EECS/ME/BioE 106B Homework 4: Vision & Estimation

Spring 2023

Note: All of the coding portions of this assignment will be done in a Colab notebook, which you can find here. Make a copy of the notebook to get started!

Problem 1: Image Filtering & Feature Detection

In robotics, images are extremely powerful tools for learning about the environment our robot interacts with. Let's develop some techniques in image processing that can be readily applied to robotic systems! Recall that we often represent black and white images as *matrices* - two dimensional arrays where the value at each (x, y) coordinate in the image represents the intensity, of light at that point. By convention, an intensity of zero corresponds to total darkness.
In the real world, we're often faced with the challenge of filtering out noise from images by smoothing out sudden jumps in intensity. We may apply a filter to a point (x, y) in an image $I(x, y)$ with the following operation, known as a *convolution*, represented by the symbol $*$:

$$(F * I)(x, y) = \sum_{i=-N}^{N} \sum_{j=-N}^{N} F(i, j) I(x - i, y - j) \tag{1}$$

A convolution filters the image $I(x, y)$ at a point by taking a weighted sum of the image's values in a square region around (x, y). The function $F(x, y)$, called the *kernel* of the convolution, may be changed to adjust the properties of the filter. We can think about a convolution sum more directly in terms of matrices by storing the convolution kernel F in a matrix:

$$F = \begin{bmatrix} F(-N, -N) & \dots & F(-N, N) \\ \vdots & \ddots & \vdots \\ F(N, -N) & \dots & F(N, N) \end{bmatrix} \tag{2}$$

Where we have simply placed each $F(i, j)$ in the entries of a matrix F. When taking the convolution of an image with this kernel at a point (x, y), we can imagine taking a sum where each element in a region around (x, y) is scaled by an element of the convolution kernel matrix. For example, if we apply a 3×3 convolution to the matrix below at the point (1, 1):

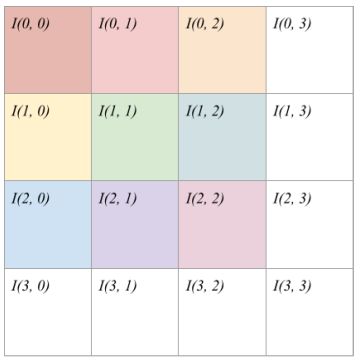


Image Matrix

Convolution Kernel

To find the convolution at the point $(1, 1)$, we perform a weighted sum of the values in the colored squares in the image matrix. By applying this operation to each entry in the image matrix, we can filter entire images!

Questions

1. Let's implement a matrix convolution sum in Python. Recall that given an $N \times N$ kernel F and an image I, we can define the convolution of the image with the kernel F as:

$$(F * I)(x, y) = \sum_{i=-N}^{N} \sum_{j=-N}^{N} F(i, j) I(x - i, y - j) \tag{3}$$

   If x, y are on the edge of the image, this sum will go "out of bounds" of the image dimensions. To take care of this, we use *zero padding*, and set $I(x - i, y - j)$ to be equal to zero when the sum goes outside of the image boundaries. Go to the Colab notebook (linked at the top of this document) and implement the *Convolutions* module.

2. Let's think about a few types of convolution kernels. The following two kernels, which are examples of *Gaussian* and *averaging* kernels, are used to smooth out noise in images:

$$F_{Gauss} = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \; F_{Avg} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{4}$$

   Provide a brief worded explanation as to why the sums of the elements of these matrices should add up to 1. Then, explain why it might be advantageous to weight the center pixel higher than the surrounding pixels, as in the Gaussian smoothing kernel. Next, go to the notebook and complete the *Gaussian smoothing* module. Attach the smoothed image to your solution.

3. We can show that we can estimate the horizontal and vertical gradients of the image intensity by taking the convolution of the image with the following two kernels:

$$G_h = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \; G_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{5}$$

   Once we've performed these convolution, we may compute the magnitude of the image gradient at each point by:

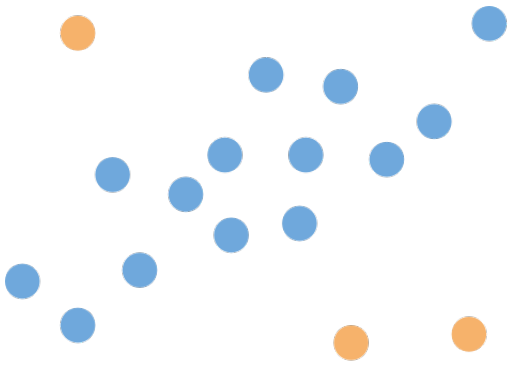
$$||\nabla I(x, y)|| = \sqrt{\nabla_h(x, y)^2 + \nabla_v(x, y)^2} \tag{6}$$

   Where $\nabla_h(x, y) = (G_h * I)(x, y)$ and $\nabla_v(x, y) = (G_v * I)(x, y)$. Go to the notebook and implement the *Image Gradient* module. Attach the gradient magnitude image to your solution.

