

Adaptive Thresholding, Segmentation and Experimenting with Robotic Arms

CSE 4422 Lab Assignment 2

Aysar Khalid (209728866) and Gonghe Shi (207865413)

10/19/2014

Introduction

In computer vision, image segmentation is the process of separating an image into multiple segments. This allows us to better analyze an image by viewing it in an easier representation than its original. One method used to segment an image is thresholding; that is to set all pixels whose intensity values are below a threshold as background values and the other pixels would be above the threshold would be set to foreground values. The simplest form of thresholding uses a global threshold for all pixels, while adaptive thresholding dynamically changes the threshold according to the pixel and its surroundings. Typically thresholding begins with a grayscale image and outputs a binary image to clearly depict the segments in the image. In this lab we developed an adaptive thresholding algorithm and compared it to its simpler counterpart.

In addition, we developed the inverse kinematics proof to move the robotic arm to a given x, y, z coordinate. The proof involved trigonometric identities and a better understanding of the robotic arm's design was acquired.

1. Adaptive Thresholding

In this first problem, we were tasked with developing an adaptive thresholding algorithm. First we created a planar image, I^t , that would serve as our thresholding plane for the original image. To do this, we know I^t is defined by:

$$I^t[i, j] = \alpha_{00} + \alpha_{10}i + \alpha_{01}j$$

And we also are given that:

$$S = \sum_{i,j} (I^t[i, j] - I[i, j])^2$$

Where $I[1..N, 1..M]$. Before minimizing, we reviewed the general summation formulas:

$$\sum_{i=1}^n i^1 = \frac{n \cdot (n+1)}{2}$$

$$\sum_{i=1}^n i^2 = \frac{n \cdot (n+1) \cdot (2n+1)}{6}$$

To minimize S , we take the derivatives with respect to the unknowns.

$$\frac{dS}{d\alpha_{00}} = 0 = 2 \sum_{i,j} (\alpha_{00} + \alpha_{10}i + \alpha_{01}j - I[i, j])$$

$$\frac{I[i, j]}{NM} = \alpha_{00} + \alpha_{10} \frac{(N+1)}{2} + \alpha_{01} \frac{(M+1)}{2}$$

$$\frac{dS}{d\alpha_{10}} = 0 = 2 \sum_{i,j} (\alpha_{00}i + \alpha_{10}i^2 + \alpha_{01}ji - I[i, j])$$

$$\frac{I[i, j]}{NM} = \alpha_{00} \frac{(N+1)}{2} + \alpha_{10} \frac{(N+1)(2N+1)}{6} + \alpha_{01} \frac{(N+1)(M+1)}{4}$$

$$\frac{dS}{d\alpha_{01}} = 0 = 2 \sum_{i,j} (\alpha_{00}j + \alpha_{10}ij + \alpha_{01}j^2 - I[i, j])$$

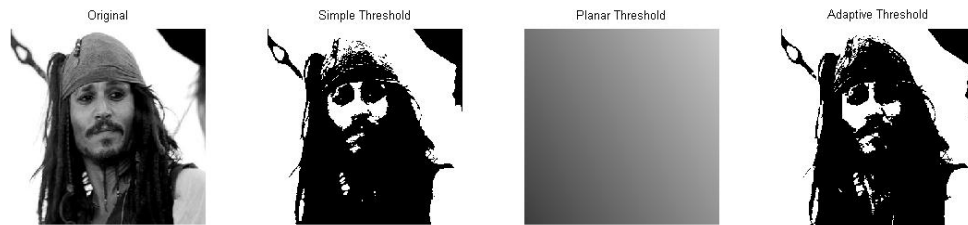
$$\frac{I[i, j]}{NM} = \alpha_{00} \frac{(M+1)}{2} + \alpha_{10} \frac{(N+1)(M+1)}{4} + \alpha_{01} \frac{(M+1)(2M+1)}{6}$$

Our results can then be represented in the form: $M\alpha=b$:

$$M = \begin{bmatrix} 1 & (n+1)/2 & (m+1)/2; \\ (n+1)/2 & ((n+1)*(2*n+1))/6 & ((n+1)*(m+1))/4; \\ (m+1)/2 & ((n+1)*(m+1))/4 & ((m+1)*(2*m+1))/6 \end{bmatrix};$$

We can then solve for α via Matlab using $M \backslash b$ operator which solves a system of linear equations $Ax=B$ for x . One thing to note is the symmetry in the M matrix which depends only on the image size. While b

is a vector of the image averages. Once we've solved for α , we can then use it to develop our thresholding planar image. Below, we see the first of our algorithm's results in comparison to a basic global thresholding algorithm based on average image intensity.



