EECE 5550 Mobile Robotics - Section 2
Lab #3: Visual navigation

Due: Nov 14, 2025 by midnight

In the following series of exercises, you will implement a basic visual mapping and navigation system using the GTSAM, OpenCV, and AprilTag libraries. (If you have not used GTSAM before, you may find it helpful to refer to the quick-start tutorial and Python examples as you implement your solutions.)

Problem 1: Camera calibration

As we saw in Lecture 5, the pinhole camera model $\pi\colon \mathrm{SE}(3) \times \mathbb{R}^3 \times \mathbb{R}^{3\times3} \to \mathbb{R}^2$ that describes the operation of a pinhole camera depends upon three pieces of data:

- The 3D position of a point $P \in \mathbb{R}^3$ in the world coordinate frame,
- The 3D pose $X \in \mathrm{SE}(3)$ of the camera in the world frame,
- The five parameters of the *camera matrix* K:

$$K = \begin{pmatrix} \alpha_u & s & p_u \\ & \alpha_v & p_v \\ & & 1 \end{pmatrix} \in \mathbb{R}^{3\times3} \tag{1}$$

  that describes the geometry of the camera itself. (Specifically, α_u and α_v describe the physical sizes of the pixels on the image sensor, $(p_u, p_v) \in \mathbb{R}^2$ the location of the principal point in the camera's images, and s the skewness of the coordinate axes.)

Therefore, if we want to estimate the positions of 3D points in the world and/or track the pose of a camera using images, it is first necessary to know the camera matrix K for the camera that we are using.

While it is possible to look up or estimate most of these values from the "nominal" camera specifications (i.e. the focal length, pixel size, and image dimensions) provided by the manufacturer, as we discussed in class, no manufacturing process is perfect, and therefore the *actual* values of these parameters on the physical camera that we have in-hand will differ (although usually only slightly) from the *nominal* (design) specifications. Therefore, in order to model our camera as accurately as possible, it is advantageous to *directly estimate* the *actual* (real-world) parameters of the camera matrix (1) directly from data, using nonlinear least-squares (NLS) parameter fitting. This procedure is called *camera calibration*.

To perform camera calibration, one typically employs a *calibration target*: an object that has a set of M easily-identifiable target points *at known positions* $P_j \in \mathbb{R}^3$ in the object's body-centric coordinate frame. (A very common example of a calibration target is a *chessboard pattern*, since

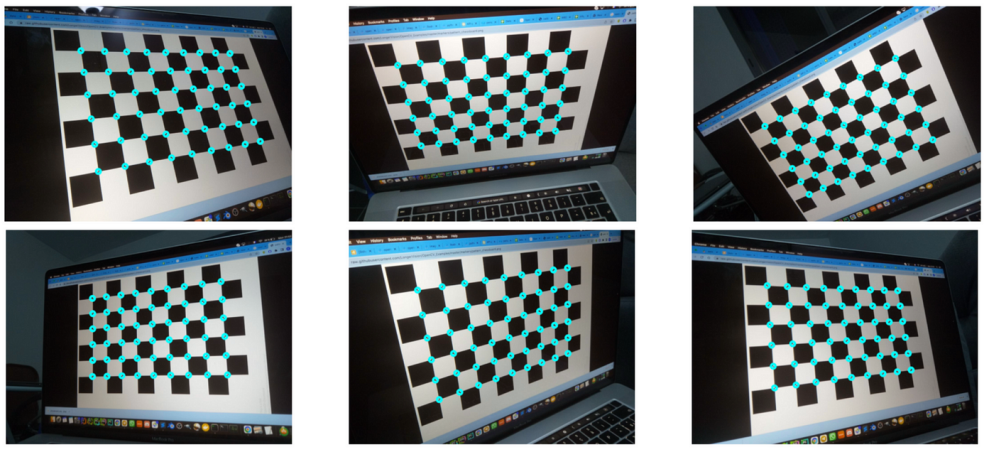

Figure 1: Chessboards are commonly used as calibration targets for camera calibration.

