PATIENT-CENTERED MONITORING AND IMAGE PROCESSING ON SMARTPHONE

An Undergraduate Research Scholars Thesis

by

CHENJIE LUO

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Tie Liu

May 2017

Major: Electrical Engineering

TABLE OF CONTENTS

Page

ABSTRACT

Patient-Centered Monitoring and Image Processing on Smartphone

Chenjie Luo
Department of Electrical & Computer Engineering
Texas A&M University

Research Advisor: Dr. Tie Liu
Department of Electrical & Computer Engineering
Texas A&M University

Surgery Site Infection (SSI) is an infection that occurs after surgery in the part of the body where the surgery took place. It typically occurs within 30 days after the surgery (and discharge from the hospital). With rapid advance of sensing and mobile technologies, today more methods may be used to control the risk of SSI. In particular, mobile phones can be one of the most convenient and effective tools for monitoring SSI prognosis. The purpose of this research was to develop an algorithm for deforming the surgery site image to help monitor SSI risk and implement the algorithm on IOS devices. With proper deformation, the surgery site image can be readily analyzed using mobile phones with limited computational power. Our work focused on the so-called thin-plate spline interpolation, which is a two-dimensional extension of the cubic spline in one dimension. Compared with other spatial interpolation functions, the thin-plate spline is smooth and numerically stable. We implemented a thin-plate spline based deformation algorithm using both Matlab (for PCs) and Object-Oriented C (for IOS devices) programming languages, and showed promising deformation results for surgery site images.

ACKNOWLEDGMENTS

I would like to express my special gratitude to Professors Tie Liu and Xiaoning Qian, who gave me the opportunity to be part of this project and served as my advisers throughout the project.

CHAPTER I

INTRODUCTION

The goal of this project is to develop a prototype system to monitor surgical region, which can predict the possibility that the patient has been infected. According to the previous research, symptoms like abnormal temperature and fast heart rate can be signs of SSI. However, many of current prognostic models of SSI have not made full use of physical information for SSI prediction. They are merely based on the medical knowledge or heuristics, and most of the models are only qualitative, rather than quantitative. Besides, most current models only incorporate static variables known as of the end of the surgery. It is thus necessary to incorporate dynamic information to improve the analytical method for SSI prediction. In order to solve this problem, we need to build up a dynamic system on smartphone which is able to process the image, analysis the image and provide risk score of the patient in the end. The first step we will do for image processing is either segmentation or deformation of the image.

1. Segmentation and Deformation

The reason why we need to operate deformation of the surgery image is that we would like to reform the surgery site into a line without losing any image information. As a result, we will be able to conduct the later image analysis. At the beginning, we attempted to use GrabCut algorithm to realize the segmentation. However, the result is far from satisfaction. Different from other images, medical images cannot offer an apparent borderline between background and foreground. Therefore, the algorithm may mark some foreground region as background by

mistake. And we use second option which is Thin Plate Spline deformation. According the surgery site, certain fixed points can be obtained in order to operate deformation.

2. Realization on the Smartphone

Due to the fact that we would like to build up a smart phone application, the algorithm need to be run on the phone correctly and precisely operate image processing before analyzing it. Because Thin Plate Spline function is hard to precisely proceed surgery site deformation, we build the function interactively. The user will be required to touch the screen along the surgery site to obtain certain fixed points. In Method section we will discuss why we need these points. After the deformation, the image will be saved to the album and display on the screen. Therefore, further image analysis will be conducted to obtain risk information.

CHAPTER II

METHODS

To analyze the surgery images, several options could be used. Firstly, Segmentation could be operated to differentiate the potential infected region and the normal region. A further analysis could mainly focus on the potential infected region. This solution will need GrabCut algorithm to differentiate two regions. Another option is analyzing the potential infection region with the help of the mesh grids. Each mesh will be analyzed separately in order to deduct if it is infected. The second option will need deformation to make sure surgery is a straight line in the center of image and hence the mesh grids is able to segment the whole image into each mesh. Therefore, further analysis based on each mesh will be easier to conduct.