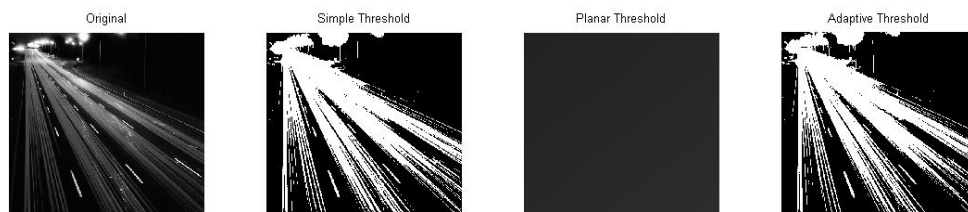
We can see in the above figure, though there is little variance, our adaptive algorithm captures more of the hat part of the image.



Above we see distinctively better results in our algorithm as it clearly shows far background objects that are not captured by the simple thresholding algorithm.



Above we see the first example of where the simple thresholding algorithm works better, this is likely due to the few objects and clean light conditions of the image.



Above we see our algorithm captures objects and omits small specks as opposed to the simple threshold.



Above we see our algorithm captures more branches/leaves of the tree.

It can be concluded that adaptive thresholding does not always perform better than simple global thresholding. Typically, good results are when the image is under varying light conditions and when the background is non-uniform.

To improve our results, we tested a localized approach of the adaptive algorithm, which broke the image into sub images and determined localized planar images; this did not render a drastic improvement in results. It is important to note that, while this approach may lead to increased quality of segmentation; it comes at the cost of time- since we would need to increase the amount of sub images proportionally.

2. Playing with the Robot

In this problem we were tasked with using inverse kinematics to move the robotic arm. The idea is that we are given the length of each link (10 inches) and the position of some point in 3D space and we must find the angles of each joint needed to obtain that position. It is important to note that since we are dealing with 3 joints, there are a lot of possible solutions. In fact, for every solution θ_3 , there is a unique θ_2 and θ_1 . To solve this problem, we must derive the equations for θ_1 , θ_2 , θ_3 :

$$\sqrt{x^2 + y^2} = L \times \cos \theta_2 + L \times \cos(\theta_2 - \theta_3) \dots\dots\dots (1)$$

$$z = L \times \sin \theta_2 + L \times \sin(\theta_2 - \theta_3) \dots\dots\dots (2)$$

Square equation (1) and (2) on both sides then add them up we will get:

$$\begin{aligned} x^2 + y^2 + z^2 &= L^2 \cos^2 \theta_2 + 2L^2 \cos \theta_2 \cos(\theta_2 - \theta_3) + L^2 \cos^2(\theta_2 - \theta_3) + \\ &\quad L^2 \sin^2 \theta_2 + 2L^2 \sin \theta_2 \sin(\theta_2 - \theta_3) + L^2 \sin^2(\theta_2 - \theta_3) \\ &= 2L^2 + 2L^2 (\cos \theta_2 \cos(\theta_2 - \theta_3) + \sin \theta_2 \sin(\theta_2 - \theta_3)) \\ &= 2L^2 + 2L^2 \cos(\theta_2 - (\theta_2 - \theta_3)) \\ &= 2L^2 (1 + \cos \theta_3) \end{aligned}$$

$$\theta_3 = \cos^{-1} \left(\frac{x^2 + y^2 + z^2}{2L^2} - 1 \right)$$

To get rid of complex number answer here, we reject negative value and take its magnitude:

$$\theta_3 = \left| \cos^{-1} \left(\frac{x^2 + y^2 + z^2}{2L^2} - 1 \right) \right|$$

$$\begin{aligned} z &= L \sin \theta_2 + L \times \sin(\theta_2 - \theta_3) \\ &= L \sin \theta_2 + L (\sin \theta_2 \cos \theta_3 - \cos \theta_2 \sin \theta_3) \\ &= L \sin \theta_2 (1 + \cos \theta_3) - L \cos \theta_2 \sin \theta_3 \end{aligned}$$

