In this tutorial, we will be building a simple and affordable stereo-endoscope system, by using a combination of regular off-the-shelf materials and Matlab to achieve real-time 3D image acquisition and stereoscopic display of the acquired images. The main goal of this project is to enhance depth perception in endoscopic applications through the aid of stereoscopic 3D vision technology, without the use of any high-cost and complicated equipment.

**OBJECTIVES**

To ensure that the final system is properly functional, we listed out a few requirements to fulfill by the end of the project:

1. The stereo-endoscope system should be able to be manufactured at a low cost, using affordable, off-the-shelf materials.
2. Development of the stereo-endoscope system should be simple enough to be made open-source.
3. The stereo-endoscope should be able to equip users with accurate binocular vision and capture relatively clear 3D endoscopic images.
4. The cameras should be able to be easily calibrated with commercially available software.
5. Images acquired from the stereo-endoscope are stereoscopically displayed on a Matlab GUI window, which is then viewed using either a standard head-mounted display or regular anaglyphic 3D glasses.

**MATERIALS (HARDWARE)**

To start off, we need to construct a stereo-rig that will hold the two endoscope cameras in place when acquiring images. This ensures that both endoscope cameras are securely fixed in a fronto-parallel orientation, which is necessary in order for stereoscopy and its governing principle of triangulation to be effective. As part of our objective for affordable development of the stereo-endoscope system, we make use of 3D printing technology to manufacture the stereo-rig. Given the growing interest in 3D printing technology today, a number of companies providing 3D printing services can be easily found online.
Design of stereo-rig:

Drawing dimensions (before and after extruding):

(The dimensions shown in this drawing are adjusted to be slightly larger than the calculated values for 3D printing.)

Basically, the stereo-rig consists of two hollow circles that can nicely fit the two endoscope cameras (of diameter 5.5mm each) into a stable fronto-parallel position. The open top allows users to attach or remove endoscopes easily from the stereo-rig just by snapping them in or out. From the equation

$$\frac{\text{Baseline distance}}{\text{Average Interpupillary distance}} = \frac{\text{Focal length of stereo-endoscope}}{\text{Average human arm length}}$$

the calculated baseline distance that separates the two cameras horizontally is approximately 6mm. Since 3D printing is used to manufacture the stereo-rig, this gives us the freedom to modify the dimensions according to the type of camera used and the effective baseline required for stereoscopic vision.

To keep things simple, we use two endoscope cameras of the same model (standard USB endoscope cameras with length 10mm) in the system. With the resolution and focal length of both cameras being kept the same, this makes the calibration process easier and also minimizes post-processing errors when using the stereo-endoscope.
When fixing the two cameras into the stereo-rig, we need to make sure that both cameras are inserted in an upright orientation. To do so, first connect the two USB endoscope cameras to your computer. Using the StereoEndoscope GUI in Matlab to view live video data from the cameras, rotate the cameras accordingly such that both of them are acquiring upright images of the scene, before inserting them into the stereo-rig.

*Note: In order for Matlab to detect and acquire images from the cameras, both Matlab Image Acquisition Toolbox and Matlab Support Package for USB Webcams must first be installed.

After 3D printing of the stereo-rig and completed assembly of the two endoscope cameras into the stereo-rig, the resulting stereo-endoscope will look like this:
Next, let’s move on to calibration of the stereo-endoscope.

**STEREO-CAMERA CALIBRATION**

In addition to estimating the intrinsic and extrinsic parameters of a camera, camera calibration is also used to describe the geometric relationship between a pair of cameras in a stereoscopic vision system, whereby it estimates the relative position and orientation of one camera with respect to the other. This then allows us to understand the amount of adjustments needed to make both optical axes of the camera pair parallel, which is in turn required to achieve good point correspondences between the stereo-images. Thus, calibration is a pivotal step in developing an accurate stereo-camera system which enables its users with clear stereoscopic vision.

