**Navigation Codes**

This file contains the main codes used for implementing the navigation algorithm for a single particle manipulation where the next steps show the correct matlab codes to be executed:

1. Snap Number of Consecutive Images
2. Accumulated Images
3. Generation of VVFMs
4. Control Code
5. Simulation Code

# **Appendix 1: Matlab Codes**

## **A.1.1 Control Code**

% FINAL MATLAB CODE FOR LATERAL BEAD'S TRAJECTORY CONTROL USING USW.

% STEPS AS THE FOLLOWING:

% 1- capture image & calculate the started position

% (find one bead's position on the window).

% 2- steering loop (capture image & return the current position)

% 3- decide where to steer the bead by calculating the required direction.

% (pattern or the trajectory is already given OR by using start & final coordinates).

% 4- select closest force map that match the required direction.

% 5- examine if the required position achieved.

% 6- if not go to step 2.( if yes print final results and end the process ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%% Initialization part %%%%%%%%%%%%%%%%%%%%

clc % clear all previous results

for n=2:5 % read 4 reference FMs ( all u & v and directions ).

U\_[n]=dlmread(['F:\9Feb16\NewFMs4\',num2str(n),'\new',num2str(n),'\iu{',num2str(n),'}.mat']);

U\_{n}=dlmread(['F:\9Feb16\NewFMs4\',num2str(n),'\new',num2str(n),'\iu{',num2str(n),'}.mat']);

V\_{n}=dlmread(['F:\9Feb16\NewFMs4\',num2str(n),'\new',num2str(n),'\iv{',num2str(n),'}.mat']);

WDir{n}=wrapTo2Pi(dlmread(['F:\9Feb16\NewFMs4\',num2str(n),'\new',num2str(n),'\dir{',num2str(n),'}.mat']));

Dir{n}=dlmread(['F:\9Feb16\NewFMs4\',num2str(n),'\new',num2str(n),'\dir{',num2str(n),'}.mat']);

end

% pixels indexing tables

CY= 65.5:64:897.5;

RX= 65.5:64:1153.5;

[XX,YY] = meshgrid(RX,CY);% rectangular full grid

% real coordinates of the start point(X(1),Y(1)) & final destination

X(1)=987;

Y(1)=522 ;

X\_final=750;

Y\_final=522;

MWP=0;% number of midway points

Total\_time=0; % total manipulation time

x= [];y= []; % initialize display vectors ( actual coordinates )

FM=[];% this matrix is the selected force maps after each direction calculation.

SELC\_D=[];% selected direction matrix

REQ\_D=[];% selected direction matrix.

All\_Dir=[];% matrix of all interpolated directions at each step.

t=[];% time matrix shows the time require for each step.

Xp=[];% predicted x coordinates matrix

Yp=[];% predicted y coordinates matrix

U = [];V = []; % turn on if quiver command is needed or some other total force calculation is required.

%%%%%%%%%%%%%% the main part %%%%%%%%%%%%%%%%%%%%

Switch\_P\_V()%switch on/off the pump/valve.

pause(27);% focus time for the first frequency.(if first voltage is 3v).

img = capture\_image();% snap an image

[X\_new,Y\_new]=Search\_particle(img,X(1),Y(1));% find the particle at starting position.

imwrite(img,'F:\9Feb16\FstCtrl42\img(1).jpg');% save the first image.

tic % start of time counter

% loop until particle arrive to the final destination

for i=1:inf% attempts number to reach the target

X(i)=X\_new;Y(i)=Y\_new;

% display real time trajectory

[x,y]=Dis\_traj(x,y,X(i),Y(i));

cur\_pos\_error=sqrt((X\_final-X(i))^2+(Y\_final-Y(i))^2);

if cur\_pos\_error <= 20

% this executes if reach the target.( true if condition )

% printing final parametric results

End\_Fun(X(i),X(1),Y(i),Y(1),X\_final,Y\_final,MWP,Total\_time)

break;% finish the manipulation loop if target is reached.

else% continue the manipulation loop if target is not reached.

MWP=MWP+1; % increase attempts number

% find the force map for the best direction

[u,v,fm,Selc\_dir,Req\_dir,d]=DirCal13(X(i),Y(i),X\_final,Y\_final,XX,YY,U\_,V\_,Dir,WDir);

U=[U,u];

V=[V,v];

FM=[FM,fm];

SELC\_D=[SELC\_D,Selc\_dir];

REQ\_D=[REQ\_D,Req\_dir];

All\_Dir=[All\_Dir,d];

%%%%%%% Time calculation for each displacement.

