Rapport de stage :

Réalisation du logiciel d'acquisition d'images pour MISOLFA
(Moniteur d'Images SOLaires Franco-Algérien)

Par : *Grigahcene zaki cherif*
Ingénieur en électronique CRAAG

Maitre de stage : Fréderic Morand

Dates : 23 juillet – 15 septembre 2007

Introduction :

Ce document présent le rapport du stage que j'ai effectué du 23 juillet au 15 septembre 2007 à l'Observatoire de la Cote d'Azur au plateau de Calern, sur le projet MISOLFA (moniteur d'image solaire franco-algérien). Ce stage est une initiation au projet MISOLFA.

Durant ce stage, il m'a été demande de réaliser un logiciel pour faire des acquisitions d'images à partir d'une camera PCO Pixelfly et de les enregistrer sur le disque dur pour cela j'ai opté pour réaliser ce programme avec le langage Matlab car il possède des bibliothèques de traitement du signal et d'images permettant de faciliter le développement du programme.

MISOLFA :(Moniteur d'Images SOLaires Franco-Algérien)

a) Objectifs :

Les mesures du diamètre solaire à l'astrolabe solaire de l'Observatoire de Calern sur plus de deux cycles solaire présentent des variations apparentes. Ces variations montrent une anti-corrélation avec l'activité solaire définie par le nombre de taches solaires (Laclare et al.1996). Ces résultats ont donné lieu à de nombreux travaux scientifiques et la mise en place du réseau international R2S3 dédié à la mesure du diamètre solaire.
Les observations effectuées sur l'astrolabe solaire ont montré l'influence des perturbations atmosphériques sur la qualité des résultats donc la disposition d'un moniteur quantifiant la qualité des images acquise lors des observations s'est imposé.
De cela l'idée de développer MISOLFA est née pour fonctionner conjointement à la mission PICARD, et à l'expérimentation au sol de l'Observatoire de Calern pour l'observation du diamètre solaire (DORAYSOL et SODISMII, réplique au sol de SODISMI), et donc MISOLFA va quantifier les effets de la perturbation atmosphérique sur les instruments d'observation au sol DORAYSOL et SODISMII. Et par la suite établire des méthodes de correction on comparant les mesures du diamètre solaire faites par SODISMI et sa réplique au sol SODISMII. Le raccordement des mesures sol et espace se fera grâce à la modélisation de la réponse impulsionnelle de l'atmosphère aux instants de mesure du diamètre, en observant les différents paramètres de la turbulence atmosphériques sur le soleil.

b) Paramètre :

Paramètre de Fried r_0 : En l'absence de turbulence, la résolution d'un télescope dépend uniquement de son diamètre. A cause des turbulences atmosphériques, la résolution d'un télescope est limitée. **r_0** correspond, théoriquement, au diamètre du télescope qui donnerait, en absence de turbulence, la même résolution qu'un télescope de diamètre infini en présence de turbulence.
Echelle externe de cohérence spatiale L_0 : Elle définit la taille maximale des perturbations du front d'onde qui restent cohérentes.
Domaine d'isoplanétisme q_0 : Il définit la zone angulaire sur laquelle les grandeurs considérées (angles d'arrivée, fluctuations de phase, ...) restent similaires.
Temps caractéristique d'évolution du front d'onde t_0 : C'est le temps durant lequel on peut considérer que les grandeurs considérées gardent leur cohérence temporelle et que l'atmosphère est comme figée.
Profil de turbulence $C_n^2(h)$: Il définit de quelle manière l'énergie turbulente varie le long du trajet optique de l'onde lors de sa traversée de l'atmosphère.

c) Principe de fonctionnement

Le principe de fonctionnement de MISOLFA est la mesure des fluctuations des angles d'arrivée qui provoquent l'agitation des images au foyer du télescope.
Les angles d'arrivée sont définis comme la normale en chaque point du front d'onde dégradé par la turbulence. Les fluctuations d'angles d'arrivée sont mesurées simultanément dans deux plans, la voie plan image et la voie plan pupille, pour permettre d'estimer tous les paramètres spatio-temporels, comme le montre le schéma suivant (fig1).