the corners on the chessboard occur at known positions and are easy to detect in images [see Fig. 1].) Given a set of N images of the target object, one can then estimate the parameters of the camera matrix K (1) by solving the following nonlinear least-squares problem:

$$\min_{K,X_i} \sum_{i=1}^{N} \sum_{j=1}^{M} \|\tilde{u}_{ij} - \pi(K, X_i, P_j)\|_2^2 \tag{2}$$

where:

- $P_j \in \mathbb{R}^3$ is the (*known*) position of the jth target point in the object's body centric coordinate frame,
- $\tilde{u}_{ij} \in \mathbb{R}^2$ is the observed position of the jth target point in the ith camera image,
- $X_i \in \mathrm{SE}(3)$ is the pose of the camera in the object's body-centric coordinate frame when the ith image was taken,
- K is the camera matrix.

Note that formulation (2) selects the parameters $K \in \mathbb{R}^{3\times3}$ and $X_1, \ldots, X_N \in \mathrm{SE}(3)$ that minimize the sum of squared norms of the *reprojection errors* $r_{ij} \triangleq \tilde{u}_{ij} - \pi(K, X_i, P_j)$: the differences between the *observed* $\tilde{u}_{ij}$ and *predicted* $\pi(K, X_i, P_j)$ projections of the points in each image.

In this exercise, you will use the OpenCV library to perform camera calibration.

(a) The `calibration_images` folder on the course website contains a set of 8 images of an 8×6 chessboard with a square sidelength of .01 m. Download these images to your computer.

(b) Write a short program that will estimate the parameters of the camera matrix K by solving the NLS problem (2), using these calibration target images as input. In detail, your program should perform the following steps:

- For each image i in the dataset, find the locations $\tilde{U}_i \triangleq \{\tilde{u}_{ij}\}_{j=1}^{48} \subset \mathbb{R}^2$ of the 48 chessboard corner points in the image using the findChessboardCorners function.

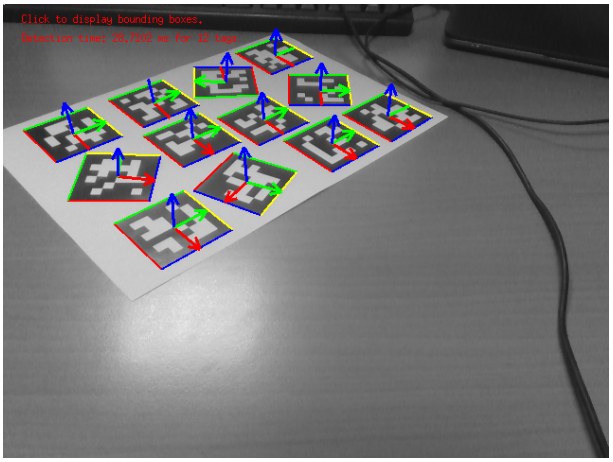

Figure 2: AprilTags are convenient fiducials for visual navigation tasks.

– Given the observed feature point locations $\{\tilde{U}_i\}_{i=1}^{8}$ in each of the 8 images and the (known) locations $\{P_j\}_{j=1}^{48} \subset \mathbb{R}^3$ of the corner points on the chessboard,[1] construct and solve the nonlinear least-squares problem (2) using the calibrateCamera function.

Submit your code, together with the estimated parameters of the camera matrix K.

Problem 2: Perspective-n-Point AprilTag pose estimation

AprilTags are 2D barcodes that are often used as fiducials in robotics and computer vision applications: they they are designed to be robustly detectable in images across a wide range of viewing angles, and the barcode on each tag encodes a unique identifier for that tag (thus solving the data association problem) [see Fig. 2]. Consequently, AprilTags are especially useful as artificial landmarks in visual mapping and navigation tasks.

One of the most useful features of AprilTags is that it is possible to determine the relative pose between the camera and the AprilTag from a *single* image[2] if we know (i) the camera matrix K for the camera and (ii) the physical size of the tag. In this exercise, you will use the AprilTag and GTSAM libraries to solve this relative pose estimation problem.

(a) The `vslam` folder on the course website contains a sequence of 500 images taken from a video captured by a camera (the same one used in Problem 1) being scanned over a collection of AprilTags. Download these images to your computer.