Both OpenCV and Matlab provide simple, open-source algorithms to perform stereo-camera calibration, along with explanations for users to better understand the calibration process. For our system, we will use Matlab’s Stereo Camera Calibrator app (available in the Computer Vision System Toolbox) to calibrate our stereo-endoscope, since we will also be using Matlab later on for stereoscopic display of the endoscopic images.

The Stereo Camera Calibrator app uses a checkerboard pattern as a calibration target. To accommodate the endoscope cameras’ focal length range of 40-60mm, the checkerboard pattern used for our stereo calibration consists of black and white squares with dimensions 3mm by 3mm each.
For more info on how to perform the stereo-calibration, please refer to the website: https://www.mathworks.com/help/vision/ug/stereo-camera-calibrator-app.html. To capture images of the checkerboard pattern for calibration, you can either create an algorithm, or manually take matching snapshots with the left and right cameras one pair at a time. You can refer to https://www.mathworks.com/help/supportpkg/usbwebcams/ug/acquire-images-from-webcams.html on how to acquire single images from webcams in Matlab.

After running the calibration, a set of calibration results is generated, and this includes reprojection errors as well as the cameras’ intrinsic and extrinsic parameters. To examine the accuracy of the calibration process, we can examine the resulting reprojection errors, which refer to the distances in pixels between detected points in the checkboard pattern and the corresponding reprojected points in the camera images. They serve as a form of quantification to determine the amount of correction required to adjust one of the two images, in order to achieve perfect correspondences in a set of image points. As such, the values of reprojection errors should ideally be as small as possible. According to Matlab, reprojection errors of less than one pixel are generally acceptable.

3D IMAGE ACQUISITION & DISPLAY

Once calibration is completed with acceptable reprojection errors, we can finally proceed on with 3D image acquisition using the calibrated stereo-endoscope and Matlab. Depending on their preference, users can choose to view the acquired images on either a side-by-side display or an anaglyphic display.
1. SIDE-BY-SIDE DISPLAY

Firstly, for side-by-side display of the acquired images, we first created a user interface in Matlab, named ‘StereoEndoscope GUI’. This Matlab GUI is a user interface that enables users to view real-time stereoscopic display of a scene by navigating the stereo-endoscope. And to ensure that users can effectively experience better depth perception from the 3D display, it is important that the Matlab GUI is able to acquire live images from both endoscope cameras simultaneously with minimal time lag.

Here is a short video on how the StereoEndoscope GUI works:

The StereoEndoscope GUI shows the live images from the left and right cameras in a stereoscopic (side-by-side) display, so that the left and right images are separately presented to the user’s left and right eyes at the same time. This enables the brain to combine the left and right images together and obtain the binocular disparities between both images to perceive depth.

To view the side-by-side endoscopic images displayed on the StereoEndoscope GUI, a head-mounted display headset is needed. I opted for Silvertec VR Magic Virtual Reality Glasses, which costs approximately USD$28.50.
Like Google Cardboard, the VR Magic Glasses is used together with a smartphone to present the stereoscopic display directly in front of the user’s eyes. So, prior to using the head-mounted display to view the stereoscopic display, the StereoEndoscope GUI displayed on the desktop Matlab needs to be first mirrored onto a smartphone screen. There are a few screen-mirroring apps that are free to download online, such as MirrorOp (http://www.mirrorop.com) and Chrome Remote Desktop (https://chrome.google.com/webstore/detail/chrome-remote-desktop/gbchcmhmahfdphkmpfmihenigjmpp?hl=en).

Video on how the StereoEndoscope GUI is mirrored from the computer screen to the smartphone screen via Chrome Remote Desktop:

When the StereoEndoscope GUI is successfully mirrored onto the smartphone screen, the stereoscopic display of images from the stereo-endoscope is now ready for viewing! Simply insert the smartphone into the designated slot of the VR Magic Glasses (or any other models of head-mounted
displays), put on the headset and adjust the headset accordingly to accommodate your personal interpupillary distance.