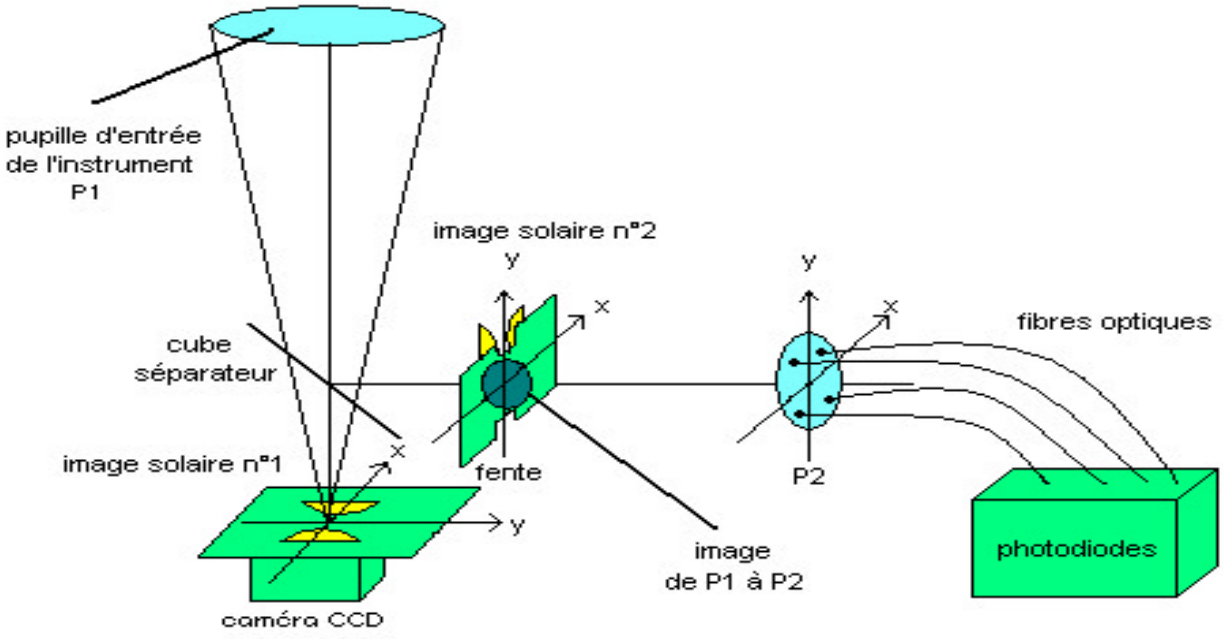


Fig1 : Principe de la mesure avec Misolfa

1. **la voie plan image :** une caméra CCD placée au foyer du télescope permet de mesurer ces fluctuations d'angles d'arrivé.
2. **la voie plan pupille :** Des photodiodes placées sur l'image de la pupille du télescope mesurent les fluctuations de phase qui provoquent les fluctuations d'intensité observées sur cette image.

d) Description de l'instrument

MISOLFA se compose d'un télescope de type Cassegrain coudé, installé sur une monture Alt-Azimutale. Elle permet d'assurer le pointage puis le suivi du Soleil au cours de la journée.
Le télescope est équipé d'un miroir primaire d'un diamètre de 254 mm, d'un miroir secondaire de diamètre utile 35.5 mm et d'un miroir plan elliptique à 45°. Au foyer de ce télescope, est située une boîte comprenant l'instrumentation focale de l'instrument. Elle porte l'ensemble de l'optique et des détecteurs (caméra CCD et photodiodes). Elle est constituée de deux voies d'analyse de types différents.

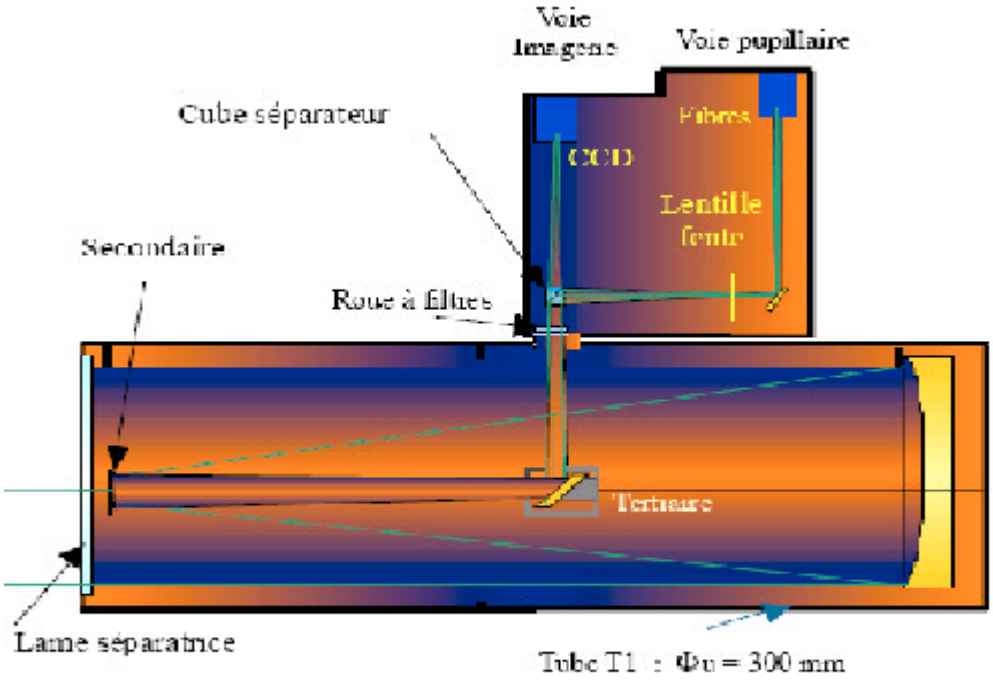
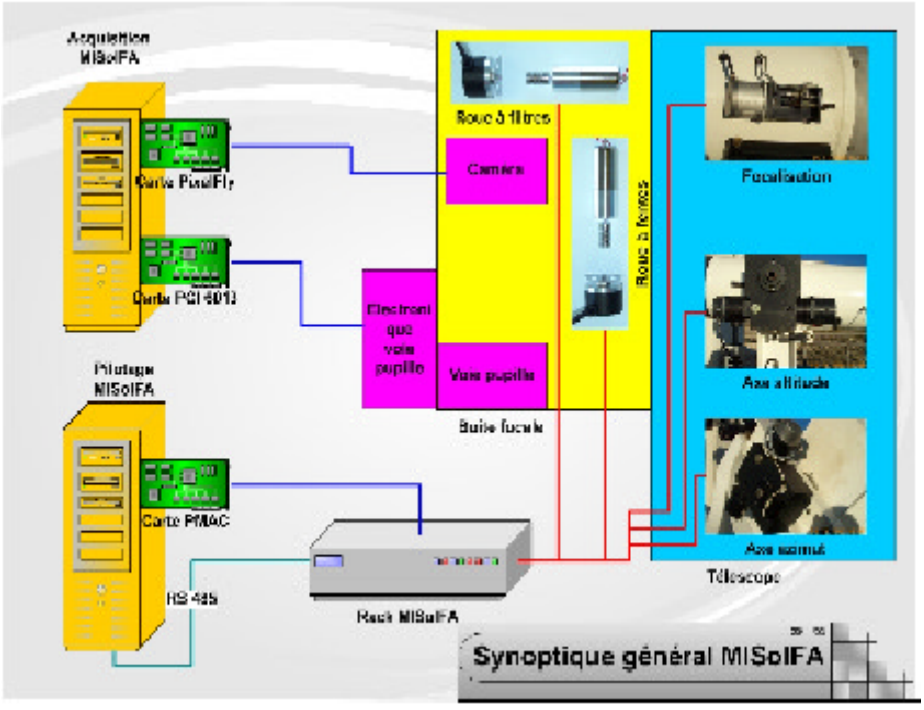