T=20/sqrt(u^2+v^2);% T=Disp/Mag(u,v)

t=[t,T];

Total\_time=Total\_time+T; % total manipulation time accumulation

%%% applying second frequency and its direction for T time

Apply\_Freq(T,fm)

% return the next calculated position

X(i+1)=T\*u+X(i);

Y(i+1)=T\*v+Y(i);

Xp=[Xp X(i+1)];

Yp=[Yp Y(i+1)];

img = capture\_image();% snap an image

imwrite(img,['F:\9Feb16\FstCtrl42\img(',num2str(i+1),').jpg']);% save the next image might have other particles.

[X\_new,Y\_new]=Search\_particle(img,X(i+1),Y(i+1));% find the particle at the next real position.

end % condition to reach target

end % number of attempts to reach target

%%%%%%%%%%%% PRINT OF FINAL PATTERN %%%%%%%%%%%%%%%

Final\_Pat(x,y)

%%%%%%%%%%%% save all data %%%%%%%%%%%%%%%

File\_N=42;

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\U.txt'],U);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\V.txt'],V);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\t.txt'],t);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\FM.txt'],FM);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\x.txt'],x);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\y.txt'],y);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\Xp.txt'],Xp);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\Yp.txt'],Yp);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\SELC\_D.txt'],SELC\_D);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\REQ\_D.txt'],REQ\_D);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\All\_Dir.txt'],All\_Dir);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\U.mat'],U);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\V.mat'],V);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\t.mat'],t);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\FM.mat'],FM);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\x.mat'],x);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\y.mat'],y);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\Xp.mat'],Xp);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\Yp.mat'],Yp);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\SELC\_D.mat'],SELC\_D);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\REQ\_D.mat'],REQ\_D);

dlmwrite(['F:\9Feb16\FstCtrl',num2str(File\_N),'\All\_Dir.mat'],All\_Dir);

toc % end of time counter

**A.1.2 Capture Image**

function img = capture\_image();

% this function snap one image by using the interface to Thorlab camera, and most of the code written in micro manager environment and return the image in such a way that can be understandable by Matlab. For more information go to https://micro-manager.org/wiki/Matlab\_Configuration.

% addpath 'C:\Program Files\Micro-Manager-1.4.21\scripts\M1.bsh'

% addpath 'C:\Program Files\Micro-Manager-1.4.21\ImageJ.exe'

javaaddpath 'F:\MATLAB\R2014a\java\mij.jar'

javaaddpath 'F:\MATLAB\R2014a\java\1j.jar'

javaaddpath 'F:\MATLAB\R2014a\java\jmi.jar'

import java.lang.System.\*;

import com.mathworks.jmi.\*;

import java.io.FileWriter;

import java.io.IOException;

import javax.swing.JTextField;

import javax.swing.JLabel;

import javax.swing.JFrame;

import java.awt.Container;

import java.awt.FlowLayout;

import java.lang.Math;

import mmcorej.StrVector.\*;

import org.micromanager.api.Autofocus.\*;

import org.micromanager.acquisition.MMAcquisition.\* ;

import org.micromanager.utils.MMScriptException.\*;

% import org.micromanager.api.AcquisitionOptions;

import java.util.Vector;

import ij.\*;

import ij.gui.GenericDialog.\*;

import java.util.Date;

import java.text.DateFormat;

import java.text.SimpleDateFormat;

% % % initialization of MMCore acquisition.

import mmcorej.\* % load the ?manager library

core = CMMCore(); % instantiate an object

core.loadSystemConfiguration('C:\Program Files\Micro-Manager-1.4.21\MMConfig\_thorlabs.cfg');

java.lang.System.clearProperty('java.util.prefs.PreferencesFactory');

% capture image

core.setExposure(75); % set exposure

core.snapImage();

img = core.getImage(); % returned as a 1D array of signed integers in row-major order

width = core.getImageWidth();

height = core.getImageHeight();

if core.getBytesPerPixel == 2

pixelType = 'uint16';

else

pixelType = 'uint8';

end

img = typecast(img, pixelType); % pixels must be interpreted as unsigned integers

img = reshape(img, [width, height]); % image should be interpreted as a 2D array

img = transpose(img); % make column-major order for MATLAB

imshow(img); % show image

## **A.1.3 Find particle**

function [X\_new,Y\_new]=Search\_particle(img,X\_cur,Y\_cur)

% This function uses the current snapped image to find the particle at specific coordinates(X\_cur, Y\_cur)and return the exact position where the position error margin is set to the lowest possible value. This function might not be designed for finding particle in group with other that make it limited.

