1. **3D Scanner (Light and camera method)**

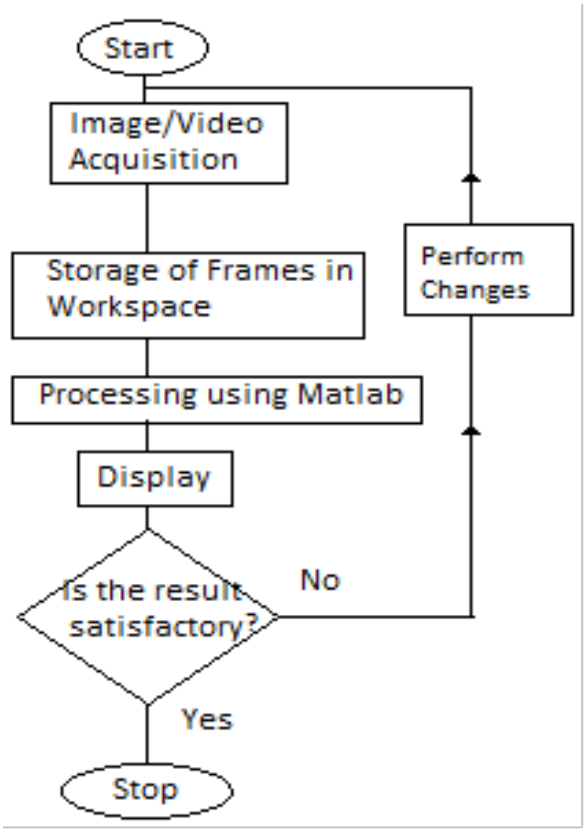

Figure 1. Flow Diagram

As per the Fig.1 flow chart we have the following modules.

1.Image/Video Acquisition

2.Storage of frames in MATLAB Workspace.

3.Processing Using MATLAB

4.Display

5.Analysis and Adjustment

1. Image Acquisition:

The first major constraint while starting coding was Image Acquisition & Storage. In the starting we began experimenting with the available Integrated Webcam.

But before acquiring images the following instructions are executed.

close all hidden

close all hidden deletes all figures including those with hidden handles. Here in all handles assigned during the previous run.

clear all

clear all clears all objects in the MATLAB workspace and closes the MuPAD engine associated with the MATLAB workspace resetting all its assumptions.

Here is an illustration of the code we inputted via the MATLAB Command Prompt.

```
>> vidobj=videoinput('winvideo',1); % Input Image Acquisition object is created.

%The object is configured.

>> inspect(vidobj)

% On execution provides the following command prompt Fig. 2. Here in the return type RGB is selected.
```

