A Study on the Curvature of Linked Pearls

Samuel Selleck
sellecksamuel@gmail.com

under the direction of
Assistant Prof. Dhrubaditya Mitra

Astrophysics and Astrobiology
Nordic Institute of Theoretical Physics

Research Academy for Young Scientists
July 13, 2016
Abstract

We study the distribution and correlation of curvature in a linked chain of beads affected by frictional forces. A MATLAB algorithm is developed to analyze pictures taken of the chain and output the curvature at each bead. The distribution of curvature was normally distributed according to a one-sample Kolmogorov-Smirnov test. Beads directly next each other have a negative curvature correlation. The effect seems to quickly disappear and the negative correlation is gone just two beads away.
Contents

1 Introduction 1
   1.1 Curvature of a Plane Curve .................................. 1
   1.2 Digital Image Analysis .................................... 2

2 Data Collection 2
   2.1 The Bead Chain ............................................. 3

3 Curvature Analyzing Algorithm 3
   3.1 Detecting Circles and Their Color ......................... 3
      3.1.1 Circular Hough Transform .......................... 4
      3.1.2 Finding Bead Color and ID .......................... 4
   3.2 Sphere Linking ............................................ 5
      3.2.1 Method ............................................. 5
      3.2.2 Systematical Errors ............................... 5
   3.3 Calculating the Curvature ................................. 6

4 Data Analysis 7

5 Results 8

6 Discussion 9
   6.1 Precision Loss .......................................... 9
   6.2 Future Development ..................................... 9

Acknowledgments 10

References 11

A The Code 12
1 Introduction

Studying the behavior of connected chains helps develop our understanding of linked systems and could in the future be expanded to analyze chemical polymers and how they react internal and external forces. Analyzing simplified cases gives a statistical basis for how linked entities behave, and could be used as a comparison while developing models and simulations of chains. This paper aims to find the distribution and correlation of curvature of a linked chain of beads connected by string when affected by frictional forces, using image analysis.

1.1 Curvature of a Plane Curve

The standard definition of curvature in euclidean, two-dimensional space is a particle’s change in direction, $\Delta \theta$, with respect to curve length, $\Delta s$, at a point $P$, Figure 1a. An equivalent way of defining curvature is the reciprocal of the radius of a circle created by the intersection of the normals of two points infinitely close to the point $P$, Figure 1b. A modified version of this second definition is what will be used in this paper to calculate the curvature. [1]

\[
k = \lim_{\Delta s \to 0} \frac{\Delta \theta}{\Delta s}
\]

(a) Particle traveling along a line.  
(b) Circle following the curvature of a line.

Figure 1: Two ways of defining curvature.
1.2 Digital Image Analysis

Digital image analysis is a branch of image processing that focuses on extracting data and information contained in an image. Exactly how this is done varies widely between different fields and depending on the nature of the image being analyzed. Everything from detecting tumors to analyzing facial expressions can be done with the help of sophisticated software. [2] The digital color images we are analyzing are made out of a $m \times n$ grid of pixels where each pixel stores three values from 0 to 255. Each value represents the intensity of the colors red, green and blue (RGB). This $m \times n \times 3$ matrix of numbers is what will be analyzed to find the curvature of the chain. [2]

2 Data Collection

A camera (Mobile, Nexus 6P) was mounted 40 cm over a level table using metal scaffolding. The chain of beads was placed on a black A3 paper. This paper was taped to a large white paper surface that could be dragged back and forth to make the chain move. Figure 2 are three examples of raw pictures. For each picture taken the white paper was pulled back and forth about 20 cm a total of 10 times. The chain was never allowed to overlap by gaining vertical motion.

![Figure 2: Three raw pictures taken with the setup specified. If the chain ever left the black paper, it was tossed back and shook a couple of extra times to return to a random state.](image)
2.1 The Bead Chain

The chain was created with a thin black twined rope made out of plastic and 30 glass beads. The beads had the colors orange, green, pink and yellow. These colors where sequentially repeated to create a pattern the image analysis algorithm could detect. The first bead was painted blue to easily be distinguished as the starting point, see Figure 3. To allow the chain to bend freely they were separated by a small gap and secured in place with small metal clamps. The unit of measure used to determine the curvature is the average length between two of the beads.