% Estimate the Background illumination

background = imopen(img,strel('disk',45));

% Subtract the Background Image from the Original Image

I2 = img - background;

% light adjust

a = imadjust(I2,[0.1 0.5],[]);

imshow(a);

% identify the coordinates of the all particles position

[centers, radii] = imfindcircles(a,[1 10], 'Sensitivity',0.9);

[r, c] = size(centers);

z=0;%internal variable for the next loop

for R = 1:r

margin\_pos\_error=sqrt((centers(R)-X\_cur)^2+(centers(R,c)-Y\_cur)^2);

if margin\_pos\_error <= 30 % error margin is 30 pixels

disp('the particle arrived to the new position');

X\_new = centers(R);

Y\_new = centers(R,c);

disp X\_new= ,disp (X\_new)

disp Y\_new= ,disp (Y\_new)

break; % terminate loop if particle is found.

else % continue loop if particle is not found.

z=z+1;

end

end

## **A.1.4 Display the Moving Trajectory**

function [x,y]=Dis\_traj(x,y,X,Y)

% this function display the instantaneous moving particle which is represented by a circle. The x & y matrices are used for gather and accumulate all calculated points in order to display the whole trajectory.

x = [x,X]; % display x vector/matrix

y = [y,Y]; % display y vector/matrix

DELAY = 0.2;% slow down the speed of calculation in order to see the trajectory.

% clf; % show only the moved particle

grid on

hold on;

scatter(x,y);% display circle at every moving position

line(x,y);% connect points by drawing line

axis([0 ,1200,0 ,990]);% define the display boundaries

xlabel('X(pixels)');

ylabel('Y(pixels)');

title('Real Time Motion ');

set(gca,'YDir','reverse');

pause(DELAY);

## **A.1.5 Display Final Manipulation Details**

function End\_Fun(X,Y,X\_final,Y\_final,MWP,Total\_time,Total\_Disp)

% this function generate the massage box which contains the final manipulation parameters, the h variable is used to be able to make all the information in one display box.

Delta\_x=X\_final-X;

Delta\_y=Y\_final-Y;

s1=0.00149; % pixels to mm scaller

h = msgbox({' SUCCESSFULL MANIPULATION ' ,' ', 'final position X ' num2str(X) ' final position Y ' num2str(Y)...'final position error in pixels(X,Y)' num2str(Delta\_x) num2str(Delta\_y) 'total displacement in pixels' num2str(Total\_Disp)...' attempts number' num2str(MWP) 'Total Experimental Time(seconds)' num2str(Total\_time)} );

set(h, 'position', [300 200 300 300]); % control display box position within the screen.

ah = get( h, 'CurrentAxes' );

ch = get( ah, 'Children' );

set( ch, 'FontSize', 15 ); % makes text font size to 15.

## **A.1.6 Direction Calculation**

function [u,v,fm,Selc\_dir,Req\_dir,d]=DirCal13(X,Y,X\_final,Y\_final,XX,YY,U\_,V\_,Dir,WDir)

%%% this function return the the force map (fm) of the closest possible direction to the required one by using the direction interpolation. And it also returns the velocity components of the best force map at the desired position.

% calculation of the desired direction using two points(current & final point)

Delta\_x=X\_final-X;

Delta\_y=Y\_final-Y;

Req\_dir = wrapTo2Pi(atan2(Delta\_y,Delta\_x));% desired direction

d = []; % initialize directions matrix/vector

for n=2:5 % read all FMs in term of direction at the specified coordinates

% read/interpolate the direction from all FMs

A= wrapTo2Pi(interp2(XX,YY,Dir{n},X,Y)); % right direction

d = [d,A]; % write the direction in directions matrix

end

% extract the closest direction from all FMs at that specified coordinates.

[~, ind]=min(abs(wrapToPi(Req\_dir-d)));

fm=ind+1;

% directions matrix internal index correction to return the fm number.

Selc\_dir=wrapTo2Pi(interp2(XX,YY,Dir{fm},X,Y));

%%%%% interpolate u&v forces for the new position X&Y.

u = interp2(XX,YY,U\_{fm},X,Y);

v = interp2(XX,YY,V\_{fm},X,Y);

end

## **A.1.7 Apply the Second Frequency**

function Apply\_Freq(T,fm)

% this function applies the lateral frequency to the levitated particle for a specified time T. The table is used to help for finding the matching frequency for a certain force map.

Frq\_Table=[2 2.52e6;3 2.53e6;4 2.54e6;5 2.56e6];

Sec\_Freq=Frq\_Table((fm),2);

a=sprintf(['second freq is ',num2str(Sec\_Freq)]);disp(a);

% % % % % % switch on lateral frequency.

disp('start lateral frequency')

s = serial('COM6');

fopen(s);

pause(0.2);

fprintf(s,[ 'WAVFREQ ' num2str(Sec\_Freq) ]);

pause(0.2);

fprintf(s,'AMPL 10' );

fprintf(s,'BEEP');

fprintf(s,'OUTPUT ON');

pause(T);

fprintf(s,'OUTPUT OFF');

pause(0.2);

fclose(s);

delete(s);

## **A.1.8 Display the Final Pattern**

function Final\_Pat(x,y)

% This function generate the final pattern after the particle has reached the target, this is fitted to a larger screen scale for more visualization , this display can also be moved to anywhere within the screen.

