UNIVERSITÄT SIEGEN

Department
Elektrotechnik und Informatik

Naturwissenschaftlich
Technische Fakultät

Lehrstuhl für Regelungs-
und Steuerungstechnik
Prof. Dr.-Ing. Hubert Roth

Master Thesis

Specularity Removal for Detecting a Flexible Manipulator for Surgical Operations with an Endoscopic Camera

Name: Mustafa Shuqair Matr.-Nr.: 976228

Supervisor: Prof. Dr.-Ing. Hubert Roth

Supervising Assistant: M. Sc. Erwin Gerz
Dipl.-Ing. Matthias Mende

Submission: 2/4/2014

DECLARATION

Hereby, I declare that I have written this master thesis report and have not used any other sources rather than the books, journals and specially internet, which are cited here.

I agree that the project report can be saved for the purpose of plagiarism test. I confirm that the electronic version is having the same contents as the printed version.

Siegen, April, 2, 2014

Place, Date

Signature of author

ABSTRACT

Disturbing light reflections are visible in images displayed on the computer monitor during endoscopic operations. They are called specular reflections and are produced when the light of the illuminant that is attached to the endoscopic camera bounces off the wet smooth surfaces of the human inner organs. The existence of these specular reflection components camouflage the surface areas beneath them and result in producing accurate results of the examined scenes.

Several approaches have been introduced by researchers and are discussed in this report. Each method is studied to show their strength and limitations for the sake of applying them in this scope of work. The selection procedure involved investigating these methods and matching those up with the defined criterion set by the work's environment of this project. The main focus is to find a method that its code is capable of running in real time while producing satisfying results.

After narrowing the selection down, the selected methods are being deeply analyzed and explained following the calculation procedures step by step. The methods are then implemented in C++, tested and evaluated as a standalone code. The testing process uses images in an uncompressed format.

The code was then merged into the existing package that process the video feed from the endoscopic camera running in Robot Operating System (ROS). This framework is used in this project to process the manipulator detection task. After the merge, two recorded video streams are used to test the package. The two video streams were recorded using the provided endoscopic camera for testing purposes. The two methods are then compared in terms of the result's quality and running speed for the two video streams.

At the end, a selection was made based on the images test results, the videos test results and the criterion specified to determine the most suitable method to be used in this work. Following, a conclusion explaining the limitations faced during applying the methods and possible future work stating where a room of improvements is presented.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

1 INTRODUCTION AND MOTIVATION

In the past years of the human history, almost everybody could notice the huge leap of technological advance in several fields in our lives, starting from mechanical systems and all the way ending with the computer science. This advanced technology helped researchers as well as end consumers and whoever in need to help or improve their lifestyle to achieve what was thought to be extremely difficult if not impossible in the past.

Computer science has remarkably gone way further than most optimists were expecting. Nowadays, computers are involved in almost everyday life in addition to research environments, aerospace, design, simulation, system modeling, artificial intelligence, robots, production machines, cell phones, personal computers and many others.

Robots have benefited largely from this rapid expansion of the computer science, allowing them to perform much more complicated mathematical operations and tasks in shorter times. Artificial intelligence, decision making, data processing, real-time applications, real life applications and the list goes on where robots are used and which fields are being implemented in.

Robots do not only perform tasks individually from their environment, but the modern ones and depending on the task they are programmed to do, need to collect data from the surroundings and process these data in order to take further decision depending on the variables present in these data.

Sensors are the most common way to collect data and transform them into electrical signals that the robot or the machine can understand. Several sensor types have been developed and used throughout time and till present. These sensors are capable of collecting various types of data, ranging from simple variables like temperature to complicated variables like images. Therefore, raised the necessity to analyze these complicated sets of data and extract the useful information after converting them into a language that robots and machines can process. The system that is capable of doing this process is called the machine vision system. [1]

Cameras are widely used for capturing images of the robot's surrounding and send these images as a machine language into the processing unit. The acquired data is then processed further in order to describe the surrounding in the images and to reconstruct its properties, such as shape, illumination and color. This is the main purpose of the machine (computer) vision system. [2] These properties may vary in their complexity and distribution in the image.

Computer vision is commonly used in the fields of optical character recognition (OCR), machine inspection, 3D model building, automotive safety, motion capture, surveillance, fingerprint recognition and biometrics and medical imaging. [2]

Computer vision can be defined as the set of artificial intelligence and image processing procedures that are concerned with computer processing the images taken from the real world. [3] According to this definition, image processing plays a significant role in the computer vision system.

There are several practical applications for computer vision including the image processing techniques; following is a list of some examples [2]:

- Optical character recognition (OCR): analyzing and reading handwritten texts and automatic number plate recognition (ANPR);
- Machine inspection: inspections for the purposes of quality control, measuring tolerances and defect detection;
- Retails: object recognition for automated checkout lanes;
- 3D model building: constructing of 3D models based on aerial images;
- Medical imaging: pre-operative and intra-operative imagery, performing studies on certain parts of human body;
- Automotive safety: detecting obstacles on the road, using it under certain conditions where other techniques do not perform well;
- Match move: combining computer generated imagery with live action footages as used in the Hollywood movies;
- Motion capture: using special markers to register the movements of the actor for a computer generated scene;
- Surveillance: watch for intruders, analyzing traffic and in swimming pools;
- Fingerprint recognition and biometrics: for authentication purposes.

As seen, medical application can also benefit from the modern techniques of computer visions. Therefore, many researches have been focused on that area in order to help improving the quality of work and results for the specialists in this field.

1.1 STIFF-FLOP PROJECT

STIFF-FLOP stands for STIFFness controllable Flexible and Learnable Manipulator for surgical Operations and is a project that aims to create a soft robotic medical arm for surgical purposes that can squeeze through a standard 12mm diameter trocar port, reconfigure itself and stiffen by hydrostatic actuation to perform compliant force control tasks while facing unexpected situations. The project takes its concept from the biological actuation and manipulation principles as the ones found in the octopus arm. The octopus can turn its links in almost any desired direction and change the degree of softness/stiffness of them according to the surroundings and the task it intends to perform. [4]

The projects involves the design and fabrication of the soft manipulator with a gripper at its tip, sensors distribution, biologically inspired actuations and controlling architectures, learning and developing cognition through interaction with a human instructor and manipulation soft objects in complex environments. Each of the tasks has its own complications and challenges to go through. [4]

In order to perform a precise controlling over the soft manipulator that is going to slide in between body organs, it is essential to design a reliable controlling algorithm. It is known that wherever control is mentioned, feedback is also present. It is unlikely to achieve a satisfying controlling without any feedback from the system. This feedback is obtained through different sensor types that depend on the controlled system and the controller used.

This concludes that in order to successfully design biologically inspired actuation and controlling architecture, it is necessary to have the suitable sensors and the most accurate feedback from the system. Sensor types vary and differ according to their function. There are motion sensors, speed sensors, displacement sensors, pressure sensors, temperature sensors, vision sensors and the list goes on. Vision sensors represented mostly by cameras are gaining a great importance nowadays as they provide detailed information of the environment. This detailed information is important for the controller to perform precise tasks as motion control, position control, object detection...Etc.

One of the goals of the STIFF-FLOP project is to design a manipulator to squeeze in between human organs. This is listed under position and motion control criteria. For this task, vision sensors are a great option to be used for providing the controller with a high quality image of the position of the manipulator and its surroundings. A special type of cameras is common in the medical field to perform such tasks of accompanying the manipulator inside the body and transmit a live feed from within. These cameras are called Endoscopic Cameras. The name came from the medical procedure itself, endoscopic operation. [5]

The transmitted live feed from the camera into the controller carries a large amount of information to be processed and analyzed. In the following chapters this topic will be further discussed.

1.2 Development Tools

The STIFF-FLOP project uses the framework Robot Operating System (ROS) as a programming and controlling tool. ROS is a flexible framework that is designed to help writing robots software. It consists of a group of libraries, tools and conventions all together help the user to create complex and robust behavior over wide variety of robotic platforms. It helps software developer to create robots applications by providing libraries, hardware abstractions, device drivers, visualizers, message passing, package management and much more. [6][7]

Software in ROS is organized in packages. A Package might contain ROS nodes, an ROS independent library, a dataset, configuration files, third-party software or anything else that logically constitutes a useful module. The aim of these packages is to provide useful functionality for easy reuse. Packages are grouped into a Stack, which goal is to simplify the code sharing. Stacks collect packages that provide functionality, like navigation stack or manipulation stack. [6][7]

The nodes in the package are the executable files. They can be written either in C++ or Python. These nodes communicate with each other using Topics or Services. The communication between nodes happens on messages. They are sent from a Publisher node to a subscriber node and may contain commands or data. Services are another way that nodes can communicate with each other. They allow nodes to send a Request and receive a Response. [7]

ROS runs under two Linux OS desktop distributions (Ubuntu and Debian), Windows® and Mac® OS X. Ubuntu is the only supported platform while the others are listed experimental. [7] For the compatibility with the used hardware, Debian OS will be used as the host environment for ROS in this work.

As nodes give you the choice to select between C++ and Python for coding, it was advisable to go for the first because of the previous knowledge and experience in working with it and not to spend longer time learning a new programming language. The code is to be written as a standalone code using Eclipse, as it provides an easy and clear interface, code highlighting, code organizing, code debugging, error detection, compiling and building projects and many other features. The written code is then transferred and integrated into ROS TubeDetect package.

For the hardware, the computer is used has a quad core Intel® Core i7™ CPU running at (1.6 - 2.8) GHz with 6 GB DDR3 of RAM. The same device was used for the standalone code and the ROS package testing.

1.3 To-date Progress Status

The main major related task is to use the endoscopic camera for the purpose of detecting the position and orientation of the manipulator inside the human body. Using the live feed from the camera it was possible to detect the dummy manipulator (tube) used in the testing stage and to identify its center using the TubeDetect package in the framework ROS. The TubeDetect package main function is to distinguish the manipulator from its environment and identify the center along its length. The algorithm then visualizes the center on the screen for the user to be able to observe.

However, a serious problem rose which prevented the designed algorithm to successfully detect the tube or the center of it due to some loss in the information. This loss of information is a result of strong light reflections caused by the light emitted by the illuminant fixed at the tip of the camera. This light source is very important, since the inner human body is dark and a light beam is necessary to get a clear view of the

examined areas. These light reflections are not easy to avoid and therefore a way must be found to eliminate them or even try to minimize their effects as much as possible.

Using the provided camera and the dummy manipulator contained in a box that has inner sides simulating a similar environment to the inner human body, it was possible to record video streams showing these reflections.

In the following chapters, the problem and its physical nature will be discussed in detail. It will also go through some literature and proposals that tried to solve similar problems and then proceed with applying and testing the selected ones in order to come up with satisfying results at the end.

2 PROBLEM DEFINITION

In the first chapter a short introduction was made into computer vision and its implementations and how they are of a great importance to today's applications by being part of many real-time and non-real-time modern systems.

However, the previously mentioned applications assume that the image or the video stream we receive from the vision sensors represented by cameras is in an ideal state. In which, this image or video stream is clear and does not contain any noises, distortion, color change, external light sources, reflections or any other undesired effects that may force themselves into the image.

Real life applications usually have this problem. It's almost very difficult to capture a completely clear image that is ready for applying the previously mentioned image processing techniques. This requires us to perform some pre-processing to the image, removing all unwanted effects and noises before outputting it to the next stage where the main processing algorithm for the desired goal is implemented.

One of the medical practices that heavily depend on the imaging technique is Endoscopy. Endoscopy is defined as a type of medical examination in which a device called endoscope is inserted and passed through an area of the body. [5] The endoscope is a flexible tube that contains beside the other things usual a camera attached to its tip and an illumination source, which can provide high detailed images and display it on a projector or screen. [8]

Due to the illumination from this source, there are usually light reflections of the organs and the surface which the endoscope is examining that cause a disturbance and confusion for the physicist. These reflections are also called highlights or specular reflections [8] and need to be removed so areas under these reflections can be restored, enabling the detection algorithm to identify the desired object.

2.1 SPECULAR REFLECTIONS IN IMAGES

First step is to study the physics of light reflection in order to proceed with the procedure of their removal. Reflection is the most noticeable property of light. It is defined by the reference to the incident rays and the reflected rays of a surface. Incident rays are the rays before they hit the surface, and reflected rays are the rays that are reflected off that surface. The angle of incidence is the angle between the incident ray and the normal draw perpendicular to the surface. The angle of reflection is the angle between the reflected ray and the normal. [9]

The reflection off a smooth surface like a mirror is called **Specular Reflection**. Specular reflection produces parallel rays upon reflection. **Diffuse Reflection** is where the reflected rays are not parallel to each other's. Specular reflection is a characteristic of smooth surfaces whereas diffuse reflection is a characteristic of rough surfaces. [9][10] It can be easily seen when looking at two surfaces, one is polished and the other is not.

The polished surface appears brighter than the other. This is due to the specular reflection component.

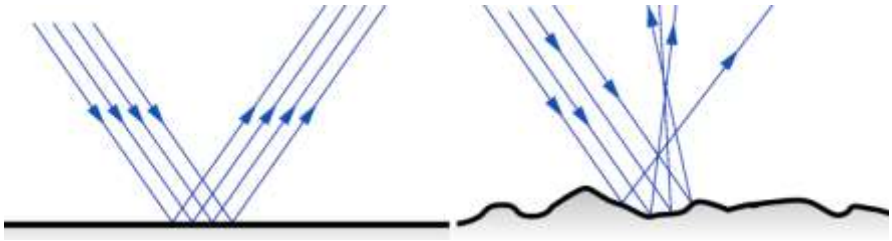


FIGURE 1 ILLUSTRATION OF LIGHT REFLECTION

Figure 1[11] is an illustration of the two types of light reflections. Left side shows the specular reflection, where the parallel reflected rays off the smooth surface can be clearly identified. Right shows the diffuse reflection, where the randomly reflected rays off the rough surface are present.

For dielectric materials, the reflection off the surface contains both specular and diffuse parts. This phenomena can be represented using a simple mathematical model, called **The Dichromatic Reflection Model**.[12] The dichromatic reflection model describes the spectral radiance of a scene $f(\lambda, i, e, g)$ as a sum of the specular reflection component $f_s(\lambda, i, e, g)$ and the diffuse reflection component $f_d(\lambda, i, e, g)$, where i, e *and* g are the angles of light incidence and reflection and the phase angle with respect to the surface normal. λ is the wavelength parameter.

The model states that these two parts are independent from each other and can be mathematically written as: [12]

$$f(\lambda, i, e, g) = f_s(\lambda, i, e, g) + f_d(\lambda, i, e, g) \qquad \text{EQUATION 2-1}$$

The model states as well that each of the two parts can be decomposed down into two parts:

- Composition, a relative spectral power distribution c_s or c_d which depends only on the wavelength and is independent of the geometry;
- Magnitude, a geometric scale factor m_s or m_d which depends on the geometry and is independent from the wavelength.

Therefore, the equation can be rewritten into:

$$f(\lambda, i, e, g) = m_s(i, e, g) * c_s(\lambda) + m_d(i, e, g) * c_d(\lambda) \qquad \text{EQUATION 2-2}$$

Cameras use the three color primaries (Red, Green, and Blue) to represent the spectrum of the light that is reflected of the objects in the images of the scene they are recording. The camera then transforms the light of the incoming ray at position (x, y) from an

infinite light spectrum into the three color components (RGB); $c(x,y) = [r,g,b]$. Since this transformation is linear and the angles $i, e \; and \; g$ depend on x and y, the dichromatic model can be applied to pixel value[13]. This allows Equation 2-2 to be written as:

$$c(x,y) = m_s(i,e,g) * c_{ss} + m_d(i,e,g) * c_{dd} \qquad \text{EQUATION 2-3}$$

c_{ss} and c_{dd} are the specular reflection and the diffuse reflection components respectively that correspond to the pixel at point (x,y) in the scene.

The dichromatic model assumes the following: [12]

- The surface is opaque, inhomogeneous medium. Most of the materials except metals and crystals are inhomogeneous materials;
- The surface has no fluorescent or thin-film properties, and it is uniformly colored;
- The reflection off the surface is isotropic with respect to rotation about the surface normal. Isotropic reflection is when the reflection does not show any change as the surface is rotated around the surface normal;[14]
- There is no inter-reflection among surface and objects;
- There is a single light source and λ is constant among the scene.

In the case of the endoscopic procedure, it can be said that all the previously mentioned assumptions for the dichromatic model to be applied are met. Therefore, a further investigation and analysis into the issue can be carried on.