1. GrabCut

GrabCut is basically an image segmentation method based on GraphCut. It defines an energy function called Gibbs Energy function[Carsten, Vladimir and Andrew 2004] so that the minimum should be able to operate a satisfactory segmentation. And the Gibbs Function is defined as:

$$E(\alpha, k, \theta, z) = U(\alpha, k, \theta, z) + V(\alpha, z) \quad (1)$$

In the above function, z which represents the grey values of the image, which is exactly the same as in Graphcut energy function. The parameter α is called opacity value and can only be either 0 and 1, which represents the pixel should be classified as foreground or background. θ is used to describe grey-level distributions of foreground and background. Variable k is a new variable imported to GrabCut to be a unique GMM component assigned to each pixel.

The Gibbs Function is composed of two key factors: U and V. Factor U is used to decide whether a pixel looks more like a foreground or not. However, color difference between adjacent pixels could also contribute to deciding whether it belongs to foreground or background. Therefore, U and V are defined as follows:

$$\mathrm{U}(\alpha, \mathrm{k}, \theta, \mathrm{z}) = \sum_n D(\alpha_n, k_n, \theta, z_n), \quad (2)$$

where $D(\alpha_n, k_n, \theta, z_n) = -\log \pi(\alpha_n, k_n) + 0.5 \times \log det \sum(\alpha_n, k_n) + 0.5 \times [z_n - \mu(\alpha_n, k_n)]^{\mathrm{T}} \sum(\alpha_n, k_n)^{-1} [z_n - \mu(\alpha_n, k_n)]$ (3)

and therefore:

$$\theta = \{\pi(a, k), u(a, k), \sum(a, k),\ a = 0,1, k = 1, \ldots, K\} \quad (4)$$

Besides, the smoothness term is the same as the $\mathrm{V}(\alpha, \mathrm{z})$ in GraphCut. And it is defined as:

$$\mathrm{V}(\alpha, \mathrm{z}) = \gamma \sum_{(m,n) \in C} [a_n \neq a_m] exp - \beta ||z_m - z_n||^2 \quad (5)$$

Therefore, we use Factor V to describe this difference. With these two factors, the minimum of the function should represent a good segmentation.

2. Segmentation by Using GrabCut

GrabCut is an algorithm to implement segmentation for a given image. We can manually input a rectangle. And the region outside rectangle is assumed background and the region inside is defined as unknown. After computation with help of Gaussian Mixture Model and mincut, the image should be segmented based on the large difference in pixel color. However, different from other kinds of images, surgery image color is relatively monotonous and they do not have a clear borderline between potential infected region and normal region. In other words, pixel color does not have a rapid change between two regions. Besides, after surgery most of patients will be sutured upon the healing regions. It will be even more complicated to handle the segmentation

with stitches in the image. Therefore, the segmentation result is not as good as we expected. Some image features will lose during the segmentation and more importantly segmentation result cannot precisely differentiate the surgery and normal regions.

3. Thin Plate Spline

Thin Plate Spline, which is short for TPS, is commonly used to describe transform of coordinate system (x , y) $\rightarrow$(x' , y') while function value at (x_i' , y_i') equals to (x_i , y_i) and i $\in$ (1 , n). Take a steel plate as an example, we assume there are M points Am on the plate and treat this plate as a 2D image. After bending the steel plate, M points are supposed to shift to Bm accordingly. Thin Plate Spline is such a kind of function that helps to solve the minimum energy we require. And this newly-generated shape can help us determine all other points' location on the plate. Firstly, it is able to generate a transform which possesses minimum bending energy where bending energy is defined as follows:

$$I[f(x,y)] = \iint \left((\frac{\partial^2 f}{\partial x^2})^2 + 2\times(\frac{\partial^2 f}{\partial x \partial y})^2 + (\frac{\partial^2 f}{\partial y^2})^2 \right) dxdy \qquad (6)$$

Given the minimum bending energy, image transform formula is able to be defined as:

$$f(x,y) = a_0 + a_1 x + a_2 y + \sum_{i=1}^{n} b_i {r_i}^2 \, lnr_i \qquad (7)$$

Where we then define U(r) = $r^2 lnr$, r_i is the distance between each corresponding point and other ordinary point so $r_i = \sqrt{(x - x_i)^2 + (y - y_i)^2}$.