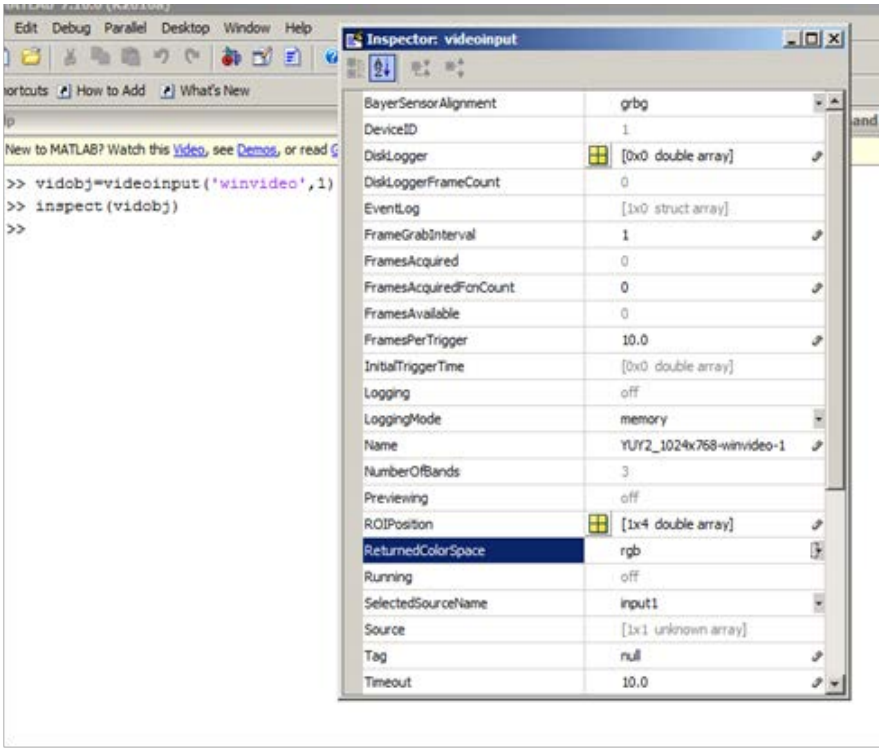

Figure 2. Video Inspector Prompt

>> set(vidobj,'FramesPerTrigger',9);% the No of Frames to taken per trigger is set.

>> start(vidobj);% The Acquisition is started.
>> data=getdata(vidobj); % Data is obtained through the object

>> imaqmontage(data)

% imaqmontage(obj) calls the getsnapshot function on video input object obj and displays a single %image frame in a MATLAB figure window using the images function. obj must be a 1-by-1 video %input object.

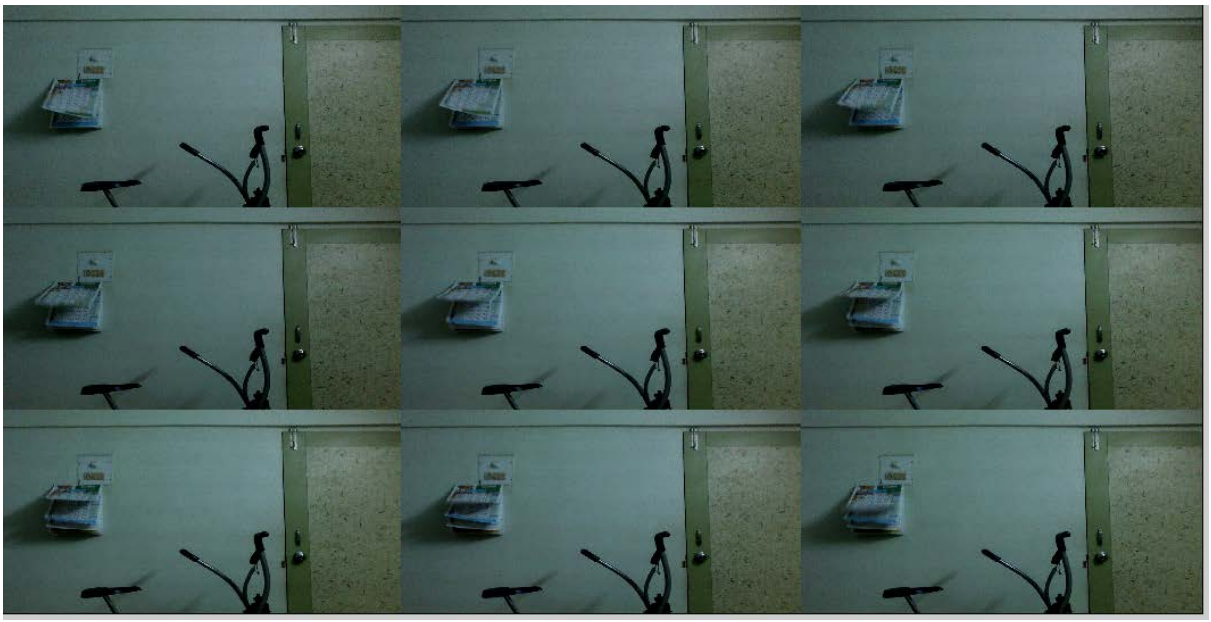

Figure 3. Sequential RGB

```
% Here in 9 sequential RGB frames (Fig. 3) are acquired as set initially.
>> delete(vidobj);   %Object is deleted
>> clear (vidobj );     %Object is cleared
```

This code block did provide us with an basic setup to acquire sequential images. But to reacquire each frame back from this multi-frame image was indeed a tedious job. Hence in a pursuit to obtain better results we searched multiple options. Finally we decided to work with Simulink.