ss = get(0,'screensize'); %The screen size

width = ss(4);

height = ss(4);

H = figure; %Make a new figure

scatter(x,y);% display circle at every moving position

line(x,y);% connect points by drawing line

grid on

axis('equal','tight');

xlabel('X(pixels)');

ylabel('Y(pixels)');

title('FinalPattern ');

set(gca,'YDir','reverse');

%Let's say we want it to take up 450 by 450 region of the screen:

vert = 450; %450 vertical pixels

horz = 450; %450 horizontal pixels

%%This will place the figure in the top-right corner

% set(H,'Position',[width-horz, height-vert, horz, vert]);

%You can now move it to the middle of the screen if you like:

set(H,'Position',[(width/2.5)-horz/1.5, (height/2.5)-vert/1.5, horz, vert]);

## **A.1.9 Simulation Code**

% THIS SIMULATION CODE IS A TOOL TO PREDICT THE PARTICLE MOVEMENT TRAJECTORY THAT USES A REAL REFERENCE FORCE MAPS FOR DIRECTION AND X,Y VELOCITY COMPONENT AS WELL. THIS CODE DEVIDED INTO TWO PART (INITIALIZATION & MAIN BODY).FOR THE HIGHER REQUIRED DISPLACEMENT,LESS ACCURACY OF REACHING THE TARGET.THIS CODE USUES THE 2D INTERPOLATION FOR BETTER PARTICLE MANIPULATION.

%%%%%%%%%%%%%% Initialization part %%%%%%%%%%%%%%%%%%%%

clc % clear all previous results

tic % start of simulation time counter

for n=2:5 % read 4 reference FMs ( all u & v and directions ).

U\_{n}=dlmread(['F:\9Feb16\NewFMs4\',num2str(n),'\new',num2str(n),'\iu{',num2str(n),'}.mat']);

V\_{n}=dlmread(['F:\9Feb16\NewFMs4\',num2str(n),'\new',num2str(n),'\iv{',num2str(n),'}.mat']);

WDir\_{n}=wrapTo2Pi(dlmread(['F:\9Feb16\NewFMs4\',num2str(n),'\new',num2str(n),'\dir{',num2str(n),'}.mat']));

Dir\_{n}=dlmread(['F:\9Feb16\NewFMs4\',num2str(n),'\new',num2str(n),'\dir{',num2str(n),'}.mat']);

end

% pixels indexing tables

CY= 65.5:64:897.5;

RX= 65.5:64:1153.5;

[XX,YY] = meshgrid(RX,CY);

% real coordinates of the start point(X(1),Y(1)) & final destination

%%% all directions

X(1)=523.5 ;

Y(1)=576.5 ;

X\_final= 666 ;

Y\_final=615;

MWP=0;% number of midway points.

Total\_time=0; % total manipulation time.

x= [];y= []; % initialize display vectors.

FM=[];% this matrix is the selected force maps after each direction calculation.

SELC\_D=[];% selected direction matrix.

REQ\_D=[];% selected direction matrix.

All\_Dir=[];% matrix of all interpolated directions at each step.

t=[];% time matrix shows the time require for each step.

U = [];V = []; % matrics of x,y velocity components for each step.

%%%%%%%%%%%%%% the main part %%%%%%%%%%%%%%%%%%%%

% loop until particle arrive to the final destination

for i=1:inf% attempts number to reach the target

% display real time trajectory

[x,y]=Dis\_traj(x,y,X(i),Y(i));

cur\_pos\_error=sqrt((X\_final-X(i))^2+(Y\_final-Y(i))^2);

if cur\_pos\_error <= 20

%this executes if reach the target.( true if condition )

break;% finish the manipulation loop if target is reached.

else% continue the manipulation loop if target is not reached.

MWP=MWP+1;

% find the force map for the best direction

[u,v,fm,Selc\_dir,Req\_dir,d]=DirCal13(X(i),Y(i),X\_final,Y\_final,XX,YY,U\_,V\_,Dir\_,WDir\_);

U=[U,u];

V=[V,v];

FM=[FM,fm];

SELC\_D=[SELC\_D,Selc\_dir];

REQ\_D=[REQ\_D,Req\_dir];

All\_Dir=[All\_Dir,d];

%%%%%%% Time calculation for each displacement.

