6.801/6.866: Machine Vision, Lecture 15

Professor Berthold Horn, Ryan Sander, Tadayuki Yoshitake
MIT Department of Electrical Engineering and Computer Science
Fall 2020

These lecture summaries are designed to be a review of the lecture. Though I do my best to include all main topics from the lecture, the lectures will have more elaborated explanations than these notes.

1 Lecture 15: Alignment, recognition in PatMAx, distance field, filtering and sub-sampling (US 7,065,262)

In this lecture, we will discuss another patent on the topic of object inspection and pose estimation, known as PatMAx. We will then look at computing distance to lines as a means to perform better edge detection, and then will investigate the role of sparse convolution for multiscale systems that perform filtering.

1.1 PatMAx

Another patent we will look at for object inspection is PatMAx.

1.1.1 Overview

Some introductory notes on this:

- This framework builds off of the previous PatQuick patent.
- This framework, unlike PatQuick, does not perform quantization of the pose space, which is one key factor in enabling sub-pixel accuracy.
- PatMAx assumes we already have an approximate initial estimate of the pose.
- PatMAx relies on an iterative process for optimizing energy, and each **attraction step** improves the fit of the configuration.
- Another motivation for the name of this patent is based off of electrostatic components, namely dipoles, from Maxwell. As it turns out, however, this analogy works better with mechanical springs than with electrostatic dipoles.
- PatMAx performs an **iterative attraction process** to obtain an estimate of the pose.
- An iterative approach (e.g. gradient descent, Gauss-Newton, Levenberg-Marquadt) is taken because we likely will not have a closed-form solution in the real world. Rather than solving for a closed-form solution, we will run this iterative optimization procedure until we reach convergence.
- Relating this framework back to PatQuick, PatMAx can be run after PatQuick computes an initial pose estimate, which we can then refine using PatMAx. In fact, we can view our patent workflow as:

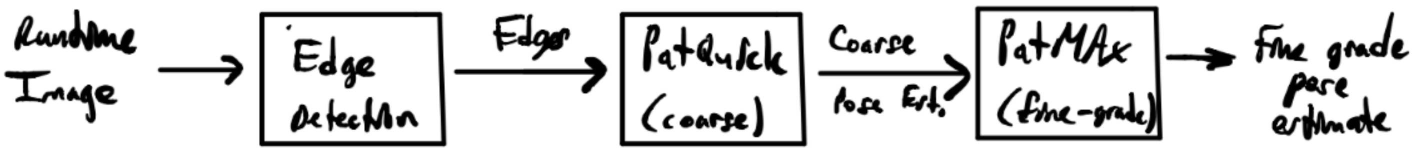

Figure 1: An overview of how the patents we have looked at for object inspection fit together.