Simulink

Simulink, developed by MathWorks, is a commercial tool for modeling, simulating and analyzing multi domain dynamic systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries. It offers tight integration with the rest of the MATLAB environment and can either drive MATLAB or be scripted from it. Simulink is widely used in control theory and digital signal processing for multi domain simulation and Model-Based Design.

It can be initialized by the following command on the MATLAB command prompt.

>> Simulink

Further after designing blocks using Image Acquisition tool box and Video and Image processing Toolbox we designed the following system (Fig.4).

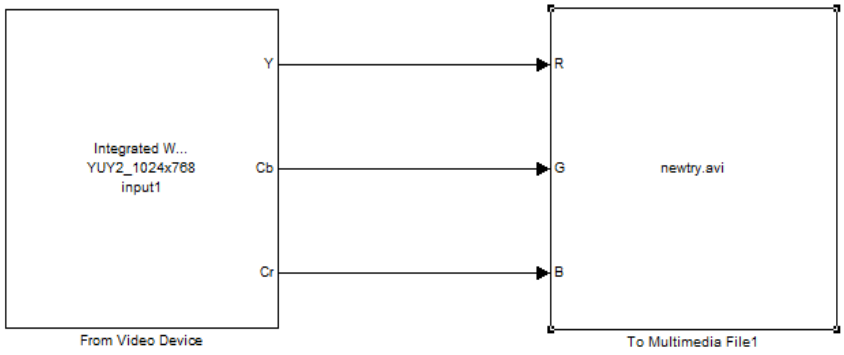

Figure 4. Acquisition Object

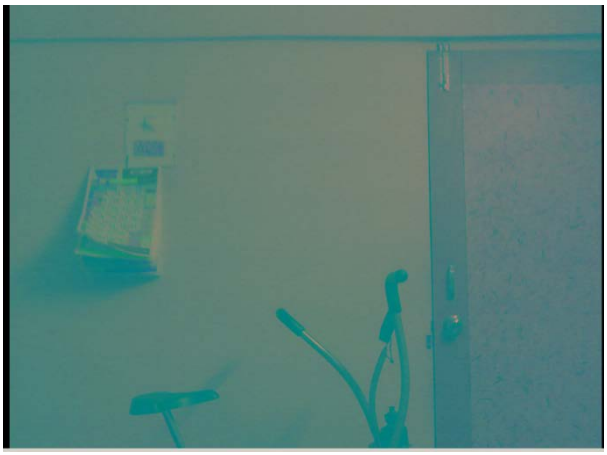

Figure 5. Acquired Video Snapshot

This is the output video (acquired image)

Here in video data is acquired from the Integrated Web Cam and is straightaway stored in a multimedia file. But feature extraction from such an image would be difficult enough. On further study we found out that the Camera module outputted images in YbCbCr format. But we need a RGB Image. This problem was solved by introducing a image converter block.

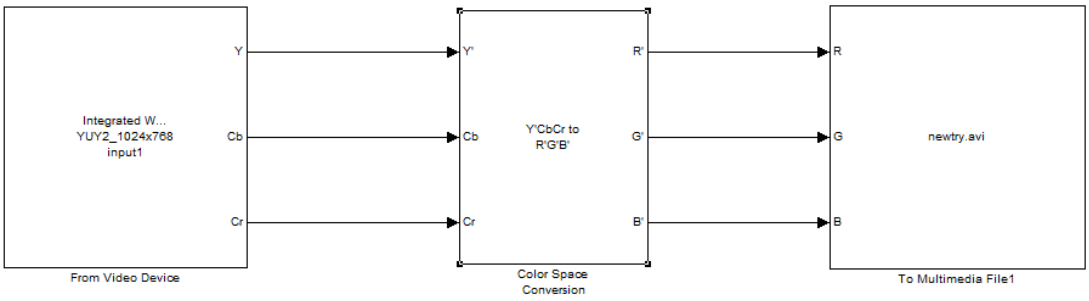

Figure 6 YbCbCr to RGB Object

The Cameras YbCbCr output is converted to RGB and stored in MATLAB workspace as newtry.avi (Fig.6).

