6.801/6.866: Machine Vision, Lecture 12

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 12: Blob Analysis, Binary Image Processing, Use of Green's Theorem, Derivative and Integral as Convolutions

In this lecture, we will continue our discussion of intellectual property, and how it relevant for all scientists and engineers. We will then elaborate on some of the specific machine vision techniques that were used in this patent, as well as introduce some possible extensions that could be applicable for this patent as well.

1.1 Types of Intellectual Property

Though it is not related to the technical content of machine vision, being familiar with different types and degrees of intellectual property (IP) is crucial to professional success. Below, we discuss some of these different types of intellectual property.

- **Patents**: One major type of these is utility and design patents. In these, the authors are required to disclose the "best mode" of performance. For convenience, here are some notes on patents from our previous lecture:
  - Can be thought of as a "contract with society" - you get a limited monopoly on your idea, and in turn, you publish the technical details of your approach.
  - Can help to reduce litigation and legal fees.
  - Can be used by large companies as "ammunition" for "patent wars".

  Some "rules" of patents:
  - No equations are included in the patent (no longer true)
  - No greyscale images - only black and white
  - Arcane grammar is used for legal purposes - "comprises", "apparatus", "method", etc.
  - References of other patents are often included - sometimes these are added by the patent examiner, rather than the patent authors
  - Most patents end with something along the lines of "this is why our invention was necessary" or "this is the technical gap our invention fills"
  - Software is not patentable - companies and engineers get around this by putting code on hardware and patenting the "apparatus" housing the code.
  - It is also common to include background information (similar to related literature in research).
- **Copyright**:
  - Books, song recordings, choreographs
  - Exceptions: presenting (fractional pieces of) information from another author
- **Trademarks**:
  - Must be unique for your field (e.g. Apple vs. Apple).

- Cannot use common words - this is actually one reason why many companies have slightly misspelled combinations of common words.
- Can use pictures, character distortions, and color as part of the trademark.
- No issues if in different fields.

- **Trade Secret**
  - No protections, but not spilled, lasts forever
  - Can enforce legal recourse with Non-Disclosure Agreement (NDA)

1.2 Edge Detection Patent Methodologies

For this next section, we will direct our attention toward covering concepts that were discussed in the edge detection patent (6,408,109). Each of these sections will be discussed in further detail below. Before we get into the specifics of the patent again, it is important to point out the importance of edge detection for higher-level machine vision tasks, such as:

- Attitude (pose estimation) of an object
- Object recognition
- Determining to position

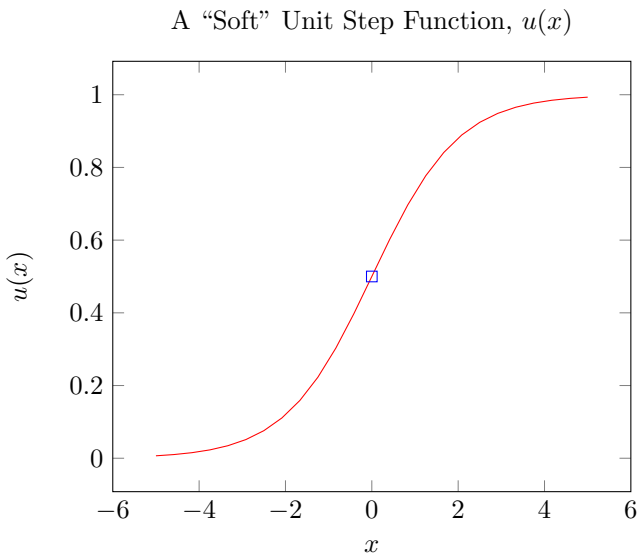
We will touch more on these topics in later lectures.

1.2.1 Finding Edge with Derivatives

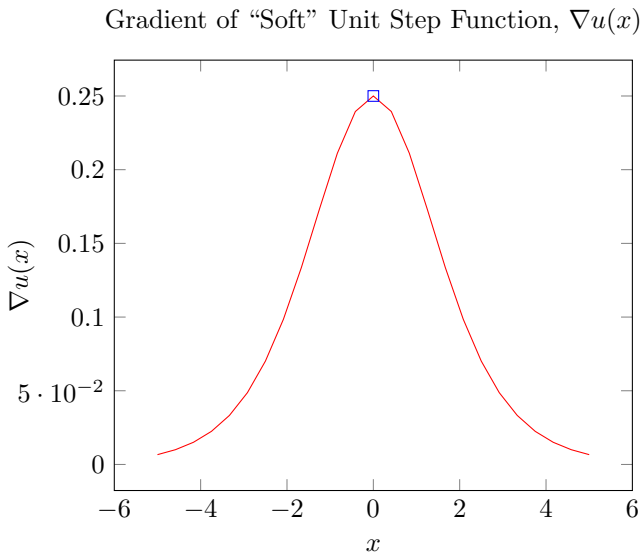
Recall that we find a proposed edge by finding an **inflection point** of the brightness $E(x, y)$. The following methods for finding this point are equivalent:

- Finding an inflection point of brightness $E(x, y)$.
- Finding maximum of brightness gradient magnitude/first derivative $|\nabla E(x, y)|$.
- Finding zero crossing of Laplacian/second derivative $\nabla^2 E(x, y)$.

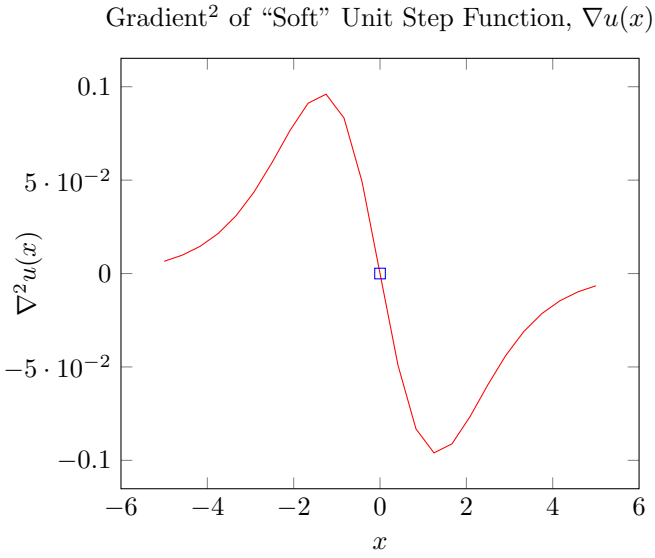
For building intuition, last time we used the following example of $u(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$:



The gradient of this brightness across the edge, given by $\nabla u(x)$ (or $\frac{du}{dx}$ in one dimension), is then given by the following. Notice that the location of the maximum matches the inflection point in the graph above:



As we mentioned above, we can find the location of this edge by looking at where the second derivative of brightness crosses zero, a.k.a. where $\nabla(\nabla u(x)) = \nabla^2 u(x) = 0$. Notice that the location of this zero is given by the same location as the inflection point of $u(x)$ and the maximum of $\nabla u(x)$:



For those curious, here is the math behind this specific function, assuming a sigmoid for $u(x)$:

1. $u(x) = \frac{1}{1+\exp(-x)}$
2. $\nabla u(x) = \frac{du}{dx} = \frac{d}{dx}\left(\frac{1}{1+\exp -x}\right) = \frac{\exp(-x)}{(1+\exp(-x))^2}$
3. $\nabla^2 u(x) = \frac{d^2u}{dx^2} = \frac{d}{dx}\left(\frac{\exp(-x)}{(1+\exp(-x))^2}\right) = \frac{-\exp(-x)(1+\exp(-x))^2+2\exp(-x)(1+\exp(-x))\exp(-x)}{(1+\exp(-x))^4}$

1.2.2 More on "Stencils"/Computational Molecules

Recall that we can use finite differences [1] in the forms of "stencils" or computational molecules to estimate derivatives in our images. For this patent, the authors used this framework to estimate the brightness gradient in order to find edges. For instance, partial derivative of brightness w.r.t. x can be estimated by

1. $E_x = \frac{1}{\epsilon}\begin{bmatrix}-1 & 1\end{bmatrix}$
2. $E_x = \frac{1}{2\epsilon}\begin{bmatrix}-1 & 0 & 1\end{bmatrix}$
3. $E_x = \frac{1}{2\epsilon}\begin{bmatrix}-1 & 1\\ -1 & 1\end{bmatrix}$

Where for molecule 2, the best point for estimating derivatives lies directly in the center pixel, and for molecules 1 and 3, the best point for estimating derivatives lies halfway between the two pixels.

How do we analyze the efficacy of this approach?