We define $r = L \sqrt{(1 + \cos \theta_3)^2 + \sin^2 \theta_3}$, this is essentially our hypotenuse:

$$\begin{aligned} \frac{z}{r} &= \frac{L \sin \theta_2 (1 + \cos \theta_3)}{L \sqrt{(1 + \cos \theta_3)^2 + \sin^2 \theta_3}} - \frac{L \cos \theta_2 \sin \theta_3}{L \sqrt{(1 + \cos \theta_3)^2 + \sin^2 \theta_3}} \\ &= \frac{\sin \theta_2 (1 + \cos \theta_3)}{\sqrt{(1 + \cos \theta_3)^2 + \sin^2 \theta_3}} - \frac{\cos \theta_2 \sin \theta_3}{\sqrt{(1 + \cos \theta_3)^2 + \sin^2 \theta_3}} \end{aligned}$$

$$\text{We define angle } q = \text{atan2} \left(\frac{1 + \cos \theta_3}{\sqrt{(1 + \cos \theta_3)^2 + \sin^2 \theta_3}}, \frac{\cos \theta_2 \sin \theta_3}{\sqrt{(1 + \cos \theta_3)^2 + \sin^2 \theta_3}} \right)$$

Then we can simplify:

$$\frac{z}{r} = \sin\theta_2 \sin q - \cos\theta_2 \cos q$$

$$= -\cos(\theta_2 - q)$$

$$\frac{\sqrt{x^2 + y^2}}{r} = \sin(\theta_2 - q)$$

$$\theta_2 - q = \text{atan2}\left(\frac{z}{r}, \frac{\sqrt{x^2 + y^2}}{r}\right)$$

Therefore:

$$\theta_2 = \text{atan2}\left(\frac{z}{r}, \frac{\sqrt{x^2 + y^2}}{r}\right) + q$$

$$\theta_1 = \text{atan}\left(\frac{y}{x}\right)$$

For the easiness of calculation, we also find the angles the following way:

$$r_a = (1 + \cos\theta_3)^2, \text{ and } r_b = \sin^2\theta_3$$

$$q_a = L \times (1 + \cos\theta_3), \text{ and } q_b = L \times \sin\theta_3$$

Then we can simplify r and q to:

$$r = L \times \sqrt{r_a + r_b}$$

$$q = \text{atan2}\left(\frac{q_a}{r} \times 1.0, \frac{q_b}{r} \times 1.0\right)$$

And finally we can get:

$$\theta_2 = q + \text{atan2}\left(\frac{z}{r}, \sqrt{\frac{x^2 + y^2}{r}}\right) \text{ and}$$

$$\theta_1 = \text{atan}\left(\frac{y}{x}\right), \theta_3 = \left| \cos^{-1}\left(\frac{x^2 + y^2 + z^2}{2L^2} - 1\right) \right|$$

To integrate our proof with the robot, we needed to identify all its joint angles and their bounds- this proved to be a nontrivial task, as the manual for the robot specified different bounds from what we found through experimentation. We know that the arm takes 5 angles, which are: waist, shoulder, elbow, wrist and twist. After further clarification from the professor, we understood that the gripper should remain in a fixed state, that is the wrist and twist angles are fixed downwards. Therefore, it was

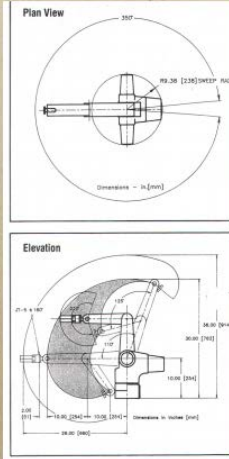
clear that we had 3 joint angles to our 3 derived angles. But the CRS A150 does not use the orientation of $\emptyset 3$ as above; the relationship between \emptyset and the arm's elbow joint is as follows:

$$A150_ \emptyset 3 = \emptyset 2 - \emptyset 3$$

The other derived angles can be used directly with the robot as our tests concluded. Below is an outline of the joint bounds and limitations, we use similar ranges to cap our derived angles to them so that the arm does not error out when given an out of range angle. An improvement we can make would be to normalize our values such that they never fall out of the bound scope, this would make for cleaner code too.

