CNS 186 Final Report
Stereo Vision Tracking System Using Webcams

Yoke Peng Leong

March 19, 2014

Abstract

In this project, a stereo vision tracking system using webcams is constructed. This system records stereo videos of moving labeled targets and extracts the 3D trajectories of these targets from the recorded videos. It consists of three major steps – stereo video capturing, post processing, and 3D trajectory construction of labeled targets. The final goal of this stereo system is to record human stick balancing experiments and extract the stick trajectories in 3D space. In this report, both the hardware and software implementations of this stereo video capturing system are described, and the system's performance is characterized.

1 Introduction

In this project, a stereo vision tracking system using webcams is constructed. This system records stereo videos of moving labeled targets and extracts the 3D trajectories of these targets from the recorded videos.

This project is complimentary to a research project that studies human sensorimotor control. More specifically, the research project uses human stick balancing as a case study to understand the role of vision in human sensorimotor control. The hypothesis based on feedback control theory is that the location where the eyes focus on affects a person's stick balancing ability [1]. To study this phenomena quantitatively in human experiments, a motion capturing system that can track the stick trajectories and an eye tracking system are required. This project focuses on setting up the stick motion capturing system.

Many motion capturing systems are available commercially or through open sources. Common video capturing hardware options are Vicon, Kinect, iPhone, webcams, and other video recording devices. Common software options are MATLAB, Python, OpenCV, C/C++, and other programming languages and packages. Despite the options, a cheap, simple, and sufficiently accurate system is desired. More precisely, the system, software and hardware included, has to cost no more than a few hundreds dollar. This cost constraint excludes the use of commercial motion capturing systems such as Vicon. The stereo system also should not require a dedicated technician to set up and use, and it has a precision in the order of a milimeter.

Based on all these requirements, the Logitech C920 HD Pro Webcam and MATLAB are chosen for buiding this motion capturing system. These HD webcams cost about $75. They are easy to set up, and they connect to computers through USB ports which are commonly available in most modern computers. The resolution of these webcams can be as high as 1080 pixels by 1920 pixels. Thus, one can achieve a precision of approximately one milimeter per pixel depending on the depth of the target relative to the cameras and

the resolution chosen. MATLAB is suitable for this project because it is available for free through campus subscription, and it is a comprehensive computational tool that many people are familiar with.

Generally, this stereo vision capturing system using webcams consists of three major steps – stereo video capturing, post processing, and 3D trajectory construction of labeled targets. During stereo video capturing step, apart from physical hardware setup, synchronization and calibration are essential components. Synchronization can be done externally or internally through software and/or hardware implementations. Due to limited budget and time for the project, the cameras are synchronized externally using a iPhone camera flash, and calibrated using the Camera Calibration Toolbox for MATLAB [2]. Further details are explained in Section 2.1. The second step, post processing, is implemented to synchronize a pair of stereo videos and cut them into segments that correspond to individual experimental trials. Lastly, 3D trajectory extraction detects the labeled target, a stick that is labeled red in this case. For each frame on the video recordings, The algorithm detects red labels on the stick and extract the stick's location in a 2D image plane based on Hough transform. Then, stereo trinagulation is implemented to obtain stick trajectories in 3D space. Refer to Section 2.3 for a more detailed description.

The rest of this report is organized as follow: Section 2 describes the hardware and software setups, system performance analysis and quantitative results are shown in Section 3, and Section 4 summarizes the project and discusses possible future directions. Appendices are included at the end of this report for those who want more details of this project.

2 Stereo Vision Tracking System

As described earlier, this stereo vision tracking system using webcams consists of three major steps. They are stereo video capturing, post processing, and 3D trajectory construction of labeled targets. This section describes the three steps with details.

2.1 Stereo Video Capturing

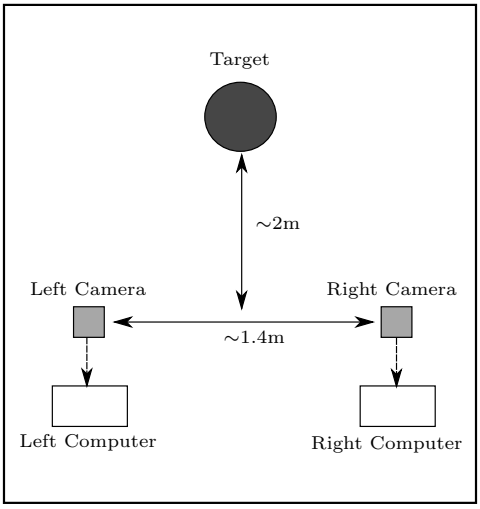
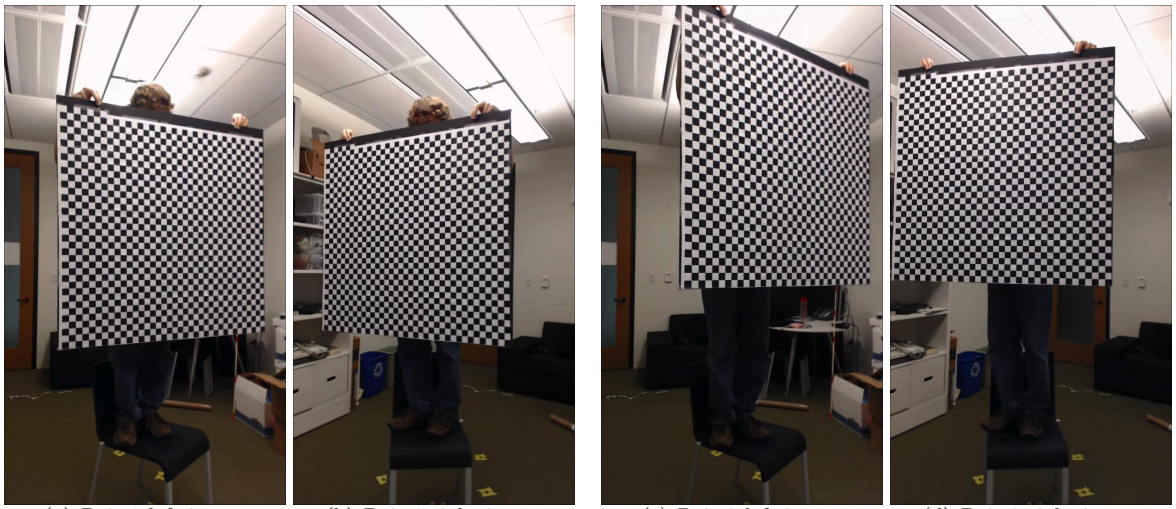