1. **Taylor Series**: From previous lectures we saw that we could use averaging to reduce the error terms from 2nd order derivatives to third order derivatives. This is useful for analytically determining the error.
2. **Test functions**: We will touch more on these later, but these are helpful for testing your derivative estimates using analytical expressions, such as polynomial functions.
3. **Fourier domain**: This type of analysis is helpful for understanding how these "stencils"/molecules affect higher (spatial) frequency image content.

Note that derivative estimators can become quite complicated for high-precision estimates of the derivative, even for low-order derivatives. We can use large estimators over many pixels, but we should be mindful of the following tradeoffs:

- We will achieve better noise smoothing/suppression by including more measured values.
- Larger derivative estimators linearly (1D)/quadratically (2D) increase the amount of computation time needed.
- Features can also affect each other - e.g. a large edge detection estimator means that we can have two nearby edges affecting each other.

We can also look at some derivative estimators for higher-order derivatives. For 2nd-order derivatives, we just apply another derivative operator, which is equivalent to convolution of another derivative estimator "molecule":

$$\frac{\partial^2}{\partial x^2}(\cdot) = \frac{\partial}{\partial x}\left(\frac{\partial(\cdot)}{\partial x}\right) \Longleftrightarrow \frac{1}{\epsilon}\begin{bmatrix}-1 & 1\end{bmatrix} \otimes \frac{1}{\epsilon}\begin{bmatrix}-1 & 1\end{bmatrix} = \frac{1}{\epsilon^2}\begin{bmatrix}1 & -2 & 1\end{bmatrix}$$

For deriving the sign here and understanding why we have symmetry, remember that convolution "flips" one of the two filters/-operators!

Sanity Check: Let us apply this to some functions we already know the 2nd derivative of.

- $f(x) = x^2$:

$$\begin{aligned} f(x) &= x^2 \\ f'(x) &= 2x \\ f''(x) &= 2 \end{aligned}$$

  Applying the 2nd derivative estimator above to this function:

$$\begin{bmatrix}1 & -2 & 1\end{bmatrix} \otimes \frac{1}{\epsilon}\begin{bmatrix}f(-1)=1 & f(0)=0 & f(1)=1\end{bmatrix} = \frac{1}{\epsilon^2}((1*1)+(-2*0)+(1*1)) = \frac{1}{\epsilon^2}(1+0+1) = 1*2 = 2$$

  Where we note that $\epsilon = 1$ due to the pixel spacing. This is equivalent to $f''(x) = 2$.

- $f(x) = x$:

$$\begin{aligned} f(x) &= x \\ f'(x) &= 1 \\ f''(x) &= 0 \end{aligned}$$

  Applying the 2nd derivative estimator above to this function:

$$\begin{bmatrix}1 & -2 & 1\end{bmatrix} \otimes \frac{1}{\epsilon}\begin{bmatrix}f(-1)=-1 & f(0)=0 & f(1)=1\end{bmatrix} = \frac{1}{\epsilon^2}((1*-1)+(-2*0)+(1*1)) = \frac{1}{\epsilon^2}(-1+0+1) = 0$$

  Where we note that $\epsilon = 1$ due to the pixel spacing. This is equivalent to $f''(x) = 0$.

- $f(x) = 1$:

$$f(x) = 1$$
$$f'(x) = 0$$
$$f''(x) = 0$$

Applying the 2nd derivative estimator above to this function:

$$\begin{bmatrix}1 & -2 & 1\end{bmatrix} \otimes \frac{1}{\epsilon}\begin{bmatrix}f(-1)=1 & f(0)=1 & f(1)=1\end{bmatrix} = \frac{1}{\epsilon^2}((1*1)+(-2*1)+(1*1)) = \frac{1}{\epsilon^2}(1+-2+1) = 0$$

Where we note that $\epsilon = 1$ due to the pixel spacing. This is equivalent to $f''(x) = 0$.

In Practice: As demonstrated in the example "test functions" above, in general a good way to test an N^{th} order derivative estimator is use polynomial test functions of arbitrary coefficients from order 0 up to order N. For instance, to calculate 4th order derivative estimator, test:

1. $f(x) = a$
2. $f(x) = ax + b$
3. $f(x) = ax^2 + bx + c$
4. $f(x) = ax^3 + bx^2 + cx + d$
5. $f(x) = ax^4 + bx^3 + cx^2 + dx + e$

Note: For derivative estimator operators, the weights of the "stencils"/computational molecules should add up to zero. Now that we have looked at some of these operators and modes of analysis in one dimension, let us now look at 2 dimensions.