T=20/sqrt(u^2+v^2);

t=[t,T];

Total\_time=Total\_time+T; % total manipulation time (accumulation)

% return the next calculated position

X(i+1)=T\*(u)+X(i);

Y(i+1)=T\*(v)+Y(i);

end

end

%%%%%%%%%%%% PRINT OF FINAL PATTERN %%%%%%%%%%%%%%%

[Total\_Disp,Step\_disp]=Displacement(MWP,x,y);

%%%%%%%%%%%% printing final parametric results

End\_Fun(X(i),Y(i),X\_final,Y\_final,MWP,Total\_time,Total\_Disp)

Final\_Pat(x,y)

%%%%%%%%%%% save all data points in mat & txt format %%%%%%%

File\_N=1;

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sU.txt'],U);

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sV.txt'],V);

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sT.txt'],t);

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sFM.txt'],FM);

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sX.txt'],x);

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sY.txt'],y);

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sSELC\_D.txt'],SELC\_D);

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sREQ\_D.txt'],REQ\_D);

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sAll\_Dir.txt'],All\_Dir);

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sU.mat'],U);

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sV.mat'],V);

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sT.mat'],t);

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sFM.mat'],FM);

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sX.mat'],x);

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sY.mat'],y);

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sSELC\_D.mat'],SELC\_D);

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sREQ\_D.mat'],REQ\_D);

dlmwrite(['F:\9Feb16\FstCtrl35\simulation\',num2str(File\_N),'\sAll\_Dir.mat'],All\_Dir);

toc % end of simulation time counter

%

## **A.1.10 Real Displacement Calculation**

function [Total\_Disp,Step\_disp]=Displacement(MWP,x,y)

% this function calculates the total real displacement that the particle moved from the starting point to the final point. Also produce a matrix of all the step displacements each time.

Step\_disp=[];

for i=1:MWP

disp =sqrt((y(i+1)-y(i))^2+(x(i+1)-x(i))^2);

Step\_disp=[Step\_disp disp];

end

Total\_Disp= sum(Step\_disp);

Step\_disp;

end

## **A.1.11 Accumulated Images**

% This code accumulate number of sequence images after modifying the contrast quality in order to show the whole particle movement.

% read 50 images

for n=1:50

img\_{n}= imread(['F:\9Feb16\FstCtrl35\img(',num2str(n),').jpg']);

% Estimate the Background illumination

background = imopen(img\_{n},strel('disk',15));

% Subtract the Background Image from the Original Image

img\_{n} = img\_{n} - background;

% light adjust

img\_{n} = imadjust(img\_{n},[0.1 0.2],[]);

% imwrite(img\_{n},['F:\9Feb16\FstCtrl38\modified\',num2str(n),'.jpg']);

end

% accumulation of all images

A=imadd(img\_{1},img\_{2});

for o=3:50

A=imadd(A,img\_{o});

end

figure;

imshow(A);

title(A);

## **A.1.12 Generation of VVFMs**

% This code generate the VVFMs after consecutive images being captured by the experiments. This code uses the µPIV package (toolbox in MATLAB).

% % read 50 images

for n=1:50

img\_{n}= imread(['F:\9Feb16\NewFMs4\Fp1\',num2str(n),'.jpg']);

% Estimate the Background illumination

background = imopen(img\_{n},strel('disk',55));

% Subtract the Background Image from the Original Image

img\_{n} = img\_{n} - background;

% light adjust

img\_{n} = imadjust(img\_{n},[0.1 0.2],[]);

end

% accumlation of odd images

A=imadd(img\_{1},img\_{3});

for o=5:2:49

A=imadd(A,img\_{o});

end

figure;

imshow(A);

title(A);

% counting number of beads

aa=double(A);

bw = im2bw(aa);

cc = bwconncomp(bw,26);

cc.NumObjects

% accumlation of even images

B=imadd(img\_{2},img\_{4});

for e=6:2:50

B=imadd(B,img\_{e});

end

figure;

imshow(B);

title(B);

% µPIV calculation

[xi,yi,iu,iv]=mpiv(A,B,128,128,0.5,0.5,0,0,0.5,'cor',0,1);

[iu\_f,iv\_f,iu\_i,iv\_i]=mpiv\_filter(iu,iv,2,2,3,1);

[iu\_s,iv\_s]=mpiv\_smooth(iu\_i,iv\_i,1);

% equalizing the matrix dimensions for Xi,Yi,X1,Y1.

Xr=transpose(iu\_s);

Yr=transpose(iv\_s);

s1=0.00149;% new axis scaller

figure;

q1=quiver(xi,yi,Xr,Yr,1);

axis([0 ,1200 ,0 ,1000]);

% axis([0 ,1.85 ,0 ,1.45]);

xlabel('X(pixels)');

ylabel('Y(pixels)');

set(gca,'YDir','reverse');

title('Fp ');

% legend('12.6 p/s');

% save magnitudinal & directional FMs to files

mag=sqrt(iu\_s'.^2+iv\_s'.^2);

dir=atan2(iv\_s',iu\_s');

wdir=wrapTo2Pi(dir);

% dlmwrite('F:\9Feb16\NewFMs4\3\mag{1.0}.txt',mag);

% dlmwrite('F:\9Feb16\NewFMs4\3\Dir{1.0}.txt',dir);