Figure 1: A schematic of the physical setup in a room. The target is the approximate location where the stick balancing experiment is conducted.

A stereo video capturing system is set up in a studio at Annenberg (Figure 1). Two 30 fps Logitech HD C920 webcams each is attached to a tripod and connected to a dedicated Window computer. Refer to [3] for full specifications of the camera. The resolution of the webcams are chosen to be 720 pixels by 1280 pixels instead of 1080 pixels by 1920 pixels (the highest possible) in order to keep the video file size managable while also achieve the desired precision. The system described in this project is implementable with any camera resolution. The Logitech Webcam Software is used to operate the camera for recording. When setting up a stereo video capturing system, there are two main components to consider, synchronization and calibration of the left and right cameras.

In order to enable synchronization of the two cameras, a light is flashed using an iPhone camera flash in front of the two webcams in the beginning and at the end of each video recording session. These flashes are markers for synchronization during the post processing step. Synchronization will be discussed further in the next section, Section 2.2.

Calibration is completed using the Camera Calibration Toolbox for MATLAB [2]. An approximately 40 inches by 40 inches checkerboard was constructed for calibration using foam board (see Figure 2). A long ruler is used to ensure that the boxes in the checkerboard are aligned when the printings of checker pattern are pasted on the foam board. Each box on the checkerboard is 27 mm by 27 mm. The checkerboard was positioned such that the whole camera frame was covered with the entire collection of calibration images. To keep the large foam board flat, the board is hold up from the top edge. Three sets of images, thirty pairs of frames each, are selected from a stereo video recordings for calibration to make sure that the calibration is consistent across different sets of images. Figure 2 shows two pairs of calibration images taken from the webcams.



(a) Pair 1 left image (b) Pair 1 right image (c) Pair 2 left image (d) Pair 2 right image

Figure 2: Examples of calibration images. Left images are taken by left camera, and right images are taken by right camera.

2.2 Post Processing

After a pair of stereo videos are recorded, post processing is necessary to synchronize and cut the videos into segments that correspond to each experimental trials. Window Movie Maker is used as the primary tool to cut the video because it is a dedicated video editing software that is cost free.

As mentioned in the last section, the two cameras are synchronized using iPhone camera flashes in the beginning and the end of the video recordings. The synchronization can be done manually by checking each frame using Window Movie Maker or automatically using MATLAB. For a small amount of videos, manual operation is sufficient. Window Movie Maker is utilized to search for the first frame after the beginning flash is not visible. This frame is used as the first frame of a captured event for both cameras. Similarly, the first frame after the ending flash is not visible is used as the last frame of a captured event for both cameras.

On the other hand, MATLAB can be used to search for the flash to do the synchronization automatically. For convenience, the location of the flash needs to be fixed at certion position relative to both webcams so that the pixels where the flash occur for both cameras' images are consistent. The locations of pixels where the flash occurs is called the flash zone. Plot the average luminance of the flash zone for some amount of frames at the beginning (or the end) of a video to obtain the flash profile of a camera. Figure 3 shows two examples of iPhone camera flash profiles in a video recording. The frames when luminance is at its peak on left and right cameras respectively are the left and right camera frames that correspond to each other. To be more precise, one can fit a curve on the luminace plot and obtain the peak of the curve with subframe accuracy. However, using synchronization with such accuracy requires interpolation of the stick trajectories in between every frames and other additional computations. Due to time constraint, subframe-accurate synchronization is not yet implemented. But, it will be implemented in the future.