4. Through examining the gradient of the image at different locations, we can extract things such as the edges and corners of features in our images! Go to the notebook and implement the *Harris Corner Detector* module, which will enable you to pick out the corners of different features in images! Attach the corner identification image to your solution.

Problem 2: Feature Extraction with RANSAC

After performing basic image processing techniques, how can we mathematically describe the different features and lines in the scene? For instance, if we've detected the edge of a table using an edge detection algorithm, how can we convert the set of bright pixels corresponding to that edge to an actual equation we can use for obstacle avoidance?
The set of tools involved in performing such tasks fall under the umbrella of *feature extraction*, the area of computer vision concerned with extracting useful geometric information from images. The Random Sample and Consensus (RANSAC) algorithm is an algorithm that helps us extract fundamental geometric information from our vision data in a way that is robust to noise and outliers. We can use it to help identify things like edges in noisy 2D images, planes in noisy 3D depth images, and many more such features. Let's discuss how it works.



Above: How can we fit a line to data while ignoring outliers?

Suppose we want to fit a line to the noisy data in the image above. How could we perform this fit in a meaningful way that avoids using the outliers in the data? Consider the following procedure, which outlines the RANSAC algorithm:

1. Suppose we begin with a set of N data points, which we call S. Pick two random points from the data set S and draw a straight line between them.

2. Define two empty sets, an *inlier* set and an *outlier* set.

3. Compute the distance from every point in S to the line between the two random points.

4. If the distance from a point to the straight line through the points is *above* a certain threshold distance d, place that point in the *outlier set*. If the distance from a point to the straight line is *below* the threshold distance d, place that point in the *inlier set*.

5. Save the straight line and the set of inlier points for the two random points picked.

6. Repeat the process above K times with the same threshold distance d, picking two new random points each time to construct the straight line through.

7. After repeating the procedure with K different pairs of random points, pick the fit with the largest number of points in the inlier set.

By carrying out this algorithm, we can successfully fit lines to noisy sets of data in a way that *rejects* outliers that are above a certain distance from our fit. This procedure may be generalized to more complex data sets and features! For example, we can use it to estimate quantities as rich as the fundamental matrix when performing 3D scene reconstruction from images.

Questions

1. Let's construct the different pieces of the RANSAC algorithm. We can begin by finding an equation for a straight line between two random points. Suppose we have two points, $(x_1, y_1), (x_2, y_2) \in \mathbb{R}^2$. Show that we can express the straight line between these two points as the following vector function of t:

$$l(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} t \\ m(t - x_1) + y_1 \end{bmatrix} \tag{7}$$

   Where $t \in \mathbb{R}$ is a parametric variable that can have a value between $-\infty$ and ∞ and $m = \frac{y_2 - y_1}{x_2 - x_1}$.

2. The next component we'll need to complete the RANSAC algorithm is a function to compute the minimum distance from a point to a line. Show that the minimum squared distance from any point $p = [p_x, p_y]^T$ to the line $l(t)$ from above is given by:

$$d^2 = \frac{(p_y - y_1 - mp_x + mx_1)^2}{m^2 + 1} \tag{8}$$

   Hint: Find the distance from a point to the line as a function of t and differentiate to find the minimum.

3. Go to the Colab notebook and implement the functions in the *Feature Extraction with RANSAC* module. Tune the number of iterations of RANSAC to get a set of inliers with at least 15 points. Attach the image of the fitted plot generated by the RANSAC code.

Problem 3: Least Squares State Estimation

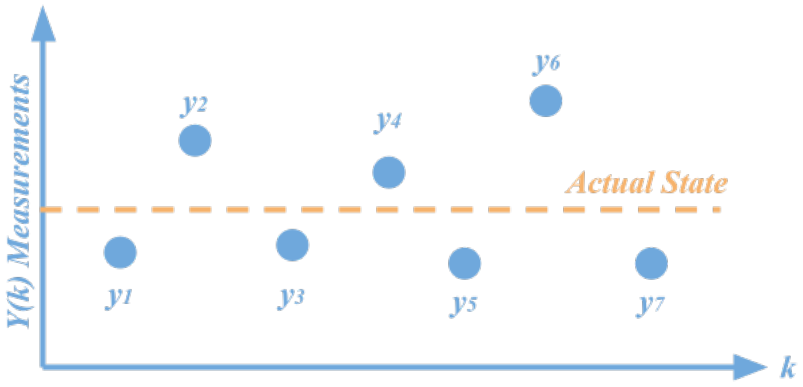
Let's now turn our attention to *state estimation*, the problem of finding the most likely state of a system given a set of noisy measurements. When accounting for random noise in a system, instead of modeling the motion of our robot in time using a *deterministic vector sequence* $x(k) \in \mathbb{R}^n$, like we have so far, we may model the state of the robot using a *random vector sequence*, $X(k) \in \mathbb{R}^n$. Note that for convenience, we'll write $X(k)$ as X_k.
For each time step $k \in \mathbb{Z}$, X_k has a probability distribution that tells us information about how likely our robot is to be in a certain state. To gain estimates of the *most likely* position of our robot given the probability distribution of its state, we may apply the techniques of probability theory. For instance, we may find the expected value of X_k by computing:

$$\mathbb{E}[X_k] = \int_R x p_x(x, k) dx \tag{9}$$

Where R is the region of possible values of X and $p_x(x, k)$ is the probability density function of the random vector X, which may change with k. To simplify our analysis, we'll assume in this question that the properties of X and our noisy measurements don't depend on k.
When predicting the state of our system from a probability distribution, it'll be important to know how the different quantities within the state vector are related! The auto-covariance of X, defined:[1]

$$\Sigma_{xx} = \mathbb{E}[(X - \mathbb{E}[X])(X - \mathbb{E}[X])^T] \in \mathbb{R}^{n \times n} \tag{10}$$

Is an $n \times n$ matrix that tells us how the elements of the state vector X are related to one another. Just like we can use covariance to express the relationship between the different elements of the state vector at different points in time, we may do the same for a set of noisy measurements of the state vector! Consider the following set of noisy measurements:



The noise in the sensors used to take the measurements may also be described by a probability distribution! We can store all of the noisy measurements we take of our system state in a large random measurement vector, which we call Y.
We define the *cross-covariance* between the state $X \in \mathbb{R}^n$ and the measurements $Y \in \mathbb{R}^m$, which expresses the relationship between our state vector and measurements, as follows:

$$\Sigma_{xy} = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])^T] \in \mathbb{R}^{n \times m} \tag{11}$$

By analyzing the auto and cross-covariances of the state vector X and our random measurement vector Y, we can gain strong estimates of the actual state of our robot from noisy measurements. Let's see how we can take advantage of these noisy measurements to best estimate the state of our system!

[1] Note that auto and cross-covariance may also be defined between different time indices k and j. Here, we consider a simpler case.

Questions

Imagine that we have a car that sits still at some position $x \in \mathbb{R}^n$ on a track. Using a sensor, we repeatedly take measurements $y_1, y_2, ..., y_k$ of the position of this stationary car, where the random vector corresponding to this collection of measurements is called Y.
Assuming the properties of X and Y remain constant in time[2], let's try and find the best possible estimate of the car's position using the probabilistic properties of these measurements.

1. Let's begin by trying to estimate the position using a vector of measurements, $y = [y_1, y_2, ..., y_k]$. We want to find an estimate z that minimizes:

$$\mathbb{E}[||X - z||^2 | Y = y] \tag{12}$$

   This is the expected value of the squared error between z and the random variable X *given* that we've taken measurements $y = [y_1, y_2, ..., y_k]$ of our system. If $z \in \mathbb{R}^n$ is a deterministic variable, prove the following statement:

$$\mathbb{E}[||X - z||^2 | Y = y] = \mathbb{E}[X^T X | Y = y] - 2z^T \mathbb{E}[X | Y = y] + z^T z \tag{13}$$

   Hint: Try expanding the squared norm and apply linearity of expectation. Remember that if z is a deterministic variable, $\mathbb{E}[z|Y = y] = z$.

2. Now, we're ready to find the value of z that minimizes the expected error! Prove that:

$$\mathbb{E}[||X - z||^2 | Y = y] \tag{14}$$

   Reaches a minimum for the following value of z:

$$z = \mathbb{E}[X | Y = y] \tag{15}$$

   This gives us an estimate of the state vector that minimizes the expected least squares error given a set of measurements! *Hint: Try taking the derivative of the answer to the previous part with respect to z to find the minimum of the function!*

3. The optimal value we just solved for is called the *conditional expectation* of X given a measurement y. When X and Y are Gaussian random vectors, it can be shown that the conditional expectation may be computed:

$$\mathbb{E}[X | Y = y] = \mathbb{E}[X] + \Sigma_{xy} \Sigma_{yy}^{-1} (y - \mathbb{E}[Y]) \tag{16}$$

   Go to the *Least Squares State Estimation* module in the notebook and use this expression to gain an optimal least squares estimate for the position of the vehicle. Attach the plot provided by the code to your solution.

