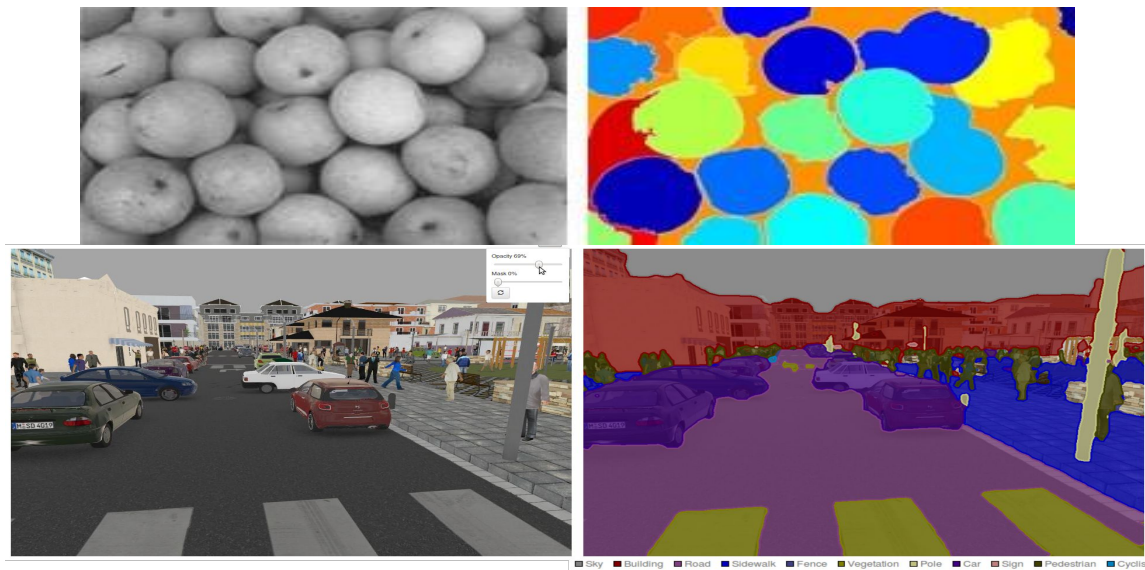
Recognizing Panoramas



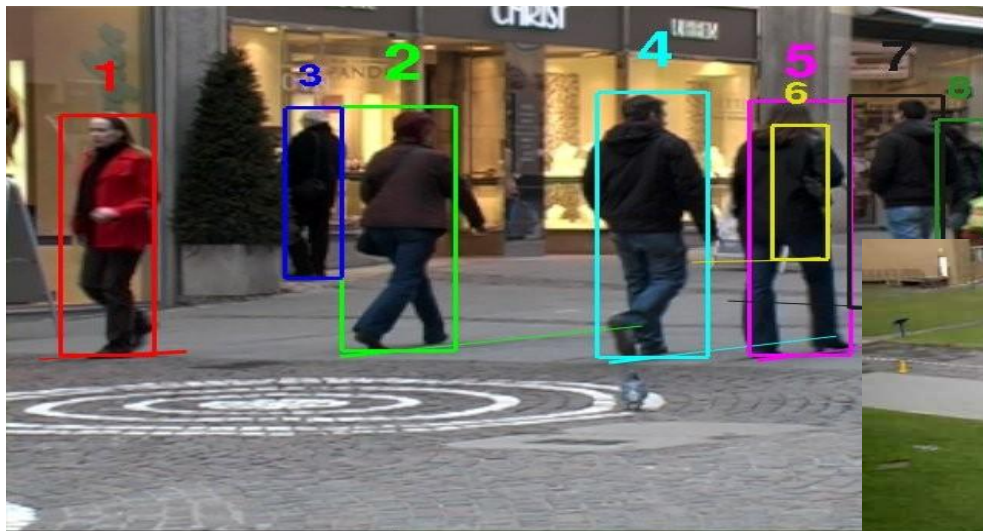
Brown, M. and Lowe, D. G., "Recognizing Panoramas", ICCV 2003.

Image Segmentation

Partition an image into multiple segments (sets of pixels) in order to make it easier to analyze



Tracking



Other Topics

- Pose Estimation: Estimate the skeleton angles for a person from an image/video
- Action and Gesture Recognition: Is a person standing, walking, or sitting in an image/video? Is he/she waving?
- Scene Understanding: Can you classify a scene? Can you recognize and/or segment each component of the scene?
- Trajectory Forecasting
- ...

Negative project examples

- Projects without components related to the course
 - Project has to have some 3D or projective Geometry component rather than only processing single images for example
- Applying Alexnet for image classification
- Finding and running an existing Github code
- Only running OpenCV libraries for a task
- ...

Where to get Project Ideas

- Course Staff: Office hours, [ideas posted on website](#)
- Computer vision papers and conferences
 - CVPR
 - ICCV
 - ECCV
- Computer vision research groups at Stanford
 - Silvio Savarese
 - Fei-Fei Li
 - Juan Carlos Niebles
- Past projects: See [course website](#)
- [Papers with Code: Computer Vision](#)
- Come up with your own!

Datasets

- Many are available on the web
- See the following aggregators:
 - CV Datasets on the Web
 - Kaggle
 - Yet Another Computer Vision Index To Datasets (YACVID)
- References found in papers

Project Advice

- Choose your team well
- Make sure the scope of your project fits a quarter
 - Set a minimum goal, desired goal, and a moonshot
- Constrain your problem smartly
- See what datasets are available if you are doing a recognition project
 - Specially for deep learning projects
- You may need to plan ahead/learn outside materials
- Use software when available
 - OpenCV, MATLAB, Deep learning frameworks
- Come ask questions – We're happy to talk!

Thanks!