% % save velocities FMs to files

% File\_N=2;

dlmwrite(['F:\9Feb16\NewFMs4\',num2str(File\_N),'\new',num2str(File\_N),'\dir{',num2str(File\_N),'}.mat'],dir);

dlmwrite(['F:\9Feb16\NewFMs4\',num2str(File\_N),'\new',num2str(File\_N),'\wdir{',num2str(File\_N),'}.mat'],wdir);

dlmwrite(['F:\9Feb16\NewFMs4\',num2str(File\_N),'\new',num2str(File\_N),'\mag{',num2str(File\_N),'}.mat'],mag);

dlmwrite(['F:\9Feb16\NewFMs4\',num2str(File\_N),'\new',num2str(File\_N),'\iu{',num2str(File\_N),'}.mat'],iu\_s');

dlmwrite(['F:\9Feb16\NewFMs4\',num2str(File\_N),'\new',num2str(File\_N),'\iv{',num2str(File\_N),'}.mat'],iv\_s');

dlmwrite(['F:\9Feb16\NewFMs4\',num2str(File\_N),'\new',num2str(File\_N),'\dir{',num2str(File\_N),'}.txt'],dir);

dlmwrite(['F:\9Feb16\NewFMs4\',num2str(File\_N),'\new',num2str(File\_N),'\wdir{',num2str(File\_N),'}.txt'],wdir);

dlmwrite(['F:\9Feb16\NewFMs4\',num2str(File\_N),'\new',num2str(File\_N),'\mag{',num2str(File\_N),'}.txt'],mag);

dlmwrite(['F:\9Feb16\NewFMs4\',num2str(File\_N),'\new',num2str(File\_N),'\iu{',num2str(File\_N),'}.txt'],iu\_s');

dlmwrite(['F:\9Feb16\NewFMs4\',num2str(File\_N),'\new',num2str(File\_N),'\iv{',num2str(File\_N),'}.txt'],iv\_s');

## **A.1.13 Snap Number of Consecutive Images**

## 

% this code snap number of images repetitive measurements in order to calculate the lateral velocity.

addpath 'C:\Program Files\Micro-Manager-1.4.21\ImageJ.exe'

javaaddpath 'F:\MATLAB\R2014a\java\mij.jar'

javaaddpath 'F:\MATLAB\R2014a\java\1j.jar'

javaaddpath 'F:\MATLAB\R2014a\java\jmi.jar'

import java.lang.System.\*;

import com.mathworks.jmi.\*;

import java.io.FileWriter;

import java.io.IOException;

import javax.swing.JTextField;

import javax.swing.JLabel;

import javax.swing.JFrame;

import java.awt.Container;

import java.awt.FlowLayout;

import java.lang.Math;

import mmcorej.StrVector.\*;

import org.micromanager.api.Autofocus.\*;

import org.micromanager.acquisition.MMAcquisition.\* ;

import org.micromanager.utils.MMScriptException.\*;

% import org.micromanager.api.AcquisitionOptions;

import java.util.Vector;

import ij.\*;

import ij.gui.GenericDialog.\*;

import java.util.Date;

import java.text.DateFormat;

import java.text.SimpleDateFormat;

clc

% % % initilization of MMCore acquisition.

import mmcorej.\* % load the ?manager library

core = CMMCore(); % instantiate an object

core.loadSystemConfiguration('C:\Program Files\Micro-Manager-1.4.21\MMConfig\_thorlabs.cfg');

java.lang.System.clearProperty('java.util.prefs.PreferencesFactory');

for i = 1:2:50

Switch\_P\_V();

pause(27);

% % % % % % switch on lateral frequency.

disp('start lateral frequency')

s = serial('COM6');

fopen(s);

pause(0.2);

fprintf(s, 'WAVFREQ 2.56e6');

pause(0.2);

fprintf(s,'AMPL 10');

pause(0.2);

fprintf(s,'BEEP');

fprintf(s,'OUTPUT ON');

fclose(s);

delete(s);

% % capture image

% For more information go to https://micro-manager.org/wiki/Matlab\_Configuration.

core.setExposure(75); % set exposure

core.snapImage();

img = core.getImage(); % returned as a 1D array of signed integers in row-major order

width = core.getImageWidth();

height = core.getImageHeight();

if core.getBytesPerPixel == 2

pixelType = 'uint16';

else

pixelType = 'uint8';

end

img = typecast(img, pixelType); % pixels must be interpreted as unsigned integers

img = reshape(img, [width, height]); % image should be interpreted as a 2D array

img = transpose(img); % make column-major order for MATLAB

imshow(img);

imwrite(img,['F:\9Feb16\NewFMs4\Fp1\',num2str(i),'.jpg']);

pause(0.5); % time between pairs

p=0;

p=i+1

% capture image

% For more information go to https://micro-manager.org/wiki/Matlab\_Configuration.