2.2 Scientific Approaches and Brief Background

Several methods have been introduced to separate these two reflection components. The main aim of all these methods is to create an image that contains no specular reflections, but only diffuse reflection. The image created is called specular free image, and was first introduced by Tan and Ikeuchi. [15]

Nayar *et al.* [16] proposed a method based on the fact that the specular reflection is polarized while the diffuse one is not. By placing a polarization filter in front of the camera they could separate the two components. Their method performs well when dealing with multicolored high textured surfaces even if inter-reflections exist. It shows some weakness in highlight areas with high brightness. Placing a filter in front of the endoscopic camera in our case is not an applicable option; since the camera needs to be inserted into the human body along with other instruments and any foreign object is not desirable.

Kilnker *et al.* [13] found that for a single colored surface the color pixels consist of matte pixels and highlight pixels. The combined color cluster of these two pixels types forms a skewed T-shaped distribution. Their proposal was to use this information to identify

these components and separate the two reflections for each pixel. The method however considers objects having one single color. In our application the scene contains several colors and surfaces.

Tan and Ikeuchi [15] avoided the segmentation and introduced the concept of the specular free image in addition to the specular to diffuse mechanism. Their method removed highlights in single textured multi colored image. They managed to create a specular free image containing only diffuse reflections while keeping the same geometrical distribution as the original one. The separation procedure is done based on chromaticity iteration with regard to the logarithmic differentiation of the specular free image. The long calculation steps will affect the running time of the program. It works well for images but might not be a good option if the code needs to be considered for real time applications.

Myazaki *et al*. [17] generated the specular free image by introducing a special color space. They substituted the saturation components into the intensity components for each pixel in the image. The resulted image has the same hue and saturation values but with intensity different to the actual diffuse component image. They used single image performing one pixel operation in very short execution time. The method consists of few calculation steps and produces satisfying results.

Yoon *et al.* [18] also presented a fast method to separate the diffuse-specular reflection and generate a specular free image by subtracting the minimum RGB value from each channel of each pixel from the same corresponding pixel. Their resulting images had slight change of color, but were free of any specular reflections.

Shen and Cai [19] expanded this approach and solved the issue of color change. They introduced the modified specular free image by adding an offset to the specular free image generated from Yoon *et al.* [18] to produce a chromaticity that is closer to the chromaticity in the diffuse image. Their method is fast and handles well multi colored textured images.

Tan *et al.* [20] managed to separate the two reflection components by analyzing the specular and diffuse pixel distribution in maximum chromaticity-intensity space in addition to the noise analysis to enhance the output image. They used the specular to diffuse mechanism they had introduced earlier. Long set of equations is used to generate the output image and that affects the running time of the code.

Mallick *et al.* [21] introduced a framework for specular-diffuse separation in images and video. Their approach is based on local spatial interactions. Separation is applied through a group of partial differential equations to erode the specular component at each pixel. This erosion is guided by local color and shading information, so that the diffuse texture is preserved without the need of a segmentation of the image. The interaction between neighbor pixels will lead to an increase of the calculation steps and therefore increase the running time of the code.

Tan and Ikeuchi [22] proposed a method to separate reflection components by relating specular pixels to diffuse pixels for the same colors by using the coefficient of reflectance linear basis functions. The algorithm performs well with textured colored imaged without requiring color segmentation. Relating specula pixels to diffuse pixels requires scanning the image multiple times. While it works well for single still images, it might cause slowdowns in video streams and real time applications.

Park and Lee [23] proposed a method to inpaint highlights and remove specular reflections by color line projection. They used two images captured at different exposure times to analyze the highlight components and figure out the colors behind these areas. The method is suitable for still image, since two images at different exposure times can be taken for the same scene. It is not the case for video streams. In video streams the scene is not fixed and each frame has different components compared to the previous and the next one. In order to apply this method, one needs to install two cameras and set different exposure time for each to record every frame. In the endoscopic procedure for this project, one camera is used. Thus, makes the implementation of this method in this project and under the current status not possible.

Das *et al.* [24] presented another method in the field of surgery to eliminate specular reflections in inspecting and detecting cervical cancer. They separated the RGB values of the images into three red, green and blue planes. Then they identified the highlights by selecting the white pixels from each plane and then logically “AND” them. They used Laplace equation in the filling algorithm where the highlights had been removed.

Tsuji [25] proposed a way to remove specular reflections by utilizing the characteristics of a high speed camera and supposing a linear change in the luminance values. The camera operates at 200 frames per second. Their method works efficiently in removing highlight from different light sources and different object surfaces and structures. The provided endoscopic camera for this project work does not have the capability to run at this high frame rate and this prevents implementing Tsuji’s method in this work.

Yang *et al.* [26] eliminated the specular reflections using a single colored image by applying bilateral filtering process. Their method computes the maximum chromaticity and the maximum diffuse chromaticity for the input image, and then applies joint bilateral filtering the image based on the previous two parameters. This process is repeated until the specular reflection is removed from the image. These iteration procedures will increase the running time of the code. They avoided that by using a GPU that runs 200 times faster than the CPU.

As it can be seen, all the previously mentioned methods managed to remove the specular reflection components from images. Some of them have limitations and some have advantages over the others. In this scope of work, the removal of the specular reflections in the image is a pre-processing step, where the image is prepared for further analysis and for other algorithms to be applied. Therefore, the technique to be used needs to be in real-time if not faster. Since the size of the input image of the video

feed from the endoscopic camera is 1920x1080 pixels running at 24 frames per second, the calculation procedures need to be as simple as possible.

The use of any additional hardware at this stage other than the provided camera and the manipulator is not applicable; for example attaching filters, extra lenses and sensors. The number of endoscopic cameras used in this scope of work is one. This camera is responsible for transmitting a video stream viewing the examined areas of the inner human body, alongside the robotic manipulator. This streamed video consists of frames (24 frames per second in this case) and each frame represents an image. Each frame is being processed as a single image independently from the previous and the following frame. Therefore, the method to be selected in this scope of work needs to be able to generate the specular free image out from one single original image of the scene. Every frame that is going to be processed contains scenes for different parts inside the human body in addition to the manipulator. These objects have different surface colors and geometries and vary in their optical properties. Any applied method needs to handle this situation well. It is also a crucial element to obtain satisfactory results in the method that is planned to be used.

In summary, the method selection procedure needs to follow certain criterion and they are:

- The use of the provided single camera;
- The frame rate is locked at 24 frames per second. It can be lowered using the provided hardware, but it can't be set to higher values;
- The need of one single image for the scene to generate the specular free image;
- The ability to remove highlights from images containing multicolored surfaces;
- Real time capability of written code of the method;
- Achieving satisfactory results.

The work by Nayar [16] requires attaching a filter to the camera, which in this case is not applicable as well as the work by Tsuji [25] by using a high speed camera. Klinker's method [13] works only with images containing one color. In [20] [21] [22] [23] removing highlights are efficiently performed in multi colored images but interaction between pixels is considered and is part of the algorithm. This increases the calculation procedures and the execution time. The need for iterative calculations like in [26] requires longer times and more computational power, since our work is performed using the CPU as the processing unit and not the GPU. Using long series of equations [21] [24] is also time consuming. Yoon's method [18] is very fast since it does not contain much calculation steps, but does not produce pleasant results.

The approach of Miyazaki [17] has defined number of calculation steps and does not contain any consideration of neighbor pixels or iterations. This increases the possibility for the code written for this method to run in real-time .It also uses one single image to generate the specular free image and works for images containing multicolored surfaces

with different geometries. That makes Miyazaki's approach a strong candidate for further studying and analyzing.

Shen and Cai's method [19] has also defined number of calculation steps and produces the specular free image from one single image. It does not consider any relations between neighbor pixels nor iteration procedures. It does not require using any additional tools or special cameras. It deals well with images having multicolored surfaces and several geometries. Such good properties for the scope of this work make the method an approach to be considered.

The two proposals from Miyazaki and Shen and Cai meet the defined criterion for selecting the approach to be applied in this work and that makes them both strong candidates for further analyzes to study their behavior and benefits achieved. In the following section, these two approaches of Miyazaki and Shen and Cai are going to be discussed in detail.

2.3 Algorithm Analysis

2.3.1 Shen and Cai's Proposal

Shen and Cai [19] introduced a simple yet effective method to separate specular reflections and diffuse reflection components in multicolored textured surfaces using a single image. However, there are three main assumptions that are used in the equations derivation process of this method. They are:

- The dichromatic reflection model is applicable;
- The camera responses are linear to the entering light flux. It means the gamma value of the camera needs to be equal to 1. Using compressed formats for images, videos or video streams by default changes the relation between the camera response and the entering light flux in the image into a nonlinear relation [27]. They must be avoided.
- The color of the light source is known. It is white in our case.

This method deals with each pixel individually. It does not require any region segmentation or it considers any relations between the neighbor pixels. Therefore the execution time dependent only on how many pixels are in the image, or in other words the size of the image in pixels. The geometry of textures in the image does not affect the running time of the method either.

The first procedure in this method includes scanning the image to determine the minimum value of RGB components of each pixel. Each minimum value is then subtracted from all three RGB components in the corresponding pixel to produce the specular free image. After that, a threshold is added to each pixel to compensate the loss of the chromaticity that occurred because of the subtraction performed earlier to produce what they introduced and called the Modified Specular Free Image. The derivation procedures are their own and will be explained step by step in the following.

As mentioned before, the behavior of the camera is linear. This means that the RGB values in the image respond linearly to the intensity of the illuminant. In the recording process the camera transforms the incoming light at the pixel position into RGB values by integrating the infinite light spectrum.[13] Let v_i denote the response of the ith channel of the pixel (RGB being 1, 2 and 3), s_i is the spectral sensitivity of the camera and $l(\lambda)$ is the spectral power distribution of the illuminant. From Equation 2-2, $f(\lambda, i, e, g)$ is the total spectral reflectance of the wavelength λ at position (i, e, g) or pixel position (x, y). Then the imaging process can be written as:

$$v_i(x,y) = \int f(\lambda, x, y) l(\lambda)\, s_i(\lambda) d\lambda \qquad \text{EQUATION 2-4}$$

The integral is applied in the visible wavelength range.

From the fact that the spectral power distribution of both the specular reflection and the illuminant are similar [28] [19], the Equation 2-2 can be re-written:

$$f(\lambda, x, y) = m_s(i, e, g) * c_{ss} + m_d(i, e, g) * c_{dd}(\lambda) \qquad \text{EQUATION 2-5}$$

Substituting Equation 2-5 in 2-4 results in:

$$v_i(x,y) = m_d(x,y) \int c_d(\lambda) l(\lambda) s_i(\lambda) d\lambda + m_s(x,y) * c_{ss} \int l(\lambda) s_i(\lambda) d\lambda \qquad \text{EQUATION 2-6}$$

$$v_i(x,y) = m_d(x,y) * V_{d,i} + m_s(x,y) * v_{s,i} \qquad \text{EQUATION 2-7}$$

Since (x, y) indicates the pixel position (p), the Equation 2-7 can be written as:

$$v_i(p) = m_d(p) * v_{d,i} + m_s(p) * v_{s,i} \qquad \text{EQUATION 2-8}$$

In the last equation v, represents the intrinsic body color of the material and v represents the illuminant color. The illuminant color is obtained by assuming a pure white panel, which corresponds to white light source in our case. The color of each pixel is normalized and rescaled into the range (0-225). This leads to $v_{s,i} = 225$ for each RGB channel. Let $m_{ss}(p) = m_s(p) v_{s,i} = 255 m_s(p)$. Equation 2-8 becomes:

$$v_i(p) = m_d(p) * v_{d,i} + m_{ss}(p) \qquad \text{EQUATION 2-9}$$

The specular free image can be obtained by subtracting the minimum channel values of the RGB components of each pixel. It can be described with:

$$v_{SF,i}(p) = v_i(p) - v_{min}(p) \qquad \text{EQUATION 2-10}$$

where $v_{min}(p) = min_i\{v_i(p)\}$. The term $v_{min}(p)$ can be described using the dichromatic model of reflection as:

$$v_{min}(p) = m_d(p) * v_{d,min} + m_s(p) \qquad \text{EQUATION 2-11}$$

For $v_{d,min} = min_i\{v_{d,i}(p)\}$, and substituting Equations 2-9 and 2-11 in Equation 2-10. It becomes:

$$v_{SF,i}(p) = m_d(p) * \left(v_{d,i} - v_{d,min}\right) \qquad \text{EQUATION 2-12}$$

It is noted that the Equation 2-12 does not contain the geometrical factor $m_s(p)$. It is also clear that by subtracting the minimum value of the RGB channels values we get an image with at least one channel having the value zero. As a consequence the specular free image will appear darker than the original.

The relation between the diffuse image and the specular free image can be written as:

$$v_{dif,i}(p) = v_i(p) - m_{ss}(p) = v_{SF,i}(p) + v_{min} - m_{ss}(p) \qquad \text{EQUATION 2-13}$$

$$v_{dif,i}(p) = v_{SF,i}(p) + \tau_s(p) \qquad \text{EQUATION 2-14}$$

The term $\tau_s(p)$ which is equal to $v_{min} - m_{ss}(p)$, is the offset of the pixel p. The value of this offset is unknown due to the existence of the geometrical factor $m_{ss}(p)$ in it.

In order to produce an image as close as possible to the diffuse image, the concept of the modified specular free image that was introduced by Shen and Cai is used. It is obtained by adding a pixel offset $\tau(p)$ to the specular free image as follows:

$$v_{MSF,i}(\tau, p) = v_{SF,i}(p) + \tau(p) \qquad \text{EQUATION 2-15}$$

$$v_{MSF,i}(\tau, p) = m_d(p) * \left(v_{d,i} - v_{d,min}\right) + \tau(p) \qquad \text{EQUATION 2-16}$$

The pixel offset $\tau(p)$ is not constant but pixel dependent. It is calculated by introducing a threshold that is able to distinguish between the specular and the diffuse pixels. The threshold t_v is calculated as follows:

$$t_v = \mu_v + \vartheta\sigma_v \qquad \text{EQUATION 2-17}$$

μ_v and σ_v are the mean and the standard deviation of all $v(p)$ in all pixels. ϑ is a factor related to the degree of the highlights in the image and it is set to 0.5 as this value is appropriate for most images [19].

To distinguish between the diffuse and the specular pixels, the minimum RGB value of each pixel is compared to the threshold t_v. If the minimum value is smaller than the threshold, then it's a diffuse pixel, and if the minimum value is larger than the threshold, then it's a specular pixel. It can be formulated as follows:

$$\tau(p) = \begin{cases} t_v & , v_{min,p} > t_v \\ v_{min,p} & , v_{min,p} \leq t_v \end{cases} \qquad \text{EQUATION 2-18}$$

After calculating the pixel offset, it can be easily added to the specular free image to achieve the desired modified specular free image.

2.3.2 *MIYAZAKI'S PROPOSAL*

Miyazaki *et al.* [17] introduced a simple method to eliminate the specular reflection components in multicolored complex textured objects using a single image. His method avoids all the complicated calculations and iterations. Like in the previous method, there are some restrictions and assumption which need to be verified before proceeding with the calculations. They are similar to the ones in Shen and Cai's method. [19] They are listed as follows: [27]

- Image saturation should not occur. The RGB values should remain in the dynamic range. If a camera has a dynamic range between 0 and 255, the pixel value 255 needs to be avoided.
- The camera's response must be linear to the intensity of the light flux coming in.
- Highlights can't be removed from white objects.
- The light source is known and to be white.

This method as Shen and Cai's [19], deals with each pixel on its own. It does not apply any region segmentation or consider any relations between the neighbor pixels. This makes the execution time is dependent only on the size of the image in pixels. The geometry of textures in the image is maintained and it does not affect the execution time either. Hue and saturation of the image do not change after the process, but the intensity does. The color changes slightly as well, but it remains similar to the color in the original image.

The method obtains the specular free image by transforming the original image from its RGB color space into another customized color space **M** introduced by Miayazki *et al*. In that color space, some calculations are carried over to eliminate the specular reflection components. The image is then transferred back into the RGB color space.

This linear transformation is done using the following relation:

$$\begin{bmatrix} m_1(p) \\ m_2(p) \\ m_3(p) \end{bmatrix} = \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} r(p) \\ g(p) \\ b(p) \end{bmatrix}$$

EQUATION 2-19

where $r(p), g(p)$ and $b(p)$ are the values of RGB channels for each pixel in the image. $m_1(p), m_2(p)$ and $m_3(p)$ are the values of the M color space channels for the respected pixel.

In this color space the specular free image is then calculated as follows:

$$\begin{bmatrix} \hat{m}_1(p) \\ \hat{m}_2(p) \\ \hat{m}_3(p) \end{bmatrix} = \begin{bmatrix} m_1(p) \\ m_2(p) \\ a\sqrt{m_1(p)^2 + m_2(p)^2} \end{bmatrix}$$

EQUATION 2-20

Where $\hat{m}_1(p), \hat{m}_2(p)$ and $\hat{m}_3(p)$ are the specular free image for the respected image pixel channel values in the M color space. The factor a is an arbitrary factor used to control the saturation of the resulting image. It is set to 1 as a default value. Increasing its value above 1 will saturate the image further and decreasing it below 1 will lower the saturation and therefore the image will look darker.