2. ANAGLYPHIC DISPLAY

Apart from using the head-mounted display headset, another low-cost option for stereoscopic viewing of the images acquired from the stereo-endoscope is through the use of regular 3D anaglyph glasses.

In this case, the left and right images acquired from the stereo-endoscope need to be first filtered before being superimposed together to form a single anaglyphic image. To create an anaglyphic display of the acquired images, another Matlab code is used instead of the StereoEndoscope GUI.

Here is a short video on how the code works:
Here is a summary of the steps to develop our simple, low-cost stereo-endoscopy system:

1. Construct a stereo-rig with the provided drawing and a 3D printer.
2. Fix two endoscope cameras (preferably of the same model) into the stereo-rig. Ensure that both cameras are inserted securely into the stereo-rig in an upright orientation.
3. Connect the two endoscope cameras to the computer via their USB cables.
4. Perform calibration of the stereo-endoscope using Matlab’s Stereo Camera Calibrator app. The mean reprojection error generated in the calibration results should be less than 1 pixels.
5. Using Image Acquisition Tool in Matlab, make sure that Matlab can detect the two endoscope cameras. Edit the StereoEndoscopeGUI script according to the detected cameras.
6. Run the StereoEndoscopeGUI script on the desktop Matlab, and ensure that the GUI shows a real-time preview of the scene with the stereo-endoscope.
7. Mirror the desktop screen onto a smartphone screen via a screen-mirroring app.
8. Insert the smartphone into a head-mounted display before putting it on to view the images.
9. This completes our development of a low-cost stereo-endoscopy system!
MATLAB CODE FOR THE STEREOENDOSCOPE GUI:

Prior to running this script, use the Image Acquisition Tool in Matlab to ensure that Matlab is able to detect the two endoscope cameras that are connected to the computer via USB cables.

*Edit lines 85 and 86 in the StereoEndoscopeGUI script according to the device name, adaptor and device ID that are provided in the Image Acquisition Tool.

```matlab
84  % Select which webcam and which mode for image acquisition
85  -  vid1 = videoinput('macvideo', 2, 'UY2_640x480');  % Left camera
86  -  vid2 = videoinput('macvideo', 3, 'UY2_640x480');  % Right camera
```
function varargout = StereoEndoscopeGUI(varargin)

% STEREOENDOSCOPEGUI MATLAB code for StereoEndoscopeGUI.fig
% STEREOENDOSCOPEGUI, by itself, creates a new STEREOENDOSCOPEGUI or raises the existing
% singleton*.
%
% H = STEREOENDOSCOPEGUI returns the handle to a new STEREOENDOSCOPEGUI or the handle to
% the existing singleton*.
%
% STEREOENDOSCOPEGUI('CALLBACK',hObject,eventData,handles,...) calls the local
% function named CALLBACK in STEREOENDOSCOPEGUI.M with the given input arguments.
%
% STEREOENDOSCOPEGUI('Property','Value',...) creates a new STEREOENDOSCOPEGUI or raises the
% existing singleton*. Starting from the left, property value pairs are
% applied to the GUI before StereoEndoscopeGUI_OpeningFcn gets called. An
% unrecognized property name or invalid value makes property application
% stop. All inputs are passed to StereoEndoscopeGUI_OpeningFcn via varargin.
%
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
% instance to run (singleton)*.
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help StereoEndoscopeGUI

% Last Modified by GUIDE v2.5 24-Jan-2017 22:49:50

% Begin initialization code - DO NOT EDIT

gui_Singleton = 1;

gui_State = struct('gui_Name', mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @StereoEndoscopeGUI_OpeningFcn, ...
    'gui_OutputFcn', @StereoEndoscopeGUI_OutputFcn, ...
    'gui_LayoutFcn', [], ...
    'gui_Callback', []);

if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before StereoEndoscopeGUI is made visible.
function StereoEndoscopeGUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to StereoEndoscopeGUI (see VARARGIN)

% Choose default command line output for StereoDisplayGUI
% Create video object to establish connection to the cameras.
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes StereoEndoscopeGUI wait for user response (see UIRESUME)
uiwait(handles.StereoEndoscopeGUI);