core.setExposure(75); % set exposure

core.snapImage();

img = core.getImage(); % returned as a 1D array of signed integers in row-major order

width = core.getImageWidth();

height = core.getImageHeight();

if core.getBytesPerPixel == 2

pixelType = 'uint16';

else

pixelType = 'uint8';

end

img = typecast(img, pixelType); % pixels must be interpreted as unsigned integers

img = reshape(img, [width, height]); % image should be interpreted as a 2D array

img = transpose(img); % make column-major order for MATLAB

imshow(img);

imwrite(img,['F:\9Feb16\NewFMs4\Fp1\',num2str(p),'.jpg']);

s = serial('COM6');

fopen(s);

fprintf(s,'OUTPUT OFF');

fclose(s);

delete(s);

end

## **A.1.14 Calculation of Real Direction**

% calculate real diraction

x=dlmread('F:\9Feb16\FstCtrl36\x.txt');

y=dlmread('F:\9Feb16\FstCtrl36\y.txt');

delta=[];

for i=1:12

ang= wrapTo2Pi(atan2((y(i+1)-y(i))',(x(i+1)-x(i))'));

delta=[delta ang];

end

dlmwrite('F:\9Feb16\FstCtrl36\Real\_Dir.txt',delta);

## **A.1.15 Display Vectors**

% produce vectors at each point, these vectors indicate towards selected direction.

clc

x=dlmread('F:\9Feb16\FstCtrl35\x.txt');

y=dlmread('F:\9Feb16\FstCtrl35\y.txt');

xp=dlmread('F:\9Feb16\FstCtrl35\Xp.txt');

yp=dlmread('F:\9Feb16\FstCtrl35\Yp.txt');

X\_target=666;

Y\_target=615;

figure;

grid on;

hold on;

for i=1:20

quiver(x(i),y(i),xp(i)-x(i),yp(i)-y(i),0.5) ;

end

scatter(X\_target,Y\_target);% display circle at the final position

% scatter(x,y);% display circle at every moving position

line(x,y);% display circle at every moving position

axis([0 ,1200 ,0 ,1000]);

xlabel('X(pixels)');

ylabel('Y(pixels)');

set(gca,'YDir','reverse');

## **A.1.16 Repeatability Test**

% the function of this code is measure the repeatability degree between two identical experiments which have same parameters, however, number of particles of the first experiment can be different from number of particles of the second experiment.

clc

% read 50 images of first experiment

for n=1:50

img\_{n}=imread(['F:\9Feb16\NewFMs4\5\',num2str(n),'.jpg']);

% Estimate the Background illumination

background = imopen(img\_{n},strel('disk',15));

% Subtract the Background Image from the Original Image

img\_{n} = img\_{n} - background;

% light adjust

img\_{n} = imadjust(img\_{n},[0.1 0.2],[]);

end

% accumlation of first 25 odd images

A1=imadd(img\_{1},img\_{3});

for o=5:2:49

A1=imadd(A1,img\_{o});

end

% accumlation of first 25 even images

B1=imadd(img\_{2},img\_{4});

for e=6:2:50

B1=imadd(B1,img\_{e});

end

[xi,yi,iu,iv]=mpiv(A1,B1,128,128,0.5,0.5,0,0, 0.5,'cor',0,1);

[iu\_f,iv\_f,iu\_i,iv\_i]=mpiv\_filter(iu,iv,2,2,3,1);

[iu\_s,iv\_s]=mpiv\_smooth(iu\_i,iv\_i);

X1=iu\_s;

Y1=iv\_s;

Xr1=transpose(X1);

Yr1=transpose(Y1);

figure;

quiver(xi,yi,Xr1,Yr1,1);

axis([0 ,1200 ,0 ,1000]);

xlabel('X(pixels)');

ylabel('Y(pixels)');

set(gca,'YDir','reverse');

title('First Vector Field');

% read 50 images of second experiment

for n=1:50

img\_{n}= imread(['F:\9Feb16\NewFMs4\6\',num2str(n),'.jpg']);

% Estimate the Background illumination

background = imopen(img\_{n},strel('disk',15));

% Subtract the Background Image from the Original Image

img\_{n} = img\_{n} - background;

% light adjust

img\_{n} = imadjust(img\_{n},[0.1 0.2],[]);

end

% accumlation of first 25 odd images

A2=imadd(img\_{1},img\_{3});

for o=5:2:49

A2=imadd(A2,img\_{o});

end

% accumlation of first 25 even images

B2=imadd(img\_{2},img\_{4});

for e=6:2:50

B2=imadd(B2,img\_{e});

end

% MPIV resultant images A2 & B2

[xi,yi,iu,iv]=mpiv(A2,B2,128,128,0.5,0.5,0,0, 0.5,'cor',0,1);

[iu\_f,iv\_f,iu\_i,iv\_i]=mpiv\_filter(iu,iv,2,2,3,1);