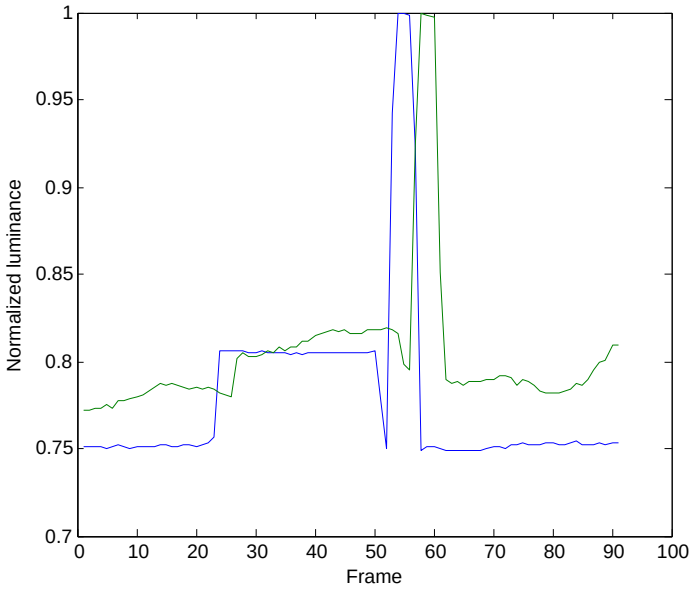


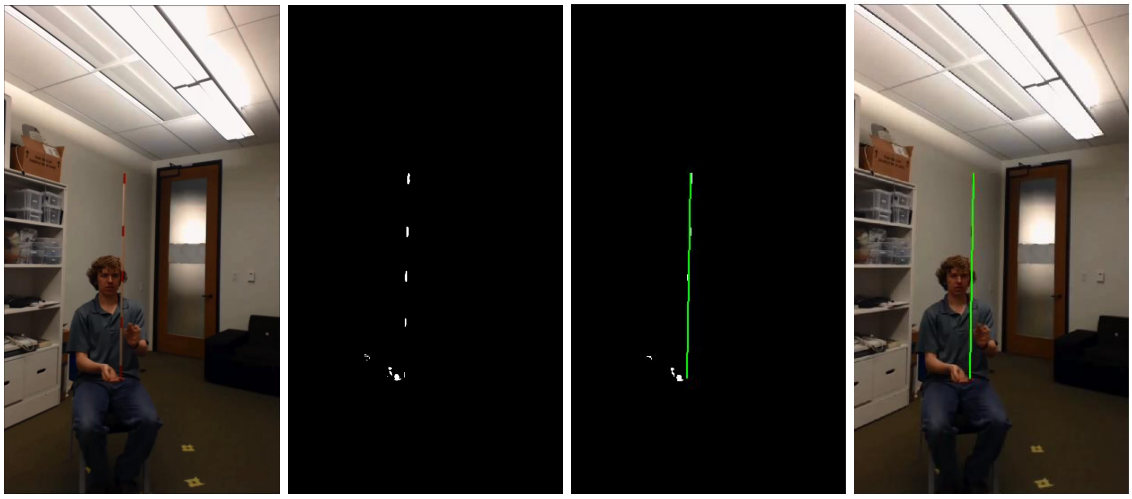
Figure 3: Two examples of flash profile for an iPhone 5 whereby its flash is used as synchronization marker. The iPhone camera has double-flash. Hence, the first bump corresponds to the first small flash and the second tall bump corresponds to the actual flash.

Once the synchronization frame is found, Window Movie Maker is used to cut the stereo videos to the correct frames. MATLAB is not used for this purpose because of long computation time as compared to the other option. After that, the synchronized videos are cut manually into short segments that correspond to the experimental trials using Window Movie Maker. There is no obvious way to automate this step at the moment.

2.3 3D Trajectory Construction

After the videos are synchronized and segmented, MATLAB is used to parse each frame for labeled targets. This process involves two major components – color and line detection to extract the 2D stick trajectories from left and right cameras, and stereo triangulation to construct the 3D stick trajectories.

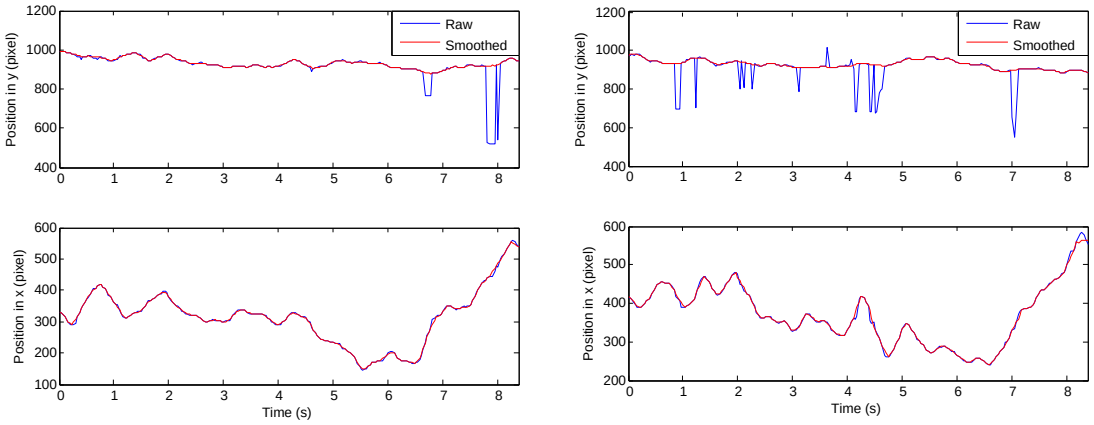
Firstly, for each frame, the image is filtered for red targets whereby the red pixels are coded white and the rest are coded black in the filtered image (Figure 4(b)). In order to reject noise in the filtered image such as the extra white patches that corresponds to the hand in Figure 4(b), Hough transform [4] is used to detect a line (i.e. the stick) in the filtered image. The technique counts the number of white pixels that falls into each bin in a parameterized space of all possible lines, and search for the bin with the maximum number of white pixels. This maxima corresponds to a line in an image. For more details, refer to [4]. In essence, this technique allows one to find the coordinates of the two ends of a line in an image. Figure 4 shows an example of the process involved to extract 2D stick coordinates from a sample camera frame.



(a) Original (b) Red Filtered (c) Hough Transform (d) Overlay

Figure 4: An example of 2D stick coordinates extraction from an image: (a) shows the original image, (b) shows the image after being filtered for red, (c) shows the line (i.e. stick) detected using Hough transform, and (d) shows the line detected overlays on the original image.