The outputted video image is now RGB in nature.

Figure 7. Acquired Video Snapshot

Later on in order to achieve portability we used a locally available Web Cam. After installing the drivers. The camera was detected into Simulink Module. The following model (Fig.8) used for video acquisition.

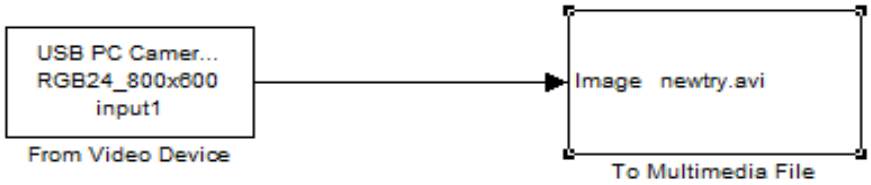

Figure 8. Simulation Object

Figure 9.Acquired Video Snapshot

The Image (Video frame) obtained from this simulation (Fig.9) was same as the previous one but it had an added privilege of focusing.

Initially all data obtained is uncompressed in nature as MATLAB works better with uncompressed data otherwise a decompressor (system specific Codec) is needed. Moreover initially we acquired video in separate signal format as it can be further utilized to extract certain color features such as the scanning line. During the simulation time the turn table is rotated using serial communication. Here in, on receiving a character from the computer the microcontroller activates the turntable for a pre calculated time duration.

2)Storage of Images in MATLAB workspace(Frame Extraction).

Once the video is acquired and stored into the workspace the next challenge was to obtain frames from this video data. For this uncompressed video we obtain numbered images and store them into the work space. For this we have used the following code.

```
Vid=aviread('newtry.avi');
for i=1:length(Vid)
   imwrite(Vid(i).cdata,[num2str(i) '.jpeg'])
 % Check the format & options
end
 nFrames = length(Vid);
```

The number of images obtained depend upon the simulation time of the acquisition module.

The above code captures data and writes them into the workspace as 1.jpeg, 2.jpeg… nFrames.jpeg.

2) Processing Using MATLAB

A)Obtaining Coordinates/Scanning images

1. Here in each image is scanned sequentially and the coordinates of the data above the threshold intensity level are stored in x_c and y_c arrays. These coordinates are separated depending upon the threshold value of scanning line& background.

b) Smoothening

After obtaining the data smoothening operation is carried out on it. Using the following instructions

```
x_c(:,k)= smooth(xi(:,k),span);
y_c(:,k)= smooth(yi(:,k),span);
```

C)Plotting

After smoothening a pseudo cylinder is plotted with variable radius, & programmable height .One contour (feature) is extracted from each image and is plotted using cylindrical geometry. The angle between respective contours is a function of number of images. Now if we have more number of images then the angle between respective contours is less and accuracy is higher. Hence we are inclined to obtain more number of images per simulation. For this we acquire almost 360 images per simulation. The timing to rotate the turntable and simulation time are thus calculated using an iterative process.

4) Display

Once the contours are plotted a surface is plotted around this structure to obtain the 3d model of the object scanned.

For this purpose we use the following function

```
h=surface(X, Y, Z);
```

This image can be further observed by enabling Rotate 3d icon. Further this 3D model can be stored in the workspace and can be accessed whenever required.

5) Analysis and Adjustment

Since calibration is not incorporated we may have to alter the threshold, height (top& bottom) and radius to acquire a better result.

After each run the frames extracted from the video are deleted from the workspace in order to provide space for freshly acquired images. For this we use the following code block.

```
for k=1:nFrames
 filename=strcat(num2str(k),'.jpeg')
delete(filename)
end
```

Moreover X and Y plots can also be displayed to observe the alterations in the contours. Suitable labelling is done wherever required.