After generating the specular free image out of the original, it needs to be transformed back into the original RGB color space. In order to do so, the matrix in the Equation 2-19 is simply inversed and then multiplied by the resulting $\hat{m}_1(p), \hat{m}_2(p)$ and $\hat{m}_3(p)$ values. The transformation looks as follows:

$$\begin{bmatrix} \hat{r}(p) \\ \hat{g}(p) \\ \hat{b}(p) \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & 0 & 1 \\ -\frac{1}{3} & -\frac{1}{\sqrt{3}} & 1 \\ -\frac{1}{3} & \frac{1}{\sqrt{3}} & 1 \end{bmatrix} \begin{bmatrix} \hat{m}_1(p) \\ \hat{m}_2(p) \\ \hat{m}_3(p) \end{bmatrix}$$

EQUATION 2-21

The values $\hat{r}(p), \hat{g}(p)$ and $\hat{b}(p)$ are the pixel RGB components of the specular free image in the RGB color space.

As it can be seen from studying both methods, they both make use of simple mathematical calculation steps and do not perform any image segmentation or iterative procedures. Miyazaki's method at the first sight appears to be simpler and requires less effort than Shen and Cai's. Though, in the following we are going to test both of them and evaluate which one is more applicable than the other, which is faster and which gives better results in our application.

2.4 COLLECTING VARIABLES AND PARAMETERS

The main goal in this project is to eliminate the specular reflection components in a live video stream from the endoscopic camera. It is usually advisable to start with the simple ideal case and then move toward more complicated ones. Therefore, starting with single images and test the selected method on them is the first step to be done.

The images selected need to meet the criterion specified by the two methods. The images need to be saved in a non-compressed format, having highlights caused by a white light source and do not contain white colored objects. Tan [15] provided sample test images for various objects on his website [32] along with the open source C++ code of his method. Yang *et al.* [26] provided some other images on their website [33] they have their coded openly uploaded there. The Computational Vision Laboratory of Simon Fraser University [29] has on their website sets of image data for various research purposes and available for free download.

Images for the purpose of this research work have been collected from the three mentioned resources. They represent pictures of various objects and with different levels of highlights. The light source is white in all of them and the formats of the digital copies are raw (*.ppm and *.tif file formats). The selected images are readable by OpenCV libraries and can be imported into any C++ program. They are not in very large sizes and contain enough information about the objects and the highlights in them.

When the two methods prove their capabilities in eliminating specular reflection components for the selected images set then the next step will take place by replacing the images with video streams. The video streams are two uncompressed digital files recorded using the provided endoscopic camera. They contain frames showing the green tube-like wire with the camera being moved around, changing the angle of view and the angle of the illuminant light falling on and reflecting off its surface. The first video stream represents the ideal case with light specular reflections present. The second contains heavy highlights and various intensity degrees of specular reflection components on the tube and the surrounding. This situation can be labeled as the worst case that could occur in practice. It is also a big challenge since there are several objects in the scene with different textures and colors including the tube itself and the organs around it. The two video files are stored in an uncompressed format (*.bag) and can be played only under ROS environment. Both video streams are in full HD format (1920x1080 pixels) and running at 24 frames per second.

After collecting all the necessary material, the next step is to implement the selected two methods using C++. It is going to be discussed in the following chapter.

3 CODING AND IMPLEMENTATION

3.1 IMPLEMENTING ALGORITHM IN C++

3.1.1 SHEN AND CAI'S METHOD

This method [19] does not require a large amount of calculations and can be implemented without major issues using any programming language.

By looking at the Equation 2-10, the specular free image can be generated by subtracting the minimum value among the RGB channels in each pixel from all three RGB values for the same pixel. Then we calculate the standard and the mean deviation of these values in order to determine the threshold and the pixel offset to be added to the image to create the modified specular free image. Doing so in C++ requires following the procedures step by step. Next I will list and explain the code for this method. The complete code is included in the Appendix A.

```
1    #include <opencv2/highgui/highgui.hpp>
2    #include <cv.h>
3    #include <iostream>
4    #include <cmath>
5
6    using namespace cv;
7    using namespace std;
```

These lines are to tell the compiler to include the OpenCV libraries and header files, `cmath` library, `isoatream` library and the compartments named `std` and `cv`. The OpenCV libraries and header files are essential for the image processing procedures. The `cmath` library is required since the program makes use of square root and power functions. These functions are part of the `cmath` library and the compiler cannot calculate the results without it. The `std` and `cv` compartments are to tell the compiler where to look in order to use the commands related to OpenCV and basic C++ function, like `cout`, `cin`, etc.

```
 9   int minValue (int anRGBvalues[3])
10   {
11         int nMinimum = anRGBvalues[0];
12         for (int iii = 1; iii < 3; iii++)
13         {
14         if (nMinimum > anRGBvalues[iii])
15         {      nMinimum = anRGBvalues[iii]; }
16         }
17         return nMinimum;
18   }
```

This function calculates and returns the minimum of three values. It is used to find the minimum of the three RGB components and returns it to the main program.

```
20   int main( int argc, const char** argv )
```

Here the main function of the program is declared with the proper variables related to the use of OpenCv libraries. Each C++ program must have a `main()` function.

```
22  int nbeta;
23  int anRGBPixelSet [3];
```

Then the two integer variables are declared; `nbeta` is where the minimum value of RGB components is stored in the main, `anRGBPixelSet` is an array where the three RGB components are stored in.

```
25  Mat image = imread("Image Path", CV_LOAD_IMAGE_UNCHANGED);
26  Mat SFImage = Mat::zeros( image.size(), image.type() );
27  Mat MSFImage = Mat::zeros( image.size(), image.type() );
```

Next, in lines 25-27, is the process of reading the image that is desired to be processed by specifying its path without any change to the format or the colors. Then create two empty containers for the specular free image and the modified specular free image with the same properties as the loaded image. The command `Mat` is part of the OpenCV libraries.

```
29  if (image.empty())
30  {     cout << "Error : Image cannot be loaded..!!" << endl;
31        return -1;
32  }
```

Lines 29-32 contain the commands to check the image for any errors or if it cannot be read or loaded and display an error message stating so.

```
34  const int nImagePixelsCount = image.rows * image.cols;
35  int anMinPixelsValues [nImagePixelsCount];
36  int nCounterZ = 0;
```

The program creates a constant and stores the image size in pixels in it, and then creates an array with the size of the image to store the calculated minimum RGB values in it. Assuming the image size is 200 pixels with each pixel has a minimum value of RGB channels. This means we have 200 minimum values and that requires 200 slots of space or an array with the size of 200. The reason why the size is defined as constant is because the size of the array must be known when declaring it in C++ and it cannot be variable. If the `nImagePixelsCount` was declared as variable the compiler would terminate and display an error message. `nCounterZ` is general variable for counting purposes.

```
38  for( int y = 0; y < image.rows; y++ )
39  {
40  for( int x = 0; x < image.cols; x++ )
41  {
42  for ( int ii = 0; ii < 3; ii++)
```

```
43  {     anRGBPixelSet[ii] = image.at<Vec3b>(y,x)[ii];   }
44  nbeta = minValue(anRGBPixelSet);
45  for( int c = 0; c < 3; c++ )
46  { SFImage.at<Vec3b>(y,x)[c] = (image.at<Vec3b>(y,x)[c] - nbeta);}
47
48  anMinPixelsValues[nCounterZ] = nbeta;
49  nCounterZ++;
50  }
51  }
```

In the lines 38 and 40, we access each pixel in the image using these two `for` commands. We then scan the RGB channel values and store them in the array `anRGBPixelSet` using 42 and 43. Then the function `minValue` is called which sends the values in the `anRGBPixelSet` to it and store the result in `nbeta` using line 44. In lines 45, 46 we start creating the specular free image subtracting the calculated minimum from the respected pixel in the original image and store the resulting image in the empty container `SFImage`. After that the calculated minimum is stored in the array `anMinPixelsValues` in a position that corresponds to the pixel which led to the use of the variable `nCounterZ` as a counter. Note that by the end of this step and after scanning the entire image, the counter `nCounterZ` will have a value equal to the image size in pixels.

After creating the specular free image, we use it to create the modified specular free image.

```
54  int nMeanValue = 0;
55  int nTempSum = 0;
56  for (int jjj = 0; jjj < nImagePixelsCount; jjj++)
57  {     nTempSum = nTempSum + anMinPixelsValues [jjj]; }
58  nMeanValue = nTempSum / nImagePixelsCount;
```

In the lines 54 to 58 we calculate the mean value of all the already stored minimum RGB values for the entire image. `nTempSum` is used as a temporary variable to store the summation and `nMeanValue` is the variable used to store the desired mean value.

```
61  nTempSum = 0;
62  double dMeanDeviation = 0.0;
63  for (int jj = 0; jj < nImagePixelsCount; jj++)
64  {     nTempSum = nTempSum + abs(anMinPixelsValues[jj] - nMeanValue); }
65  dMeanDeviation = nTempSum / nImagePixelsCount;
```

Next, the calculation of the mean deviation of the same minimum RGB values is performed. The variable `ntempSum` is used again instead of declaring a new variable for a clear optimized coding. The `dMeanDeviation` is declared as a double for a purpose that will clarify in the following steps. The mean deviation [30] can be calculated using the following equation:

$$Mean\ Deviation = \frac{1}{n}\sum_{i=1}^{n}|x_i - \bar{x}| \qquad \text{EQUATION 3-1}$$

Here, n represents the sample size and $\bar{x}$ is the mean value of the sample.

```
67  nTempSum = 0;
68  double dStandardDeviation = 0.0;
69  for (int j = 0; j < nImagePixelsCount; j++)
70  {     nTempSum = nTempSum + ((anMinPixelsValues[j] - nMeanValue) *
71  (anMinPixelsValues[j] - nMeanValue)); }
72  dStandardDeviation = sqrt (nTempSum / nImagePixelsCount);
```

Now, the standard deviation of these minimum RGB values is calculated and stored them in the double type variable `dStandardDeviation`. Note that the variable `nTempSum` was used again for the reason mentioned before. The standard deviation [30] can be calculated using:

$$Standard\ Deviation = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2} \qquad \text{EQUATION 3-2}$$

```
74  double dZeta = 0.50;
75  double dPixelThreshold = 0.0;
76  dPixelThreshold = dMeanDeviation + dZeta * dStandardDeviation;
```

Lines 74-75 contain commands for applying the Equation 2-17 to calculate the threshold T_v or `dPixelThreshold` in the program structure. The variable `dZeta` is the factor ϑ that depends on the level of specular reflection components in the image. It is set to 0.5 here as this value suits most of the image types with moderate highlights. Because `dZeta` is a fraction it must be declared as double or float but not integer. In C++ programing language the arithmetic operands require that all operands to have the same type. This is to avoid calculation errors and confusing that might occur. Looking at line 76 and since `dZeta` is double; the other operands including the standard deviation, mean deviation and the threshold need to have the same data type. Hence, they were declared as double previously.

```
78  double dTau = 0.0;
79  int nCounterX = 0;
80  for( int y = 0; y < image.rows; y++ )
81  {     for( int x = 0; x < image.cols; x++ )
82  {
83  if ( anMinPixelsValues[nCounterX] > dPixelThreshold)
84  {     dTau = dPixelThreshold; }
85  else
86  {     dTau = anMinPixelsValues[nCounterX];      }
87
88  for( int c = 0; c < 3; c++ )
89  {     MSFImage.at<Vec3b>(y,x)[c] = saturate_cast<uchar>(
90  SFImage.at<Vec3b>(y,x)[c] + dTau );}
```

```
91   nCounterX++;
92
93   }
94   }
```

The modified specular free image is generated using the Equation 2-18. The variable `dtau` is the pixel offset$\tau(p)$. The variable `nCounterX` is used as a counter to access the array `anMinPixelsValues` that we created earlier which contains the minimum RGB values of each pixel in the original image. By comparing these values to the threshold we can determine the pixel offset and add this to the specular free image to create the modified specular free image. Since the threshold and the pixel offset are both double type, the command `saturate_cast<uchar>` is used to assure that the resulted pixel has a valid value without factional components as 8-bit unsigned integer.

```
97   namedWindow("Original Image", CV_WINDOW_AUTOSIZE);
98   imshow("Original Image", image);
99
100  namedWindow("Modified Specular Free Image", CV_WINDOW_AUTOSIZE);
101  imshow ("Modified Specular Free Image", MSFImage);
```

The command `namedWindow` creates a window in order to load the wanted image and the command `imshow` shows the selected image in the specified created window. It is used two times to view the original image and the specular free image in separate windows.

```
103  waitKey(0);
104  destroyWindow("Original Image");
105  destroyWindow("modified Specular Free Image");
106
107  return 0;
108  }
```

The command `waitKey(0)` waits for any key to be pressed on the keyboard before resuming executing the program. Placing the `destroyWindow` command, which closes the specified window, after the `waitKey(0)` command will result in that the windows will close only after any key is pressed. At the end, the `return 0` command will terminate and close the program.

3.1.2 MIYAZAKI'S METHOD

Miyazaki's method [17] is simpler and should have a shorter execution time. It can be summed up in three main steps. First, the image is transformed from the RGB color space into the M color space, Equation 2-19. Then, we create the specular free image in the respected color space, Equation 2-20. Finally, the specular free image is transformed back to the RGB color space, equation 2-21. The complete code is included in the Appendix B.

The code will be explained below.

```
12   Mat image = imread("Image Path", CV_LOAD_IMAGE_UNCHANGED);
13   Mat SFImage = Mat::zeros( image.size(), image.type() );
```

```
14  double adTransformMatrix [3][3] = { {1.0, -1.0/2.0, -1.0/2.0,}, {0.0,
15  sqrt(3.0)/2.0, -sqrt(3.0)/2.0,}, {1.0/3.0, 1.0/3.0, 1.0/3.0} };
16  double adInverseTMatrix [3][3] = { {2.0/3.0, 0.0, 1.0,}, {-1.0/3.0,
17  1.0/sqrt(3.0), 1.0,}, {-1.0/3.0, -1.0/sqrt(3.0), 1.0} };
18  double adMColorSpace [3] = {0.0, 0.0, 0.0};
19  double adSFMColorSpace [3] = {0.0, 0.0, 0.0};
20  double adBGRPixelSet [3] = {0.0, 0.0, 0.0};
21  double adSFRGBPixelSet [3] = {0.0, 0.0, 0.0};
22  double adSFBGRPixelSet [3] = {0.0, 0.0, 0.0};
23  double dFactor = 1.0;
```

As a first step, the variables that are needed in the program are defined. In Line 12 the desired image is loaded while in line 13 an empty container is created that has the same size and type as the loaded image. The lines 14 and 15 represent the transformation matrix that is needed for the transformation from BGR to the M color space as seen in Equation 2-19. Since OpenCV reads the image in the BGR and not in the RGB order, the matrix introduced by Miyazaki needed to be altered as above. The inverse transformation matrix from M to RGB color space as seen in Equation 2-21 is represented in the lines 16 and 17. Other important variables used in the transformation process between the two color spaces and in the calculation of the specular free image are defined in the lines 18-22. The array `adMColorSpace` actually contains the variables $m_1(p), m_2(p)$ and $m_3(p)$ from Equation 2-19. The array `adSFMcolorSpace` contains the variables $\hat{m}_1(p), \hat{m}_2(p)$ and $\hat{m}_3(p)$, `adBGRPixelSet` contains $b(p), g(p)$ and $r(p)$, `adSFRGBPixelSet` has $\hat{r}(p), \hat{g}(p)$ and $\hat{b}(p)$ and `adSFBGRPixelSet` has $\hat{b}(p), \hat{g}(p)$ and $\hat{r}(p)$ from Equations 2-19 to 2-21. The variable `dFactor` is the factor a in Equation 2-20. The reason behind defining all the variables as double is because the transformation matrices contain fractions and square root operations.

```
25  if (image.empty())
26  {     cout << "Error : Image cannot be loaded..!!" << endl;
27  return -1;
28  }
```

The lines 25-28 contain the commands to check whether the image was loaded successfully or not.

```
30  for( int y = 0; y < image.rows; y++ )
31  {
32  for( int x = 0; x < image.cols; x++ )
33  {
34  for ( int iii = 0; iii < 3; iii++)
35  {     adBGRPixelSet[iii] = double(image.at<Vec3b>(y,x)[iii]);}
36
37  adMColorSpace[0] = adTransformMatrix[0][0]*adBGRPixelSet[2] +
38  adTransformMatrix[0][1]*adBGRPixelSet[1]
39  +adTransformMatrix[0][2]*adBGRPixelSet[0];
40  adMColorSpace[1] = adTransformMatrix[1][0]*adBGRPixelSet[2] +
41  adTransformMatrix[1][1]*adBGRPixelSet[1]
42  +adTransformMatrix[1][2]*adBGRPixelSet[0];
```

```
43  adMColorSpace[2] = adTransformMatrix[2][0]*adBGRPixelSet[2] +
44  adTransformMatrix[2][1]*adBGRPixelSet[1]
45  +adTransformMatrix[2][2]*adBGRPixelSet[0];
```