![Figure 3: The Bead Chain that was used under the experiment. Orange, yellow, pink and green beads where ordered in a repeating sequence.](image)

3 Curvature Analyzing Algorithm

Analyzing the images was done with an algorithm created in MATLAB. Images were collected in a folder and processed individually. The algorithm works by automatically finding the pixel coordinates of the colored beads, indexing them in order and then calculating the curvature in each bead. In this way large amounts of data could be processed in a short timespan. For the actual code see the Appendix. An explanation of every part of the software can be found in the following subsections.

3.1 Detecting Circles and Their Color

The circle detection was performed using a MATLAB toolbox called the Image Processing Toolbox™ [4]. The toolbox gives the user access to a function called `imfindcircles` that finds circles in a given radius interval and outputs their centers and radii. The algorithm
uses a method called *circular Hough Transform* and has a number of different settings that can be changed to fine tune the sensitivity.

### 3.1.1 Circular Hough Transform

Circular Hough Transform works by finding the edges in an image, and letting these vote on possible circle centers based on their position, see Figure 4. Different variations of this method are used depending on the type and clarity of image being analyzed. More levels of complexity such as noise removal can be added to optimize for both speed and/or precision. The theory of circular Hough Transform is derived from a general case where other geometric figures can be found. [3]

**Figure 4:** Circular Hough Transform. The black dots represent three detected edge transitions that each vote for all pixels along their respective dotted circles. The pixel with the highest number of votes is where the center of the circle can be found.

**Figure 5:** The RGB values of the pixels within the black outlined circle are averaged to find the mean color of the bead. The radius is set to be slightly smaller to not unintentionally contaminate the color.

### 3.1.2 Finding Bead Color and ID

To find the color of a bead as accurately as possible, the mean RGB value color of all the RGB values within its radius is calculated, see Figure 5. The color of each bead is used to match specific beads with identification numbers (ID) from one to four depending on color, see Figure 3.
The ID of each bead is calculated by comparing the individual rgb channels with reference colors predefined by the user in the form of a matrix. This matrix was created manually by averaging the rgb values of one bead of each color. It ensures that, even if small changes in lighting occur, all the beads can be categorized.

3.2 Sphere Linking

Linking the beads is done purely based on knowledge of position and color ID of the beads. The reason for not searching for the physical segments of string between the beads is a combination of speed, computational complexity and sources of error. Even if a method that incorporates both systems would be the most reliable one, linking the beads purely by color and placement was deemed sufficient for the present study.

3.2.1 Method

The program creates a tree and searches through all the possible paths that can be taken from the blue starting bead, see Figure 6. Every time the current bead has two or more beads of the next sequence color within its searching radius that it can jump to, a tree branch is created. If all the beads are a part of the chain when the algorithm finds a dead end, a list of all the beads in order is returned. If all the beads are not a part of the chain, the program jumps back to the last branching point where all the possible paths have not been explored, and keeps searching. The code segment responsible for the circle linking is a recursive function and works by calling itself for every new possible path.

3.2.2 Systematical Errors

The method gives room for a number of systematical errors. There is a small chance that more than one working path can be found, see Figure 7. In this case, the algorithm finds the chain with the closest connections and outputs this as the result. These situations do not seem to be frequent enough to effect the result in a significant way. The problem
could however be avoided if either a human intervened and chose the right result, or if it was combined with a method of detecting the physical connections between the beads.

(a) Simple case where no branching occurs. The algorithm searches all the way through the chain. When it finds the endpoint it realizes it has been to all points and returns the order list.

(b) A branch is created at the second bead. If the algorithm decided to go down to the yellow bead first, it would soon realize it has no where to go. Since it did not found all the beads it jumps back to the last branch and tries the other path.

Figure 6: Two different bead linking scenarios.

Figure 7: Two different chains can be created from the same bead configuration.

Figure 8: A circle is drawn to approximate the curvature at the orange bead.

3.3 Calculating the Curvature

Approximating the curvature could be done in a number of different ways. In its essence we are given ordered points on a plane that have a maximum distance of one unit between each other. One option considered was fitting a parametric curve to the points. But in the
end, due to the computational complexity and the non-ambiguous numerical uncertainty of the method, we opted for a mathematically simpler approach.