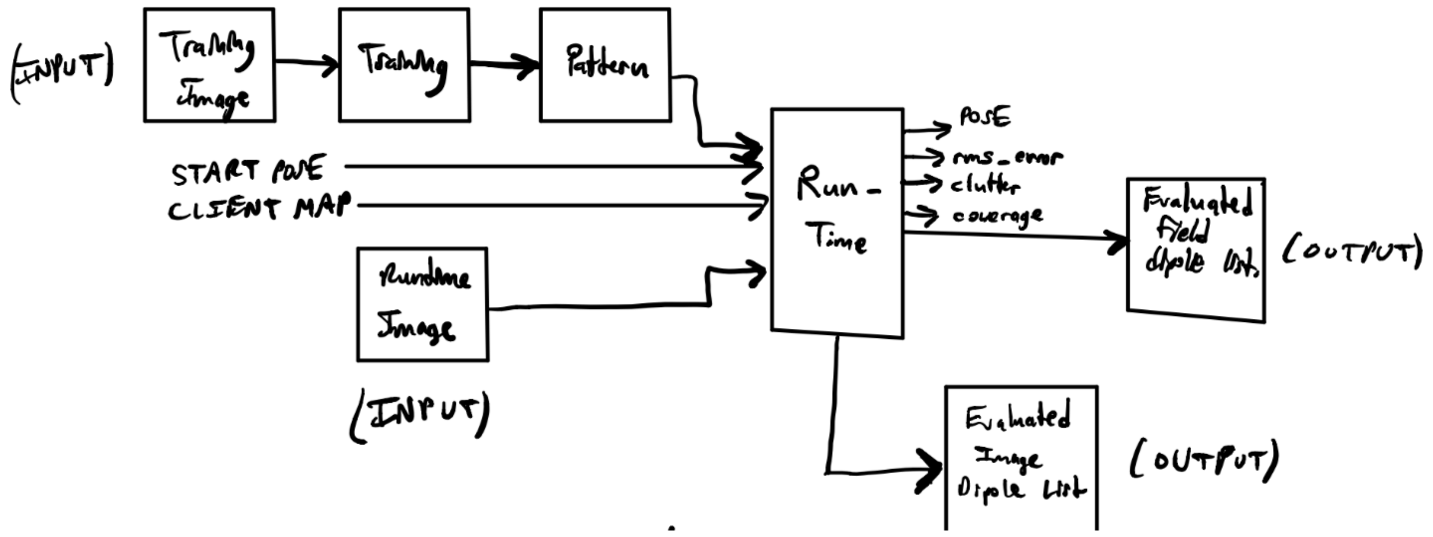

Figure 2: High-level diagram of the PatMAx system.

A diagram of the system can be found here: Now that we have a high-level overview, we are now ready to dive more into the specifics of the system.

1.1.2 Training PatMAx

: The training process can be classified as three distinct steps:

1. We begin with edge detection, which produces a **field dipole list** (essentially the probe points from the PatQuick patent framework).
2. Training also produces a **field**. We compare runtime features with template features and determine the attraction of these features between the images using this field as a vector field.*
3. We map the feature-detected runtime image's features back to the field (this is more computationally-efficient than mapping the field to the runtime image).

*For field generation, we can in turn discuss the steps needed to generate such a field:

1. Initialize
2. Seed
3. Connect
4. Chain
5. Filter
6. Segment
7. Propagate

Many of the steps outlined in this field generation process were also leveraged in the PatQuick method.

Another important aspect of training is **computing field dipoles**. A few notes on this:

- Field dipoles correspond to edge fragments.
- Field dipoles are created as a data structure of flags that provide information about proximity to other components, such as the edge.

Some other notes on this framework:

- Edge detection is largely the same procedure that we have seen in the previous patents (e.g. PatQuick). However, note that because this framework seeks to obtain highly-precise estimates accurate to the sub-pixel level, PatMAx does not use CORDIC or quantized gradient directions.
- **Field dipoles** are computed during training.
- The chaining procedure used in PatMAx is similar to the process we saw before: (i) Link chains, and then (ii) Remove short (weak) chains.
- For **initialization**, the array contains a vector field, but the vectors do not cover the entire array.

We will now explore some specific elements of this framework:

1.1.3 Estimating Other Pixels

- To estimate other pixels, we need to fill in pixels near the edge in an iterative fashion.
- To accomplish this, PatMAx uses a **distance map**, which is common in machine vision applications.
- We can compute the distance to the edge accurately with Manhattan distance, but we use Euclidean distance, which is non-trivial to compute, particularly, as we will see shortly, for corner edges.
- Intuitively, we want the system to be drawn to a "lower energy state", hence the idea of this algorithm being an energy minimization algorithm.
- Identical copies can be resolved via averaging.

1.1.4 Attraction Module

The diagram of the attraction module is given below: **Intuition with Mechanical Springs:** Scaling adjustments via scaled