Each pixel in the image is accessed using the commands in the lines 30-35. Then these pixels are transformed from the BGR color space into the M color space according to the Equation 2-19 in the lines 37-45.

```
46  adSFMColorSpace[0] = adMColorSpace[0];
47  adSFMColorSpace[1] = adMColorSpace[1];
48  adSFMColorSpace[2] = dFactor * sqrt(adMColorSpace[0]*adMColorSpace[0]
49  + adMColorSpace[1]*adMColorSpace[1]);
```

Eliminating of the specular reflection components within the M color space as described in Equation 2-20 and storing the result in the `adSFMColorSpace` array is done using the commands in the lines 46-49

```
51  adSFRGBPixelSet[0] = adInverseTMatrix[0][0]*adSFMColorSpace[0] +
52  adInverseTMatrix[0][1]*adSFMColorSpace[1] +
53  adInverseTMatrix[0][2]*adSFMColorSpace[2];
54  adSFRGBPixelSet[1] = adInverseTMatrix[1][0]*adSFMColorSpace[0] +
55  adInverseTMatrix[1][1]*adSFMColorSpace[1] +
56  adInverseTMatrix[1][2]*adSFMColorSpace[2];
57  adSFRGBPixelSet[2] = adInverseTMatrix[2][0]*adSFMColorSpace[0] +
58  adInverseTMatrix[2][1]*adSFMColorSpace[1] +
59  adInverseTMatrix[2][2]*adSFMColorSpace[2];
```

Transformation of the image is performed back from the M color space to the RGB color space as in Equation 2-21 and its values are stored in the array `adSFRGBPixelSet`.

```
61  adSFBGRPixelSet[0] = adSFRGBPixelSet[2];
62  adSFBGRPixelSet[1] = adSFRGBPixelSet[1];
63  adSFBGRPixelSet[2] = adSFRGBPixelSet[0];
```

The commands in lines 61-62 convert from RGB to BGR by swapping the B and the R components in order for OpenCV to display the image correctly.

```
65  for( int c = 0; c < 3; c++ )
66  {
67  SFImage.at<Vec3b>(y,x)[c] = saturate_cast<uchar>(adSFBGRPixelSet[c]);}
68  }
```

It is important to make sure that the resulting image contains valid RGB values since all the transforming operations are done using double data type variables; Lines 65-68.

```
71  namedWindow("Original Image", CV_WINDOW_AUTOSIZE);
72  imshow("Original Image", image);
73
74  namedWindow("Specular Free Image", CV_WINDOW_AUTOSIZE);
75  imshow ("Specular Free Image", SFImage);
```

The lines 71-75 create two separate windows and display the original image and the specular free image in each of them.

```
77  waitKey(0);
78  destroyWindow("Original Image");
79  destroyWindow("Specular Free Image");
80
81  return 0;
82  }
```

The wait for any key to be pressed before closing the two windows and terminating the program is executed by the commands in lines 77-82.

3.2 Integrating The Code into ROS STIFF-FLOP Arm Detection Package

Implementing the algorithms in ROS and integrating them into the desired package require that the code has to be modified a bit in order to be compatible with the other functions in the TubeDetect package.

3.2.1 Shen and Cai's Method

Two main changes need to be applied here. First, the variable names in order to match the corresponding names in the original code of the package. Secondly, a function inside the main code was created which is called first and before any other in the image processing section. That is because the highlights removal is a pre-processing step before passing the image to the main processing procedures.

Evaluation and testing of the performance of this method in addition to the quality of the achieved results are shown in the next chapter.

3.2.2 Miyazaki's Method

Integrating the code into ROS followed similar steps as mentioned above in the previous method but with some additions. The additions include adding an enable/disable option for the highlight removal step. This is useful when there are no large amounts of highlights or not at all in the image that are viewed and examined. Skipping this pre-processing step will increase the speed of the entire process. The second addition is adding a sliding bar to provide the ability to control the arbitrary factor a. Since changing this factor will remarkably change the saturation of the resulted image, it is very useful when the image contains several colors and degrees of highlights. It can be adjusted according to the input image to get the most suitable desired output. The third addition was being inserted after noticing a considerable change in the colors when there are large amounts of specular reflection components in the image. It is demonstrated by combining the original image that has the original colors with the pre-resulted image with a certain ratio that can be controlled easily using a sliding bar.

These three additions were merged with the TubeDetect ROS package settings using the ROS package Dynamic Reconfigure. The following figure 2 shows these three options alongside the ones that are related to the TubeDetect package as it is displayed in the Reconfigure window.

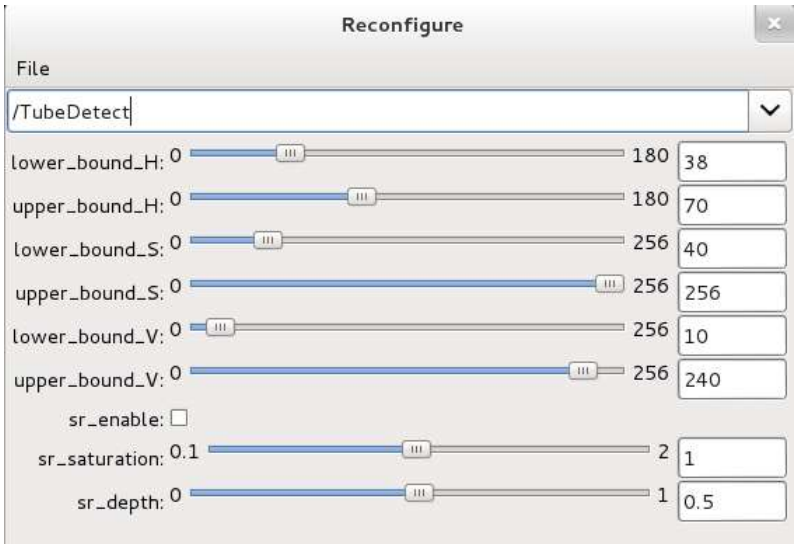


FIGURE 2 DYNAMIC RECONFIGURE PACKAGE SCREEN CAPTION

The last three options sr_enable, sr_saturation and sr_depth are the ones that are related to the highlights removal algorithm. They are set to their default values when launching the package with sr_enable being unchecked, sr_saturation having the value 1 and sr_depth at 0.5.

The sr_enable can easily checked and unchecked to turn the algorithm on and off. The sr_saturation which represents the arbitrary factor a has the range from 0.1 to 2. It is clear that it cannot be set to 0 since the factor is in the denominator, according to Equation 2-20. Setting it to a value higher than 2 will result in an image with a hugely different saturation than the original, which is not useful in our case. The sr_depth has the range from 0 to 1 and it actually represents the weight of the original image in the combination process to produce the final output. The combining process is done using the command `addWeighted`. The weight of the pre-resulted image is being kept at 100% or 1 while the weight of the original is adjusted. The following table describes this process briefly.

TABLE 1 SR_DEPTH VALUES

sr_depth	0	0.25	0.5	0.75	1
Weight of original image	0%	25%	50%	75%	100%
Weight of pre-resulted image	100%	100%	100%	100%	100%

Using this technique helped significantly enhancing the resulted image colors while keeping the same geometry and the highlight still being removed. It is noted that all these three settings can be changed in real-time during the execution of the package.

They will keep their values as long as the ROS Core is on and will be reset to their defaults when the ROS Core is turned off.

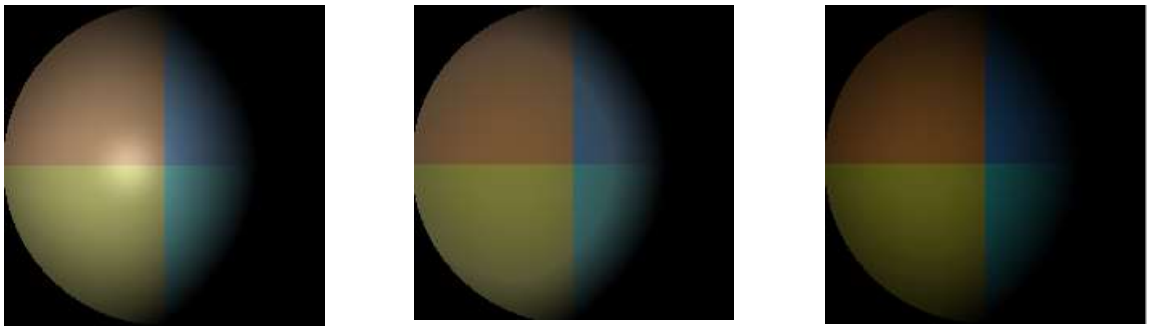
4 TESTING AND RESULTS

4.1 IMAGE RESULTS

Both methods were tested on sample images as a standalone C++ code without integrating it or any part of it into any ROS package. Since experimenting with images is done to determine the execution time and the efficiency of each method, it was sufficient to do so.

The samples collected from the sources [15] [26] [29] [32] [33] are a set of nine images representing different objects with different geometries, colors and degree of highlights. Next, a comparison between the original image and the result image using both methods is shown. Hereby, the original image is presented on left side, the resulted using Shen and Cai's method in the middle and the resulted using Miyazaki's method on the right side.

In Figure 3 we can see a sphere with four different colors and a spot of highlights almost in the middle. Both methods performed very well in this case and removed all the specular reflection components off the surface. In shen and Cai's method (3-b) the colors have been slightly altered but yet very close to the original. There is a strip that can be noticed and exists along the area that lies between the dark and the bright side of the object. The colors there have a different intensity perhaps due to the sudden change in brightness in the original image. Miyazaki's method (3-c) also managed to eliminate all the highlights. The resulted image appears darker than the original one and the color's intensity has changed as well.



(a) (b) (c)

FIGURE 3 SPHERE

In Figure 4 a continuous line of specular reflection can be seen along the four tapes. Each tape has a different color. Both methods show similar results, removing most of the highlights with some remnants. This is probably due to the high degree of specularity in that area.

(a) (b) (c)

FIGURE 4 TAPE

The scene in Figure 5 presents a harder challenge to both methods with more complicated geometry and colors for the objects. In both cases the highlights were completely removed. The color intensity difference can be noticed in (4-c) while in (4-b) it almost was kept the same.

(a) (b) (c)

FIGURE 5 FRUIT

The pear in Figure 6 and the head in Figure 6 test the methods when there is a single colored object in the scene. Both managed to eliminate the specular reflections with difference in color intensity that is noticeable in the head. The resulting image appears darker in (5-b) and brighter in (5-c).

(a) (b) (c)

FIGURE 6 PEAR

(a) (b) (c)

FIGURE 7 HEAD

With the group of plastic objects in Figure 7 both methods removed highlights completely. Shen and Cai's method (7-b) almost kept the original colors while Miyazaki's (7-c) altered the intensity.

(a) (b) (c)

FIGURE 8 PLASTIC

In the fish example Figure 8, the highlights were removed perfectly by both methods and the original colors were kept as in the original.

(a) (b) (c)

FIGURE 9 FISH

The group of toys in Figure 9 is another challenge. All the specular reflection components were removed, but the slight change in the color intensity in (9-c) is visible.

(a) (b) (c)

FIGURE 10 TOYS

The last image represents a painting rolled on its side to create a reflective surface. Since there are white areas in the original image, the two methods are not expected to perform well. White areas can be easily mixed with highlights areas and the methods might not be able distinguish between them. The specular reflections were removed completely but with a loss of the original white color which turned to a yellow like color in the results. The other colors were kept nearly unchanged.

(a) (b) (c)

FIGURE 11 PAINT

After viewing the results of the both methods, it's time to test how long the execution time is for each. It is known from previous discussion that both methods depend on the size if the image in pixels. This will lead to a slower processing time as the image gets

larger. The sample images were not very large; therefore usually the execution time was pretty short.

Following shows a table with the execution time of each method for each image as a standalone C++ code, and not included in any ROS package.

TABLE 2 RUNNING TIME COMPARISON

File	Size (pixels)	Execution Time (s)	
		Shen & Cai's Method	Miyazaki's Method
Sphere	200x200	0.04	0.03
Tape	637x468	0.10	0.08
Fruit	637x468	0.10	0.08
Pear	450x560	0.10	0.09
Head	190x287	0.04	0.04
Plastic	637x468	0.10	0.08
Fish	640x480	0.10	0.08
Toys	353x387	0.06	0.05
Paint	400x800	0.11	0.09

It can be clearly noticed that the running time depends only in the size of the image in pixels. The complexity of the geometry of the scene has no effect at all. This can be a huge advantage since the size of the image is controllable while the environment or the content of the scene is not predictable. The running time can be reduced by reducing the image size or by using a faster CPU.

The execution time of Miyazaki's method is on average and for all nine images 17% shorter than the execution time for the method presented by Shen and Cai. This is due to the shorter and the lower amount of calculations involved. Both methods alter the intensity color of the original images in some scenes with different degrees.

The reason behind the change in the color intensity for Shen and Cai's method is caused mainly by the threshold that was added in Equation 2-18 to create the modified specular free image. The purpose of this threshold is to restore the color and the brightness loss in the image after applying the subtracting procedure in Equation 2-10.

The value of this threshold remains constant after being calculated for the image as in Equation 2-17. The value of this threshold might not in all cases be able to restore the original color intensity. This occurs when the first condition in Equation 2-18 is met $V_{min,p} > T_v$ making $\tau(p) = T_v$. Producing the modified specular free image would require using Equation 2-15 to add the pixel offset to the specular free image. If this calculated pixel offset $\tau(p)$had a much smaller value than the minimum RGB pixel value $V_{min}(p)$ then it would lower the color intensity of that pixel in the produced modified specular free image. A possible solution for this issue might be to alter the value of the factor ϑ to change the value of the threshold T_v and the pixel offset $\tau(p)$.

By looking at Equation 2-20 for Miyazaki's method, the change of the color intensity can be traced down to the third term $\hat{m}_3(p) = a\sqrt{m_1(p)^2 + m_2(p)^2}$. The first two terms do not modify anything during the calculation of the specular free image. The procedure in the third term is essential to generate the specular free image in the M color space. The term $m_3(p)$ in the M color space is actually related to the intensity of the RGB image. From Equation 2-19, the three values of Hue, Saturation and intensity can be calculated as follows [27]:

$$\text{Hue} = arctan\frac{m_2(p)}{m_1(p)} \qquad \text{EQUATION 4-1}$$

$$\text{Saturation} = \sqrt{m_1(p) + m_2(p)} \qquad \text{EQUATION 4-2}$$

$$\text{Intensity} = m_3(p) \qquad \text{EQUATION 4-3}$$

It is clear that the intensity of the RGB image is equal to the term $m_3(p)$ in the M color space and any change to its value will lead to a change by default in the color intensity of the RGB image.

Calculating the specular free image in the M color spaces as in Equation 2-20 requires keeping the two terms $m_1(p)$ and $m_2(p)$ as they are by performing $\hat{m}_1(p) = m_1(p)$ and $\hat{m}_2(p) = m_2(p)$. According to this, the hue and saturation of the image does not change. The third term however involves calculating a new value for $\hat{m}_3(p)$. This new value is $\hat{m}_3(p) = a\sqrt{m_1(p)^2 + m_2(p)^2}$ and is not equal to $m_3(p)$. This will produce the specular free image when converted back to the RGB color space with the same values of hue and saturation but different value for the intensity.

The degree of the intensity change in the specular free image in the RGB color space depends on the three variables a, $\hat{m}_1(p)$ and $\hat{m}_2(p)$. The last two are image dependent while the factor a is a controllable parameter and can be adjusted using the option sr_satuartion in the Reconfigure Package in ROS. The expected color intensity change is described by Miyazaki [27] as when there are green objects and red objects in the scene, they will appear green and red respectively in the resulted image. But when there are light green objects and dark green objects, they will appear having the same green color in the output image.