(b) **Formulating the PnP problem:** Suppose that we are given 2D projections $\{\tilde{u}_i\}_{i=1}^{N} \subset \mathbb{R}^2$ (in some camera image) of N points whose 3D positions $\{P_i\}_{i=1}^{N} \subset \mathbb{R}^3$ in the world coordinate system are known, and we *also* know the camera matrix K for the camera. Write down the nonlinear least-squares problem that estimates the pose of the camera $X \in \mathrm{SE}(3)$ by minimizing the sum of squared norms of the reprojection errors of the observed points [you may leave your answer in terms of the camera projection function π, as in (2)].

[1] Note that part of your task will be to determine the locations of these corner points in some body-fixed coordinate system on the chessboard. You are free to use whatever coordinate system that you like, **subject to the condition that it is defined in units of meters**.

[2] In computer vision, this estimation problem is called the *Perspective-n-Point* (PnP) problem.

(c) The first image (`frame_0.jpg`) in the `vslam` dataset contains an observation of the AprilTag with ID 0. Using this image, write a short program that performs the following operations:

- Using the AprilTag library, extract the locations $\{\tilde{u}_i\}_{i=0}^{3} \subset \mathbb{R}^2$ of tag 0's four corners in the image.
- Using the GTSAM libary, construct and solve the factor graph representing the non-linear least-squares problem you derived in part (b) to estimate the pose $X \in \mathrm{SE}(3)$ of the camera with respect to AprilTag 0, given the observed positions of its 4 corner points in the image.

For the purposes of this question, you should assume that the (3D) body-centric coordinate frame attached to AprilTag 0 is defined as follows: the origin of the coordinate system is at the center of the tag, with the $+X$-axis pointing to the right, the $+Y$-axis pointing down, and the $+Z$-axis pointing into the page (away from the viewer) as you are facing the tag. The AprilTag corner points are labeled sequentially, starting with the lower-left corner (corner 0) and proceeding counterclockwise. The tags appearing in the `vslam` dataset all have sides of length .01 m. Similarly, the camera's body-centric coordinate system has the $+X$-axis pointing to the right, the $+Y$-axis pointing down, and the $+Z$-axis pointing forward (coinciding with the optical axis). The camera used to collect the `vslam` dataset is the same one used to collect the `calibration` images, so you should use the same calibration matrix K you estimated in Problem 1.

You may find the following pointers useful in implementing your solution:

- The Point3 class models a 3D point $P \in \mathbb{R}^3$.
- The Pose3 class models a 3D pose $X \in \mathrm{SE}(3)$.
- The Cal3_S2 class models the camera matrix K for a pinhole camera.
- The GenericProjectionFactorCal3_S2 class models observations from a pinhole camera with a *fixed and known* calibration K; it calculates the reprojection error for a point observation $\tilde{u} \in \mathbb{R}^2$, given estimates for the 3D position $P \in \mathbb{R}^3$ of the observed point and the pose $X \in \mathrm{SE}(3)$ of the camera.
- You can use a PriorFactor with a Constrained noise model to fix the positions of the AprilTag's corner points.

Report the estimated pose of the camera in AprilTag 0's body-centric coordinate frame, and submit your code.

Problem 3: Visual SLAM

In Problem 2, you showed how one can estimate the relative pose between an AprilTag and a calibrated camera by solving the PnP problem. The AprilTag library can actually perform this relative pose estimation for you as part of its tag detection operation:[3] specifically, it can return the estimated translation $\tilde{t}_{CT} \in \mathbb{R}^3$ and rotation $\tilde{R}_{CT} \in \mathrm{SO}(3)$ of the rigid motion $\tilde{X}_{CT} \triangleq (\tilde{t}_{CT}, \tilde{R}_{CT}) \in \mathrm{SE}(3)$ that maps points from the AprilTag's body-centric coordinate system to the camera's body-centric coordinate system. In this exercise, you will exploit this functionality to implement a simple visual SLAM system that uses AprilTags as landmarks.

[3] Although the AprilTag library uses a different method than the PnP approach you developed in Problem 2.