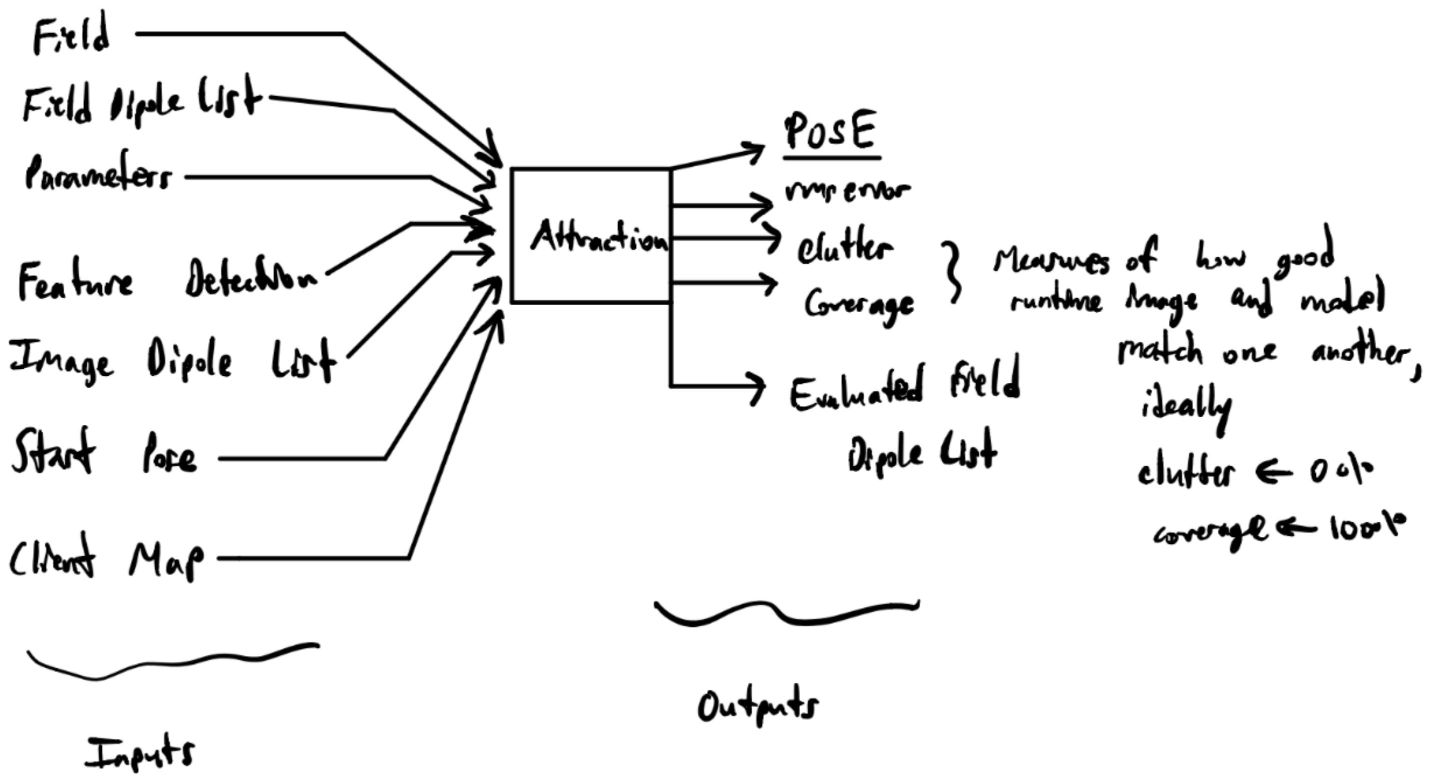

Figure 3: The Attraction module for the PatMAx system. Note that this produces a refined estimate of the pose at the output, which is one of the main goals of the PatMAx system.

transformations can be conceptualized as a set of mechanical springs (rather than electrostatic dipoles) that are adjusted until an optimal configuration of the degrees of freedom is found.

Solving this system is equivalent to solving a large **least squares problem**:

- Each runtime dipole has force exerted on it in a certain direction. The goal here is to use diffent movements, which comprise our Degrees of Freedom, to create a configuration of these dipoles that minimizes the overall energy/tension of this system.
- The movements/degrees of freedom that are allowed: (i) Translation in 2D (2 DOF), (ii) Rotation in 2D (1 DOF), (iii) Scaling (1 DOF). Therefore with these admissable movements we have 4 degrees of freedom.
- A closed-form solution of this does not exist, but we can compute a solution to this least squares problem using an **upper triangular matrix accumulator array**. This array is scaled additionally by weights, and can also be used to compute image moments.
- With our movements (translation, rotation, and scaling), we have 4 DOF. With a full set of movements comprising affine linear transformations, we have 6 DOF.
- Local linearization around the operating point is also used when solving for the least squares solution to this problem.
- One computational technique the authors of this patent make use of is the use of **doubly linked lists** for the image dipoles.

1.1.5 PatMAx Claims

As we have seen, another good way to get a sense of the "abstract" of a patent is to look through its claims. The big claim of PatMAx: PatMAx is a geometric pattern-matching method used for iteratively refining the estimte of the true pose (relative to the orientation of the object in the training image) in a runtime image.

