Name: _____ **UID:** _____

CS188: Introduction to Robotics

Worksheet 2

Problem 1: Image Filters

(a) What will the following convolution kernels do when they are used on an image?

$$\begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

(b) What is gamma correction? Why do we need that?

Solution 1: Image Filters

(a) Convolution Kernels

1. **Vertical Edge Detector (Prewitt Operator):**

$$\begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}$$

This kernel approximates the gradient in the x-direction ($\frac{\partial I}{\partial x}$). It responds strongly to vertical edges (transitions from dark to light horizontally).

2. **Sharpening Filter:**

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

This kernel subtracts the average of the neighbors from the center pixel (scaled). It enhances the contrast around edges, making the image appear sharper. It can be viewed as $2 \times$ Identity $-$ Box Blur.

3. **Box Blur / Smoothing Filter:**

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

   This is an unnormalized box filter. When multiplied by $\frac{1}{9}$, it calculates the average of the 3×3 neighborhood. It suppresses noise but blurs the image (low-pass filter).

4. **Horizontal Edge Detector (Sobel Operator):**

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

   This kernel approximates the gradient in the y-direction ($\frac{\partial I}{\partial y}$) with some Gaussian smoothing. It responds strongly to horizontal edges.

(b) Gamma Correction

Definition: Gamma correction is a non-linear operation used to encode and decode luminance or tristimulus values in video or still image systems. It is defined by the power-law expression:

$$V_{\text{out}} = AV_{\text{in}}^{\gamma}$$

where $\gamma < 1$ is usually used for encoding (compression) and $\gamma > 1$ for decoding (expansion).

Why do we need it?

1. **Display Characteristics (Historical):** Legacy CRT monitors had a non-linear relationship between input voltage and output brightness (approximately $\gamma \approx 2.2$). Gamma correction was originally required to pre-compensate the signal so the displayed image would look linear to the viewer.

Problem 2: CNN

Conceptual

In a neural network (say, for image recognition), why do we use convolutional filters? What advantages do they have over just using fully-connected layers?

Solution. This is an open-ended question and there are a variety of answers. There are two main reasons that I can think of:

1. Convolution as an operation operates over small neighborhoods, since the kernel weights only apply to a small neighborhood. We know that manually-designed filters can capture important features of an image, such as edges and corners. Learning these weights can find even more expressive features for the purposes of the model. You can read these wonderful blog posts by Chris Colah to learn more (1, 2) if you're curious.
2. Another reason is that the same set of weights are applied at every pixel (for an image) location, so the model needs fewer parameters than an equivalent fully-connected layer.

On the other hand, fully-connected layers largely ignore these properties; they treat every pixel as unrelated and require more parameters.

Computational

Given a convolutional layer with the following configuration: an **input feature map** size of 32×32, a **kernel size** of 5×5, a **stride** of 1 and a **padding** of 0, what is the spatial size of the output feature map?

Solution. The output size of a convolution is calculated as:

$$\text{Output size} = \left\lfloor \frac{N - K + 2P}{S} \right\rfloor + 1$$

And we have

- N = input size (32)
- K = kernel size (5)
- P = padding (0)
- S = stride (1)

This gives us the following result

$$\left\lfloor \frac{32 - 5 + 0}{1} \right\rfloor + 1 = 28$$

Problem 3: PF vs. KF

Conceptual

- Why do we use particle filters and Kalman filters in robotics? Alternatively, how, or where do we use them?
- When might we want to use a Kalman filter over a particle filter?
- When might we want to use a particle filter over a Kalman filter?

Solution.

- Both algorithms perform *state estimation*: they take noisy sensor data, some process model, and use this to estimate a robot's pose, velocity, etc. over time.
- When the system dynamics and observation models are approximately *linear* and the noise is *Gaussian*. This is when Kalman filters are optimal for the system.
- When the motion or sensor models are *non-linear* and/or *non-Gaussian*. PFs approximate the posterior with weighted samples, so they can represent *arbitrary distributions*. However, this is at the cost of increased computation.

Problem 4: SLAM

Conceptual

What makes the *Simultaneous Localization and Mapping* (SLAM) problem difficult? Describe a challenge that can arise when a robot tries to build a map of an unknown environment while simultaneously localizing itself within that map.

Solution. This is another open-ended questions with a lot of answers. When designing this question, the main answer I was thinking of was the chicken-and-egg nature of the SLAM problem: to localize accurately, the robot needs a good map, but to build a good map, it needs to know its location accurately. This interdependence means estimation errors can feed on each other. For example, if the robot's pose estimate drifts, it will place new landmarks in the wrong locations on the map; conversely, an incorrect map can mislead the localization. Of course, however, we have several known algorithms (KF, PF) to estimate these equations tractably.

Multiple Choice

In a SLAM system, why is it important to account for both the robot's motion and the sensor's measurement noise when constructing the map?