(a) Let $X_{WC} \in \mathrm{SE}(3)$ denote the pose of the camera in some world coordinate frame W, and $X_{WT} \in \mathrm{SE}(3)$ denote the pose of the AprilTag in that same coordinate system. Derive an expression for X_{CT} in terms of X_{WC} and X_{WT}.

(b) We know that the *estimated* relative pose $\tilde{X}_{CT} \in \mathrm{SE}(3)$ returned by the AprilTag library will not *exactly* coincide with the *true* relative pose $X_{CT} \in \mathrm{SE}(3)$ because it is calculated using (noisy) camera images. Being good probabilistic roboticists, we thereore want to model $\tilde{X}_{CT}$ as a *noisy measurement* of X_{CT}. As we saw in Lecture 6 (Review of Probability), we can do so by assuming the following measurement model for SE(3)-valued data:

$$\tilde{X}_{CT} = X_{CT} \exp(\xi), \qquad \xi \sim \mathcal{N}(0, \Sigma), \tag{3}$$

where $\xi \in \mathrm{Lie}(\mathrm{SE}(3)) \cong \mathbb{R}^6$ is an element of the (6-dimensional) Lie algebra of SE(3) and $\Sigma \succeq 0$ is a 6×6 covariance matrix.

Using the model (3) and your result from part (a), derive an expression for the negative log-likelihood $-\log p(\tilde{X}_{CT} | X_{WC}, X_{WT})$ of the estimated relative pose $\tilde{X}_{CT}$ in terms of X_{WC} and X_{WT}, the poses of the camera and AprilTag in the world frame. (For the purposes of this question, and to simplify the math a bit, you may ignore the normalization constant in the Gaussian probability distribution.)

(c) **Formulating the visual SLAM problem:** In this exercise you will derive the form of the maximum-likelihood estimation problem corresponding to visual SLAM using AprilTags as landmarks. Suppose that we collect N images of a camera with known calibration K in an environment containing M AprilTags. For each image $i \in [N]$, some subset $S_i \subseteq [M]$ of the AprilTags will be visible in that image, and each such AprilTag detection produces a noisy measurement $\tilde{X}_{ij} \in \mathrm{SE}(3)$ of the transformation from the coordinate system of tag j to the body centric coordinate system of the camera at the time image i was taken.

Using your result from part (b), write down the form of the maximum-likelihood estimation that *jointly* estimates the camera poses $X_1, \ldots, X_N \in \mathrm{SE}(3)$ associated with the N images *and* the M poses $Y_1, \ldots, Y_M \in \mathrm{SE}(3)$ of the AprilTags.

(d) Write a short program that uses the AprilTag and GTSAM libraries to construct and solve the particular instance of the maximum-likelihood estimation from part (c) that is determined by the images in the `vslam` dataset. Specifically, your program should perform the following steps:

- For each image $i \in [N]$ in the dataset, run AprilTag detection to identify (i) the set of AprilTags S_i that are visible in that image, and (ii) the relative poses $\tilde{X}_{ij} \in \mathrm{SE}(3)$ of each visible tag $j \in S_i$ with respect to the camera.
- Using the GTSAM library, construct and solve the factor graph representating the maximum-likelihood estimation problem you derived in part (c) to *jointly* estimate the camera poses $X_1, \ldots, X_N \in \mathrm{SE}(3)$ and the tag poses $Y_1, \ldots, Y_M \in \mathrm{SE}(3)$.

You may find the following pointers useful in implementing your solution:

- The GTSAM BetweenFactor factor class can be used to model a noisy observation of the relative transformation between two elements $X, Y \in G$ of a Lie group G. [For the purposes of this question, you may assume an Isotropic noise model, which corresponds to taking $\Sigma = \sigma^2 I$ in (3).]
- You should also fix the gauge of your SLAM problem by constraining the pose of AprilTag 0 to be the origin [i.e. $Y_1 = (0, I_3) \in \mathrm{SE}(3)$] using a PriorFactor with a Constrained noise model.

Plot the resulting estimated positions of the AprilTags (in red) and camera poses (in blue), and submit this plot together with your code.