[2] More formally, we will assume X and Y are stationary random processes.

Problem 4: The Kalman Filter

Let's build upon our results from the previous section to develop a method for optimally estimating the state of systems which *move*. In this case, we'll focus on systems of the form:

$$x_{k+1} = Ax_k + Bu_k + w_k \tag{17}$$

Where $w_k \in \mathbb{R}^n$ is a vector called the *process noise*. The process noise w_k represents a random disturbance applied to the system at a timestep k. In addition to having noise in the process itself, we have noise in our sensors. We can model this noise with the following output equation:

$$y_k = Cx_k + v_k \tag{18}$$

Where $v_k \in \mathbb{R}^m$ is the *measurement noise* and $y_k \in \mathbb{R}^m$ is the measurement of the system taken by the sensors at timestep k. In our derivations, we'll assume that w_k and v_k are Gaussian random vectors with covariances Σ_w, Σ_v and zero expected value.

$$w(k) \sim \mathcal{N}(0, \Sigma_w),\ v(k) \sim \mathcal{N}(0, \Sigma_v) \tag{19}$$

Let's develop some pieces of the Kalman filter, which may be used to optimally estimate the state of this noisy system as time passes. To write the equations of the Kalman filter in a convenient manner, we'll introduce the following notation for conditional expectation:

$$\mu_{k|j} = \mathbb{E}[X_k | y_0, ..., y_j] \tag{20}$$

This refers to the expected value of the state vector, X_k, given that the measurements $y_0, ..., y_j$ have been taken. We may use similar notation to express the covariance of X_k given that measurements $y_0, ..., y_j$ have been taken:

$$\Sigma_{k|j} = \mathbb{E}[(X_k - \mu_{k|j})(X_k - \mu_{k|j})^T] \tag{21}$$

Let's derive two key components of the Kalman filter and implement the filter in code!

Questions

1. By applying least squares estimation, we may show that we can find an expression for $\mu_{0|0}$, the expected initial state of the system given an initial measurement y_0. Prove that we may compute the expected value of the state at $k = 1$ given the measurement y_0 using:

   $$\mu_{1|0} = \mathbb{E}[X_1 | y_0] = A\mu_{0|0} + Bu_0 \tag{22}$$

   Where $\mu_{0|0} = \mathbb{E}[X_0 | y_0]$ is the expected initial condition given the first measurement and u_0 is the first (deterministic) input we send to the system. *Hint: Can you express X_1 in terms of X_0, u_0 using the system dynamics? Remember to apply linearity of expectation!*

2. Let's see if we can estimate the state covariance at time step 1 given the first measurement. Using the following shorthand notation for covariance:

   $$\Sigma_{0|0} = \mathbb{E}[(X_0 - \mu_{0|0})(X_0 - \mu_{0|0})^T] \tag{23}$$

   $$\Sigma_{1|0} = \mathbb{E}[(X_1 - \mu_{1|0})(X_1 - \mu_{1|0})^T] \tag{24}$$

   If X_0 and w_k are uncorrelated, which means that $\mathbb{E}[(X_0 - \mu_{0|0})w_k^T] = \mathbb{E}[w_k(X_0 - \mu_{0|0})^T] = 0$, prove that $\Sigma_{1|0}$ may be calculated by:

   $$\Sigma_{1|0} = A\Sigma_{0|0}A^T + \Sigma_w \tag{25}$$

   Where $\Sigma_w = \mathbb{E}[w_k w_k^T]$ is the auto-covariance of the process noise. *Hint: Use the expression for $\mu_{1|0}$ from the previous part. Try expanding the expression inside the expectation.*

3. By *recursively* applying the two equations above at each time step, we may we may show that the following equations:

$$\mu_{k+1|k} = A\mu_{k|k} + Bu_k \tag{26}$$
$$\Sigma_{k+1|k} = A\Sigma_{k|k}A^T + \Sigma_w \tag{27}$$
$$\mu_{k+1|k+1} = \mu_{k+1|k} + \Sigma_{k+1|k}C^T(C\Sigma_{k+1|k}C^T + \Sigma_v)^{-1}(y_{k+1} - C\mu_{k+1|k}) \tag{28}$$
$$\Sigma_{k+1|k+1} = \Sigma_{k+1|k} - \Sigma_{k+1|k}C^T(C\Sigma_{k+1|k}C^T + \Sigma_v)^{-1}C\Sigma_{k+1|k} \tag{29}$$

Give us the best possible estimate of the state vector, $\mu_{k|k}$, for a linear system with Gaussian noise! These equations form the famous *Kalman filter*.[3] Go to the notebook and implement the *Kalman Filtering* module. Attach the plots generated by the code to your solution.

[3]Note that the second two equations may be derived using some more advanced results in random processes.