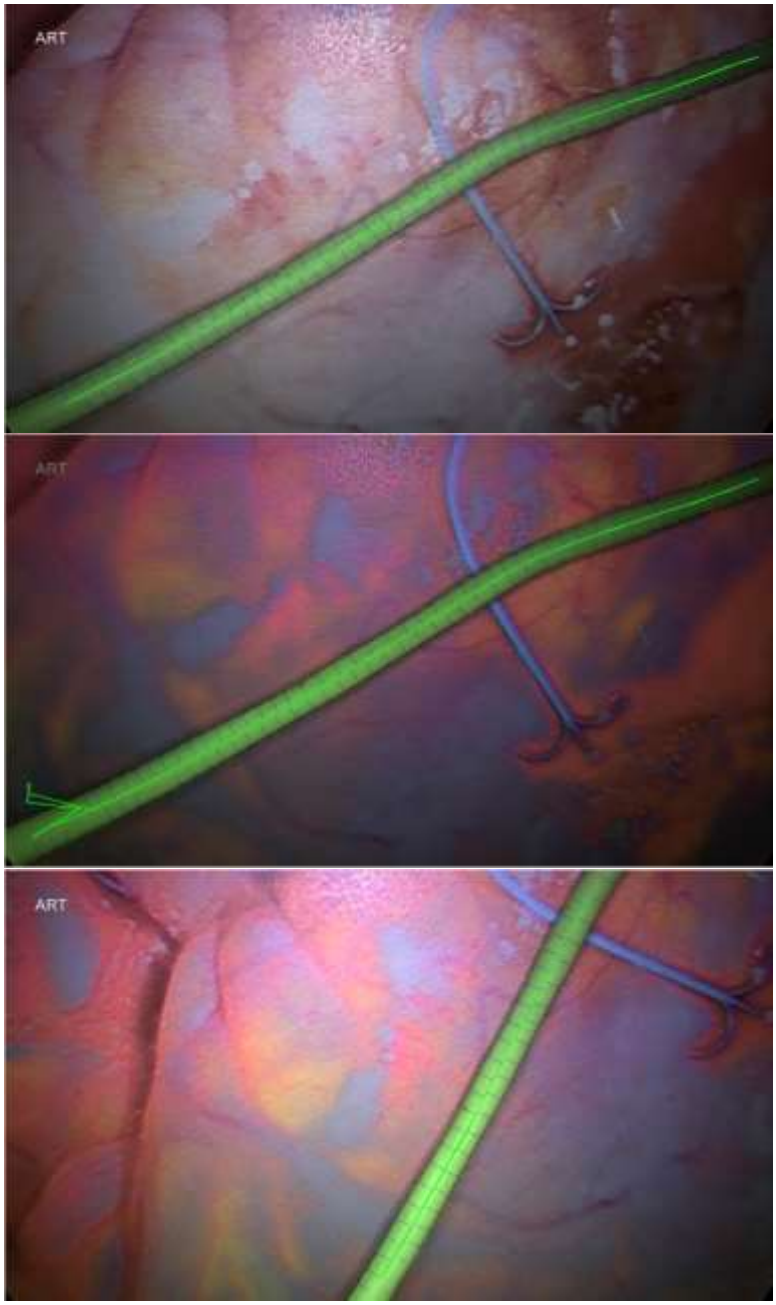
To reduce the effect of the color change even more, the method introduced in section 3.2.2 was implemented by combining the original image with the specular free image generated by Miyazaki's method to introduce the final output image.

4.2 VIDEO RESULTS

Speed is a very important aspect when applying the selected algorithm on a video stream. Since both methods show very similar results, it makes more sense to use Miyazaki's method when it comes to videos in order to achieve a real-time processing or as much as close to it. Shen and Cai's method will be tested in order to compare the results.

As previously mentioned, there are two video streams for testing purposes. The first represents the ideal case in which a weak presence of specular reflection components. The second one contains much stronger highlights and can represent the worst case to face.

Starting with Miyazaki's method, both Figures 12 and 13 show preview of screenshots taken from the resulting output video stream. Note that, it is possible to try setting different values for the specularity removal depth and saturation factor in real-time to get the best desired results.

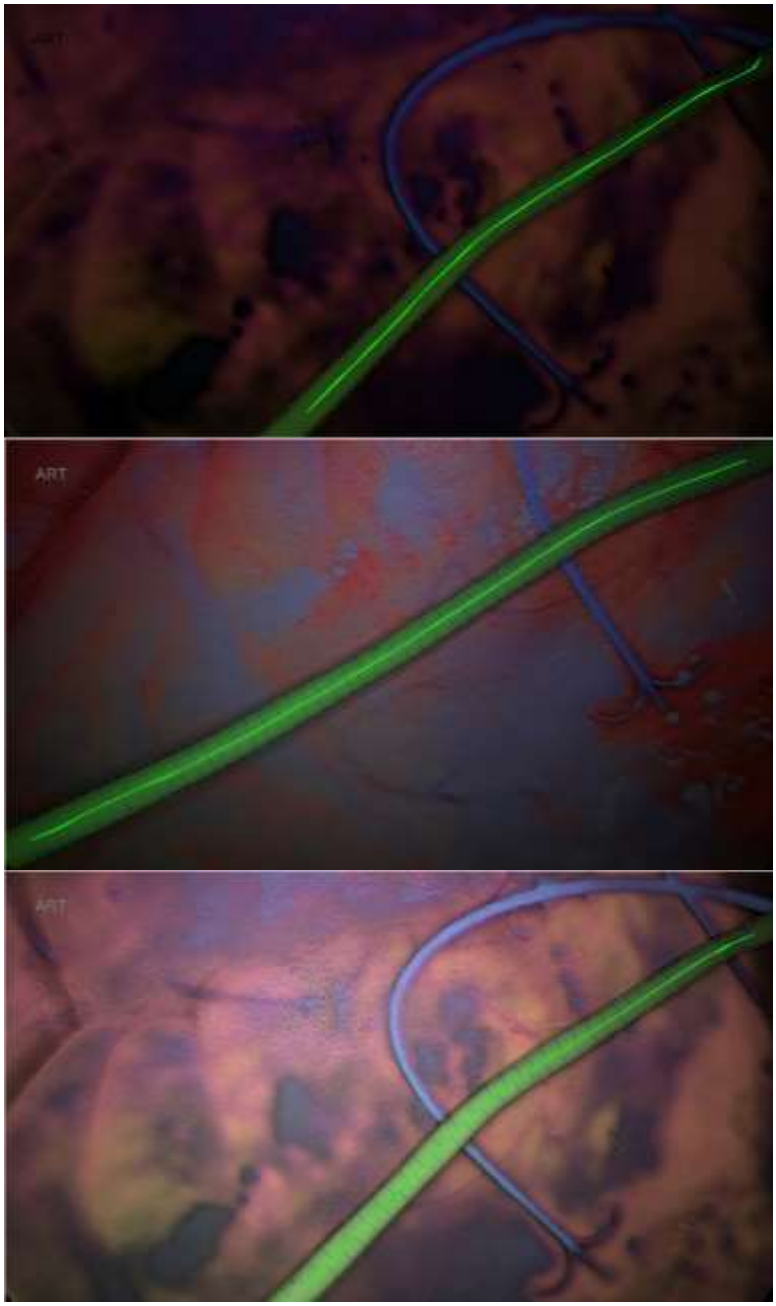


(a)
Specularity removal mechanism is turned off showing the original video stream.

(b)
The mechanism is turned on at default values, depth =0.5, saturation factor=1.

(c)
The mechanism is turned on at depth =0, saturation factor=1.

FIGURE 12 LIGHT HIGHLIGHTS VIDEO MIYAZAKI 1



(a)
The mechanism is turned on at depth =1, saturation factor=1.

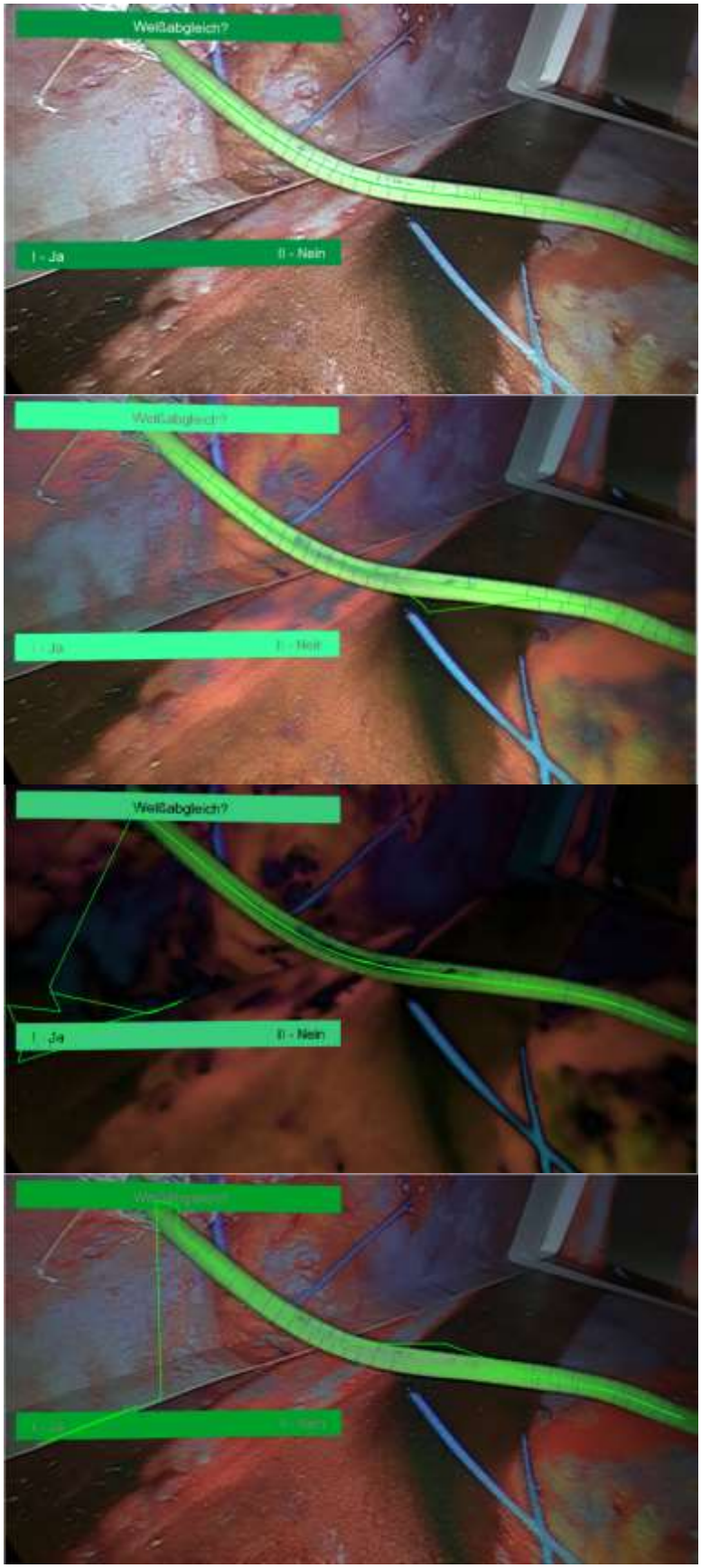
(b)
The mechanism is turned on at depth =0.5, saturation factor=0.1.

(c)
The mechanism is turned on at depth =0.5, saturation factor=2.

FIGURE 13 LIGHT HIGHLIGHTS VIDEO MIYAZAKI 2

The resulting stream appears darker than the original with a change in the intensity color values in each case. The geometry was maintained without any alteration. The reason behind the notable change in the color intensity is probably because the background contains surfaces and objects that have white color, or because the gamma value in the camera was not set to 1.

Then the performance when heavy highlight components exist in the video stream will be tested. The following Figures 14, 15 and 16 show the results. Because the video stream contains different levels of highlights at different frames, three main cases were set. Case 1 is where the specular reflection is high. Case 2 is where it is very high and case 3 where the specular reflection is extremely high.



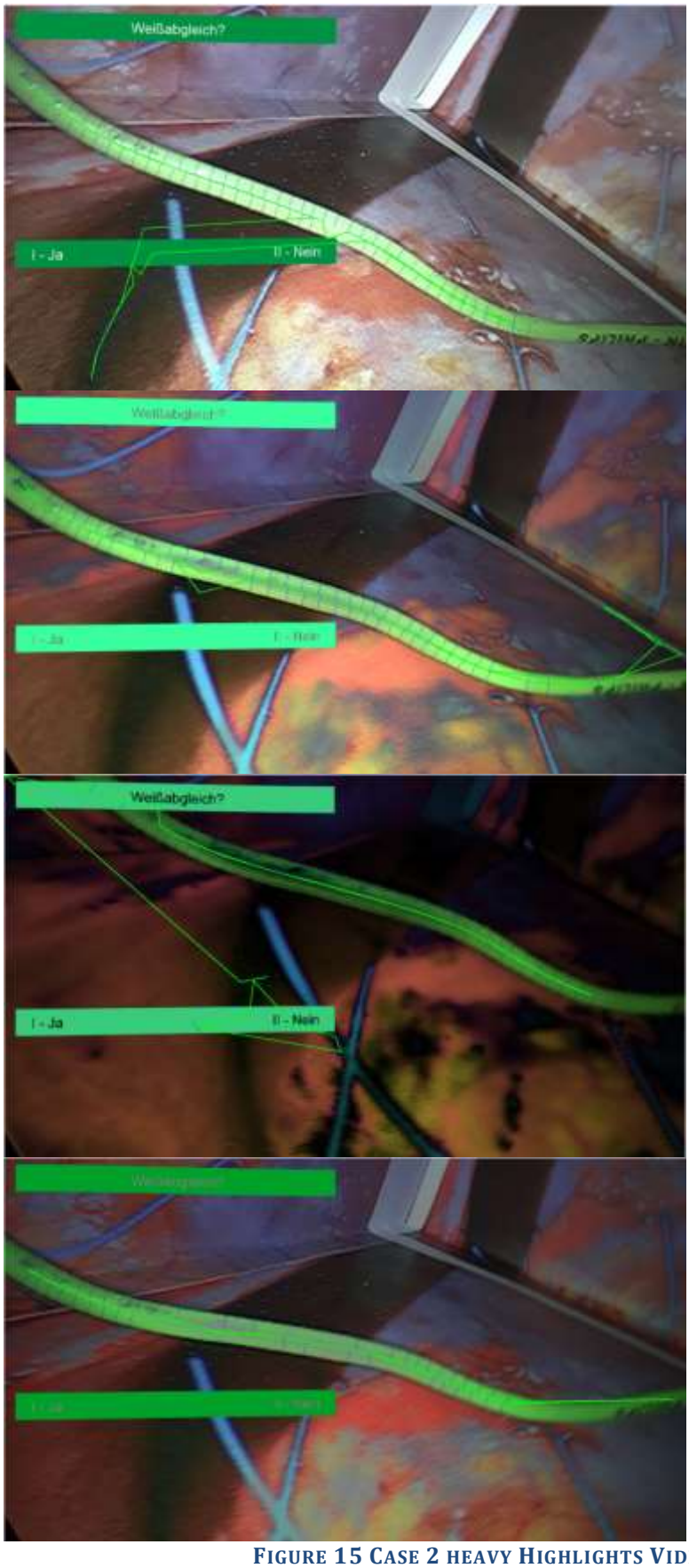
(a)
Case 1. Specularity removal mechanism is turned off showing the original video stream.

(b)
Case 1. The mechanism is turned on at default values, depth =0.5, saturation factor=1.

(c)
Case 1. The mechanism is turned on at depth =1, saturation factor=1.

(d)
Case 1. The mechanism is turned on at depth =0.5, saturation factor=0.1.

FIGURE 14 CASE 1 HEAVY HIGHLIGHTS VIDEO MIYAZAKI



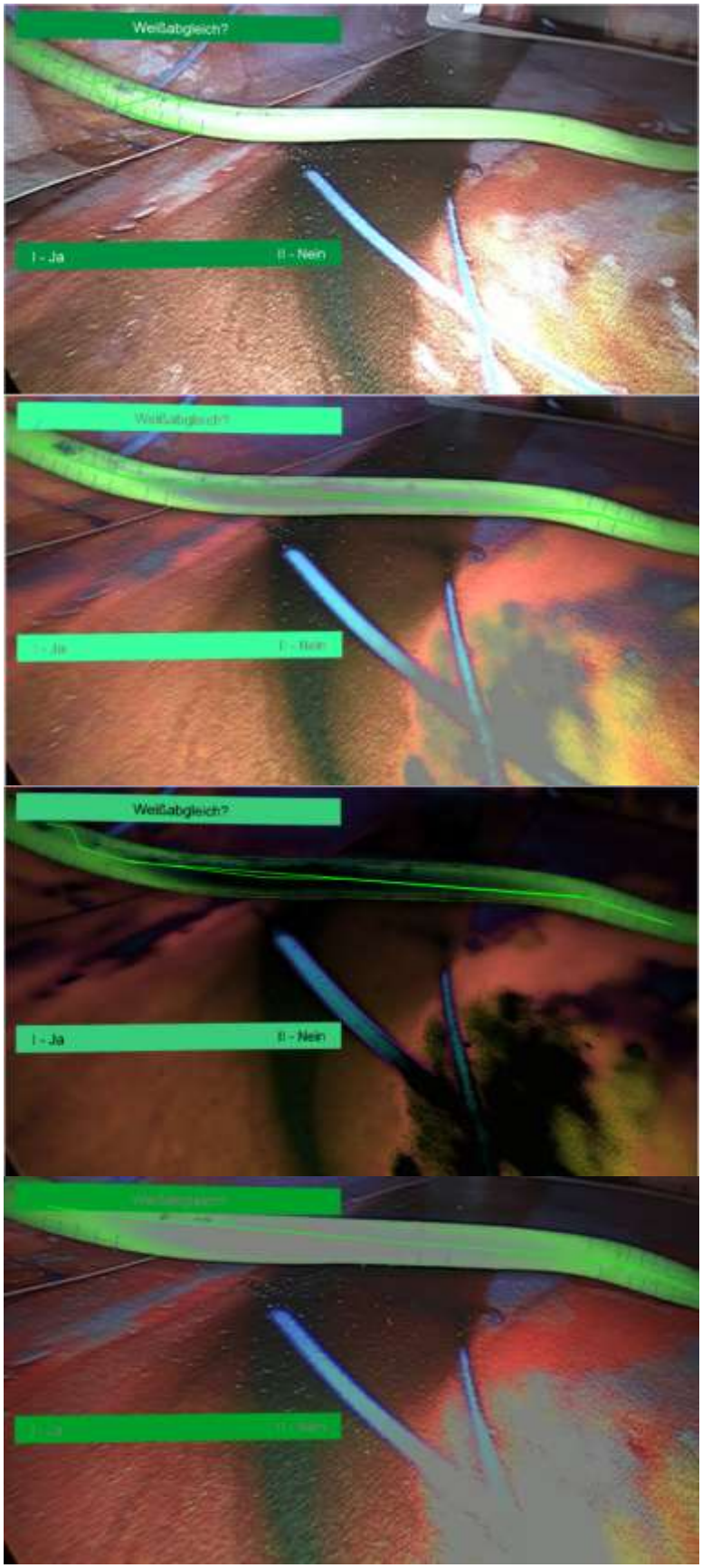
(a)
Case 2. Specularity removal mechanism is turned off showing the original video stream.