According to Thin Plate Spline function, certain number (N) of points need to be picked out and obtain image grey values on each point. The grey values should be mapped with corresponding $f(x,y)$. Plug into the formula we are able to obtain N equations to solve N + 3 unknowns. Therefore, further conditions will be needed to solve all unknowns. We have another condition, which is:

$$\sum_{i=1}^{n} b_i = \sum_{i=1}^{n} b_i x_i = \sum_{i=1}^{n} b_i y_i = 0 \quad (8)$$

Therefore, all the unknowns in (7) are able to be solved. We can then use the equation (7) to obtain all values at every point in image. The new values obtained by the above equation generated the transformed image.

4. Deformation by Using Thin Plate Spline

Deformation method basically uses the mesh grids to segment the image. Different from GrabCut segmentation, no information will lose because we will not extract the portion of the image out but directly focus on each mesh in the original image. However, most surgery are not a straight line and nearly all risky regions are surrounding surgery site. Therefore, a deformation will be needed to change the surgery outlook and we use Thin Plate Spline deformation. We firstly divide the image into $m \times n$ meshes. Assuming the length of the original image is L, width is W. Considering the following matrix calculation, the meshes should all be square. Suppose the mesh length is α. Therefore, m = roundup(L/α), n = roundup(L/α). And then we can operate the deformation using Thin Plate Spline function. After deformation, we can start analyze the image based on the generated image. In the end, we will operate another reverse transform to the original image. And then we will be able to tell which region may have been infected.

CHAPTER III

RESULTS

Here are the results by Thin Plate Spline, GrabCut, and simulations on the smartphone. To begin with, the user will need to either take a picture over the surgery or load an existing image from smartphone's album. In the following simulation, we implemented with an open source patient image (Figure 1). Secondly, the user will interact with the app by tapping certain number of points as reference. And then smart phone will implement Thin Plate Spline based on these fixed points (Figure 2). After that, a mesh grid will be added on the image (Figure 3) and do the deformation again (Figure 4). Figure 5 and Figure 6 are the simulation result on a IOS device. The output image will be both displayed on the screen and saved into album on smartphone.

Figure 1. Original image. Accessed http://photobucket.com/images/wound%20infection?page=1, August, 2016

Figure 2. Image deformed using Thin Plate Spline. Based on http://photobucket.com/images/wound%20infection?page=1, August, 2016

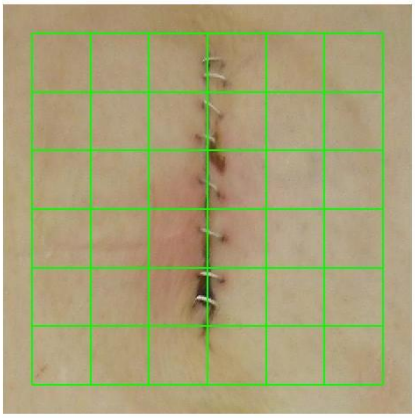

Figure 3. Deformed image with mesh.
Based on http://photobucket.com/images/wound%20infection?page=1, August, 2016