1.1.6 Comparing PatMAx to PatQuick

To better understand these frameworks (and how they potentially fit together for cascaded object inspection, let us draw some comparisons between PatQuick and PatMAx):

- PatQuick searched all pose space and does not require an initial guess - PatMAx does require an initial estimate/guess of the pose in order to produce a refined estimate.
- For PatMAx, there is repeated emphasis on avoiding techniques such as thresholding and quantization of gradient directions that have been used in the previous patents we have looked at. This makes sense, since PatMAx aims to output a more refined estimate of the pose than these other frameworks (i.e. reach sub-pixel accuracy).
- Using "dipoles" for PatMAx is misguided - using physical springs as an analog is much more physiclly consistent.
- For PatMAx, we use evidence collected for determining the quality of alignment, which in turn determines the quality of our refined pose estimate.
- PatMAx and PatQuick each have different methods for assigning weights.
- PatMAx is a nonlinear optimization problem, and therefore does not have a closed-form solution. PatMAx is also iterative - alignment quality and matched edges get closer with more iterations of optimization.

1.1.7 Field Generation for PatMAx

Here, we look at the field generation. For building intuition and simplifying, we will only take into account distance. See the figure below for some examples of distance fields as circles, lines, and (hardest to compute) corner edges. A few notes about these:

- If we were working with **Manhattan** (L1-norm) distance rather than **Euclidean** (L2-norm) distance, this would be a much easier problem to solve, especially for edge corners. However, unfortunately this is not an option we have.
- We can weight the forces of **individual dipoles** in the runtime image. Weight is computed for beliefs/evidence for (i) **forces**, (ii) **torques**, and (iii) **scaling**, where $\{w_i\}_{i=1}^N$ is the set of weights:

  1. **Forces (Translation)**: $\mathbf{F} = \frac{\sum_{i=1}^N \mathbf{F}_i w_i}{\sum_{i=1}^N w_i} \in \mathbb{R}^2$
  2. **Torques (Rotation)**: $\tau = \frac{\sum_{i=1}^N w_i(\mathbf{r}_i \times \mathbf{F}_i)}{\sum_{i=1}^N w_i} \in \mathbb{R}$, where r_i is the radial vector
  3. **Scaling**: $s = \frac{\sum_{i=1}^N w_i(\mathbf{r}_i \cdot \mathbf{F}_i)}{\sum_{i=1}^N w_i} \in \mathbb{R}$ where r_i is the radial vector

  Together, these three elements composed of weighted evidence from the dipoles compose our 4 DOF.

Distance Field:

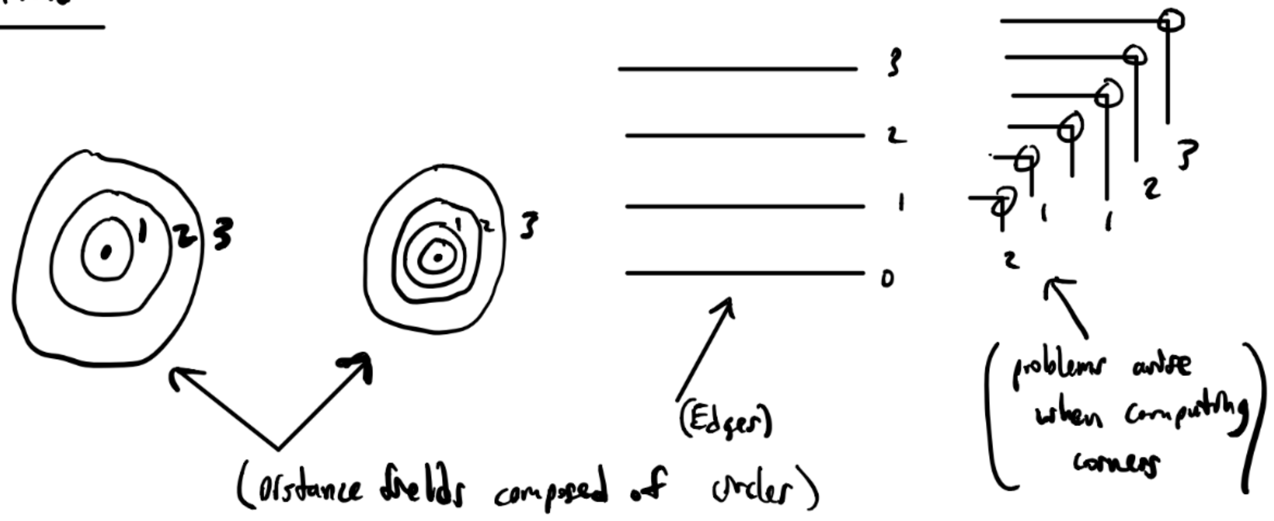
1
2
3
0
(Edges)
(Distance fields composed of circles)
(problems arise when computing corners)

Figure 4: Examples of distance fields.