Schéma technique de MISOLFA

MISOLFA comprend aussi une partie électronique divisée en deux blocs :

- Une électronique d'acquisition qui assure l'acquisition des données de la caméra CCD et des photodiodes.
- Une électronique de pilotage qui a pour fonction de fournir la puissance aux moteurs qui assurent les mouvements des mécanismes suivants : les deux axes du télescope, la mécanique interne de la boîte et la mise au point par déplacement du secondaire.



e)mécanique du télescope

D'un point de vue mécanique, MISOLFA est composé de deux parties distinctes :

- Le télescope qui permet d'assurer le pointage puis le suivi du soleil au cours de la journée :
- La boite focale, qui porte l'ensemble de l'optique et des détecteurs.

f) Méthode de mesure

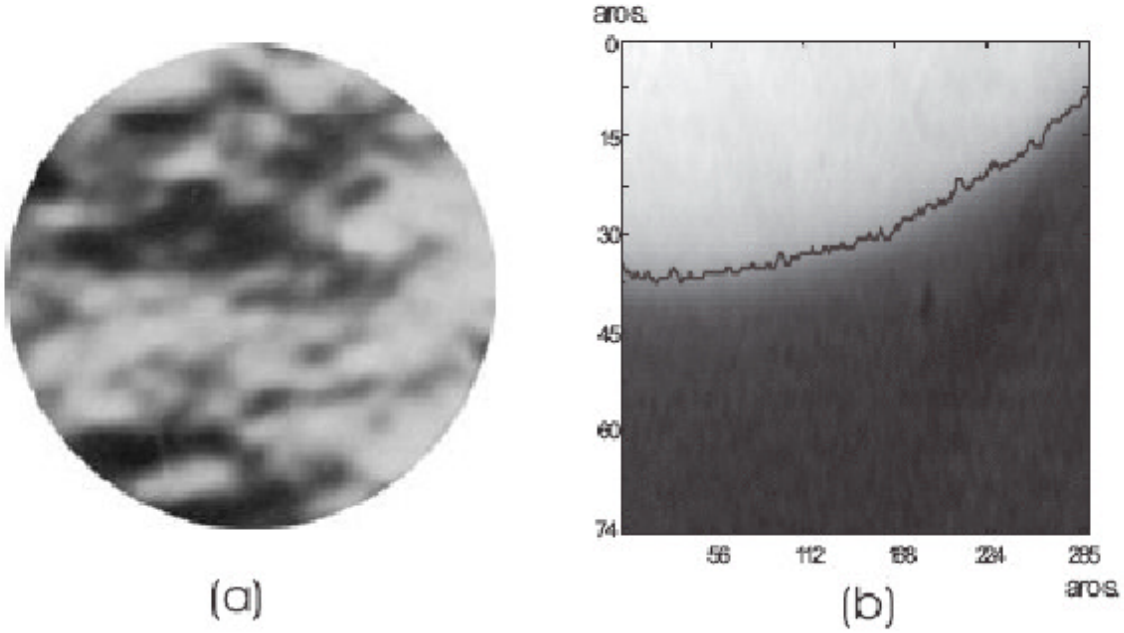
Grâce aux deux voies MISOLFA (voie image et voie pupille) tous les paramètres spatio-temporels de la turbulence peuvent être estimés, mais pour des raisons techniques seule la voie pupille pourra estimer les constantes de temps de l'atmosphère. En effet, le temps entre deux acquisitions de la caméra est plusieurs fois supérieur au temps d'évolution du front d'onde.

La voie plan pupille

L'image de la pupille est formée à travers une fente, de quelques secondes d'arc de largeur et de quelques dizaines de secondes d'arc de hauteur, positionnée perpendiculairement sur le bord de l'image du soleil. L'image ainsi obtenue présente des fluctuations d'intensité appelées *ombres volantes* qui au premier ordre, sont proportionnelles aux fluctuations d'angles d'arrivée sur la pupille d'entrée, la voie plan pupille est utilisée pour la mesure du temps d'évolution du front d'onde grâce à l'utilisation de photodiodes.

Fluctuation des angles d'arrivée dans la voie plan image

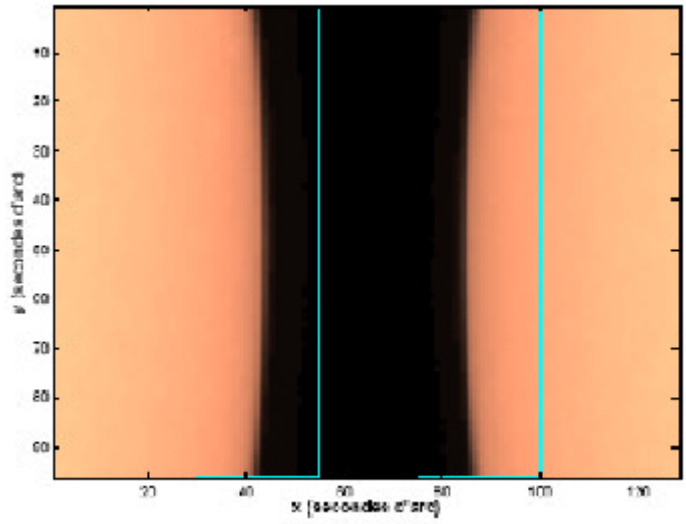
L'image du soleil est directement formée sur la caméra CCD placée dans le plan de l'instrument. Les fluctuations d'angles d'arrivée sont celles que l'on observe sur le contour de l'image du soleil. Chaque point du contour est formé par le front d'onde dégradé par la turbulence limitée à la pupille de l'instrument. Les fluctuations d'angle d'arrivée ainsi mises en évidence dans le plan focal sont filtrées mais les méthodes d'estimation des paramètres tiennent compte de ce filtrage.



