# Team A: Final Project: Structure From Motion

**Team members:** Vincent Hock, Tomas Arevalo, Callum Taylor

The goal of structure from motion is to use images to build sparse 3D models of a scene and to localize the cameras with respect to such models. This process involves estimating both 3D geometry (“structure”) and camera pose (“motion”) at the same time. An appropriate name would be structure *and* motion but for historical reasons, it is commonly known as structure *from* motion. It has proven to be immensely useful in applications such as virtual tourism, autonomous navigation, and augmented reality. There are some pretty good open-source pipelines available (see [COLMAP](https://colmap.github.io/)), but they all suffer from failures. So SFM remains an important and vibrant area of research.

SFM input:

* A set of images of a scene taken from multiple unknown viewpoints

SFM outputs:

* A sparse 3D point cloud of scene points
* A set of intrinsic and extrinsic parameters for each of the cameras

**To learn more**: Read the introduction to Szeliski Chapter 11 (Structure from motion and SLAM) and §11.4.6 (Application: Reconstruction from internet photos) and §11.5.2 (Application: Smartphone augmented reality). For deeper reading, proceed to §11.3 (Two-frame structure from motion), §11.2.4 (Triangulation) and §11.4.2 (Bundle adjustment).

## Minimum Expected Achievements

* Calibrate a smartphone camera’s intrinsic parameters
* Use the calibrated camera to capture structure-from-motion sequences of at least three scenes, with each sequence including at least five images from different viewpoints
* Use openCV to Implement keypoint detection and robust keypoint-matching between two images
* Use openCV to implement these steps to automatically reconstruct a “base scene” from two views:
  + Estimate essential matrix from matched keypoints and known intrinsic parameters
  + Extract rotation and translation from essential matrix
  + Triangulate matched keypoints to obtain 3D scene points
* Use openCV to implement these steps for automatically adding another view to the base scene:
  + Estimate the new camera pose from matched keypoints using a perspective-n-point (PnP) solver
  + Triangulate new 3D points
* For each scene, use plotly to create an interactive visualization of the reconstructed 3D scene points and cameras
* Apply [COLMAP](https://colmap.github.io/)’s SFM pipeline to your image sequences and compare its results to the ones that you obtained with your pipeline.
* Implement at least one of these extensions (or at least one other extension that you propose and get approved by the teaching staff):
  + Use scipy, pytorch or jax to implement bundle adjustment to iteratively refine a set of camera parameters and 3D points to minimize the reprojection error
  + Generalize your pipeline to eliminate the requirement for pre-calibrating the camera, so that the intrinsic parameters are estimated automatically from the image sequence

## 

## Project Plan

### Resources (*edit and add to this list*)

* [openCV tutorial: epipolar geometry](https://docs.opencv.org/4.x/da/de9/tutorial_py_epipolar_geometry.html)
* [openCV tutorial: Feature Matching](https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html)
* openCV functions:
  + cv2.findEssentialMat()
  + cv2.decomposeEssentialMat()
  + …

### Equipment (*edit and add to this list*)

* One smartphone with a camera app that supports focal-distance locking (e.g. [Adobe Lightroom for mobile](https://helpx.adobe.com/lightroom-cc/using/capture-photos-mobile-ios.html), [CameraPixels](https://camerapixels.app/))
* Planar calibration target (aprilboard)
* Scenes to be captured

### Capture protocol (*edit and add to this*)

1. Choose a scene, place the camera at the desired distance from it, and lock the focus.
2. With the focus locked, capture 10-20 images of an aprilboard for calibration
3. With the focus still locked, capture a set of images of the scene from different viewpoints, making sure there is overlap between the fields of view, and keeping the camera at the right distance for the scene to remain in focus.

### Communication plan (*edit and add to this*)

*Describe the locations and times of team meetings. Describe the tools for managing code and project files (e.g., Github and google-doc URLs). Continually update this section and be precise.*

We will meet weekly during and after the lab in the SEC on Thursdays. We’ll also have time scheduled on Monday afternoon for continued work on the functions; these meetings will take place in the Quad, as all members of the team live there.

For managing code, we will use a shared Google Drive folder with a Jupyter Notebook; team members will test and implement their functions on individual notebooks, but copy completed code over once it has been verified. We used a similar communication plan for the Scanner project, and it worked very well.