This 2D line coordinates extraction process is repeated for every frame in each pair of videos and the coordinates of the line are recorded. At the end of this process, 2D stick tracjetories are obtained relative to the image coordinate frame (in pixels) of the right and left camera respectively. To reject noise and outliers in the stick trajectories, the raw data are smoothed using the "smooth" function in MATLAB that implements robust local regression using weighted linear least squares and a first degree polynomial model. The noise and outliers are results of video compression and stick motion that is too quick, especially when the subjects are about to fail to balance the stick. When the stick moves across the space, lights that fall on the red markers change, and thus threshold for red filtering varies. This error is accounted for by refiltering with a more appropriate threshold for the segment of frames that is severely affected. The rest of the color filtering errors are treated as noise. Figure 5 shows a reconstruction of a 2D trajectory of a target (the point where the stick and finger are in contact) from left and right camera. Go to [5] for a video showing the stick extracted overlays on the original videos.



(a) Left Camera (b) Right Camera

Figure 5: An example of 2D trajectories reconstructed from the left and right cameras. The blue lines indicate raw data, and the red lines indicate smoothed data.

Using the 2D trajectories from left and right camera, stereo triangulation is performed to compute the stick trajectories in 3D space relative the cameras. Figure 6 shows the trajectory of the target in 3D space relative to the right camera after stereo triangulation. Go to [6] for a video showing the reconstructed 3D trajectory of a stick. A sample MATLAB code that implements the 3D trajectory construction is attached in Appendix A. The code can be downloaded from [7].

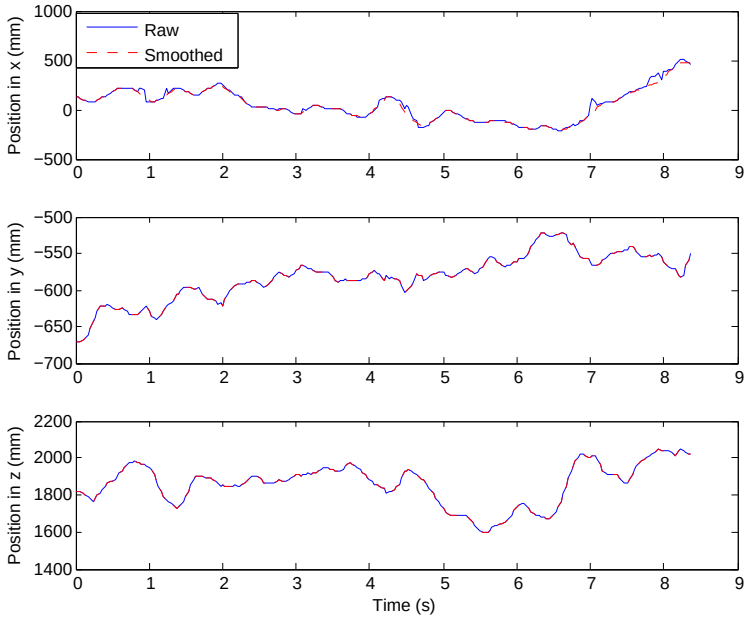


Figure 6: An example of 3D reconstruction from the 2D trajectories in Figure 5. The blue lines indicate a reconstruction using the raw data. The red lines indicate a reconstruction using the smoothed data. The coordinate frame is defined such that x is pointing to the right, y is pointing upwards and z is pointing into the image relative to the right camera image plane.

3 Performance Analysis and Results

This section presents the calibration results and describes the system performance analysis. To quantify the performance of this stereo vision tracking system, two volunteers are recruited to balance a stick while being recorded by the stereo system.

3.1 Calibration

For stereo camera calibration, the important extrinsic parameters are translation and rotation of cameras relative to each other. A summary of the calibration results is given by Table 1 showing the rotation and translation vectors with respect to the left camera for three sets of calibration images. The mean translation vector is (-25, 1438, 640) mm, and the mean rotation vector is (44.02, 4.59, 0.15) deg. The values for all three sets of calibration images show that the calibration process is consistent. Based on physical measurements of the setup, the values are as expected. Figure 7 shows the orientation of the camera with respect to each other and the images taken computed using one set of the calibration images. Refer to Appendix B for other camera parameters computed for each set of calibration images and the errors in the calibrations.

Images Set	1	2	3	Mean
Translation Vector (mm)				
x	-17	-30	-27	-25
y	1425	1449	1441	1438
z	653	622	644	640
Rotation Vector (deg)				
x	44.79	44.02	43.25	44.02
y	3.57	4.92	5.29	4.59
z	0.19	0.07	0.19	0.15

Table 1: Translation and rotation vectors relative to the left camera obtained from calibration for three sets of calibration images (30 images each). Rotation vector is defined as rotation around x, y, and z. Camera coordinate frame is defined such that x is pointing downwards, y is pointing to the right and z is pointing into the image relative to the camera image plane.

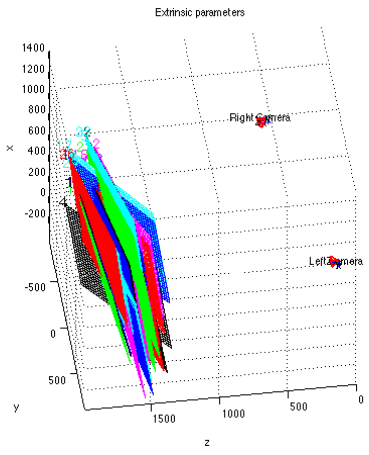


Figure 7: Orientation of cameras and calibration images.

3.2 System Performance

In order to understand the performance of this system, the following analysis is considered and the results are presented.