Observation des fluctuations d'angle d'arrivée

dans la voie *plan pupille* (a) et *plan image* (b)

MISOLFA est conçu de sorte à fournir les paramètres de la turbulence sur des lignes de visée ayant une séparation angulaire de plusieurs dizaines de minutes. Pour cela, l'idée retenue est de former une image sur la camera CCD constituée des deux bords du soleil diamétralement opposées. La figure montre le type d'image que l'on souhaite observer avec la voie plan image, pour limiter le flot de données à acquérir, seules deux fenêtres contenant l'information utile seront sauvegardées.

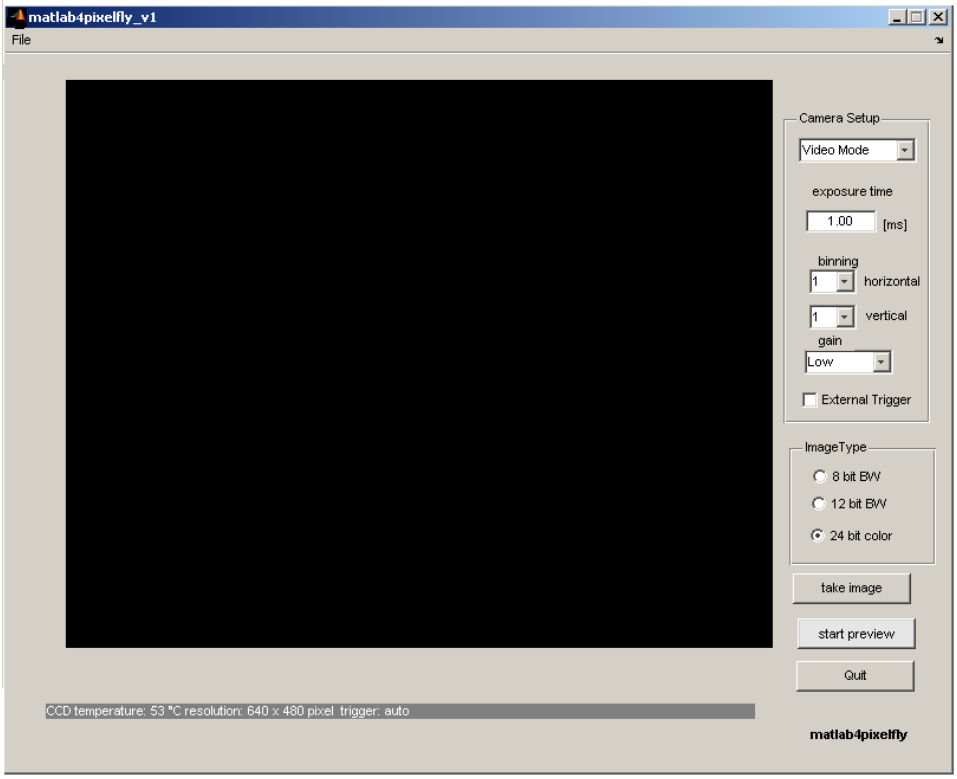


Images du soleil telles qu'elles seront observées

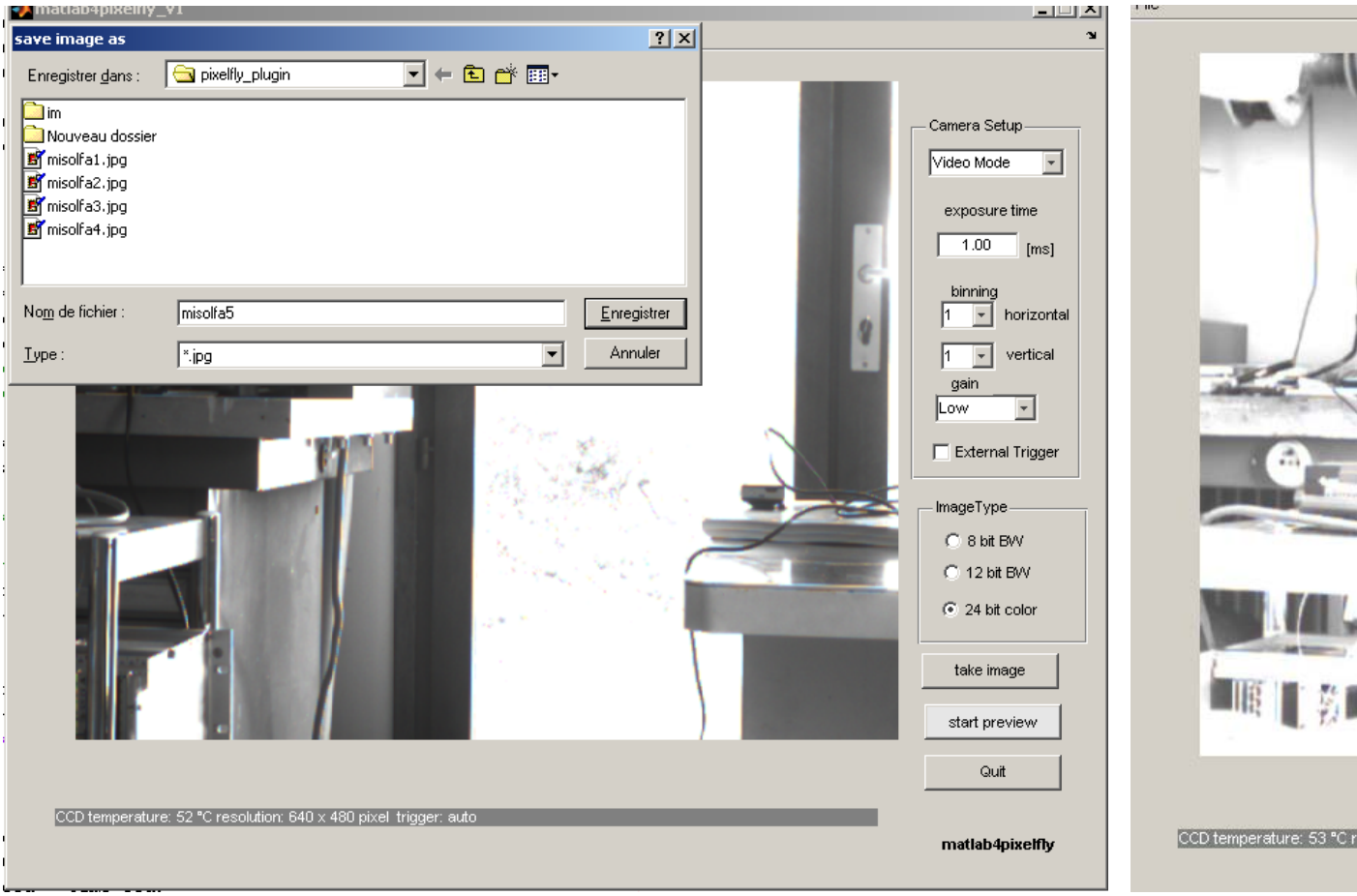
dans la voie plan image de MISOLFA

Fonctionnement du programme

A partir de l'espace de travail de Matlab, on lance la fonction matlab4pixelfly_v1. La fenêtre suivante s'ouvre. Le programme effectue les taches 'initialisation de la carte d'acquisition d'images au démarrage de l'application.

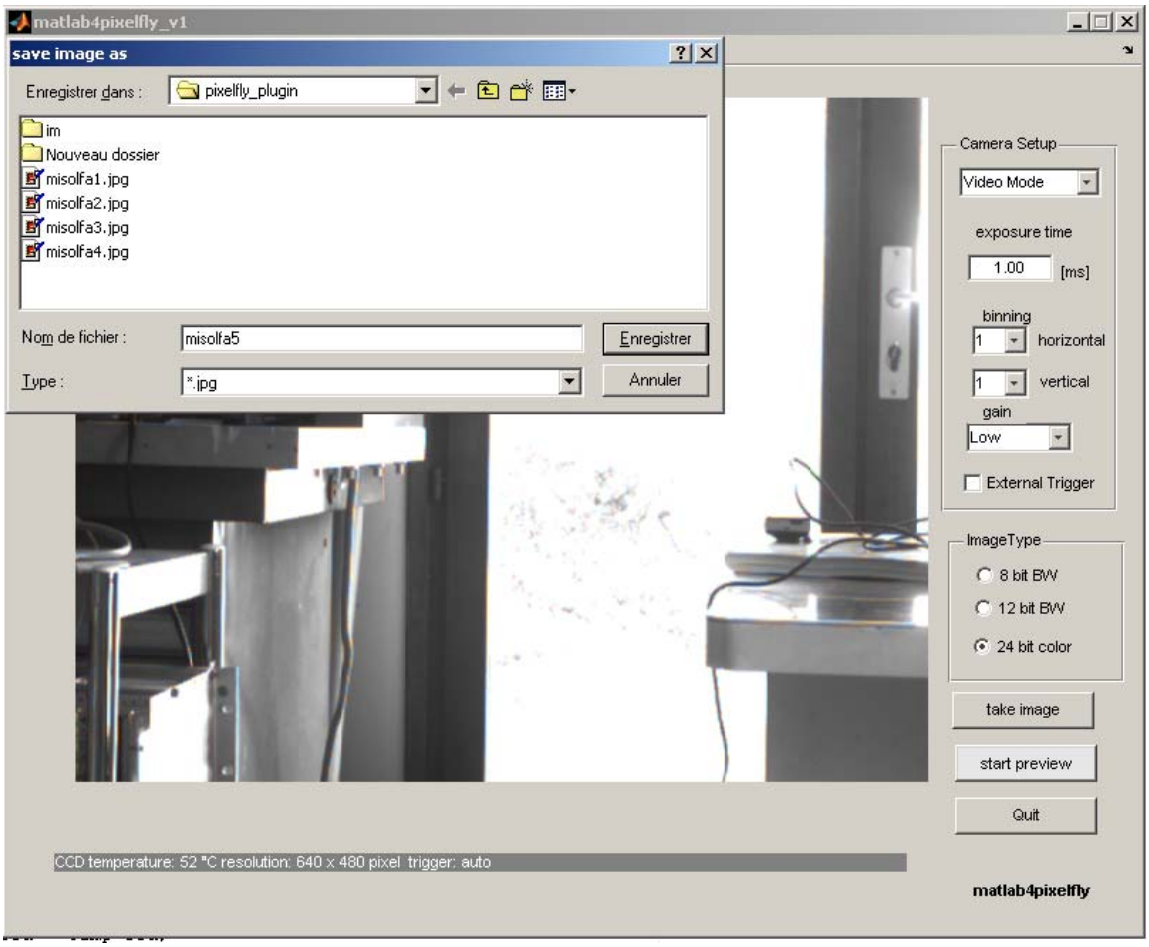


Pour lancer la capture vidéo on clique sur le bottons '*start preview'* on obtient alors l'affichage des images en continu sur la zone d'affichage.