The curvature at each bead was calculated by fitting a circle to its center and the centers of the two beads on either side, see Figure 8. A circle has a constant curvature (the reciprocal of its radius) and can therefore be used to give an approximation of the curvature of the link. The direction of curvature was found by evaluating the cross product of two vectors drawn between the beads in the direction of increasing ID, see Figure 9.

![Figure 9: When the vector between bead one and two is a clockwise to reach the vector between bead two and three, the curvature is negative. When the vector between bead one and two is rotated counter-clockwise to reach the vector between bead two and three, the curvature is positive.](image)

4 Data Analysis

Two different distributions were deemed interesting to study: The probability distribution and the correlation distribution. The probability distribution was visualized by creating a normalized histogram using MATLAB. Calculating the correlation distribution was done using Equation (1),

\[
C(m) = \frac{\sum_{j=1}^{n-m} k(j)k(j + m)}{\sum_{j=1}^{n-m} k(j)k(j)},
\]

where \( n \) is the number of beads, \( k(j) \) the curvature at bead number \( j \) and \( m \) the distance
from one bead to the other. The function is derived from a statistical correlation function, but normalized to be unitary at $m = 0$ for ease of readability.

5 Results

A normalized histogram created from a data set of 3528 curvature values extracted from 126 photos (Figure 10). Figure 11 shows the correlation between beads where the function is the mean value of $C(m)$ for all 126 chains and the error bars are the standard deviation.
6 Discussion

The probability distribution in Figure 10 strongly resembles a normal distribution mildly shifted towards the positive x direction, and was confirmed so by a one-sample Kolmogorov-Smirnov test. A possible explanation of why the distribution seems to be shifted could be because of a tilted table. The end of the beads move more easily in the direction of declining height and result in more positive curvature values than negative.

The correlation between beads seem to drop quickly from $0 < m < 1$ to then flat out at around $m = 3$, as seen in Figure 11. Some kind of force seems to work to create a negative correlation in curvature between beads close to each other.

6.1 Precision Loss

The intuitive threats to accuracy loss are the camera resolution, the distance to the objects measured, the small size of the beads and the camera's ability to focus. Another factor that could effect the results is the accuracy of the function `imfindcircles`, that depends on how distorted the image is.

6.2 Future Development

There is still a lot of features that could be added to the algorithm. Many of the parameters that currently need to be manually customized could technically be automated, such as automatic color differentiation. Prompting the user to sort images with more than one working path would remove a small, but existing source of error.

Future experiments could be conducted where different beads, surface textures and ropes could be tested. Similar experiments could be conducted in a fluid to try and approach the forces a polymer affected by turbulence would experience. Expanding the concept to the third dimension would require more than one camera and new software to map the points in three dimensions. The knowledge that curvature in this type of system follows a normal distribution can also be used to compare with how curvature in models
and simulations of linked systems behave, and thereby give valuable insight into their validity.

Acknowledgments

I would like to thank assistant professor Dhrubaditya Mitra for a wonderful two weeks at The Nordic Institute of Theoretical Physics. I would like to thank Jacob Kuhlin for being an excellent partner. And last but not least I would like to thank Rays, and their sponsors Europaskolan and AstraZeneca, for giving me the opportunity to work on this amazing project.
References


A  The Code

The Flowchart in Figure 12 shows the respective calls each function makes during one run of the program.

Listing 1: The Script ImCAnalysis

```matlab
src = dir('Images new/*.jpg');
curvatures = []; %comment this out if you wish to save for next run
for i = 39:1:length(src)
    curvature = Image_Curvature_Analysis(strcat('Images new/',src(i).name));
    if(~isempty(curvature))
```

Figure 12: Flowchart of the respective function calls

Note that this code was created under a limited timespan and is not properly commented, optimized, or formatted according to conventions. Use it as you see fit.
curvatures = [curvatures, curvature];

disp(strcat(src(i).name, ': Processed'));
disp(i);
else
disp(strcat(src(i).name, ': Failed'));
disp(i);
end
disp(strcat(int2str((i/length(src))*100), '%'));
end

h = histogram( curvatures, 'Normalization', 'probability');