Rolling Shutter The webcams used in this system have a rolling shutter. Rolling shutter camera record the light sensor values line by line across the plane instead of all at the same time. The scan line is the same orientation to the stick orientation. This orientation reduces the rolling shutter effect, and a typical error is in the range of a milimeter. This is equivalent to $\sim 12\%$ error relative to stick crosssection diameter which is 9.53 mm. The calculation can be found in Appendix C. The rolling shutter error is considered small because it is in the order of the precision of the stereo system, which is approximately a milimeter depending on the exact depth.

Computation Time The computation time for processing each frame of image is computed on a MacBook Pro with 2.5 GHz Intel Core i5. The average time to read a frame from a video file and process for the stick location is about 109 ($\pm$ 12) ms per frame. Figure 8 shows a sample of the time it take for each frame in a video to find a line. The computation time to detect a line in each frame is less than the frame rate. Therefore, in principle, this 3D trajectory reconstruction method can be implemented in real time if the stereo cameras are already pre-synchronized either through software or hardware implementations.

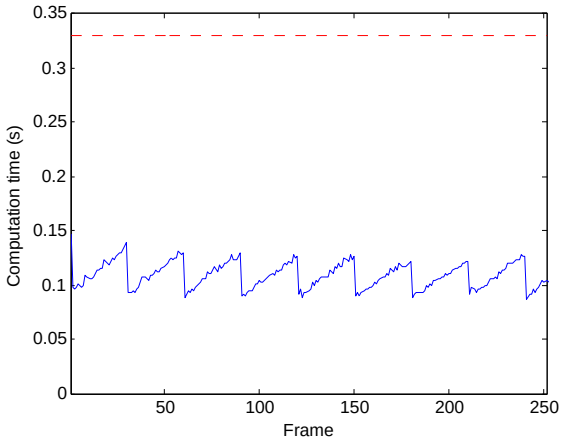


Figure 8: A test of computation time per frame for 251 frames in a video recording to locate 2D stick coordinates in each frame. The mean time is 109 ms with standard deviation of 12 ms. The red line indicate the frame rate of the video, and the blue line is the computation time for each frame.

Stick Length To measure the accuracy of this system, stick length computed from the 3D trajectory reconstruction algorithm is compared with the actual stick length. Figure 9 shows the computed stick length in a 251-frame video as compared to the actual stick length. Table 2 summarizes the results for three sticks with different lengths. Notice the small offset ($\sim$ 16 mm) of computed stick length from the actual stick length. This offset is due to the error in line detection. The filtering algorithm cannot distinguish between a fingertip and the bottom of a stick, and thus, counting the fingertip as part of the stick. One way to avoid this error is to use another color such as green that is less similar with skin tone. In the research project, this offset is not critical because only relative differences in trajectories are considered. The rest of the noise inherent from the filtering and smoothing noise during 2D stick trajectories extraction.

Actual Stick Length (m)	1.200	1.050	0.900
Computed Stick Length (m)			
Mean	1.217	1.064	0.919
Standard Deviation	0.013	0.005	0.007
Offset (m)	0.017	0.014	0.019

Table 2: A comparison of computed stick length and actual stick length for three sticks with different lengths. The average offset is approximately 16 mm.

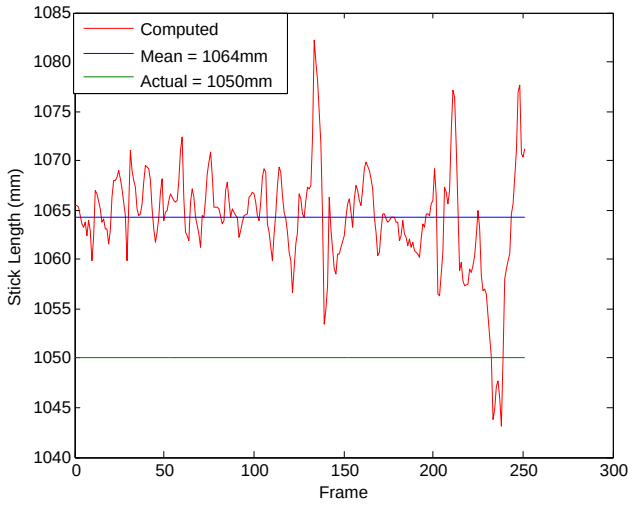


Figure 9: Stick length during a video recording. There are 251 frames in total. The computed length are calculated from the results of 3D trajectory reconstruction. The actual length is the measurement of a physical stick length. The average computed stick length is 1.064 m with a standard deviation of 5 mm.

4 Conclusion

A stereo vision tracking system using webcams are constructed for stick balancing experiments. Both hardware and software implementations are discussed, and system performance is analyzed. The three main parts for this stereo tracking system are stereo video capturing, post processing, and 3D trajectory recontruction of labeled targets. In general, it is feasible to build a stereo system that is relatively cheap, simple and precise.

The next step for this stereo system is to streamline the proces and create a user friendly graphic user interface. In addition, to achieve more automation, more sophisticated software packages such as Robotic Operating System (ROS) and OpenCV can be used to replace MATLAB and Window Movie Maker. Some of these software packages will allow for online camera synchronization. As mentioned earlier, the 3D trajectory reconstruction step can be extended to real time implementation if the cameras are pre-synchronized.

Lastly, in a broader sense, future work includes adding head pose estimation and eye tracking into the setup in order to conduct the human stick balancing experiment.

References

[1] J. Doyle, B. Francis, and A. Tannenbaum, *Feedback Control Theory*. Macmillan Publishing Co., 1990.