On peut aussi avoir de captures images en cliquant sur le bouton ‘take image’.

on peut faire la captures de deux images car on fait la capture de deux bord pour réduire l’espace mémoire comme c’est indiqué dans les rapports précédents (tel que celui de M. Saidi Yacine en 2004).



on obtient alors les deux images suivantes :

Misolfa 5

Misolfa 6

On peut modifier la taille des images et le nombre de bits par pixel. le Matlab donne la possibilité d’enregistrer les images sous format 'fits' grâce à la fonction fitswrite.

Conclusion

Ce stage n'est qu'une initiation au projet MSOLFA. Il m'a permis d'établir un contact avec ce projet, de mettre en pratique mes connaissances et d'apprendre d'avantage sur la programmation. En plus j'ai pu me familiariser avec l'acquisition d'images en utilisant la camera « PCO pixelfly » et le logiciel d'acquisition «CamWare ».
J'ai pu réaliser un programme d'acquisition en langage MATLAB qui permet de d'acquérir des images depuis la camera CCD, de les afficher et de sauvegarder les régions utiles. Par la suite, je compte le réécrire le code en langage C++.

Annexe

```
function varargout = matlab4PIXELFLY_v1(varargin)
% pixelfly driver for Matlab: version 1.1.0 from The Cooke Corporation.
% This simple program demonstrates the basic camera control and image acquisation using
% pixelfly. No association with any toolbox is assumed.
%
% check for updates from www.cookecorp.com


% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @matlab4PIXELFLY_v1_OpeningFcn, ...
                   'gui_OutputFcn',  @matlab4PIXELFLY_v1_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 matlab4PIXELFLY_v1 is made visible.
function matlab4PIXELFLY_v1_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 matlab4PIXELFLY_v1 (see VARARGIN)


% check whether library dll is already loaded
if not(libisloaded('hpccamvb'))
    loadlibrary('hpccamvb','hpccamvb')
end

% Choose default command line output for matlab4PIXELFLY_v1
handles.output = hObject;

% default parameters for camera setmode
handles.hbin = 0; % horizontal binning
handles.vbin = 0; % vertical binning
handles.exposure = 33; % exposure time in ms
```

```
handles.gain = 1;
handles.mode = 49;  % video mode and software trigger
handles.bit_pix = 12;
board_number=0;
handles.board_number=board_number;

error_code = pfINITBOARD(handles.board_number);
if error_code ~= 0
   error('....initialize camera, check connection and device driver');
end

error_code = pfSETMODE(handles.board_number, handles.mode, 0, handles.exposure,...
                  handles.hbin,handles.vbin,handles.gain, 0,handles.bit_pix,0);

if error_code ~= 0
   error('....initial setmode failed!');
end

[ccd_width, ccd_height, image_width, image_height, bit_pix, error_code] =
pfGETSIZES(board_number);
handles.ccd_width = ccd_width;
handles.ccd_height = ccd_height;
handles.image_width = image_width;
handles.image_height = image_height;
handles.imagesize=image_width*image_height*2;
handles.bit_pix=bit_pix;
[temp_ccd, error_code] = pfREADTEMPERATURE(board_number);
if error_code == 0
   handles.temp_ccd = temp_ccd;
else
   handles.temp_ccd = -1;
end
%to add double shot here
[caminfo, error_code] = pfGETBOARDPAR(board_number, 33*4);
if error_code ~= 0
   error('....pfGETBOARDPAR failed!');
end
if bitand(caminfo(12),128)==128
   handles.color=1;
   handles.depth=3; %color sensor
   if not(libisloaded('Pcocnv'))
      loadlibrary('Pcocnv','Pcocnv')
   end
   set(handles.radiobuttonrgb,'Value',1);
   handles.colorlutptr=calllib('Pcocnv','CREATE_COLORLUT',12,0,255);
   handles.image= zeros(double(handles.image_height), double(handles.image_width),3);
  % handles.image_buffer_map = image(handles.image, 'EraseMode', 'none');
   handles.image_buffer_map = imagesc(handles.image);
else
   handles.depth=2; %BW
```

```
    handles.color=0;
    set(handles.radiobutton12,'Value',1);
    set(handles.radiobuttonrgb,'Enable','off');
    handles.image= zeros(double(handles.image_height), double(handles.image_width));
   % handles.image_buffer_map = image(handles.image, 'EraseMode', 'none');
    handles.image_buffer_map = imagesc(handles.image);
end
bufnr=-1;
bufsize=ccd_width*ccd_height*2;
[bufnr,bufsize, error_code] = pfALLOCATE_BUFFER(board_number,bufnr,bufsize);

if error_code ~= 0
    error('....memory allocation failed!');
end
handles.bufnr=bufnr;
[bufaddress, error_code] =
pfMAP_BUFFER(board_number,bufnr,ccd_width*ccd_height*2,0);
if error_code ~= 0
    error('....map buffer error!');
end
handles.bufaddress = bufaddress; %this value is only for debugging purpose
error_code = pfSTART_CAMERA(board_number);
if error_code ~= 0
    error('....initial start_camera error!');
end

%**************************************************************************
handles.is_camera_image = 'yes';
axis off
hold on
handles.res_str = [' resolution: ' num2str(double(handles.image_width)) ' x '
num2str(double(handles.image_height)) ' pixel '];
handles.trig_str = ' trigger: auto ';
handles.ccd_temp_str = ['CCD temperature: ' num2str(double(handles.temp_ccd)) ' °C'];
handles.output_string = [handles.ccd_temp_str handles.res_str handles.trig_str];
set(handles.status_text,...
    'String', handles.output_string);