1.2.3 Mixed Partial Derivatives in 2D

First, it is important to look at the linear, shift-invariant property of these operators, which we can express for each quality:

- **Shift-Invariant**:

$$\frac{d}{dx}(f(x+\delta)) = f'(x+\delta), \text{ for some } \delta \in \mathbb{R}$$
$$\text{Derivative of shifted function } = \text{ Derivative equivalently shifted by same amount}$$

- **Linear** :

$$\frac{d}{dx}(af_1(x) + bf_2(x)) = af_1'(x) + bf_2'(x) \text{ for some } a, b \in \mathbb{R}$$
$$\text{Derivative of scaled sum of two functions } = \text{ Scaled sum of derivatives of both functions}$$

We will exploit this **linear, shift-invariant** property frequently in machine vision. Because of this joint property, we can treat derivative operators as convolutions in 2D:

$$\frac{\partial^2}{\partial x \partial y}(\cdot) = \frac{\partial}{\partial x}\Big(\frac{\partial}{\partial y}(\cdot)\Big) \iff \frac{1}{\epsilon}\begin{bmatrix}-1 & 1\end{bmatrix} \otimes \frac{1}{\epsilon}\begin{bmatrix}1\\-1\end{bmatrix} = \frac{1}{\epsilon^2}\begin{bmatrix}-1 & +1\\+1 & -1\end{bmatrix}$$

A few notes here:

- The second operator corresponding to E_y has been flipped in accordance with the convolution operator.
- If we project this derivative onto a "diagonal view", we find that it is simply the second derivative of x', where x' is x rotated 45 degrees counterclockwise in the 2D plane: $x' = x\cos 45 + y\cos 45 = \frac{\sqrt{2}}{2}x + \frac{\sqrt{2}}{2}y$. In other words, in this 45-degree rotated coordinate system, $E_{x'x'} = E_{xy}$.
- **Intuition for convolution**: If convolution is a new concept for you, check out reference [2] here. Visually, convolution is equivalent to "flipping and sliding" one operator across all possible (complete and partial) overlapping configurations of the filters with one another.

1.2.4 Laplacian Estimators in 2D

The Laplacian $\nabla^2 \triangleq \Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ is another important estimator in machine vision, and, as we discussed last lecture, is the **lowest-order rotationally-symmetric** derivative operator. Therefore, our finite difference/computational molecule estimates should reflect this property if they are to be accurate. Two candidate estimators of this operator are:

1. **"Direct Edge"**: $\frac{1}{\epsilon^2}\begin{bmatrix}0 & 1 & 0\\ 1 & -4 & 1\\ 0 & 1 & 0\end{bmatrix}$
2. **"Indirect Edge"**: $\frac{1}{2\epsilon^2}\begin{bmatrix}1 & 0 & 1\\ 0 & -4 & 0\\ 1 & 0 & 1\end{bmatrix}$

Note that the second operator has a factor of $\frac{1}{2\epsilon^2}$ in front of it because the distance between edges is $\sqrt{2}$ rather than 1, therefore, we effectively have $\frac{1}{\epsilon'^2}$, where $\epsilon' = \sqrt{2}\epsilon$.

How do we know which of these approximations is better? We can go back to our analysis tools:

- Taylor Series
- Test functions
- Fourier analysis

Intuitively, we know that neither of these estimators will be optimal, because neither of these estimators are rotationally-symmetric. Let us combine these intelligently to achieve rotational symmetry. Adding four times the first one with one times the second:

$$4\Big(\frac{1}{\epsilon^2}\begin{bmatrix}0 & 1 & 0\\ 1 & -4 & 1\\ 0 & 1 & 0\end{bmatrix}\Big) + 1\Big(\frac{1}{2\epsilon^2}\begin{bmatrix}1 & 0 & 1\\ 0 & -4 & 0\\ 1 & 0 & 1\end{bmatrix}\Big) = \frac{1}{6\epsilon^2}\begin{bmatrix}1 & 4 & 1\\ 4 & -20 & 4\\ 1 & 4 & 1\end{bmatrix}$$

Using Taylor Series, we can show that this estimator derived from this linear combination of estimators above results in an error term that is **one derivative higher** than suing either of the individual estimators above, at the cost of more computation. Note that the sum of all the entries here is zero, as we expect for derivative estimators.