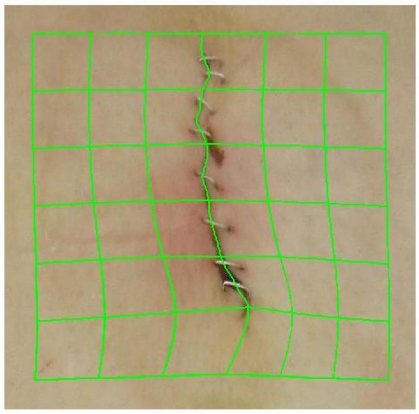

Figure 4. the image after 2nd deformation.
Based on http://photobucket.com/images/wound%20infection?page=1, August, 2016

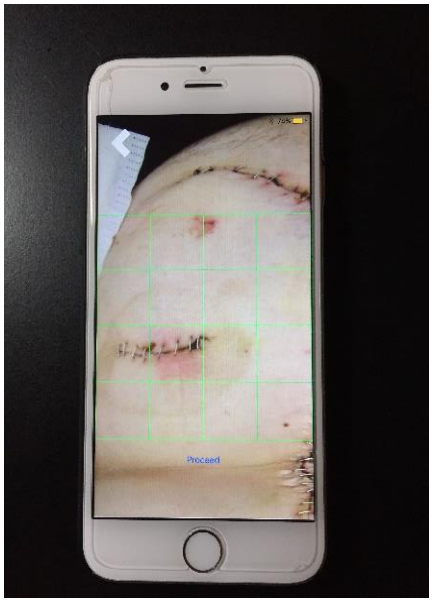

Figure 5. Simulation on the smartphone.
Based on http://photobucket.com/images/wound%20infection?page=1, August, 2016

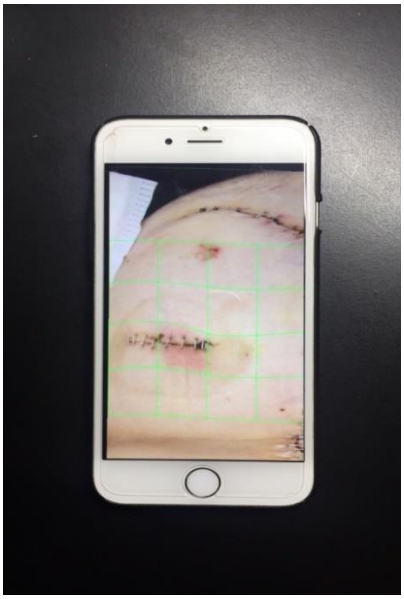

Figure 6. Deformation on the smartphone.
Based on http://photobucket.com/images/wound%20infection?page=1, August, 2016

CHAPTER IV

CONCLUSIONS

This project aims to assist medical centers and hospitals easily track their patients' surgery site infection remotely. With the help of smartphones, general image processing and medical diagnosis are able to be accomplished dynamically. The application will be embedded more functions in order to enrich its features and increase its preciseness. For image processing, both Thin Plate Spline and GrabCut are famous methods and are widely used in the relative field. The difficulty in this project is to maintain the consistency for patients when taking different skin colors and ambient conditions into account. However, because overlapping and missing points still occur, the present function need to be improved. Furthermore, whether there exists a smoother and faster algorithm is still to be verified. Since the present approach is still time consuming and my focus will switch to comparison between different interpolation algorithms on the given surgery site image.

REFERENCES

1) F. L. Bookstein, “Principal warps: thin-plate splines and the decomposition of deformations,” in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, no. 6, pp. 567-585, Jun 1989. doi: 10.1109/34.24792.

2) K. Rohr and S. Wörz, "An extension of thin-plate splines for image registration with radial basis functions," 2012 9th IEEE International Symposium on Biomedical Imaging (ISBI), Barcelona, 2012, pp. 442-445. doi: 10.1109/ISBI.2012.6235579Bartoli, Adrien et al. “Generalized Thin-Plate Spline Warps.” International Journal of Computer Vision.

3) Gary Bradski and Adrian Kaehler, Learning OpenCV , pp. 163-172, September, 2008.

4) Carsten Rother, Vladimir Kolmogorov and Andrew Blake. “‘GrabCut’ — Interactive Foreground Extraction using Iterated Graph Cuts.” ACM Transactions on Graphics, 2004.