1.2 Finding Distance to Lines

One application of performing this task is to improve the performance of edge detection systems by combining shorter edge fragments of objects into longer edge fragments.

For this procedure, we break this down into two steps:

1. Rotate the 2D cartesian coordinate system into a system that is parallel to the line of interest:

$$\begin{aligned} x' &= x\cos\theta + y\sin\theta \\ y' &= -x\sin\theta + y\cos\theta \\ \text{I.e.} \begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \end{aligned}$$

2. Translate the origin to the new location to match the line:

$$\begin{aligned} x'' &= x' \\ y'' &= y' - \rho \\ &= -x\sin\theta + y\cos\theta - \rho \end{aligned}$$

   Together, these equations imply:

$$\begin{aligned} y'' + x\sin\theta - y\cos\theta + \rho = 0 \\ y'' + x''\sin\theta - y\cos\theta + \rho = 0 \end{aligned}$$

   Which in turn has the properties:

   - There are no redundant solutions
   - The system is parameterized by (ρ, θ)
   - There are no singularities

From here, we can use the above framework to find a line by minimizing the following objective over our parameterized degrees of freedom ρ and θ:

$$\begin{aligned}\rho^*, \theta^* &= \arg\min_{\rho,\theta} \sum_{i=1}^{N} (y_i^{''})^2 \\ &= \arg\min_{\rho,\theta} \sum_{i=1}^{N} (x_i \sin\theta - y_i \cos\theta + \rho)^2 \\ &\triangleq \arg\min_{\rho,\theta} J(\rho, \theta)\end{aligned}$$

This problem can be solved through our standard calculus approaches of finding the first-order conditions of our objective $J(\rho, \theta)$ on our degrees of freedom ρ and θ. Since we have two degrees of freedom, we have two First-Order Conditions:

1. $\frac{\partial J(\rho,\theta)}{\partial \rho} = 0$:

$$\begin{aligned}\frac{\partial}{\partial \rho}(J(\rho,\theta)) &= \frac{\partial}{\partial \rho}\Big(\sum_{i=1}^{N}(x_i \sin\theta - y_i \cos\theta + \rho)^2\Big) \\ &= 2\sum_{i=1}^{N}(x_i \sin\theta - y_i \cos\theta + \rho) = 0 \\ &= \sin\theta\Big(\sum_{i=1}^{N} x_i\Big) - \cos\theta\Big(\sum_{i=1}^{N} y_i\Big) + \Big(\sum_{i=1}^{N} \rho\Big) = 0 \\ &= N\bar{x}\sin\theta - N\bar{y}\cos\theta + N\rho = 0 \\ &= \bar{x}\sin\theta - \bar{y}\cos\theta + \rho = 0\end{aligned}$$

   (Where $\bar{x} \triangleq \frac{1}{N}\sum_{i=1}^{N} x_i$ and $\bar{y} \triangleq \frac{1}{N}\sum_{i=1}^{N} y_i$.)

   Though this does not give us the final answer, it does provide information on how our solution is constrained, i.e. the line must pass through the centroid given by the mean $(\bar{x}, \bar{y})$. Let us now look at the second FOC to combine insights from that FOC with this FOC in order to obtain our solution.

2. $\frac{\partial J(\rho,\theta)}{\partial \theta} = 0$:

   Before computing this derivative, let us move our coordinates to the centroid, i.e. subtract the mean:

$$\begin{aligned}x_i^{'} = x_i - \bar{x} \longrightarrow x_i = \bar{x} + x_i^{'} \\ y_i^{'} = y_i - \bar{y} \longrightarrow y_i = \bar{y} + y_i^{'}\end{aligned}$$

   Plugging this substituted definition into our equations renders them such that the centroid cancels out. Let us now compute the second FOC:

$$\begin{aligned}\frac{\partial}{\partial \theta}(J(\rho,\theta)) &= \frac{\partial}{\partial \theta}\Big(\sum_{i=1}^{N}(x_i \sin\theta - y_i \cos\theta + \rho)^2\Big) \\ &= 2\sum_{i=1}^{N}(x_i \sin\theta - y_i \cos\theta + \rho)(x' \cos\theta + y' \sin\theta) = 0 \\ &= \sum_{i=1}^{N} x'^2 \sin\theta\cos\theta + x'y'\sin^2\theta - x'y'\cos^2\theta - y'^2\cos\theta\sin\theta) = 0 \\ &= \sum_{i=1}^{N}(x_i^2 - y_i^2)\sin\theta\cos\theta = \sum_{i=1}^{N} x_i y_i(\cos^2\theta - \sin^2\theta) = 0 \\ &= \frac{1}{2}\sum_{i=1}^{N}(x_i^2 - y_i^2)\sin(2\theta) = \sum_{i=1}^{N} x_i y_i \cos(2\theta) = 0 \\ &= \frac{\sin(2\theta)}{\cos(2\theta)} = \tan(2\theta) = \frac{2\sum_{i=1}^{N} x_i y_i}{\sum_{i=1}^{N}(x_i^2 - y_i^2)}\end{aligned}$$