For a hexagonal grid, this is scaled by $\frac{1}{2\epsilon^2}$ and has entries of all 1s on the outer ring, and an entry of -6 in the center. An example application of a hexagonal grid - imaging black holes! Leads to $\frac{4}{\pi}$ greater efficiency.

As food for thought, what lowest-order rotationally-symmetric *nonlinear* operators?

$\sqrt{E_{x'}^2 + E_{y'}^2} = \sqrt{E_x^2 + E_y^2}$ Where this is the l2 normal of the estimated brightness gradient

1.2.5 Non-Maximum Suppression

Another technique leveraged in this patent was **Non-Maximum Suppression (NMS)**. Idea: Apply edge detector estimator operator everywhere - we will get a small response in most places, so what if we just threshold? This is an instance of **early decision-making**, because once we take out these points, they are no longer considered edge candidates in downstream steps.

It turns out the authors discourage thresholding, and in their work they remove all but the maximum estimated gradient (note that this is quantized at the octant level). Note that the quantized gradient direction is perpendicular to the edge. In this case, for a candidate gradient point G_0 and the adjacent pixels G_- and G_+, we must have:

$$G_0 > G_-, G_0 \geq G_+$$

This forces $-\frac{1}{2} \leq s \leq \frac{1}{2}$. Note that we have the asymmetric inequality signs to break ties arbitrarily. Next we plot the quantized profile that has been interpolated parabolically - i.e. sub-pixel interpolation.

1.2.6 Plane Position

Note that we have not yet done any thresholding. How can we improve this, given that we quantized the edge gradient direction? Could we try not quantizing the edge direction? If we have the true gradient direction, we can find the intersection of this line with the edge (at 90 degrees to the edge gradient) to find a better solution.

To find this point above (please take a look at the handwritten lecture notes for this lecture), we project from the quantized gradient direction to the actual gradient direction. This is the "plane position" component.

1.2.7 Bias Compensation

Another component of the patent focuses on the interpolation technique used for sub-pixel gradient plotting for peak finding. To find an optimal interpolation technique, we can plot s vs. s', where $s' = s|2s|^b$, where $b \in \mathbb{N}$ is a parameter that determines the relationship between s and s'.

In addition to cubic interpolation, we can also consider piecewise linear interpolation with "triangle" functions. For some different values of b:

- $b = 0 \rightarrow s' = s$
- $b = 1 \rightarrow s' = 2\,\text{sign}(s)s^2$
- $b = 2 \rightarrow s' = 4\,\text{sign}(s)s^3$

Where different interpolation methods give us different values of b.

1.2.8 Edge Transition and Defocusing Compensation

Another point of motivation: most edge detection results depend on the actual edge transition. Why are edges fuzzy (note that some degree of fuzziness is needed to prevent aliasing)? One major cause of fuzziness is "defocusing", in which the image plane and "in focus" planes are slightly off from one another. This causes a "pillbox" of radius R to be imaged (see handwritten lecture notes), rather than the ideal case of an impulse function $\delta(x, y)$. This radius is determined by:

$$R = \delta\frac{d^2}{f}\ (\textbf{Point Spread Function (PSF)})$$

This pillbox image is given mathematically by:

$$\frac{1}{\pi R^2}(1 - u(r - R))$$

Where $u(\cdot)$ is the unit step function. Where f is the focal length of the lens, d is the diameter of the lens (assumed to be conic), and δ is the distance along the optical axis between the actual image plane and the "in focus" plane.

1.3 Multiscale

Note: We will discuss this in greater detail next lecture.

Multiscale is quite important in edge detection, because we can have edges at different scales. To draw contrasting examples, we could have an image such that:

- We have very sharp edges that transition over $\approx$ only 1 pixel
- We have blurry edges that transition over many pixels

1.3.1 Effect on Image Edge

Here is one possible extension not included in the edge detection patent.