Listing 2: The Function Image_Curvature_Analysis

function [curvatures] = Image_Curvature_Analysis(Image_Path)
    Im = imread(Image_Path);
    %find dark and bright circles
    [centers, radii] = imfindcircles(Im,[20 30],'ObjectPolarity','bright',
                                    'Sensitivity', 0.92, 'EdgeThreshold', 0.03);
    %disp to screen
    imshow(Im);
    h = viscircles(centers, radii);
    %calculate colors and find starting point (blue ball)
    colors = zeros(length(radii), 3);
    current_index = -1;
    for i = 1:length(centers)
        colors(i,:) = Ball_Color(Im, round(centers(i, :)), round(radii(i))
                                 );
        if(Find_Color_Index(colors(i,:)) == 5)
current_index = i;
end
end

%find pearl order
order_list = find_order(centers, colors, [current_index]);

%test??
if (isempty(order_list))
curvatures = [];
else
curvatures = calc_curvatures(centers, order_list);
end
end

Listing 3: The Function Find_Order

function [ order_list ] = Find_Order(centers, colors, order_list )
    for i = length(order_list)+1:length(centers)
        pos_nextindex = Find_Neighbor_Circles(order_list, centers, colors);
        if length(pos_nextindex) == 1
            order_list(i) = pos_nextindex;
        elseif length(pos_nextindex) > 1
            for j = 1:length(pos_nextindex)
                pos_order_list = find_order(centers, colors, [order_list, pos_nextindex(j)]);
                if length(pos_order_list) == length(centers)
                    order_list = pos_order_list;
                end
            end
        end
    end
else
    return;
end
end

distances = zeros(length(pos_nextindex), 1);
for i = 1:length(pos_nextindex)
    distances(i) = pdist([centers(pos_nextindex(i), :);centers(order_list(end), :))];
end
%sort them in order closest -> furthest away
function [color] = Ball_Color( Im, center, radius )
    x = size(Im,2);
    y = size(Im,1);
    pCount = 0;
    r = 0; g = 0; b = 0;
    for i = round(center(1) − radius):1: round(center(1) + radius)
        if((x > i) && i > 0)
            for j = round(center(2) − radius):1: round(center(2) + radius)
                if((y > j) && j > 0)
                    if( pdist([center; i, j]) < radius)
                        r = r + int32(Im(j,i, 1));
                        g = g + int32(Im(j,i, 2));
                        b = b + int32(Im(j,i, 3));
                        pCount = pCount + 1;
                    end
                end
            end
        end
    end
end
end

Listing 5: The Function Ball_Color
color = [r/pCount,g/pCount,b/pCount];
end

Listing 6: The Function Find_Color_Index

```matlab
function [ color_index ] = Find_Color_Index(color)
    smallest_dif = 1000;
    ref_colors = [5, 213, 53; 250, 8, 105; 220, 236, 12; 251, 67, 28;
    33, 91, 154]; %Green = 1, Pink = 2, Yellow = 3, Orange = 4, Blue = 5, Cycles: 5, 4, 3, 2, 1, 4, 3...
    for i = 1:1:length(ref_colors)
        dif = mean(abs(ref_colors(i, :) - color));
        if(smallest_dif > dif)
            smallest_dif = dif;
            color_index = i;
        end
    end
end
```

Listing 7: The Function Curvature

```matlab
function [ k ] = Curvature(point1,point2,point3 )
    % Generates the radius of the circle determined by three given
    a = double(pdist([point1;point2]));
    b = double(pdist([point1;point3]));
    c = double(pdist([point2;point3]));
    k = double(93.8768*(((a+b+c)*(a+b-c)*(a-b+c)*(-a+b+c)).^(1/2))/(a*b*c));
end
```

Listing 8: The Function Calc_Curvatures
function [ curvatures ] = Calc_Curvatures(list_of_points, index_order)
    curvatures = zeros(length(index_order) - 2, 1);
    for i = 2:length(curvatures) + 1
        curvatures(i-1) = curvature(list_of_points(index_order(i-1), :),
                                    list_of_points(index_order(i), :), list_of_points(index_order(i+1), :));
    end
end