(b)
Case 2. The mechanism is turned on at default values, depth =0.5, saturation factor=1.

(c)
Case 2. The mechanism is turned on at depth =1, saturation factor=1.

(d)
Case 2. The mechanism is turned on at depth =0.5, saturation factor=0.1

FIGURE 15 CASE 2 HEAVY HIGHLIGHTS VIDEO MIYAZAKI



(a)
Case 3. Specularity removal mechanism is turned off showing the original video stream.

(b)
Case 3. The mechanism is turned on at default values, depth =0.5, saturation factor=1.

(c)
Case 3. The mechanism is turned on at depth =1, saturation factor=1.

(d)
Case 3. The mechanism is turned on at depth =0.5, saturation factor=0.1

FIGURE 16 CASE 3 HEAVY HIGHLIGHTS VIDEO MIYAZAKI

Looking at the Figures (14-a) (15-a) and (16-a) we can easily identify the three cases. The image in (16-a) has a remarkably high amount of specular reflection components on both the tube, being focused intensely in the middle, and on the surroundings.

In the results (14-b) (15-b) and (16-b) it can be seen the some of the original color under the highlights were restored while in other areas were turned into grey, especially in the areas where the highlights are really intense; case (16-b). The resulted image appears a little darker than the original one as well.

In (14-c) (15-c) and (16-c) much darker image has been observed and a larger change of colors. This is due to the sr_depth is set to zero, which means that the pre-resulted image is displayed without combining it to the original one to perform color restoration. The grey areas that appeared previously are now black.

The case (14-d) (15-d) and (16-d) show more promising results with the color being fully restored with even a calmer tone. More of the areas under the highlights were recovered, but still not all of them. Case (16-d) still represents a big challenge for the algorithm to eliminate the huge amount of specular reflection components. Grey color can be seen in these areas as in case (16-b).

The user has the freedom to change the controllable variables in the allowed range according to the needs and the current scene to obtain the best results. Each scene differs from the other and therefore, each might have different values and settings. Being able to change them in real-time is an advantage where the output can be observed directly on the screen without the need to restart anything.

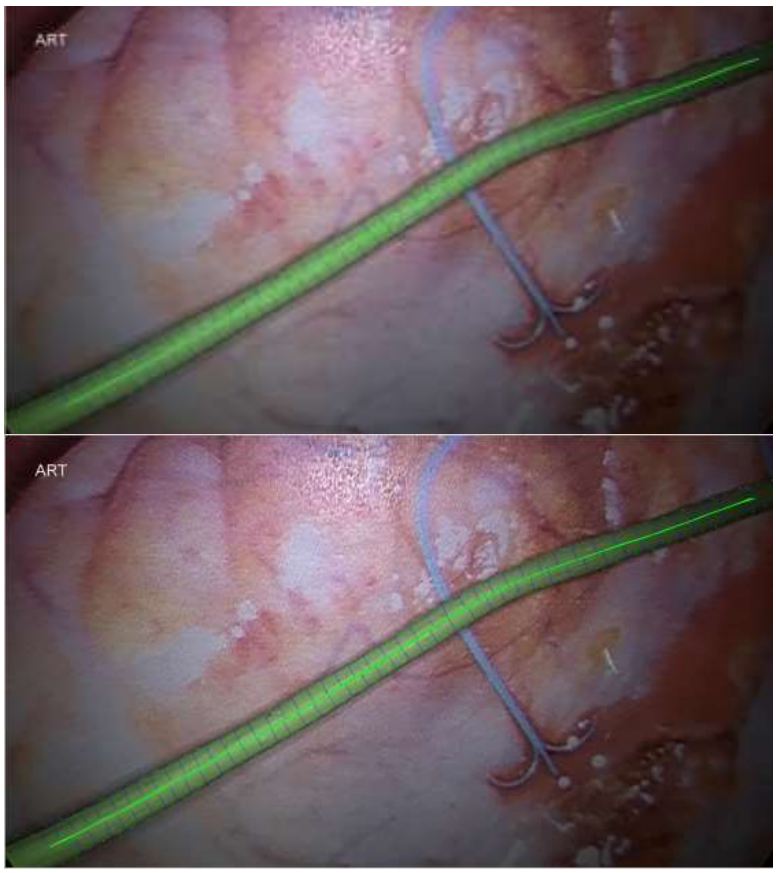
The reason for having gray areas in the output screen is a side effect of the two main steps performed when applying this method. The first step is calculating the pre-resulted specular free image using Miyazaki's method itself. The second step is producing the final image by merging the pre-resulted image with the original image to enhance the colors as shown in Table 1.

The first step involves using Miyazaki's calculations to produce the pre-resulted image. The original video frame contains areas having heavy specular reflection components that appear close to the color white. One of Miyazaki's conditions is that the image must not contain any white surfaces. If the image does contain any white surfaces, the algorithm will not be able to completely eliminate the specular reflections resulting in black color spots in these areas. He provided some sample results on his website for when his method fails in removing highlights [27].

These white areas cause a problem. The original image has white color in these areas. The pre-resulted image after applying Miyazaki's calculations produced an image with black color in those areas. The color enhancing method applied earlier in section 3.2.2 combines the original image with the pre-resulted image at adjustable weights. Combining the white color with the black color in these areas produces the gray color

visible in the final image. Adjusting the parameter sr_depth will alter the weight at which the two images are combined. Changing the value of this weight sway the ration between the white and the black color and thus altering the intensity of the gray color.

For testing Shen and Cai's proposal, the same videos were used. The following Figure 17 shows the result of the ideal case compared to the original video.



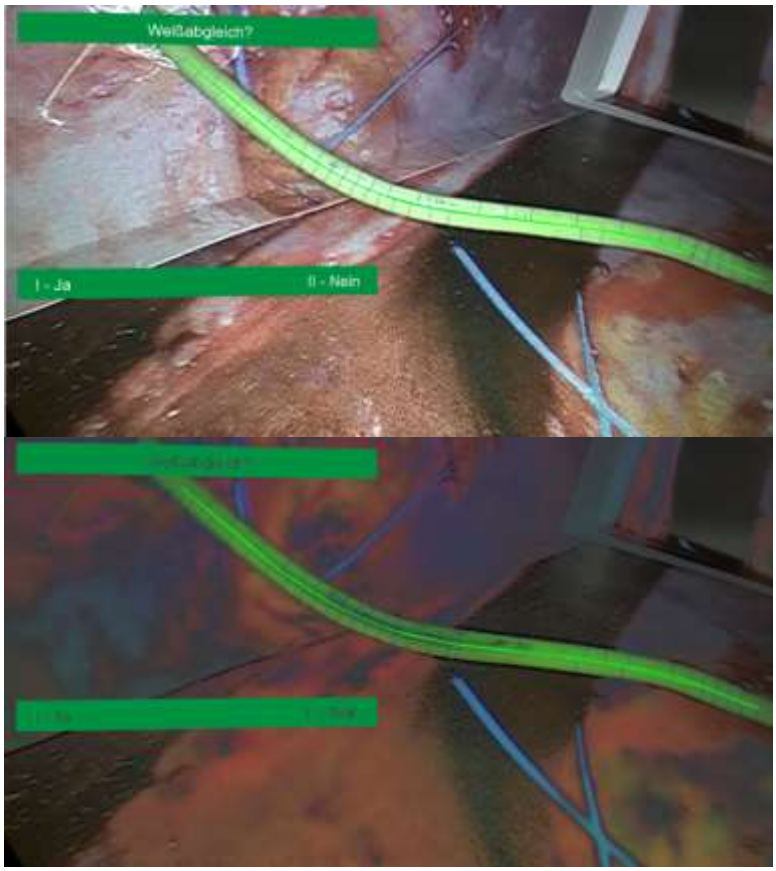
(a)
The original videos stream.

(b)
The algorithm is turned on.

FIGURE 17 LIGHT HIGHLIGHTS VIDEO SHEN & CAI

An interesting observation here is that both video streams show exactly the same image, as if the algorithm did not modify the image at all. This is probably due to the image being free of any remarkable specular reflection components. This can be of an advantage and disadvantage at the same time. Having the image not touched while there are no highlights actually keeps the image as it is providing the original form of it. On the other hand this does not mean that the code is not being executed. The running speed of the program will not increase and will remain as it.

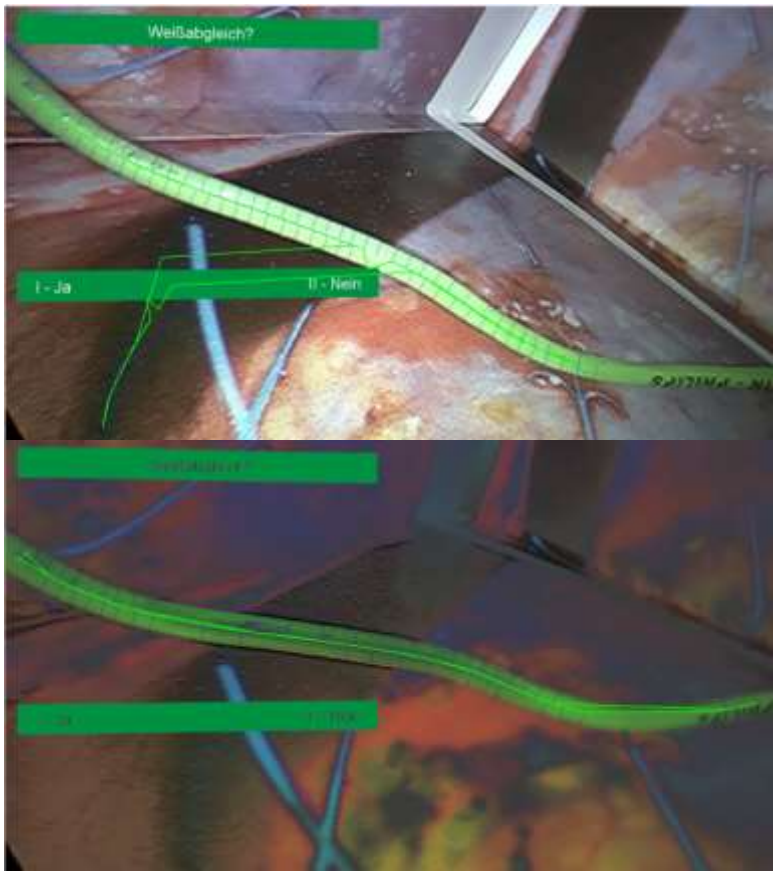
The Figures 18, 19 and 20 display the algorithm in act when the video stream with heavy highlights is fed in.



(a)
Case 1. Original stream

(b)
Case 1. Output stream.

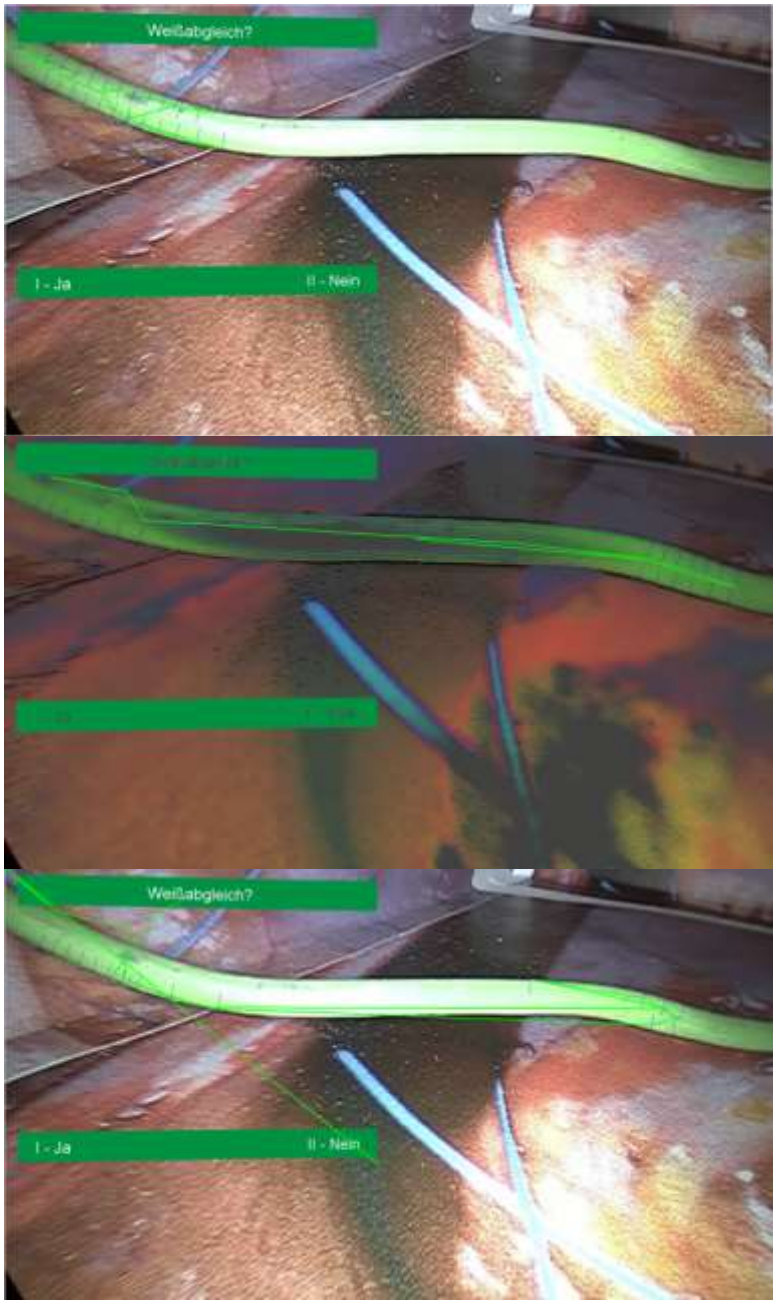
FIGURE 18 HEAVY HIGHLIGHTS VIDEO SHEN & CAI CASE 1



(a)
Case 2. Original Stream.

(b)
Case 2. Output stream.

FIGURE 19 HEAVY HIGHLIGHTS VIDEO SHEN & CAI CASE 2



(a)
Case 3. Original stream.

(b)
Case 3. Output stream.

(c)
Case 3. Output stream with weird behavior.

FIGURE 20 HEAVY HIGHLIGHTS VIDEO SHEN & CAI CASE 3

By looking at (18-b) and (19-b) we can see that the output image appears darker than the original with color alteration as well. Most of the highlights were removed successfully except for some areas appearing in grey color that had strong specular reflections before. This is similar to what was observed with Miyazaki's method.

In Figure (20-b) the challenge is at its peak the colors have been changed as well but with a higher degree. The grey areas are still visible where strong highlights were present. An odd behavior can be seen in (20-c). It occurs when the video frame reaches case 3, the algorithm seems to fail to work and the output displays the original frame image.