% --- Outputs from this function are returned to the command line.
function varargout = StereoEndoscopeGUI_OutputFcn(hObject, eventdata, handles)
% varargout    cell array for returning output args (see VARargout);
% hObject      handle to figure
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
handles.output = hObject;
varargout{1} = handles.output;
% --- Executes on button press in pbStart.
function pbStart_Callback(hObject, eventdata, handles)
% hObject    handle to pbStart (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Select which webcam and which mode for image acquisition
vid1 = videoinput('macvideo', 2, 'YUY2_640x480'); % Left camera
vid2 = videoinput('macvideo', 3, 'YUY2_640x480'); % Right camera
% Set Inf to capture the video continuously
vid1.FramesPerTrigger = Inf;
vid2.FramesPerTrigger = Inf;
% Output would be portrayed in RGB color space
vid1.ReturnedColorspace = 'YCbCr';
vid2.ReturnedColorspace = 'YCbCr';
% Start the webcam on user request, not automatically
triggerconfig(vid1, 'immediate');
triggerconfig(vid2, 'immediate');
% To know the image height and width
vidRes = get(vid1, 'VideoResolution');
vidRes2 = get(vid2, 'VideoResolution');
% Image width
imWidth = vidRes(1);
imWidth2 = vidRes2(1);
% Image height
imHeight = vidRes(2);
imHeight2 = vidRes2(2);
% Number of bands of our image (should be 3 because it's RGB)
nBands = get(vid1, 'NumberOfBands');
nBands2 = get(vid2, 'NumberOfBands');
% Create an empty image container and show it on axPreview
hImage = image(zeros(imHeight, imWidth, nBands), 'parent', handles.PreviewLeft);
hImage2 = image(zeros(imHeight2, imWidth2, nBands2), 'parent', handles.PreviewRight);
if strcmp(get(handles.pbStart, 'String'), 'Start Camera')
    % Camera is off. Change button string and start camera.
    set(handles.pbStart, 'String', 'Stop Camera')
    % The object acquire every 3rd frame in the video stream.
    vid1.FrameGrabInterval = 3;
    vid2.FrameGrabInterval = 3; % begin the webcam preview
    preview(vid1, hImage);
    preview(vid2, hImage2);
else
    % Camera is on. Stop camera and change button string.
    set(handles.pbStart, 'String', 'Start Camera')
    stoppreview(vid1);
    stoppreview(vid2);
end

MATLAB CODE FOR REAL-TIME ANAGLYPHIC DISPLAY OF THE STEREO-ENDOSCOPIC IMAGES

Similar to the StereoEndoscopeGUI script, edit lines 2 and 5 in the above script according to the device name, adaptor and device ID that are provided in the Image Acquisition Tool.

% Create the Video Device System objects.
vidDevice1 = imaq.VideoDevice('macvideo', 2, 'YUY2_640x480', ...
    'ROI', [1 1 640 480], ...
    'ReturnedColorSpace', 'rgb'); % Left camera
vidDevice2 = imaq.VideoDevice('macvideo', 3, 'YUY2_640x480', ...
    'ROI', [1 1 640 480], ...
    'ReturnedColorSpace', 'rgb'); % Right camera
% Create VideoPlayer System objects to display the videos.
hVideoIn = vision.VideoPlayer;
% Create a processing loop to perform motion detection in the input video.
% This loop uses the System objects you instantiated above.
% Set up for stream
nFrames = 0;
while (nFrames < 20000)  % Process for the first 20000 frames.
    % Acquire single frame from imaging device.
    data1 = step(vidDevice1);
    data2 = step(vidDevice2);

    % processes the color channels of the videos
    data1(:,:,1) = 0;
    data2(:,:,2:3) = 0;
    anaglyphData = data1 + data2;

    % Send image data to video player
    % Display original video.
    step(hVideoIn, anaglyphData);

    % Increment frame count
    nFrames = nFrames + 1;
end