A) To improve the map resolution and reduce the computational cost of mapping

B) To minimize the effect of drift and inaccuracies in the robot's localization over time

C) To ensure the robot can detect obstacles in unknown environments

D) To increase the speed at which the map is generated

Solution. B

Problem 5: PRM & RRT

Conceptual

Consider the following conceptual questions.

- Describe, in words, how the PRM (Probabilistic Roadmap) algorithm finds a path from *start* to *goal*.
- What aspect of the RRT (Rapidly-Exploring Random Tree) algorithm allows it to efficiently search high-dimensional spaces for a path? Said another way, why do grid-based approaches perform poorly in high-dimensional spaces?

Solution.

- This is a general question, but a reasonably short answer might look like the following: PRM constructs a graph (roadmap) of the free space by randomly sampling a large number of collision-free configurations in the environment and connecting nearby samples with collision-free edges. Once this roadmap is built, the algorithm searches the graph for a path connecting the start configuration to the goal configuration (i.e., using a shortest path search), resulting in a feasible path through the free space if one exists. This answer is, for the most part, stolen from the Wikipedia page.
- This is a more difficult question. RRT efficiently explores high-dimensional spaces by using *random sampling* to bias the tree's growth towards unexplored regions. In a high-dimensional space, the volume grows exponentially with each additional dimension. So, a grid-based approach or algorithm will have an input which is exponential, exploding its runtime and essentially making the problem intractable for it. On the other hand, random sampling is cheap in comparison. The reason random sampling works well is a bit more complex, and I will refer to the wikipedia page if you want to learn more.

Problem 6 EKF SLAM

Part A: Conceptual

Q2. Evolution of Uncertainty

1. **Prediction (Motion):**
   - **Robot-Robot Block: Changes (Increases).** We add motion noise R_t to the robot's uncertainty.
   - **Robot-Map Block: Changes.** The robot moves, so the geometric relationship (correlations) between the robot and the map transforms.
   - **Map-Map Block: No Change.** The motion of the robot does not affect the relative uncertainty between static landmarks in the map.
2. **Correction (Observation):**
   - **All Blocks: Change (Decrease).** Observing a known landmark provides information about the robot's location. Because the robot pose is correlated with the map (via the off-diagonal terms), correcting the robot pose also updates the estimates for all landmarks in the map.

Part B: 2D Numerical Update

Step 1: Motion (Prediction)

1. Predicted Mean $\bar{\mu}_t$: The motion model is $x' = x + d\cos\theta, y' = y + d\sin\theta, \theta' = \theta$. Given $\mu_{t-1} = [0, 0, 0]^T$ and control $u = [2, 0]^T$:

$$\bar{\mu}_t = \mu_{t-1} + \begin{bmatrix} 2\cos(0) \\ 2\sin(0) \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 10 \\ 0 \end{bmatrix}$$

Explanation: The robot moves 2 meters along the x-axis. The landmark coordinates (indices 4 and 5) do not change during prediction.

2. Predicted Covariance $\bar{\Sigma}_t$: Since the motion is linear (pure translation) and $\theta = 0$, the Jacobian G_t is the Identity matrix. The update is simply adding the process noise R_t to the robot's top-left block.

$$\bar{\Sigma}_t = \Sigma_{t-1} + F_x^T R_t F_x = \left[\begin{array}{ccc|cc} 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0.01 & 0 & 0 \\ \hline 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \end{array}\right]$$

Explanation: The robot becomes uncertain due to motion noise. The map uncertainty remains 0.5 (from initialization). The cross-covariance remains 0 because we started with 0 and the motion was aligned with the axes.

Step 2: Measurement (Correction)

1. Expected Measurement $\hat{z}_t$: We calculate what the sensor *should* see based on the predicted state $\bar{\mu}_t$.

- Robot position: $(2, 0)$
- Landmark position: $(10, 0)$
- $\Delta x = 10 - 2 = 8$
- $\Delta y = 0 - 0 = 0$

$$\hat{z}_t = \begin{bmatrix} \sqrt{8^2 + 0^2} \\ \tan(0/8) - 0 \end{bmatrix} = \begin{bmatrix} 8 \\ 0 \end{bmatrix}$$

Explanation: We expect the landmark to be 8 meters directly ahead.

Step 3: Adding New Landmarks
If a new landmark is initialized:

1. **State Vector μ:** The size increases by 2 rows ($N = 1 \rightarrow N = 2$).

   $$\text{Old Size: } 5 \times 1 \quad \rightarrow \quad \text{New Size: } 7 \times 1$$

   The new entries will be the estimated global coordinates $(6, 12)$.

2. **Covariance Matrix Σ:** The matrix expands to include the variance of the new landmark and its covariance with the robot and existing landmarks.

   $$\text{Old Size: } 5 \times 5 \quad \rightarrow \quad \text{New Size: } 7 \times 7$$

3. **Initialization:** The new diagonal block (variance of the new landmark) is initialized based on the robot's current uncertainty plus the measurement noise (Q_t).