CRS A150

- *Waist -175..175 (0=straight ahead)*
- *Shoulder 0..110 (0=horizontal)*
- *Elbow -125..0 (0==straight)*
- *Wrist bend -110..110 (0=straight)*
- *Wrist twist -180..180 (0=no*



The image contains two technical drawings of the CRS A150 robot arm. The top drawing is labeled 'Plan View' and shows a top-down perspective of the arm's joints and base, with dimensions such as 200, 100, and 150. The bottom drawing is labeled 'Elevation' and shows a side view of the arm, highlighting the elbow and wrist joints, with dimensions such as 100, 150, and 200. Both drawings include dimension lines and numerical values in parentheses, likely representing units or specific measurements.

Conclusion

We have demonstrated just a glimpse of the wide range of applications and capabilities of basic computer vision methods. Namely, adaptive thresholding which can be used to reliably segment an image to show objects and features distinctly. We have also explored a more localized approach that improved results proportional to its time complexity. This approach is likely not to be used in a real-time application context. Overall, we have also demonstrated that the simple global thresholding algorithm still holds its own in simple images with uniform backgrounds.

Using trigonometric identities and inverse kinematics principles, we developed a program that moves the robotic arm to a given 3D space coordinate. Although the proof was somewhat trivial, moving from the theoretical values to arm adaptation was not. Since the robotic arm took in five angle values for its; waist, shoulder, elbow, wrist and twist, some additional arithmetic and further understanding of the arm's orientation was needed. All in all, the lab was satisfying at the end however the difficulty.

User Manual

All implementations were done using MATLAB 2013.

1. Adaptive Thresholding

Open p1.m and run it.

2. Playing with the Robot

Run p2.m, providing an X, Y, Z coordinate and an A150 instance.

References

"Adaptive Thresholding." Point Operations -. N.p., n.d. Web. 19 Oct. 2014.

"Thresholding (image Processing)." Wikipedia. Wikimedia Foundation, 19 Oct. 2014. Web. 19 Oct. 2014.

Code

```
% Aysar Khalid and Gonghe Shi
% Problem 1: Adaptive Thresholding
% The following function takes an image, fits a plane, It, and then
% thresholds the image with the plane.
% Ensure an image, I, is uncommented first before running.
function [ I ] = p1( )
    close all;
%     I = rgb2gray(imread('images/2x2.png'));
%     I = rgb2gray(imread('images/small2.png')); b_1=140; b_2=200;
%     I = rgb2gray(imread('images/jack.png')); b_1=140; b_2=200;
%     I = rgb2gray(imread('images/brain.jpg')); b_1=140; b_2=200;
%     I = rgb2gray(imread('images/virus.jpg')); b_1=140; b_2=200;
%     I = rgb2gray(imread('images/bacteria.jpg')); b_1=140; b_2=200;
%     I = rgb2gray(imread('images/light.jpg')); b_1=240; b_2=260;
    I = rgb2gray(imread('images/winter.jpg')); b_1=100; b_2=150;

    [n m] = size(I);

    It = ones(n,m);
    It_adaptive = zeros(n,m);
    w_size = floor(n/2); %create an NxN window

    % Adaptive Threshold v2
    for p= 1+w_size: 50 : n-w_size
        for q= 1+w_size: 50 : m-w_size
            sub_n = p+w_size;
            sub_m = q+w_size;
            subimage = I(p-w_size:sub_n, q-w_size:sub_m);
            I_avg = mean(mean(subimage));
            b = [I_avg; I_avg*130; I_avg*160];
            M = [1 (sub_n+1)/2 (sub_m+1)/2;
                (sub_n+1)/2 ((sub_n+1)*(2*sub_n+1))/6
                ((sub_n+1)*(sub_m+1))/4;
                (sub_m+1)/2 ((sub_n+1)*(sub_m+1))/4
                ((sub_m+1)*(2*sub_m+1))/5];
            %
            %         alpha = M\b; %solves for a in Ma=b
            %
            %         get the threshold
            for i=1:sub_n
                for j=1:sub_m
                    It_temp = alpha(1) + alpha(2)*i + alpha(3)*j;
                    if (It(i,j) == 1)
                        It(i,j) = It_temp;
                        if (I(i,j) < It(i,j))
                            It_adaptive(i,j) = 0; %black
                        elseif (I(i,j) > It(i,j))
                            It_adaptive(i,j) = 255;
                        end
                    end
                end
            end
        end
    end
end
```

```

%      end

% Adaptive Threshold v1
I_avg = mean(mean(I));
b = [I_avg; I_avg*b_1; I_avg*b_2];
M = [1 (n+1)/2 (m+1)/2;
      (n+1)/2 ((n+1)*(2*n+1))/6 ((n+1)*(m+1))/4;
      (m+1)/2 ((n+1)*(m+1))/4 ((m+1)*(2*m+1))/5];

alpha = M\b; %solves for a in Ma=b

It = I;
for i=1:n
    for j=1:m
        It(i,j) = alpha(1) + alpha(2)*i + alpha(3)*j;
        if (I(i,j) < It(i,j))
            It_adaptive(i,j) = 0;
        elseif (I(i,j) > It(i,j))
            It_adaptive(i,j) = 255;
        end
    end
end

% Simple thresholding
It_simple = I;
for i=1:n
    for j=1:m
        if (I(i,j) < I_avg)
            It_simple(i,j) = 0;
        elseif (I(i,j) > I_avg)
            It_simple(i,j) = 255;
        end
    end
end

subplot(1,4,1);
imshow(I);
title('Original');

subplot(1,4,2);
imshow(It_simple);
title('Simple Threshold');

subplot(1,4,3);
imshow(It);
title('Planar Threshold');

subplot(1,4,4);
imshow(It_adaptive);
title('Adaptive Threshold');
set(gcf, 'units', 'normalized', 'outerposition', [0 0 1 1])
end