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% UIWAIT makes matlab4PIXELFLY_v1 wait for user response (see UIRESUME)
% uiwait(handles.matlab4PIXELFLY_v1_figure);

%**************************************************************************
% --- Outputs from this function are returned to the command line.
function varargout = matlab4PIXELFLY_v1_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
varargout{1} = handles.output;

% --- Executes on button press in takeimage_pushbutton.
function takeimage_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to takeimage_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
image_ready = uint32(2);          % if b0010, DMA done
buffer_error = uint32(hex2dec('0000f000'));   % 4 types of errors
%buffer_queued=uint32(4);
bit_test = 1;
error_code =pfADD_BUFFER_TO_LIST(handles.board_number,handles.bufnr,
handles.imagesize,0,0);
if error_code ~= 0
   error('....takeimage_pushbutton_Callback: add buffer error!')
end
error_code = pfTRIGGER_CAMERA(handles.board_number);
if error_code ~= 0
   error('....takeimage_pushbutton_Callback: pfTRIGGER_CAMERA error!')
end
% Check image status
while bit_test == 1
   [ret_image_status, error_code] =
pfGETBUFFER_STATUS(handles.board_number,handles.bufnr,0,4);
   if error_code ~= 0
       error('....takeimage_pushbutton_Callback: pfGETBUFFER_STATUS error!')
   end
   image_status=cast(ret_image_status,'uint32');
   if bitand(image_status,image_ready)==image_ready
      bit_test =0; %image ready
      break;
   end
   if bitand(image_status, buffer_error)>0

error_code=pfREMOVE_BUFFER_FROM_LIST(handles.board_number,handles.bufnr);
         if error_code ~= 0
            disp('....takeimage_pushbutton_Callback: pfREMOVE_BUFFER_FROM_LIST
error!');
            break;
         end
         error_code =pfADD_BUFFER_TO_LIST(handles.board_number,handles.bufnr,
handles.imagesize,0,0);
         if error_code ~= 0
            disp('....takeimage_pushbutton_Callback: pfADD_BUFFER_TO_LIST error!');
            break;
         end
         error_code = pfTRIGGER_CAMERA(handles.board_number);
         if error_code ~= 0
```

```
            disp('....takeimage_pushbutton_Callback: pfTRIGGER_CAMERA error!');
            break;
        end
    end
end

if bit_test == 0
    [result_image, error_code] = pfTRANSFER_IMAGE(handles.board_number,
handles.bit_pix,handles.image_width,handles.image_height);
    if (handles.depth==3)&((handles.hbin+handles.vbin)==0)
        result_image_ptr = libpointer('uint16Ptr', result_image);
        rgbimage=uint8(zeros(handles.image_width*3,handles.image_height));
        rgbimage_ptr=libpointer('uint8Ptr', rgbimage);
        calllib('Pcocnv','CONV_BUF_12TOCOL',0,handles.image_width, ...
            handles.image_height, result_image_ptr,rgbimage_ptr,handles.colorlutptr);
        rgbimage = uint8(get(rgbimage_ptr, 'Value'));
        RGB_image = uint8(zeros(handles.image_height, handles.image_width, 3));
        for i=1:handles.image_width
            %you may want to balance the color here, multiply a factor
            RGB_image(:,i,3)=rgbimage((i-1)*3+1,:)';
            RGB_image(:,i,2)=rgbimage((i-1)*3+2,:)';
            RGB_image(:,i,1)=rgbimage(i*3,:)';
        end
        handles.image = RGB_image;
        set(handles.image_buffer_map,'CData',RGB_image);
        [filename,pathname]=uiputfile ({'*.jpg';'*.bmp';'*.tiff';'*.fits';'*.*'},'save image as');
        filepath1 = sprintf('%s\%s',pathname,filename);
        [filename,pathname]=uiputfile ({'*.jpg';'*.bmp';'*.tiff';'*.fits';'*.*'},'save image as');
        filepath2 = sprintf('%s\%s',pathname,filename);
        fenetre1 = RGB_image(1:480,1:32);
        fenetre2 = RGB_image(1:480,608:640);
        %imwrite(fenetre1,filepath1);
        %imwrite(fenetre2,filepath2);
       fitswrite(fenetre1, filepath1);
        fitswrite(fenetre2, filepath2);
    else
        handles.image = result_image';
        set(handles.image_buffer_map,'CData',handles.image);colormap(gray);

        %imagesc(handles.image);
    end
else %try to reset camera
    error_code = pfSTOP_CAMERA(handles.board_number);
    if error_code ~= 0
        error('....takeimage_pushbutton_Callback: pfSTOP_CAMERA error!');
    end
    error_code = pfSTART_CAMERA(handles.board_number);
    if error_code ~= 0
      error('....takeimage_pushbutton_Callback: pfSTART_CAMERA error!')
    end
```

```
end

[temp_ccd, error_code] = pfREADTEMPERATURE(handles.board_number);
if error_code == 0
   handles.temp_ccd = temp_ccd;
else
   handles.temp_ccd = -1;
end

handles.ccd_temp_str = ['CCD temperature: ' num2str(double(handles.temp_ccd)) ' °C'];
handles.output_string = [handles.ccd_temp_str handles.res_str handles.trig_str];
set(handles.status_text,...
   'String', handles.output_string);
% Update handles structure
guidata(hObject, handles);


%**************************************************************************

% --- Executes on button press in preview_togglebutton.
function preview_togglebutton_Callback(hObject, eventdata, handles)
% hObject    handle to preview_togglebutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of preview_togglebutton
% if a color camera is connected, it is necessary to load a proper
% look-up-table to read out the camera correct.