[2] J.-Y. Bouguet, *Camera Calibration Toolbox for MATLAB*. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/

[3] *Logitech C920 HD Pro Webcam*. [Online]. Available: http://www.logitech.com/en-us/support/hd-pro-webcam-c920?crid=405

[4] *Hough Transform*. [Online]. Available: http://en.wikipedia.org/wiki/Hough_transform

[5] *Stereo Stick Tracking*. [Online]. Available: http://youtu.be/cyAevMFHrdo

[6] *3D Stick Reconstruction*. [Online]. Available: http://youtu.be/otGTySqKdW8

[7] *Stereo Vision Tracking System MATLAB Code*. [Online]. Available: http://www.cds.caltech.edu/~yleong/codes/stereo_webcam_code.zip

Appendices

A MATLAB Implementation of 3D Trajectory Reconstruction

This is a sample code for implementing 3D trajectory reconstruction in MATLAB. The code requires user to download and use the Camera Calibration Toolbox for MATLAB at [2] because it uses the stereo triangulation function in the toolbox. This function relies on calibration data format that are output by the toolbox. The following code can be downloaded from [7].

Main 3D trajectory reconstruction code

```
%% Analyze stereo videos to obtain 3D stick trajectories

clear

% Read video files
subset = num2str(1);

leftcam = VideoReader(['sub4_cut1_left_set',subset,'.mp4']);
rightcam = VideoReader(['sub4_cut1_right_set',subset,'.mp4']);

% Extract 2D trajectories from each video
[traj_right,err_r] = extractTraj(rightcam,0.18);
[traj_left,err_l] = extractTraj(leftcam,0.19);

% Smooth raw trajectories
traj_right_s.point1 = smooth(traj_right.point1(:,1),'rlowess');
traj_right_s.point1(:,2) = smooth(traj_right.point1(:,2),'rlowess');
traj_right_s.point2 = smooth(traj_right.point2(:,1),'rlowess');
traj_right_s.point2(:,2) = smooth(traj_right.point2(:,2),'rlowess');

traj_left_s.point1 = smooth(traj_left.point1(:,1),'rlowess');
traj_left_s.point1(:,2) = smooth(traj_left.point1(:,2),'rlowess');
traj_left_s.point2 = smooth(traj_left.point2(:,1),'rlowess');
traj_left_s.point2(:,2) = smooth(traj_left.point2(:,2),'rlowess');

% Stereo triangulation
[Xc_s_left,Xc_s_right] = get3Dtraj([traj_left_s.point1;...
    traj_left_s.point2]'-1,[traj_right_s.point1;traj_right_s.point2]'-1,...
    'Calib_Results_stereo_2', 2);

% Compute stick length and its statistics
pen_length_right_s = Xc_s_right{1}-Xc_s_right{2};
pen_length_right_s = sqrt(pen_length_right_s(1,:).^2 + ...
    pen_length_right_s(2,:).^2+pen_length_right_s(3,:).^2);

mean(pen_length_right_s)
std(pen_length_right_s)

% Show stick trajectories on top of original video
animateVideoNTraj(leftcam,rightcam,traj_left_s,traj_right_s,1:nn,2,1,...
    'example_track.mp4')

% Show stick 3D trajectories relative to one of the stereo cameras
animate3DTraj(Xc_s_right,1:nn,6,'Right Camera',1,'example_3D.mp4')
```

Relevant functions

Extract 2D trajectory from a video

```
% extractTraj Extracts 2D trajectories from a videoReader object

% Inputs:
% vidobj - VideoReader object
% thresh - threshold for detecting red (should be around 0.15-0.2)
% frame_range - range of frame to analyze (optional)
% traj - original data input (optional)
%
% Outputs:
% trajout - trajectories of 4 fields that represent lines
%       -- point1 - starting end of a line
%       -- point2 - ending end of a line
%       -- theta - angle from top left corner clockwise from horizontal
%       -- rho - distance from top left corner
% err - index of frames with too many or no line detected

function [trajout,err] = extractTraj(vidobj,thresh,frame_range,traj)

if nargin == 2
    t = vidobj.NumberOfFrames; % find number of frame
    frame_range = 1:t;

    % initialize points
    traj.point1 = zeros(t,2);
    traj.point2 = zeros(t,2);
    traj.theta = zeros(t,1);
    traj.rho = zeros(t,1);
end

noline=0;

for ii = frame_range

    % pick a frame
    data = read(vidobj,ii);

    % subtract the red component from the grayscale image to extract the
    % red components in the image
    diff_im = imsubtract(data(:,:,1), rgb2gray(data));

    % use a median filter to filter out noise
    diff_im = medfilt2(diff_im, [3 3]);

    % convert the resulting grayscale image into a binary image.
    diff_im = im2bw(diff_im,thresh);

    % remove all those pixels less than 20px
    diff_im = bwareaopen(diff_im,10);

    % detect line in image
    [H, theta, rho] = hough(diff_im,'Theta',[-70:-0.5:-90,89.5:-0.5:70]);
    peaks = houghpeaks(H);
    lines = houghlines(diff_im,theta,rho,peaks,'FillGap',250,'MinLength',150);

    if length(lines)>1
        err(ii) = ii;
        fprintf('Step %g: Too many lines detected.\n', ii);
    elseif length(lines)<1
        err(ii) = ii;
```

```
        fprintf('Step %g: No line detected.\n', ii);
        noline =1;
    end

    % save data
    if noline % if no line detected, use the previous result
        traj.point1(ii,:)=traj.point1(ii-1,:);
        traj.point2(ii,:)=traj.point2(ii-1,:);
        traj.theta(ii,1)=traj.theta(ii-1,1);
        traj.rho(ii,1)=traj.rho(ii-1,1);
        noline=0;
    else
        traj.point1(ii,:)=lines(1).point1;
        traj.point2(ii,:)=lines(1).point2;
        if lines(1).theta>0
            traj.theta(ii,1)=lines(1).theta;
        else
            traj.theta(ii,1)=180+lines(1).theta;
        end
        traj.rho(ii,1)=abs(lines(1).rho);
    end
end

trajout = traj;

if exist('err','var')
    err(err==0)=[];
else
    err = -1;
end
end
```