The reason behind this behavior can be found in Equation 2-18. The output image appears exactly as the original without any change. The first condition is not met in this case at all. It states that $\tau(p) = T_v$ when $V_{min,p} > T_v$. The threshold would have altered the colors of the image, but in this case it did not. What happened is the opposite; none of the colors has changed. This means that the second condition is met in this case and for all the pixels in the image resulting in generating the original image back as in Equations 2-10, 2-15. The calculated modified specular free image is exactly equal to the original image. The condition in Equation 2-18 states that $\tau(p) = V_{min,p}$ when $V_{min,p} \leq T_v$. Then, for all pixels $\tau(p) = V_{min,p}$ and $V_{min,p} \leq T_v$. This concludes to that the values of the threshold T_v for all pixels are always higher than the values of $V_{min,p}$. The threshold is calculated as in Equation 2-17 by computing the mean and the standard deviation and the factor ϑ. Having a high value of the threshold is a result of a high value of both the standard and the mean deviation. The high values of the standard and the mean deviation result from the image having high values of the calculated minimum values of the RGB channels in each pixel. This is due to the image having a high brightness or large areas of white colors. An image with high brightness has high values of the RGB channels for each pixel. An image with large areas of white colors can be considered a similar case since the RGB channel values of the color white are [255, 255, 255]. High RGB channel values lead to high values f the calculated RGB minimum which leads to high values of the standard and the mean deviation. The high values of the standard and the mean deviation increase the value of the calculated threshold and thus resulting in meeting the second condition in Equation 2-18 for all pixels. This is possible to avoid by decreasing the value of the factor ϑ in the Equation 2-17 below the set value 0.5, which in return will decrease the value of the threshold.

For comparing the running speed for both Miyazaki's and Shen and Cai's methods when processing video streams, a program to measure the frame rate output was used. The test was run on the same machine was used for coding and implementing which was mentioned in the first chapter. Miyazaki's method recorded an average of 20 frames per second while Shen and Cai's method recorded an average of 15 frames per second. This shows a speed advantage of Miyazaki's method over Shen and Cai's method.

4.3 Benefits and Improvements Achieved

After getting complete results from both methods, it is easy now to compare between them and point out the strengths and weaknesses for each one. Miyazaki's method as well as Shen and Cai's method showed good quality results in still images. Though, Miyazaki's method performed better than Shen and Cai's method with video streams in addition for it being simpler and faster.

The light to moderate specular reflections components were removed completely in almost every case. The problem appears when the scene contains really high intensity of the specular reflection components. It becomes really difficult to restore the original surface color behind these spotlights as they appeared like a white colored surface. As

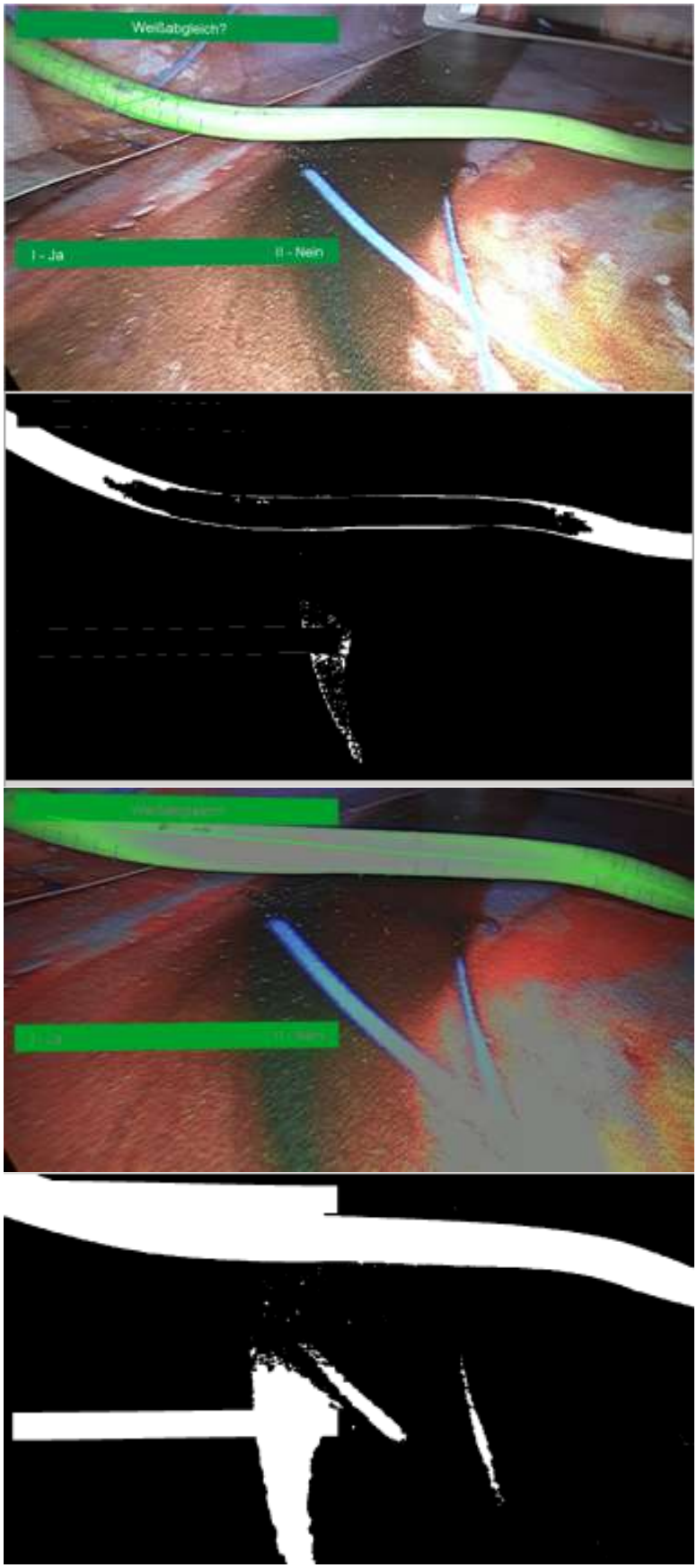
explained earlier, one condition of the method is that specular reflections can't be removed from white objects and therefore the resulted image has gray color in these areas.

In some case and by using the appropriate settings, the output video stream's colors can be even enhanced to get a warmer or softer image as in Figures (14-d) and (15-d). The running speed is nearly real-time for both methods with an advantage to Miyazaki's. This however, can be easily boosted up by using a faster computer to reduce the execution time to the half or even more.

Due to the low running time, less calculation steps and better results, the proposal of Miyazaki has been adopted and applied for the purpose of this work.

The main goal behind removing the highlights in the output video stream is to provide an enhanced image for the purpose of detecting the tube. After detecting the body of the tube, its center is defined along its axis and a green line is drawn on the screen visualizing it as a center line. Studying the effects of the highlight removal technique is important to see how far they contribute to a successful detection and elimination of any unwanted results those were obtained before.

Following, Figure 21 shows the effect of Miyazaki's method that can be seen and compared to the original video stream. Case 3 is considered here since it is the worst case among them all.



(a)
Original stream.

(b)
Black/White output for the original stream.

(c)
Highlights removal is turned on.

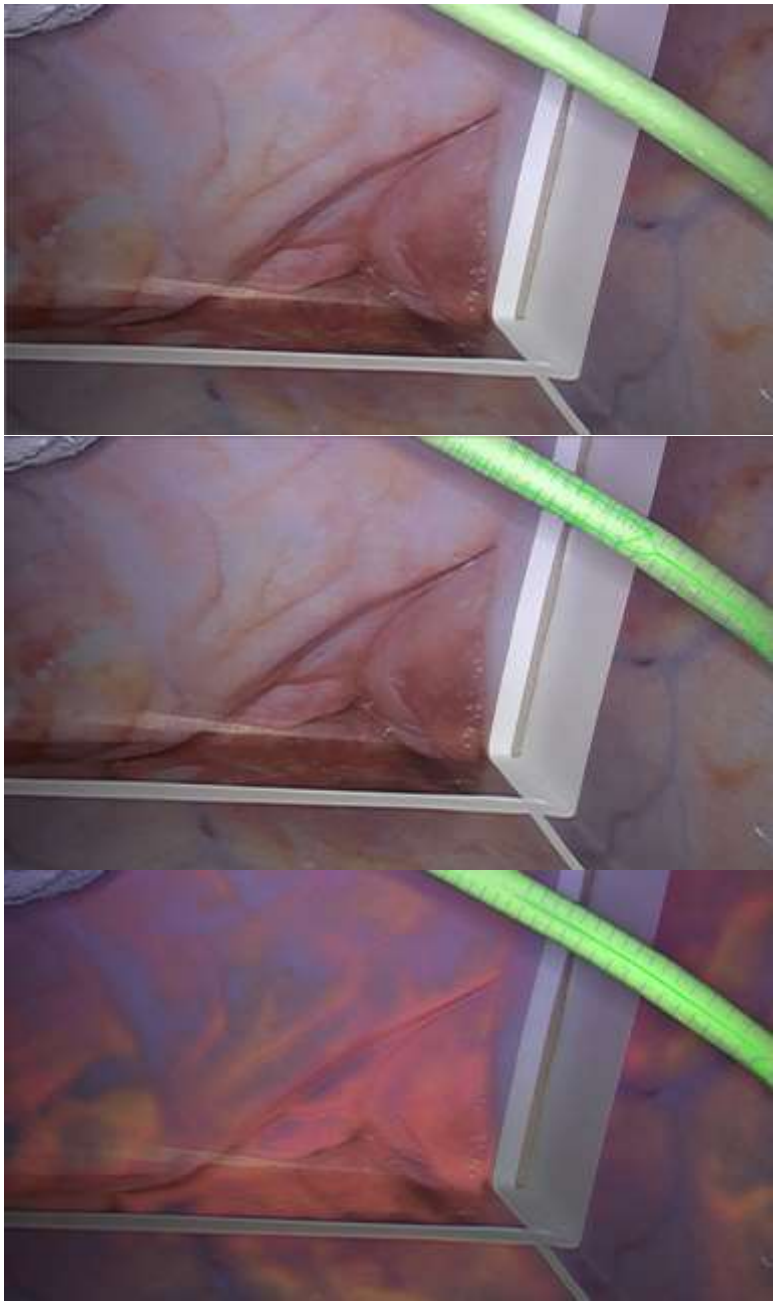
(d)
Black/White output when the highlights removal is on.

FIGURE 21 TUBE DETECTION COMPARISON 1

By examining the previous screen shots it becomes easy to notice the positive effect of the highlights removal technique on the tube detection process. When the highlights removal option is turned off, the resulting black and white image contains some cavities along the detected tube shape. This sometimes causes the program to crash and if not it results in a false detection of the object, as it can be seen in (21-b). But, when the highlights removal is enabled, even when grey areas are present in particular regions; the black and white image that is displayed on the output screen does not contain any cavities and/or deformation of the original shape of the tube, as shown in (21-d).

The center line has automatically been corrected and connected after a clear disconnection, that was caused by the specular reflection, as in (21-a). It is seen in (21-c) how the line is being drawn along the center of the tube.

Another example can be seen in Figure 22. In (22-a), the original video stream is shown with visible highlights along the green tube. When the tube detection algorithm is turned on, it fails to correctly detect the center of the tube and instead considers its two edges as two centers, (22-b). This is problematic and leads to wrong results. The highlight removal algorithm is turned on then and a clear improvement can be noticed. The detection of the tube center is successful and the center line can be seen clearly.



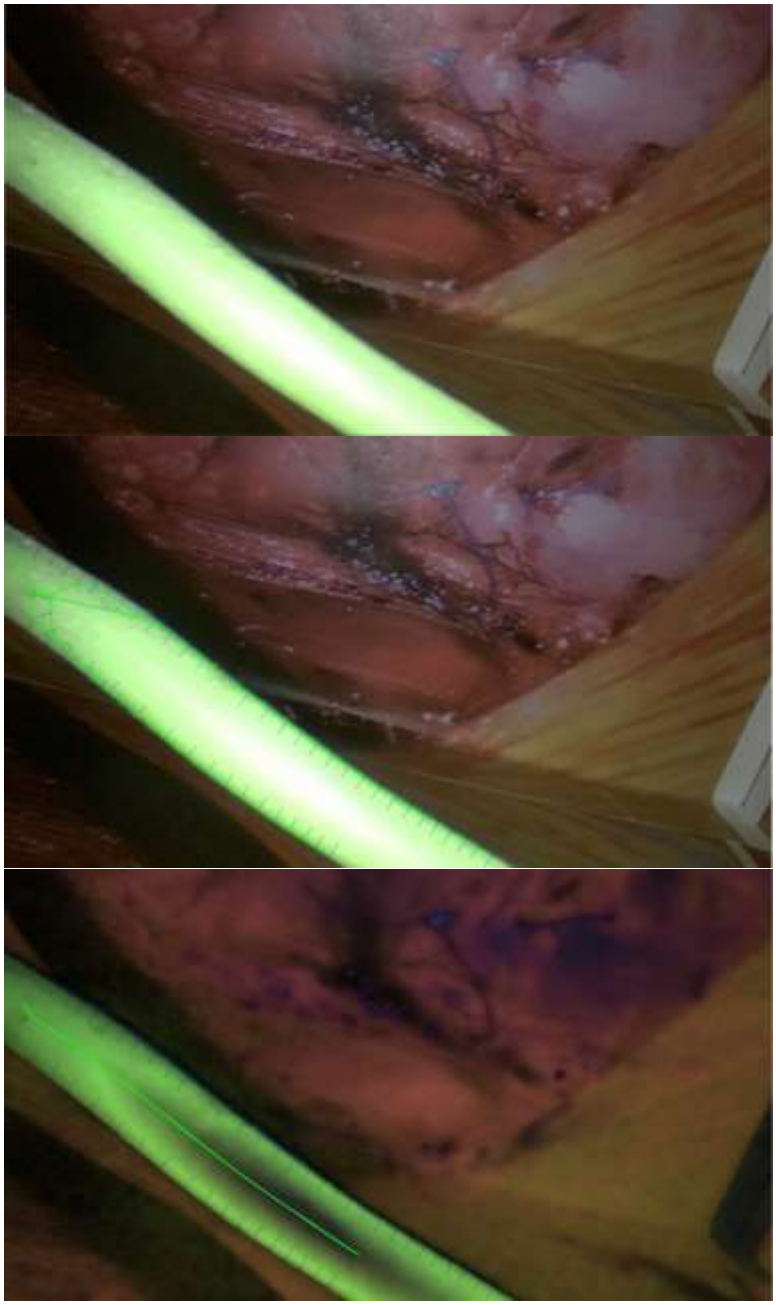
(a)
Original stream.

(b)
Tube detection is on, highlights removal is off.

(c)
Tube detection is on, highlights removal is on.

FIGURE 22 TUBE DETECTION COMPARISON 2

The next Figure 23 demonstrates another example. Strong specular reflection components are found in the scene and along the tube. When the detection algorithm is on and the highlights removal is ticked off in (23-b), the detection fails and two centers of the tube are identified instead of the single original one. After switching on the highlights removal, the algorithm can now successfully detect the true center of the tube, (23-c).



(a)
Original stream.

(b)
Tube detection is on, highlights removal is off.

(c)
Tube detection is on, highlights removal is on.

FIGURE 23 TUBE DETECTION COMPARISON 3

As a result of applying the presented method, some improvements have been achieved and are summed up in the following:

- The main issues of having specular reflection components in the scene was resolved and the output video stream is free of any of them;
- The fail to detect the center of the dummy green tube (manipulator) problem was enhanced after eliminating the highlights. The center is properly detected now and displayed on the monitor as a drawn green line along the tube center;
- The incorrect detection of the tube center by identifying two centers along the edges of the tube instead of the single real center was resolved as well. The algorithm is able now to avoid this inaccurate detection;
- The running speed of the implemented code is in real-time and therefore does not cause lag time between the input and the output stream;
- The color intensity and tone in the output stream were improved in some case and at certain setups as show in Figures (14-d) and (15-d).

5 CONCLUSION AND FUTURE WORK

5.1 SUMMARY

Several methods to eliminate these specular reflection components from images and videos have been proposed. A group of the most common proposals was discussed, studied and compared to show the ability to apply them in the scope of this work. The main aspects for the method selection were the use of a single endoscopic camera, the use of a single input image, ability to process multicolored images, real time capability of the written code and producing satisfying results.

After analyzing and studying, the selection was narrowed down to two proposals; Miyazaki's method and Shen and Cai's method [17] [19]. These two methods are the most suitable approaches to be implemented in this project. They both were analyzed and investigated in detail. They both avoid the use of complicated mathematical operations like iterations or image segmentations, which leads to the increase of the execution time of the code. The use of a single input image and the ability to process multicolored images are properties of both methods as well.

The two proposals are then implemented in C++. The written code is tested using a set of 9 images obtained from different sources. The running time of the code for both methods was compared as well as the achieved results. Both showed good capability of removing the specular reflections from these images at real-time capable speeds. For processing single images, Miyazaki's method was about 17% faster compared to Shen and Cai's method.