A few notes about this:

- In the second-to-last step, we used the two following trigonometric identities: (i) $\sin\theta\cos\theta = \sin(2\theta)$, and (ii) $\cos^2\theta - \sin^2\theta = \cos(2\theta)$.
- Notice that we can separate the angle θ from the sums because it does not depend on the sum index and is a degree of freedom in this optimization problem.

From here, we can solve for the optimal value of θ (which will also constrain and give us an optimal value of ρ) by taking the arctangent that returns quadrant ("atan2"):

$$\theta^* = \frac{1}{2}\arctan 2\Big(2\sum_{i=1}^{N} x_i y_i, \sum_{i=1}^{N}(x_i^2 - y_i^2)\Big)$$

Therefore, solving the FOCs gives us a closed-form least squares estimate of this line parameterized by (ρ, θ). This solution, unlike the Cartesian $y = mx + b$ fitting of a line, is independent of the chosen coordinate system, allowing for further flexibility and generalizability.

1.3 Fast Convolutions Through Sparsity

Next, we will switch gears and revisit multiscale, which is a common procedure needed in machine vision. Multiscale motivates the need of filtering methods that are computationally-agnostic to the scale or resolution being used. Since filtering is just convolution, this motivates the exploration of another patent, namely on *fast convolutions.* The patent we explore is **"Efficient Flexible Digital Filtering, US 6.457,032**.

The goal of this system is to efficiently compute filters for multiscale. For this, we assume the form of an N$^{\text{th}}$-order piecewise polynomial, i.e. a N$^{\text{th}}$-order spline.

1.3.1 System Overview

The block diagram of this system can be found below: A few notes on this system:

- Why is it of interest, if we have N$^{\text{th}}$-order splines as our functions, to take N$^{\text{th}}$-order differences? The reason for this is that the differences create **sparsity**, which is critical for fast and efficient convolution. Sparsity is ensured because:

  $$\frac{d^{N+1}}{dx^{N+1}} f(x) = 0 \;\forall\, x \text{ if } f(x) = \sum_{i=0}^{N} a_i x^i, a_i \in \mathbb{R} \;\;\forall\, i \in \{1, \cdots, N\}$$

  (I.e, if $f(x)$ is a order-N polynomial, then the order-(N+1) difference will be 0 for all x.

  This sparse structure makes convolutions much easier and more efficient to compute by reducing the size/cardinality of the support (we will discuss what a support is in greater detail in the next lecture, as well as how the size of a support affects computational efficiency, but effectively the support is the subset of the domain of a function that is not mapped to zero).

- Why do we apply an order-(N+1) summing operator? We apply this because we need to "invert" the effects of the order-(N+1) difference. Intuitively, this makes sense that the order-(N+1) difference and the order-(N+1) sum commute, because we are simply performing iterative rounds of subtraction and addition (respectively), which we know are commutative algebraic operations. I.e, representing differencing and summing as linear operators where their order is the same, we have:

$$\begin{aligned}
&\textbf{First Order}: \; \mathcal{D}\mathcal{S} = I \\
&\textbf{Second Order}: \; \mathcal{D}\mathcal{D}\mathcal{S}\mathcal{S} = \mathcal{D}\mathcal{S}\mathcal{D}\mathcal{S} = (\mathcal{D}\mathcal{S})(\mathcal{D}\mathcal{S}) = II = I \\
&\vdots \\
&\textbf{Order K}: \; (\mathcal{D})^K(\mathcal{S})^K = (\mathcal{D}\mathcal{S})^K = I^K = I
\end{aligned}$$

1.3.2 Integration and Differentiation as Convolutions

Conceptualizing these differencing/differentiation and summing/integration as linear operators that are commutative and associative, we can then extend this framework to conceptualizing these operators as convolutions:

- **Integration**: This corresponds to the convolution of our piecewise polynomial $f(x)$ with a unit step function $u(x)$.