[iu\_s,iv\_s]=mpiv\_smooth(iu\_i,iv\_i,1);

X2=iu\_s;

Y2=iv\_s;

Xr2=transpose(X2);

Yr2=transpose(Y2);

figure;

quiver(xi,yi,Xr2,Yr2,1);

axis([0 ,1200 ,0 ,1000]);

xlabel('X(pixels)');

ylabel('Y(pixels)');

set(gca,'YDir','reverse');

title('Second Vector Field');

% error calculation & generating error plot

% calculating the magnitude velocity

mag1=sqrt(X1.^2+Y1.^2);

mag2=sqrt(X2.^2+Y2.^2);

%Error vector

ex= X1-X2;

ey= Y1-Y2;

Emag=sqrt(ex.^2+ey.^2);

emag= mean2(sqrt(ex.^2+ey.^2));

averror\_percent3 = mean2(Emag/mag1)\*100

averror\_percent4 = mean2(Emag/mag2)\*100

%avereging the magnitude velocity

av1=mean2(mag1);

av2=mean2(mag2);

averror\_percent1 = (emag/av1)\*100

# **Appendix 2 : Arduino Codes**

## **A.2.1 Levitation Frequency Generation**

// Control a AD9851 DDS based on code of others including:

// Peter Marks http://marxy.org

// Mike Bowthorpe, http://www.ladyada.net/rant/2007/02/cotw-ltc6903/ and

// http://www.geocities.com/leon\_heller/dds.html

//

#define DDS\_CLOCK 125000000

const long f1 = 2552000;

byte LOAD = 9;

byte CLOCK = 10;

byte DATA = 8;

byte LED = 13;

byte LED\_state=0;

void setup()

{

pinMode (DATA, OUTPUT); // sets pin 10 as OUPUT

pinMode (CLOCK, OUTPUT); // sets pin 9 as OUTPUT

pinMode (LOAD, OUTPUT); // sets pin 8 as OUTPUT

pinMode (LED, OUTPUT);

delay(200);

{ //enter serial mode: (This seems to be unneccesary)

digitalWrite (LOAD, LOW);

digitalWrite (CLOCK, LOW);

digitalWrite (CLOCK, HIGH);

digitalWrite (CLOCK, LOW);

digitalWrite (LOAD, HIGH);

digitalWrite (LOAD, LOW);

}

}

void loop()

{

// Do a frequency sweep in Hz

sendFrequency(f1);

delay(1000);

LED\_state=!LED\_state;

if(LED\_state==1) digitalWrite(LED, HIGH); else digitalWrite(LED, LOW);

}

void sendFrequency(unsigned long frequency)

{

unsigned long tuning\_word = (frequency \* pow(2, 24)) / (DDS\_CLOCK/256); //comment on page

digitalWrite (LOAD, LOW); // take load pin low

for(int i = 0; i < 32; i++)

{

if ((tuning\_word & 1) == 1)

outOne();

else

outZero();

tuning\_word = tuning\_word >> 1;

}

//byte\_out(0x08); //changed for AD9850 PGJ

byte\_out(0x00);

digitalWrite (LOAD, HIGH); // Take load pin high again

digitalWrite (LOAD, LOW);

}

void byte\_out(unsigned char byte)

{

int i;

for (i = 0; i < 8; i++)

{

if ((byte & 1) == 1)

outOne();

else

outZero();

byte = byte >> 1;

}

}

void outOne()

{

digitalWrite(CLOCK, LOW);

digitalWrite(DATA, HIGH);

digitalWrite(CLOCK, HIGH);

digitalWrite(DATA, LOW);

}

void outZero()

{

digitalWrite(CLOCK, LOW);

digitalWrite(DATA, LOW);

digitalWrite(CLOCK, HIGH);

}

## **A.2.2 Controlling Modes of Gilson Valve.**

/\*

CONTROL the GILSON 819 valve.

IMPORTANT: Connect 0.5k Ohm resistors on both lines to the valve

Arduino GND -> right hand input pin (via resistor)

Arduino pin -> left hand input pin (via resistor)

\*/

#define OUT\_pin 12

#define ledPin 13

// the loop function runs over and over again forever

//Channel B = Valve

#define communicationTimer 50

void setup() {

Serial.begin(9600);

pinMode(13, OUTPUT); //Initiates Valve Channel B pin

}

void loop()

{

char controlPosition;

delay(communicationTimer);

if (Serial.available() > 0)

{

controlPosition = Serial.read();

posExecute(controlPosition);

}

}

void posExecute(char controlPosition)

{

if (controlPosition == 'L')

{

Serial.println(controlPosition);

digitalWrite(13, HIGH);

}

if (controlPosition == 'I')

{

Serial.println(controlPosition);

digitalWrite(13, LOW);

}

}