5) Gianluca Donato and Serge Belongie. “Approximate Thin Plate Spline Mappings.” In Proceedings of the 7th European Conference on Computer Vision-Part III (ECCV '02), Anders Heyden, Gunnar Sparr, Mads Nielsen, and Peter Johansen (Eds.). Springer-Verlag, London, UK, 21-31, 2002.

6) Open Source Links:
Accessed http://photobucket.com/images/wound%20infection?page=1, August, 2016
Accessed http://photobucket.com/images/surgical%20incision%20infection, August, 2016

APPENDIX

1. Matlab Source Code

```
%Simulation of thin plate spline function
%Created by Chenjie Luo
%Created On October, 23rd, 2016


input_image=imread('1.jpg');
figure
imshow(input_image);
get_image2 = input_image;
N = size(input_image);
I = input_image;
[a,b] = ginput(1);
a=round(a);
b=round(b);
get_image = input_image;
n = 3;
len = -n:n;
X = b + (len)*100;
Y = a + (len)*100;
for i = 1: size(X)
      if X[i] > 0
      break;
      end
end
X = (i+1) * 100 + X;
for j = 1: size(Y)
      if X[j] > 0
      break;
      end
end
Y = j * 100 + Y;

round(X);
round(Y);

for m =1:7

   for k = [1, 3]

      input_image(X(m)+len, Y(1):Y(7), k) = 0;
      input_image(X(1):X(7), Y(m)+len, k) = 0;
   end

   input_image(X(m)+len, Y(1):Y(7), 2) = 255;
   input_image(X(1):X(7), Y(m)+len, 2) = 255;
end
```

```
img1 = input_image( X(1)-49:X(7)+49, Y(1)-49:Y(7)+49,:);% with mesh and mask

img2 = get_image( X(1)-49:X(7)+49, Y(1)-49:Y(7)+49,:);% with mask

img3 = get_image2( X(1)-49:X(7)+49, Y(1)-49:Y(7)+49,:);% with nothing

imshow(img1)
x = ginput;

x = x(:,[2,1]);

y = x;
d = [];
Zp = [];

for s = 1:100:length(x)
    Zp = [Zp; 1,s];
    Zp = [Zp; s,1];
end
Zp = [Zp;x];
for r = 1:(length(x(:,1))-1)
   d = [d, sqrt((x(r,1)-x(r+1,1))^2+(x(r,2)-x(r+1,2))^2)];
end
y(1,1) = x(1,1);
for r = 2:length(x(:,1))
   y(r,1) = d(r-1) + y(r-1,1);
end
y(:,2) = length(y)/2;
for s = 1:100:length(x)
    Zs = [Zs; 1,s];
    Zs = [Zs; s,1];
end
 Zs = [Zs; y];

img = img1;
outDim = [699, 699];

interp.method = 'invdist';
interp.radius = 10;
interp.power = 8;
[imgw2, imgwr2, map2] = tpswarp(img1, outDim, Zp, Zs, interp);
imgw2 = uint8(imgw2);
figure('Position',[230 250 800 500])
get_image = imgw2;
n = 1;
len = -n:n;
X = 50 + (0:6)*100;
Y = 350 + (-3:3)*100;

img4 = get_image;
imshow(img4);
```

```
[imgw3, imgwr3, map3] = tpswarp(img3, outDim, Zp, Zs, interp);
imgw3 = uint8(imgw3);
figure('Position',[230 250 800 500]);
imshow(imgw3);
for m =1:7

   for k = [1, 3]

      imgw3(X(m)+len, Y(1):Y(7), k) = 0;
      imgw3(X(1):X(7), Y(m)+len, k) = 0;
   end

   imgw3(X(m)+len, Y(1):Y(7), 2) = 255;
   imgw3(X(1):X(7), Y(m)+len, 2) = 255;
end
imshow(imgw3);

Zt = Zp;
Zp = Zs;
Zs = Zt;
[imgw3, imgwr3, map3] = tpswarp(imgw3, outDim, Zp, Zs, interp);
imgw3 = uint8(imgw3);
figure('Position',[230 250 800 500]);
imshow(imgw3);
```