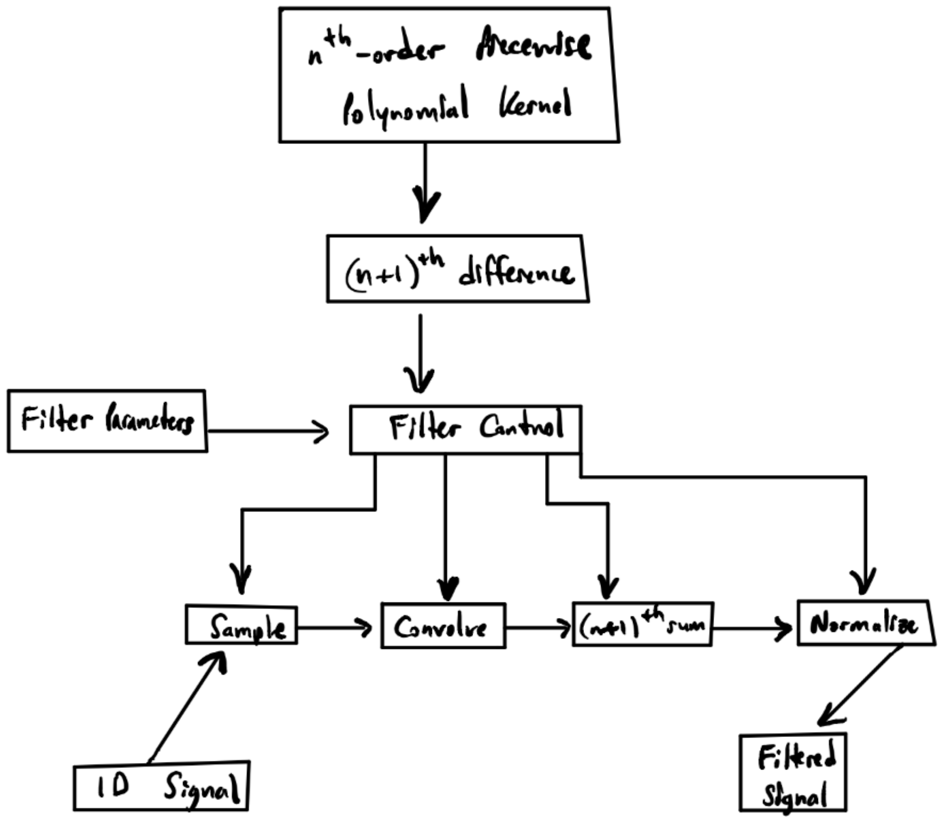

Figure 5: Block diagram of this sparse/fast convolution framework for digital filtering. Note that this can be viewed as a compression problem, in which differencing compresses the signal, and summing decompresses the signal.

- **Differentiation**: This corresponds to the convolution of our piecewise polynomial $f(x)$ with two scaled impulses in opposite directions: $\frac{1}{2}\delta(x+\frac{\epsilon}{2})+\frac{1}{2}\delta(x-\frac{\epsilon}{2})$.

This motivates the discussion of some of the properties of convolution this system relies on in order to achieve high performance. For operators $\mathcal{A}, \mathcal{B}$, and $\mathcal{C}$, we have that:

1. **Commutativity**: $\mathcal{A} \otimes \mathcal{B} = \mathcal{B} \otimes \mathcal{A}$
2. **Associativity**: $\mathcal{A} \otimes (\mathcal{B} \otimes \mathcal{C}) = (\mathcal{A} \otimes \mathcal{B}) \otimes \mathcal{C}$

These properties stem from the fact that in the Fourier domain, convolution is simply multiplication. Therefore, convolution obeys all the algebraic properties of multiplication.

1.3.3 Sparse Convolution as Compression

As aforementioned, another way to consider this efficient convolution system is as a compression mechanism, in which the **differencing** operation acts as a compression mechanism, and the **summing** operation acts as a decompression mechanism. We can contrast this with standard convolution in the block diagram below. This sparse convolution approach ends up being a much more efficient way to filter a signal.

1.3.4 Effects on Scaling

This fast/efficient convolution framework yields desirable results for scaling, and, as we will see shortly, for problems in which multiscale approaches are necessary. Because the only places in which we need to do work are now the locations where the piecewise polynomial segments are stitched together, the scaling can be changed to coarser and finer levels without affecting the amount of computation that is required! This improves the efficiency of multiscale approaches.