```

```

% Aysar Khalid and Gonghe Shi
% Problem 2: Playing with the Robot
% The following function moves the arm to the given XYZ coordinate
% x,y,z are inputs which represents the coordinates of the target position.
% f1,f2,f3,are the angles as represented in the graph in the assignment
question.
% f1,f2,a150_f3 are the angles based on which the arm would move(waist
shoulder elbow).
function [f1,f2,f3] = p2(x,y,z, a150)
%function [f1,f2,f3] = move_arm(x, y, z, a150)
l = 10; % length of link is 10 inches.

% based on the calculation in the report
%  $x^2 + y^2 + z^2 = 2L^2(1+\cos f3)$ 
% so  $f3 = \arccos((x^2 + y^2 + z^2)/(2*1^2)) - 1$ 
% the abs() is meant to get rid of the complex solution of f3.
f3 = abs(acos((x^2 + y^2 + z^2)/(2*1^2)) - 1));

% definitions for r_a, r_b, which makes
%  $r = l * \sqrt{(1+\cos f3)^2 + (\sin f3)^2}$  easier to write.
r_a = (1+cos(f3))^2;
r_b = (sin(f3))^2;
r = l * sqrt(r_a+r_b);

% definitions for q_a, q_b, which makes
%  $q = \text{atan2}((l * (1 + \cos(f3))/r)*1.0, (q_b = l * \sin(f3)/r)*1.0)$ 
% easier to write.
q_a = l * (1 + cos(f3));
q_b = l * sin(f3);

% calculate for q
q = atan2((q_a/r)*1.0,(q_b/r)*1.0);

% calculate for f_1 f_2 and a150_f3.
f2 = q + atan2(z/r, sqrt(x^2 + y^2)/r);
f1 = atan(y/x);

%input angle for the elbow joint of arm
a150_f3 = f2 - f3;

% the range of angles allowed for waist joint is -175 degree to 175 degree
if (f1 >= 175*(pi/180))
    disp('angle for waist > the allowed max value, angle setted to 175
degree');
    f1 = 175*(pi/180);
elseif (f1 <= -175*(pi/180))
    disp('angle for waist < the allowed min value, angle setted to -175
degree');
    f1 = -175*(pi/180);
end

% the range of angles allowed for shoulder joint is 0 degree to 110 degree
if (f2 >= 110*(pi/180))

```

```

        disp('angle for shoulder > the allowed max value, angle setted to 110
degree');
        f2 = 110*(pi/180);
elseif (f2 <= 0)
        disp('angle for shoulder < the allowed min value, angle setted to 0
degree');
        f2 = 0;
end

% the range of angles allowed for elbow joint is -125 degree to 0 degree
if (a150_f3 >= 0)
        disp('angle for elbow > the allowed max value, angle setted to 0
degree');
        a150_f3 = 0;
elseif (a150_f3 <= -125*(pi/180))
        disp('angle for elbow < the allowed min value, angle setted to -125
degree');
        a150_f3 = -125*(pi/180);
end

% angles of the inputs for a150 in degrees
K = [f1*(180/pi) f2*(180/pi) a150_f3*(180/pi) 0 0]

% angles of the inputs for a150 in radians
J = [f1 f2 a150_f3 0 0]
a150.madeg(K)
end

```