2. Objective-C code

```
//
//  DeformationViewController.m
//  Mobile Healthcare System
//
//  Created by Chenjie Luo on 1/3/17.


#import "DeformationViewController.h"
#import "MeshView.h"
#import <MobileCoreServices/UTCoreTypes.h>
#import <AVFoundation/AVFoundation.h>
#import "MeshCaptureViewController.h"
#import <opencv2/opencv.hpp>
#import "SegmentationManager.h"
#import "CThinPlateSpline.h"
#include <iostream>
#include <vector>

@interface DeformationViewController ()

@property (strong, nonatomic) UITapGestureRecognizer *tapGestureRecognizer;
@property (weak, nonatomic) IBOutlet UIImageView *deformedImageView;
@property MeshView * meshView;
@property SegmentationManager * segmentationManager;
@property NSMutableArray * tapLocations;
@property (nonatomic) UIActivityIndicatorView *spinner;
@property (nonatomic) UIView* dimmedView;
@property CGPoint taplocation;

@end

@implementation DeformationViewController

const int majorGrid = 4;
const int minorGrid = 4;

- (void)viewDidLoad {
    [super viewDidLoad];

    self.deformedImageView.image = self.loadedImage;
    self.meshView = [[MeshView alloc] initWithFrame:self.view.frame andMajorGrid:majorGrid
andMinorGrid:minorGrid];
    self.segmentationManager = [[SegmentationManager alloc] init];
    [self.view addSubview:self.meshView];
    [self.view bringSubviewToFront:self.backButton];
    [self.view bringSubviewToFront:self.TakeScreenshot];
    [self.view bringSubviewToFront:self.AnalysisRecognizer];

    NSLog(@"%f", self.meshView.frame.size.width);
    self.tapLocations = [[NSMutableArray alloc ] init];
```

```
    // Do any additional setup after loading the view.
}

-(void) touchesBegan:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [[event allTouches] anyObject];
    CGPoint taplocation = [touch locationInView: touch.view];
    NSLog(@"x:%f y:%f", taplocation);
    NSValue *valueToStore = [NSValue valueWithCGPoint:taplocation];
    [self.tapLocations addObject: valueToStore];
NSLog(@"okok %lu", (unsigned long)self.tapLocations.count);


- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}


- (IBAction)screenshotPressed:(UIButton *)sender {
    NSLog(@"Button pressed");
    UIImage * screenshot = [self screenshot];
    self.deformedImageView.image = screenshot;

    [self showLoadingIndicatorView];

    __weak typeof(self)weakSelf = self;

    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, (unsigned
long)NULL), ^(void) {

        cv::Mat img = [self.segmentationManager cvMatFromUIImage:self.deformedImageView.image];
        NSLog(@"ASDASD %d, %d", img.cols, img.rows);

        int i = 0;
        int j = 0;
        std::vector<cv::Point> iP, iiP;

        for(NSValue * tapPoint in self.tapLocations) {
            CGPoint currentPoint = [tapPoint CGPointValue];
            iP.push_back(cv::Point(currentPoint.x, currentPoint.y));
        }

        for(NSValue * tapPoint in self.tapLocations) {
            CGPoint currentPoint = [tapPoint CGPointValue];
            iiP.push_back(cv::Point(currentPoint.x, self.deformedImageView.image.size.height/2.0));
            NSLog(@"%f", self.deformedImageView.image.size.height/2.0);
        }


        for(i = 1; i <= self.view.frame.size.width; i+=60)
        {
            iP.push_back(cv::Point(i,1));