1.3.5 Filtering (For Multiscale): Anti-Aliasing

We have now reduced the amount of computation needed to compute efficient digital filtering. We now only need final ingredient for multiscale that is motivated by Shannon and Nyquist: anti-aliasing methods (filters).

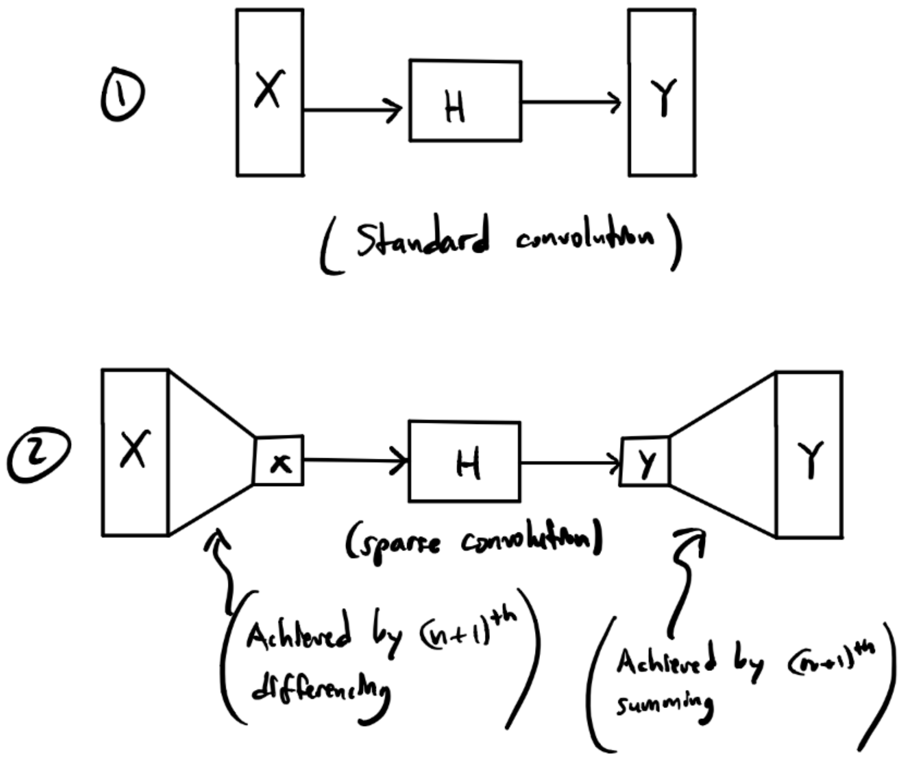
1
X
H
Y
(Standard convolution)
2
X
x
H
y
Y
(sparse convolution)
(Achieved by (n+1)th differencing)
(Achieved by (n+1)th summing)

Figure 6: Comparison of standard filtering and efficient/sparse filtering procedures, where the sparse filtering approach is illustrated as a compression problem. Here, H represents the filter, X and Y represent the uncompressed inputs and outputs, respectively, and x and y represent the compressed inputs and outputs.

Recall from Shannon/Nyquist (the Sampling Theorem) that in order to sample (for instance, when we subsample in multi-scale problems) without aliasing and high-frequency artifacts, it is critical that we first remove high-frequency components from the signal we are sampling. This high-frequency component removal can be achieved with approximate low pass filtering (which we will cover in greater detail during the next lecture).

We will see in the next lecture that one way we can achieve approximate low pass filtering is by approximating a spatial sinc function (which transforms into an ideal low pass filter in the frequency domain) as a spline.

1.3.6 Extending Filtering to 2D and An Open Research Problem

This framework is built for 1D, but we note that we can extend it to 2D simply by approximating 2D convolution as a cascaded set of 1D convolutions, if we are to continue using this sparse convolution mechanism. This requires some additional run-time memory as we must store an image corresponding to the 1D convolution of the image with a 1D filter, but this allows us to continue using this efficient sparse convolution structure.

One open research problem: how can we extend this sparse convolution structure to 2D?

Finally, we will finish this lecture with a fun fact on **calcite crystals**. Calcite crystals are a type of **birefringent material**, which means that they have two indices of refraction that depend on two polarizations (one in the x-direction and one in the y-direction), and therefore reflect light into two different ways. As we will see in the next lecture, adding birefringent lenses in the analog domain can prevent aliasing affects from occurring that would otherwise be unavoidable. DSLRs have these birefringent lenses affixed to them for this specific anti-aliasing purpose.