### Breakdown of tasks and timeline (*edit and add to this*)

| What | Who | When (completion deadline) |
| --- | --- | --- |
| Categorize minimum expected achievements by difficulty / time required | Vincent Hock | Pre-lab 4/11 |
| Find/define dataset to serve as unit-testing input | Tomas Arevalo | Pre-lab 4/11 |
| Brainstorm scene ideas for image capture | Callum Taylor | Pre-lab 4/11 |
| Complete “deeper reading” background readings | Full team | Pre-lab 4/11 |
| Define, describe, and assign main Python functions  Instantiate a shared project workflow for your team that includes (at least): (i) a project-specific github repository; (ii) at least one editable notebook (.ipynb) file within that repository; and (iii) at least one datafile (image or ZIP) and source file (.py) within that repository that are loaded and used in the notebook  Add rows to this table that outline the expected milestones for: (i) the end of Lab Session on 4/18; and (ii) presentation on 4/23. Include in this the name of the leader for each of: the final written report, the final demo notebook, and the presentation | Full team | Post-lab 4/11 |
| **Capture datasets.** Capture calibration images and structure-from-motion sequences of at least three scenes, with each sequence including at least five images from different viewpoints |  |  |
| Scene from two images | | |
| **Keypoint Detection.** Use openCV to Implement keypoint detection and robust keypoint-matching between two images. | Vinny |  |
| **Essential matrix.** Estimate essential matrix from matched keypoints and known intrinsic parameters. Extract rotation and translation from the essential matrix. | Vinny |  |
| **Triangulate.** Triangulate matched keypoints to obtain 3D scene points. |  | End of lab 4/18 |
| Adding Additional Images | | |
| **PnP Solver**. Using openCV and a PnP solver, estimate the new camera pose from matched keypoints. |  | End of lab 4/18 |
| **Triangulate again.** |  | End of lab 4/18 |
| Visualization | | |
| **Plotly.** Create plotly visualizations for each scene, including *both* the scene points and multiple camera positions. |  |  |
| **Comparison to COLMAP.** Apply [COLMAP](https://colmap.github.io/)’s SFM pipeline to image sequences and compare its results to the plotly visualizations. |  |  |
| Extension | | |
| **Eliminate fiducials.** Generalize pipeline to eliminate the requirement for pre-calibrating the camera, so that the intrinsic parameters are estimated automatically from the image sequence. |  |  |
| Deliverables | | |
| **Slides for Pres.** ten minutes or less |  | 4/23 |
| **Final Written Report.** PDF format, 4-6 pages including references, compiled in latex/overleaf using the CVPR Latex templateLinks to an external site. |  | 5/7 |
| **Demo Notebook.** a URL to a Colab notebook, which includes connections to your team's source files in a github repository |  | 5/7 |

### Description of Main Python Functions (*edit and add to this*)

*A list of the project’s principal functions. For each one, describe the inputs, outputs, who is responsible, and a plan for testing the function’s correctness.*

#### detectKeypoints(image)

Purpose

* Finds all the interesting points in the image.

Arguments

* image: H x W image of scene

Output

* keypoint\_locations: N x 2 matrix, with one row for each of N key points. Each row contains the x-coord, y-coordinate.
* keypoints\_SIFT: N x (8 \* num\_subregions) matrix, with one row for each of N key points. Each row i contains the SIFT vector for the ith key point.

Verify correctness by

* *add description here*

#### matchKeypoints(keypoint\_locations1, keypoint\_locations2, keypoints\_SIFT1, keypoints\_SIFT2, threshold=0.15)

Purpose

* Match keypoints from image 1 and image 2 using their corresponding SIFT vectors, and return the locations of the pairs of the top threshold% of matching points.

Arguments

* keypoint\_locations1: N x 2 matrix returned by detectKeypoints on image 1
* keypoint\_locations2: N x 2 matrix return by detectKeypoints on image 2
* keypoints\_SIFT1: N x (8 \* num\_subregions) matrix returned by detectKeypoints on image 1
* keypoints\_SIFT2: N x (8 \* num\_subregions) matrix returned by detectKeypoints on image 2
* threshold=0.15: the percentage of the matched key points with the smallest distances that we will return

Output

* matched\_keypoints: N x 2 x 2 matrix containing the coordinates of N matched pairs of key points

Verify correctness by

* *add description here*

#### getEssentialMatrix(points1, points2)

Purpose

* Use openCV function to

Arguments

* points1:
* points2:

Output

* matched\_keypoints: N x 2 x 2 matrix containing the coordinates of N matched pairs of key points

Verify correctness by

* *add description here*

#### triangulate\_images(points1, points2,)

Purpose

* Use openCV function to

Arguments

* points1:
* points2:

Output

* matched\_keypoints: N x 2 x 2 matrix containing the coordinates of N matched pairs of key points

Verify correctness by

* *add description here*