Reconstruct 3D trajectory from left and right trajectory

```
% get3Dtraj computes 3D trajectories based on trajectories computed from
% two camera

% Input:
% left_cam_pt,right_cam_pt - trajectories from left and right camera
% calib - stereo calibration data
% nTar - number of points

% Output:
% Xc_1_left,Xc_1_right - trajectories in 3D space

function [Xc_1_left,Xc_1_right] = get3Dtraj(left_cam_pt,right_cam_pt,...
    calib, nTar)

% load calibration data
load(calib)

% initialization
Xc_1_left = cell(nTar,1);
Xc_1_right = cell(nTar,1);

len = length(left_cam_pt)/nTar;

% stereo triangulation
[Xc_1_left_all,Xc_1_right_all] = stereo_triangulation(left_cam_pt,...
    right_cam_pt,om,T,fc_left,cc_left,kc_left,alpha_c_left,fc_right,...
    cc_right,kc_right,alpha_c_right);

```

```
for ii = 1:nTar
    Xc_1_left{ii} = Xc_1_left_all([2 1 3],len*(ii-1)+1:len*ii);
    Xc_1_left{ii}(2,:) = -Xc_1_left{ii}(2,:);
    Xc_1_right{ii} = Xc_1_right_all([2 1 3],len*(ii-1)+1:len*ii);
    Xc_1_right{ii}(2,:) = -Xc_1_right{ii}(2,:);
end

end
```

Create an animation of stick in 3D

```
% animate3DTraj shows an animation of the stick detected in 3D relative to
% one of the stereo cameras
%
% Inputs:
% Xc_1_left - coordinates of stick relative to one of the stereo cameras
% frame_range - range of frame
% fig - figure number
% titl - figure title
% save_movie - 1 - save, 0 - don't save
% filename - saved movie file name
%
% Outputs:
% An animation of the stick detected in 3D relative to one of the stereo
% cameras

function animate3DTraj(Xc_1_left,frame_range,fig,titl,save_movie,filename)

figmov=figure(fig);

if save_movie == 1
    writerObj = VideoWriter(filename,'MPEG-4');
    open(writerObj);
end


for ii = frame_range
    plot3([Xc_1_left{1}(1,ii) Xc_1_left{2}(1,ii)],[Xc_1_left{1}(3,ii)...
        Xc_1_left{2}(3,ii)],[Xc_1_left{1}(2,ii) Xc_1_left{2}(2,ii)],...
        'LineWidth',5);
    axis([-1000 1500 1560 2000 -600 600])
    xlabel('x (mm)')
    ylabel('z (mm)')
    zlabel('y (mm)')
    title(titl)
    view(20, 10)
    grid on
    drawnow

    if save_movie == 1
        frame = getframe(figmov);
        writeVideo(writerObj,frame);
    end
    if save_movie ==0
        pause(0.05);
    end
end

end
```

Overlay stick detected on video

```
% animateVideoNTraj shows an animation of the video and stick detected
%
% Inputs:
% leftcam, rightcam - VideoReader objects
% traj_left, traj_right - outputs of extractTraj
% frame_range - range of frame
% fig - figure number
% save_movie - 1 - save, 0 - don't save
% filename - saved movie file name
%
% Outputs:
% An animation of the sticks detected overlaid on top the videos.

function animateVideoNTraj(leftcam,rightcam,traj_left,traj_right,frame_range,...
    fig,save_movie,filename)

figmov=figure(fig);
height = leftcam.Height;

if save_movie == 1
    writerObj = VideoWriter(filename,'MPEG-4');
    open(writerObj);
end

for ii = frame_range

    % left camera
    data = read(leftcam,ii);
    subplot(1,2,1)
    imshow(imrotate(data,-90));
    hold on
    plot(height-[traj_left.point1(ii,2),traj_left.point2(ii,2)],...
        [traj_left.point1(ii,1),traj_left.point2(ii,1)],'g','Linewidth',2)
    hold off
    title('Left Camera')

    % right camera
    data = read(rightcam,ii);
    subplot(1,2,2)
    imshow(imrotate(data,-90));
    hold on
    plot(height-[traj_right.point1(ii,2),traj_right.point2(ii,2)],...
        [traj_right.point1(ii,1),traj_right.point2(ii,1)],'g','Linewidth',2)
    hold off
    title('Right Camera')

    if save_movie == 1
        frame = getframe(figmov);
        writeVideo(writerObj,frame);
    end
    if save_movie == 0
        pause(0.005)
    end
end

end
```

B Calibration Results

Calibration Set 1

Intrinsic parameters of left camera:

Focal Length:	fc_left =	$[968.03122 \quad 957.59714] \pm [3.49931 \quad 2.68883]$
Principal point:	cc_left =	$[681.70233 \quad 361.10460] \pm [1.13693 \quad 3.51549]$
Skew:	alpha_c_left =	$[0.00000] \pm [0.00000]$ => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:	kc_left =	$[0.04047 \quad -0.12609 \quad 0.01719 \quad 0.00340 \quad 0.00000] \pm$ $[0.00330 \quad 0.00859 \quad 0.00050 \quad 0.00048 \quad 0.00000]$

Intrinsic parameters of right camera:

Focal Length:	fc_right =	$[940.31927 \quad 935.73421] \pm [4.06831 \quad 3.86186]$
Principal point:	cc_right =	$[640.63975 \quad 360.19436] \pm [1.32970 \quad 2.73572]$
Skew:	alpha_c_right =	$[0.00000] \pm [0.00000]$ => angle of pixel axes = 90.00000 ± 0.00000degrees
Distortion:	kc_right =	$[0.03310 \quad -0.08164 \quad -0.01151 \quad 0.00111 \quad 0.00000] \pm$ $[0.00367 \quad 0.01002 \quad 0.00055 \quad 0.00056 \quad 0.00000]$

Extrinsic parameters (position of right camera with respect to left camera):

Rotation vector:	om =	$[0.78180 \quad 0.06229 \quad -0.00329] \pm [0.00372 \quad 0.00190 \quad 0.00087]$
Translation vector:	T =	$[-17.05754 \quad 1424.81444 \quad 653.06808] \pm [1.19876 \quad 2.39382 \quad 5.79266]$

Calibration Set 2

Intrinsic parameters of left camera:

Focal Length:	fc_left =	$[971.66176 \quad 966.94492] \pm [4.18198 \quad 3.47311]$
Principal point:	cc_left =	$[688.20551 \quad 376.88444] \pm [1.27313 \quad 3.58761]$
Skew:	alpha_c_left =	$[0.00000] \pm [0.00000]$ => angle of pixel axes = 90.00000 ± 0.00000degrees
Distortion:	kc_left =	$[0.04369 \quad -0.12351 \quad 0.01743 \quad 0.00892 \quad 0.00000] \pm$ $[0.00341 \quad 0.00859 \quad 0.00057 \quad 0.00053 \quad 0.00000]$

Intrinsic parameters of right camera:

Focal Length:	fc_right =	$[925.49682 \quad 919.93815] \pm [4.09589 \quad 3.59904]$
Principal point:	cc_right =	$[635.43093 \quad 351.28293] \pm [1.43807 \quad 3.51640]$
Skew:	alpha_c_right =	$[0.00000] \pm [0.00000]$ => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:	kc_right =	$[0.04751 \quad -0.13602 \quad -0.01623 \quad -0.00236 \quad 0.00000] \pm$ $[0.00433 \quad 0.01212 \quad 0.00064 \quad 0.00060 \quad 0.00000]$

Extrinsic parameters (position of right camera with respect to left camera):

Rotation vector:	om =	$[0.76825 \quad 0.08580 \quad -0.00124] \pm [0.00419 \quad 0.00200 \quad 0.00096]$
Translation vector:	T =	$[-30.28130 \quad 1448.87887 \quad 622.09604] \pm [1.31420 \quad 2.20786 \quad 5.07684]$

Calibration Set 3

Intrinsic parameters of left camera:

Focal Length:	fc_left =	$[953.88517 \quad 950.87995] \pm [4.01808 \quad 3.31867]$
Principal point:	cc_left =	$[687.48430 \quad 388.01209] \pm [1.12759 \quad 3.52284]$
Skew:	alpha_c_left =	$[0.00000] \pm [0.00000]$ => angle of pixel axes = 90.00000 ± 0.00000degrees
Distortion:	kc_left =	$[0.03603 \quad -0.09137 \quad 0.01800 \quad 0.01034 \quad 0.00000] \pm$ $[0.00341 \quad 0.00848 \quad 0.00051 \quad 0.00049 \quad 0.00000]$

Intrinsic parameters of right camera:

Focal Length:	fc_right =	$[927.92425 \quad 924.20190] \pm [3.98476 \quad 3.55081]$
Principal point:	cc_right =	$[631.12051 \quad 343.23754] \pm [1.37664 \quad 3.29102]$
Skew:	alpha_c_right =	$[0.00000] \pm [0.00000]$ => angle of pixel axes = 90.00000 ± 0.00000degrees
Distortion:	kc_right =	$[0.03993 \quad -0.10319 \quad -0.01730 \quad -0.00536 \quad 0.00000] \pm$ $[0.00396 \quad 0.01094 \quad 0.00062 \quad 0.00059 \quad 0.00000]$

Extrinsic parameters (position of right camera with respect to left camera):

Rotation vector:	om =	$[0.75494 \quad 0.09232 \quad -0.00338] \pm [0.00396 \quad 0.00186 \quad 0.00093]$
Translation vector:	T =	$[-26.73469 \quad 1440.78117 \quad 644.40467] \pm [1.27179 \quad 2.15776 \quad 4.75556]$

C Rolling Shutter Effect

A simple calculation of the rolling shutter effect is described here. Based on the calculation, a typical error of rolling shutter is in the range of a milimeter.

Given that the frame rate of the webcam is 30 fps, assume that the camera takes less than $\frac{1}{30}$ seconds to read the whole image. The image is acquired line by line from the right to the left relative to the image taken. The line orientation is vertical on the image plane (i.e. similar to the stick orientation). There are 720 pixels across the width of the frame. Therefore, the recording period per line is $\frac{1}{30} \times \frac{1}{720}$ seconds.

Based on pilot experiments, the mean number of pixels a stick crosses horizontally relative to the image is 100. Thus, the average delay is no greater than $\frac{1}{30} \times \frac{1}{720} \times 100 = 4.63$ miliseconds.

Similarly, based on pilot experiments, the average speed of stick motion is about 0.25 m/s. Then, the stick's travelling distance is approximately 0.25 m/s $\times$ 4.63 ms = 1.16 mm. The stick's crosssection diameter is about 9.53 mm. Hence, the error is approximately 12% relative to the stick's crosssection diameter.