The written code has then been merged into the TubeDetect package in the framework of the Robot Operating System (ROS). The package was tested for the pre-recorded two video streams showing the dummy tube inside a simulated inner human body environment. One video represents a case in which the specular reflections are light. The second video represents a condition in which the specular reflections are heavy. This was done to examine the behavior of each method in more than one case. A comparison was then held between the two methods with regard to output quality and running time. Miyazaki's method showed very good results compared to Shen and Cai's. It also runs faster tha Shen and Cai's method.

Based on the achieved results it can be said that using Miyazaki's method has improved and helped the detection of the manipulator using the TubeDetect package even when strong highlights exist. The detection of the manipulator's center is now made possible after having issues identifying the center of the tube due to the highlights in the scene. These highlights caused a false detection of the center of the tube, resulting in achieving inaccurate output. The output stream after applying Miyazaki's method in addition to the color enhancing method enabled the TubeDetect package to successfully identify the center of the tube and visualize the results on the monitor.

However, a slight change in the color intensity of the output image has occurred. This color intensity change is a result of applying Miyazaki's method and can't be avoided. To resolve this issue, two configurable parameters were added to minimize the effect of the color intensity change. The running speed of the implemented code is almost in real time. The output video stream has now color intensity closer to the original stream and free of any specular reflections at the same time.

The total gain of applying the highlights removal technique presented here is that now the detection of the tube center is made possible when there are specular reflection components in the scene. A change in the intensity of the colors occurs and its negative effect is minimized by the use of the configurable parameters those were integrated into the program.

5.2 OUTLOOK

There is a considerable place for future enhancements on the proposed work to improve its functionality and the quality of results. One of them is to try to recover more of the lost color of the surfaces with strong highlights, either by filling operations depending on the neighbor pixels [24] or using a certain unique color to compensate the lost color in these areas. Further image processing techniques can be introduced in to find a way to eliminate the negative effects of having white colored objects in the scene and restore them afterwards. However, applying these improvements might be costly with respect to computational power and running time.

Additional improvement can be done with regard to the execution time. Modifying the written code to enable it to run on a modern GPU (Graphics Processing Unit) for processing proposes instead of the currently used CPU would boost up the running speed enormously. That's because GPUs are capable of performing massive parallel operations simultaneously. [31]

Another model of the manipulator that is very similar to the original one to be used in the real life application was arrived lately after most of the work in this project was done. The main issue with the manipulator is its color. It has a grey surface color which is close to white. To overcome this problem it is required to investigate other methods. For example, the work by Tsuji [25] showed great results for images containing surfaces having white and light colors. It is possible to apply this method if a high speed camera is used in this project in the future.

The algorithm of Miyazaki can be used in any image processing application where the need for removing the specular reflections from images and videos is desired. It is important to pay attention to the environment in which this method is used. It should meet the requirements of the method in order for it to function properly.

6 REFERENCES

1 Horn, B. *Robot Vision.* Cambridge, MA: The MIT Press, 1986.

2 Szeliski, R. *Computer Vision: Algorithms and Applications.* London: Springer-Verlag London Limited, 2011.

3 Nagabhushana, S. *Computer Vision and Image Processing.* New Delhi: New Age International, 2005.

4 STIFF-FLOP Official Site. Accessed: 21/3/2014. http://stiff-flop.eu/.

5 The Medical Dictionary. Accessed: 21/3/2014. http://medical-dictionary.thefreedictionary.com.

6 ROS Official Site. Accessed: 21/3/2014. http://www.ros.org/.

7 ROS Documentation Wiki Site. Accessed: 21/3/2014. http://wiki.ros.org/.

8 Stehle, T. *Removal of Specular Reflections in Endoscopic Images*. Czech Technical University Publishing House, Acta Polytechnica Vol. 46 No. 4/2006. 2006.

9 Myers, R. L. *The Basics of Physics*. Westport, CT: Greenwood Publishing Group, 2006.

10 Serway, R. A., Jewett, J. W. *Principles of Physics: A Calculus-Based Text fourth edition*. Scarborough, ON: Nelson Education, 2006.

11 Reis, M. Reflection of Light Rays: A Simple Diagram, Deflection of Light Rays: A Simple Diagram. Wikimedia Commons. Accessed: 21/3/2014. http://commons.wikimedia.org/wiki/User:Marcelo_Reis.

12 Shafer, S. A. *Using Color to Separate Reflection Components*. Rochester, NY: Computer Science Department, University of Rochester. 1984.

13 Klinker, G. J., Shafer, S. A. and Kanade, T. *Using a Color Reflection Model to Separate Highlights from Object Color*. Proceedings First International Conference on Computer Vision, pp. 145-150. 1991.

14 Kurachi, N. *The Magic of Computer Graphics*. Boca Raton, FL: CRC Press. 2011.

15 Tan, R. T., Ikeuchi, k. *Separating Reflection Components of Textured Surfaces Using a Single Image*. IEEE Transactions Pattern Analysis and Machine Intelligence, Vol. 27, No. 2. 2005.

16 Nayar, S. K. Fang, X. S. and Boult, T. *Separation of reflection components using color and polarization*. International Journal of Computer Vision, Vol. 21, Issue 3, pp. 163-186. 1997.

17 Miyazaki, D., Tan, R. T., Hara K. and Ikeuchi K. *Polarization-based Inverse Rendering from a Single View*. Proceedings of International Conference on Computer Vision, pp. 982-987. 2003.

18 Yoon, K. J., Choi, Y. and Kweon, S. I. *Fast Separation of Reflection Components using a Specularity-Invariant Image Representation*. Image Processing, IEEE International Conference, pp. 973-976. 2006.

19 Shen, H. L., Cai, Q. Y. *Simple and efficient method for specularity removal in an image*. Applied Optics, Vol. 48, Issue 14, pp. 2711-2719. 2009.

20 Tan, R. T., Ikeuchi, K. and Nishino, K. *Separating Reflection Components Based on Chromaticity and Noise Analysis*. IEEE Transactions Pattern Analysis and Machine Intelligence, Vol. 26, No. 10. 2004.

21 Mallick, S. P., Zickler, T., Belhumeur, P. N. and Kriegman, D. J. *Specularity Removal in Images and Videos: A PDE Approach*. IEEE Conference on Computer Vision and

Pattern Recognition, Vol. 2, pp. 619–626. 2005.

22 Tan, R. T., Ikeuchi, k. *Reflection components decomposition of textured surfaces using linear basis functions*. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 1, pp. 125-131. 2005.

23 Park, J. W., Lee, K. H. *Inpainting Highlights Using Color Line Projection*. Transactions on Information and Systems, Vol. E90-D Issue 1, pp. 250-257. 2007.

24 Das, A., Kar, A. and Bhattacharyya, D. *Elimination of Specular Reflection and Identification of ROI: The First Step in Automated Detection of Cervical Cancer Using Digital Colposcopy*. IEEE International Conference on Imaging Systems and Techniques, pp. 137-241. 2011.

25 Tsuji, T. *An Image-Correction Method for Specular Reflection Removal Using a High-Speed Stroboscope*. 37th Annual Conference on IEEE Industrial Electronics Society, pp. 4498 – 4503. 2011.

26 Yang, Q., Wang, S. and Ahuja, N. *Real-Time Specular Highlight Removal Using Bilateral Filtering*. Proceedings of the 11th European conference on Computer vision, Part IV, pp. 87-100. 2010.

27 Miyazaki, D. Specular-Free Image. Hiroshima City University. Accessed: 21/3/2014. *http://www.cg.info.hiroshima-cu.ac.jp/~miyazaki/knowledge/teche40.html*.

28 Lee, H. C., Breneman, D. J. and Schulte, C. O. *Modeling Light Reflection for Computer Color Vision*. IEEE Transactions Pattern Analysis and Machine Intelligence. Vol. 12, pp.402–409. 1990.

29 Computational Vision Laboratory, School of Computer Science, Simon Fraser University. Accessed: 21/3/2014. *http://www.cs.sfu.ca/~colour/data/colour_constancy_test_images/index.html*.

30 Miller, G. J., Yang, K. *Handbook of Research Methods in Public Administration, Second Edition*. Boca Raton, FL: CRC Press. 2007.

31 Sanders, J., Kandrot, E. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Ann Arbor, MI: Edwards Brothers. 2010.

32 Tan, R. T. Singapore Institute of Management. Accessed: 21/3/2014. http://php-robbytan.rhcloud.com/code.html

33 Yang, Q., Yang, T. Department of Computer Science, City University of Hong Kong. Accessed: 21/3/2014. http://www.cs.cityu.edu.hk/~qiyang/

APPENDIX A - SHEN & CAI'S METHOD CODE

```
1   #include <opencv2/highgui/highgui.hpp>
2   #include <cv.h>
3   #include <iostream>
4   #include <cmath>
5
6   using namespace cv;
7   using namespace std;
8
9   int minValue (int anRGBvalues[3])
10  {
11        int nMinimum = anRGBvalues[0];
12        for (int iii = 1; iii < 3; iii++)
13        {
14        if (nMinimum > anRGBvalues[iii])
15        {     nMinimum = anRGBvalues[iii]; }
16        }
17        return nMinimum;
18  }
19
20  int main( int argc, const char** argv )
21  {
22        int nbeta;
23        int anRGBPixelSet [3];
24
25        Mat image = imread("Image Path", CV_LOAD_IMAGE_UNCHANGED);
26        Mat SFImage = Mat::zeros( image.size(), image.type() );
27        Mat MSFImage = Mat::zeros( image.size(), image.type() );
28
29        if (image.empty())
30        {     cout << "Error : Image cannot be loaded..!!" << endl;
31              return -1;
32        }
33
34        const int nImagePixelsCount = image.rows * image.cols;
35        int anMinPixelsValues [nImagePixelsCount];
36        int nCounterZ = 0;
37
38        for( int y = 0; y < image.rows; y++ )
39        {
40        for( int x = 0; x < image.cols; x++ )
41        {
42        for ( int ii = 0; ii < 3; ii++)
43        {     anRGBPixelSet[ii] = image.at<Vec3b>(y,x)[ii];   }
44        nbeta = minValue(anRGBPixelSet);
45        for( int c = 0; c < 3; c++ )
46        { SFImage.at<Vec3b>(y,x)[c] = (image.at<Vec3b>(y,x)[c] -
47  nbeta);}
48
49        anMinPixelsValues[nCounterZ] = nbeta;
50        nCounterZ++;
51        }
52        }
53
54
55        int nMeanValue = 0;
56        int nTempSum = 0;
```

```
57        for (int jjj = 0; jjj < nImagePixelsCount; jjj++)
58        {     nTempSum = nTempSum + anMinPixelsValues [jjj]; }
59        nMeanValue = nTempSum / nImagePixelsCount;
60
61
62        nTempSum = 0;
63        double dMeanDeviation = 0.0;
64        for (int jj = 0; jj < nImagePixelsCount; jj++)
65        {     nTempSum = nTempSum + abs(anMinPixelsValues[jj] -
66  nMeanValue); }
67        dMeanDeviation = nTempSum / nImagePixelsCount;
68
69        nTempSum = 0;
70        double dStandardDeviation = 0.0;
71        for (int j = 0; j < nImagePixelsCount; j++)
72        {     nTempSum = nTempSum + ((anMinPixelsValues[j] -
73  nMeanValue) * (anMinPixelsValues[j] - nMeanValue)); }
74        dStandardDeviation = sqrt (nTempSum / nImagePixelsCount);
75
76        double dZeta = 0.50;
77        double dPixelThreshold = 0.0;
78        dPixelThreshold = dMeanDeviation + dZeta * dStandardDeviation;
79
80        double dTau = 0.0;
81        int nCounterX = 0;
82        for( int y = 0; y < image.rows; y++ )
83        {     for( int x = 0; x < image.cols; x++ )
84        {
85        if ( anMinPixelsValues[nCounterX] > dPixelThreshold)
86        {     dTau = dPixelThreshold; }
87        else
88        {     dTau = anMinPixelsValues[nCounterX];      }
89
90        for( int c = 0; c < 3; c++ )
91        {     MSFImage.at<Vec3b>(y,x)[c] = saturate_cast<uchar>(
92  SFImage.at<Vec3b>(y,x)[c] + dTau );}
93        nCounterX++;
94
95         }
96         }
97
98
99        namedWindow("Original Image", CV_WINDOW_AUTOSIZE);
100       imshow("Original Image", image);
101
102       namedWindow("Modified Specular Free Image",
103 CV_WINDOW_AUTOSIZE);
104       imshow ("Modified Specular Free Image", MSFImage);
105
106       waitKey(0);
107       destroyWindow("Original Image");
108       destroyWindow("modified Specular Free Image");

      return 0;
  }
```

APPENDIX B - MIYAZAKI'S METHOD CODE

```
#include <opencv2/highgui/highgui.hpp>
#include <cv.h>
#include <iostream>
#include <cmath>


using namespace cv;
using namespace std;

int main( int argc, const char** argv )
{
      Mat image = imread("Image Path", CV_LOAD_IMAGE_UNCHANGED);
      Mat SFImage = Mat::zeros( image.size(), image.type() );
      double adTransformMatrix [3][3] = { {1.0, -1.0/2.0, -1.0/2.0,},
{0.0, sqrt(3.0)/2.0, -sqrt(3.0)/2.0,}, {1.0/3.0, 1.0/3.0, 1.0/3.0} };
      double adInverseTMatrix [3][3] = { {2.0/3.0, 0.0, 1.0,}, {-
1.0/3.0, 1.0/sqrt(3.0), 1.0,}, {-1.0/3.0, -1.0/sqrt(3.0), 1.0} };
      double adMColorSpace [3] = {0.0, 0.0, 0.0};
      double adSFMColorSpace [3] = {0.0, 0.0, 0.0};
      double adBGRPixelSet [3] = {0.0, 0.0, 0.0};
      double adSFRGBPixelSet [3] = {0.0, 0.0, 0.0};
      double adSFBGRPixelSet [3] = {0.0, 0.0, 0.0};
      double dFactor = 1.0;

      if (image.empty())
      {     cout << "Error : Image cannot be loaded..!!" << endl;
      return -1;
      }

      for( int y = 0; y < image.rows; y++ )
      {
      for( int x = 0; x < image.cols; x++ )
      {
      for ( int iii = 0; iii < 3; iii++)
      {     adBGRPixelSet[iii] = double(image.at<Vec3b>(y,x)[iii]);}

      adMColorSpace[0] = adTransformMatrix[0][0]*adBGRPixelSet[2] +
adTransformMatrix[0][1]*adBGRPixelSet[1]
+adTransformMatrix[0][2]*adBGRPixelSet[0];
      adMColorSpace[1] = adTransformMatrix[1][0]*adBGRPixelSet[2] +
adTransformMatrix[1][1]*adBGRPixelSet[1]
+adTransformMatrix[1][2]*adBGRPixelSet[0];
      adMColorSpace[2] = adTransformMatrix[2][0]*adBGRPixelSet[2] +
adTransformMatrix[2][1]*adBGRPixelSet[1]
+adTransformMatrix[2][2]*adBGRPixelSet[0];
      adSFMColorSpace[0] = adMColorSpace[0];
      adSFMColorSpace[1] = adMColorSpace[1];
      adSFMColorSpace[2] = dFactor *
sqrt(adMColorSpace[0]*adMColorSpace[0] +
adMColorSpace[1]*adMColorSpace[1]);

      adSFRGBPixelSet[0] = adInverseTMatrix[0][0]*adSFMColorSpace[0] +
adInverseTMatrix[0][1]*adSFMColorSpace[1] +
adInverseTMatrix[0][2]*adSFMColorSpace[2];
      adSFRGBPixelSet[1] = adInverseTMatrix[1][0]*adSFMColorSpace[0] +
adInverseTMatrix[1][1]*adSFMColorSpace[1] +
adInverseTMatrix[1][2]*adSFMColorSpace[2];
```

```
58        adSFRGBPixelSet[2] = adInverseTMatrix[2][0]*adSFMColorSpace[0] +
59  adInverseTMatrix[2][1]*adSFMColorSpace[1] +
60  adInverseTMatrix[2][2]*adSFMColorSpace[2];
61
62        adSFBGRPixelSet[0] = adSFRGBPixelSet[2];
63        adSFBGRPixelSet[1] = adSFRGBPixelSet[1];
64        adSFBGRPixelSet[2] = adSFRGBPixelSet[0];
65
66        for( int c = 0; c < 3; c++ )
67        {
68        SFImage.at<Vec3b>(y,x)[c] =
69  saturate_cast<uchar>(adSFBGRPixelSet[c]);}
70        }
71        }
72
73        namedWindow("Original Image", CV_WINDOW_AUTOSIZE);
74        imshow("Original Image", image);
75
76        namedWindow("Specular Free Image", CV_WINDOW_AUTOSIZE);
77        imshow ("Specular Free Image", SFImage);
78
79        waitKey(0);
80        destroyWindow("Original Image");
81        destroyWindow("Specular Free Image");
82
          return 0;
    }
```