```

```
            iP.push_back(cv::Point(i,self.view.frame.size.height-1));
            iiP.push_back(cv::Point(i,1));
            iiP.push_back(cv::Point(i,self.view.frame.size.height-1));
        }

        for(j = 1; j <= self.view.frame.size.height; j+=90)
        {
            iP.push_back(cv::Point(1,j));
            iP.push_back(cv::Point(self.view.frame.size.width-1,j));
            iiP.push_back(cv::Point(1,j));
            iiP.push_back(cv::Point(self.view.frame.size.width-1,j));
        }


        CThinPlateSpline tps(iP,iiP);
        NSLog(@"Hey It works!");
        Mat dst;
        tps.warpImage(img,dst,0.01,INTER_CUBIC,BACK_WARP);

        UIImage * deformedImage = [self.segmentationManager UIImageFromCVMat:dst];
        self.deformedImageView.image = deformedImage;
        UIImageWriteToSavedPhotosAlbum(deformedImage, nil, nil, nil);
        NSLog(@"It works completely!");

        dispatch_async(dispatch_get_main_queue(), ^(void) {
            [weakSelf hideLoadingIndicatorView];
        });
    });


    NSLog(@"%f, %f", self.deformedImageView.image.size.height,
self.deformedImageView.image.size.width);
}


//Screen shot
- (UIImage *) screenshot {
    [self.backButton setHidden:YES];
    [self.TakeScreenshot setHidden:YES];

    UIGraphicsBeginImageContext(self.view.bounds.size);
    [self.view.layer renderInContext:UIGraphicsGetCurrentContext()];
    UIImage *viewImage = UIGraphicsGetImageFromCurrentImageContext();

    UIGraphicsEndImageContext();
    UIImageWriteToSavedPhotosAlbum(viewImage, nil, nil, nil);

    [self.backButton setHidden:NO];
    [self.TakeScreenshot setHidden:NO];

    return viewImage;
}
```

```
- (void)showLoadingIndicatorView
{
    [self showLoadingIndicatorViewWithStyle: UIActivityIndicatorViewStyleWhite];
}

- (void)showLoadingIndicatorViewWithStyle:(UIActivityIndicatorViewStyle)activityIndicatorViewStyle
{
    if (self.spinner != nil) {
        [self hideLoadingIndicatorView];
    }

    self.dimmedView = [[UIView alloc] initWithFrame:CGRectMake(0, 0, self.view.frame.size.width,
self.view.frame.size.height)];
    [self.dimmedView setBackgroundColor:[UIColor colorWithRed:0 green:0 blue:0 alpha:0.7]];
    [self.view addSubview:self.dimmedView];

    UIActivityIndicatorView *spinner = [[UIActivityIndicatorView alloc]
initWithActivityIndicatorStyle:activityIndicatorViewStyle];
    spinner.frame = CGRectSetOrigin(spinner.frame, CGPointMake(floorf(CGRectGetMidX(self.view.bounds)
- CGRectGetMidX(spinner.bounds)), floorf(CGRectGetMidY(self.view.bounds) -
CGRectGetMidY(spinner.bounds))));
    spinner.autoresizingMask =
UIViewAutoresizingFlexibleLeftMargin|UIViewAutoresizingFlexibleRightMargin|UIViewAutoresizingFlexib
leTopMargin|UIViewAutoresizingFlexibleBottomMargin;
    [spinner startAnimating];
    [self.view addSubview:spinner];
    self.spinner = spinner;

    [self.view setUserInteractionEnabled:NO];
}

CG_INLINE CGRect
CGRectSetOrigin(CGRect rect, CGPoint origin)
{
    rect.origin = origin;
    return rect;
}

- (void)hideLoadingIndicatorView
{
    [self.spinner stopAnimating];
    [self.spinner removeFromSuperview];
    self.spinner = nil;

    [self.dimmedView removeFromSuperview];
    self.dimmedView = nil;

    [self.view setUserInteractionEnabled:YES];
}

@end
```