We can slide a circle across a binary image - the overlapping regions inside the circle between the 1-0 edge controls how bright things appear. We can use this technique to see how accurately the algorithm plots the edge position - this allows for

error calculation since we have ground truth results that we can compute using the area of the circle. Our area of interest is given by the area enclosed by the chord whose radial points intersect with the binary edge:

$$A = R^2\theta - \frac{2\sqrt{R^2 - X^2}X}{2}$$
$$\theta = \arctan\left(\frac{\sqrt{R^2 - x^2}}{x}\right)$$

Another way to analyze this is to compute the analytical derivatives of this brightness function:

1. $\frac{\partial E}{\partial x} = 2\sqrt{R^2 - x^2}$
2. $\frac{\partial^2 E}{\partial x^2} = \frac{-2x}{\sqrt{R^2 - x^2}}$

What can we do with this? We can use this as input into our algorithm to compute teh error and **compensate** for the degree of defocusing of the lens. In practice, there are other factors that lead to fuzzy edge profiles aside from defocusing, but this defocusing compensation helps.

1.3.2 Addressing Quantization of Gradient Directions

Here is another possible extension not included in the edge detection patent.

Recall that because spaces occurs in two sizes (pixel spacing and $\sqrt{2}$ pixel spacing), we need to sample in two ways, which can lead to slightly different error contributions. We do not want quantized gradient directions. To do this, we can just interpolate values G_-, G_0, G_+ along the true edge gradient!

Linear 1D Interpolation:

$$\tilde{f}(x) = \frac{f(a)(b - x) + f(b)(x - a)}{b - a}$$

We can also leverage more sophisticated interpolation methods, such as **cubic spline**.

Why did the authors not leverage this interpolation strategy?

- this requires the spacing of any level, i.e. not just pixel and $\sqrt{2}$ pixel spacing, but everything in between.
- Since you interpolate, you are not using measured values. Introduces some uncertainty that may be too much to achieve 1/40th pixel accuracy.

What can we do to address this? $\rightarrow$ **Project gradient onto unit circle!** This requires 2D interpolation, which can be done with methods such as bilinear or bicubic interpolation.

1.3.3 CORDIC

As we discussed in the previous lecture, CORDIC is an algorithm used to estimate vector direction by iteratively rotating a vector into a correct angle. For this patent, we are interested in using CORDIC to perform a change of coordinates from cartesian to polar:

$$(E_x, E_y) \rightarrow (E_0, E_\theta)$$

Idea: Rotate a coordinate system to make estimates using test angles iteratively. Note that we can simply compute these with square roots and arc tangents, but these can be prohibitively computationally-expensive:

$$E_0 = \sqrt{E_x^2 + E_y^2}$$
$$E_\theta = \arctan\left(\frac{E_y}{E_x}\right)$$

Rather than computing these directly, it is faster to iteratively solve for the desired rotation θ by taking a sequence of iterative rotations $\{\theta_i\}_{i=1,2,\ldots}^n$. The iterative updates we have for this are, in matrix-vector form:

$$\begin{bmatrix} E_x^{(i+1)} \\ E_y^{(i+1)} \end{bmatrix} = \begin{bmatrix} \cos\theta_i & \sin\theta_i \\ -\sin\theta_i & \cos\theta_i \end{bmatrix} \begin{bmatrix} E_x^{(i)} \\ E_y^{(i)} \end{bmatrix}$$

Gradients at next step = Rotation R by $\theta_i \times$ Gradients at current step

How do we select $\{\theta_i\}_{i=1,2,\cdots}^{n}$? We can select progressively smaller angles. We can accept the candidate angle and invoke the iterative update above if each time the candidate angle reduces $|E_y|$ and increases $|E_x|$.

The aggregate rotation θ is simply the sum of all these accepted angles: $\theta = \sum_i \theta_i$

One potential practical issue with this approach is that it involves a significant number of multiplications. How can we avoid this? We can pick the angles carefully - i.e. if our angles are given successively by $\frac{\pi}{2}$, $\frac{\pi}{4}$, $\frac{\pi}{8}$, ..., then:

$$\frac{\sin\theta_u}{\cos\theta_i} = \frac{1}{2^i} \rightarrow \text{ rotation matrix becomes : } \frac{1}{\cos\theta_i}\begin{bmatrix} 1 & 2^{-i} \\ -2^{-i} & 1 \end{bmatrix}$$

Note that this reduces computation to 2 additions per iteration. Angle we turn through becomes successively smaller:

$$\cos\theta_i = \sqrt{1+\frac{1}{2^{2i}}} \rightarrow R = \prod_i \cos\theta_i = \prod_i \sqrt{1+\frac{1}{2^{2i}}} \approx 1.16 \text{ (precomputed)}$$

1.4 References

1. Finite Differences, https://en.wikipedia.org/wiki/Finite difference

6.801 / 6.866 Machine Vision
Fall 2020