Results:

For a scanned object & the rendered 3D model can be shown in Fig.10

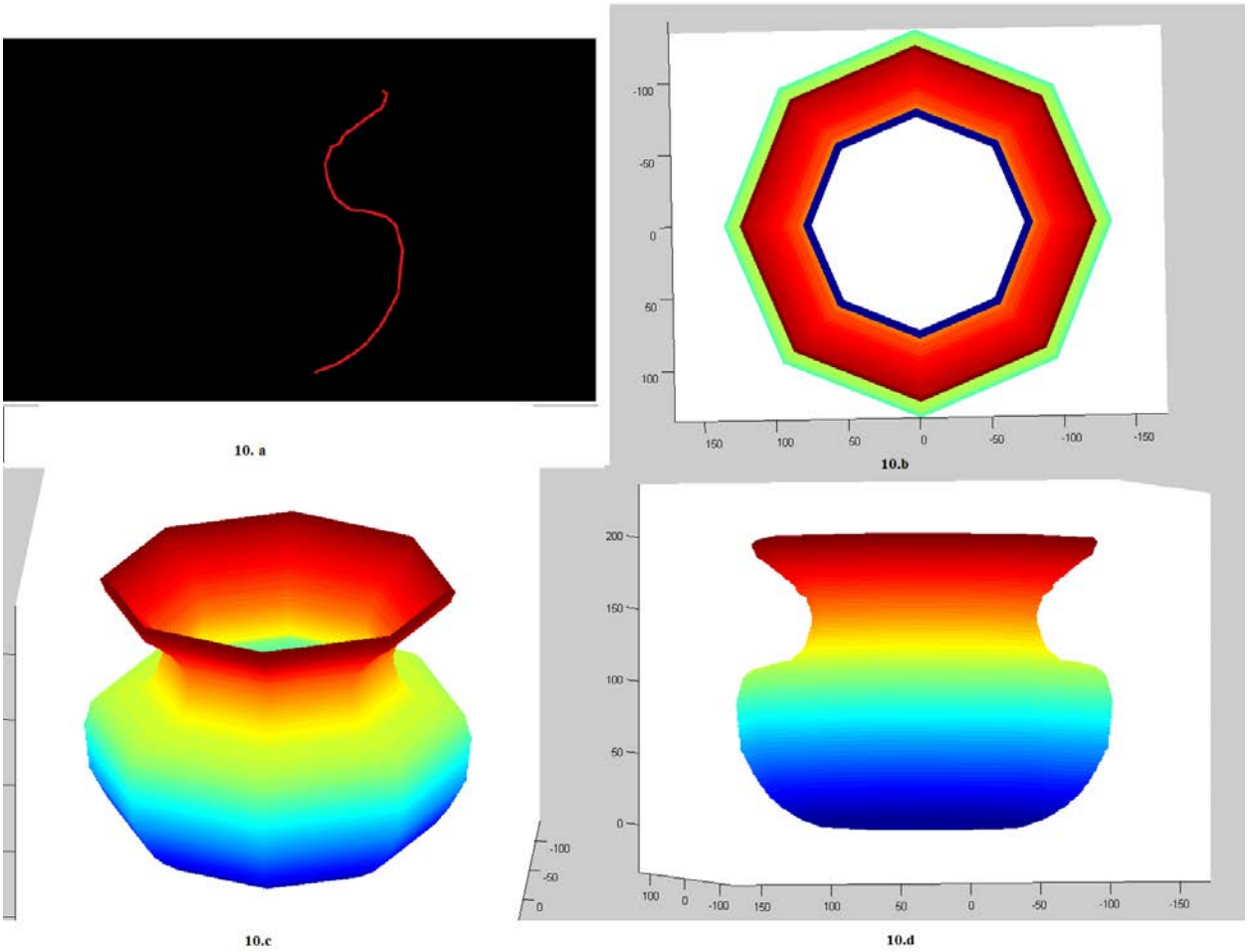

Figure 10. Scanned Object & 3D Model