if get(handles.preview_togglebutton,'Value')  % 1 if pressed, 0 if not

   error_code =pfADD_BUFFER_TO_LIST(handles.board_number,handles.bufnr,
handles.imagesize,0,0);
   if error_code ~= 0
       disp('....preview_togglebutton_Callback: add buffer error!')
       return
   end

   error_code = pfTRIGGER_CAMERA(handles.board_number);
   if error_code ~= 0
        disp('....preview_togglebutton_Callback: pfTRIGGER_CAMERA error!')
        return
   end
   handles.t = timer('timerfcn',{@update,handles},...
               'Period',1/10,...
               'ExecutionMode','fixedRate');
   guidata(hObject, handles);
   set(hObject,'BackgroundColor',[0,0.8,0.5]);
   set(hObject, 'String','stop preview');
   set(hObject, 'Foregroundcolor',[1,0,0]);
   start(handles.t)
```

```
else
    set(hObject, 'String','start preview');
    set(hObject, 'Foregroundcolor',[0,0,0]);
    set(hObject,'BackgroundColor',[0.9,0.9,0.9]);
    stop(handles.t);
    delete(handles.t);
end

%% This will be the code that updates the GUI
function update(hObject, eventdata, handles)
image_ready = uint32(2);          % if b0010, DMA done
buffer_error = uint32(hex2dec('0000f000'));   % 4 types of errors
if handles.depth==3 && handles.hbin+handles.vbin==0
    rgbimage=uint8(zeros(handles.image_width*3,handles.image_height));
    rgbimage_ptr=libpointer('uint8Ptr', rgbimage);
    RGB_image = uint8(zeros(handles.image_height, handles.image_width, 3));
end

[ret_image_status, error_code] =
pfGETBUFFER_STATUS(handles.board_number,handles.bufnr,0,4);
if error_code ~= 0
    error('....preview_togglebutton_Callback: pfGETBUFFER_STATUS error!');
end
image_status=cast(ret_image_status,'uint32');
if bitand(image_status,image_ready)==image_ready
    [result_image, error_code] = pfTRANSFER_IMAGE(handles.board_number,
handles.bit_pix,  handles.image_width,handles.image_height);
    handles.image = result_image';
    %drawnow
    if (handles.depth==3) && ((handles.hbin+handles.vbin)==0)
        result_image_ptr = libpointer('uint16Ptr', result_image);
        calllib('Pcocnv','CONV_BUF_12TOCOL',0,handles.image_width, ...
            handles.image_height, result_image_ptr,rgbimage_ptr,handles.colorlutptr);
        rgbimage = uint8(get(rgbimage_ptr, 'Value'));
        for i=1:handles.image_width
                RGB_image(:,i,3)=rgbimage((i-1)*3+1,:)';
                RGB_image(:,i,2)=rgbimage((i-1)*3+2,:)';
                RGB_image(:,i,1)=rgbimage(i*3,:)';
        end
       set(handles.image_buffer_map,'CData',RGB_image);
    else
      % image(handles.image);
      set(handles.image_buffer_map,'CData',handles.image);
    end
    [temp_ccd, error_code] = pfREADTEMPERATURE(handles.board_number);
    if error_code == 0
          handles.temp_ccd = temp_ccd;
    else
          handles.temp_ccd = -1;
    end
```

```
    handles.ccd_temp_str = ['CCD temperature: ' num2str(double(handles.temp_ccd)) ' °C'];
    handles.output_string = [handles.ccd_temp_str handles.res_str handles.trig_str];
     set(handles.status_text,'String', handles.output_string);
    error_code =pfADD_BUFFER_TO_LIST(handles.board_number,handles.bufnr,
handles.imagesize,0,0);
    if error_code ~= 0
            error('....preview_togglebutton_Callback: add buffer error!')
    end
    error_code = pfTRIGGER_CAMERA(handles.board_number);
    if error_code ~= 0
            error('....preview_togglebutton_Callback: pfTRIGGER_CAMERA error!');
    end
else
    if bitand(image_status, buffer_error)~=0

error_code=pfREMOVE_BUFFER_FROM_LIST(handles.board_number,handles.bufnr);
        if error_code ~= 0
             error('....preview_togglebutton_Callback: pfREMOVE_BUFFER_FROM_LIST
error!');
        end
        error_code =pfADD_BUFFER_TO_LIST(handles.board_number,handles.bufnr,
handles.imagesize,0,0);
        if error_code ~= 0
             error('....preview_togglebutton_Callback: pfADD_BUFFER_TO_LIST error!');
        end
        error_code = pfTRIGGER_CAMERA(handles.board_number);
        if error_code ~= 0
             error('....preview_togglebutton_Callback: pfTRIGGER_CAMERA error!');
         end
     end
end

% --- Executes during object creation, after setting all properties.
function mode_CreateFcn(hObject, eventdata, handles)
% hObject    handle to hbin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes on selection change in hbin.
function mode_Callback(hObject, eventdata, handles)
% hObject    handle to mode_listbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns mode_listbox contents as cell array
%        contents{get(hObject,'Value')} returns selected item from mode_listbox

mode_index = get(hObject,'Value');
if mode_index==1
   handles.mode = bitor(handles.mode,hex2dec('30'));
   %set(handles.exposure_unit_text, 'String', '[ms]');
else
   handles.mode = bitand(handles.mode,hex2dec('1f'));
   %set(handles.exposure_unit_text, 'String', '[us]');
end

error_code = pfSTOP_CAMERA(handles.board_number);
if error_code ~= 0
    error('....camera_utility_trigger_external_Callback: pfSTOP_CAMERA error!')
end
error_code = pfSETMODE(handles.board_number, handles.mode, 0, handles.exposure,...
                  handles.hbin,handles.vbin,handles.gain, 0,handles.bit_pix,0);
if error_code ~= 0
     error('....camera_utility_trigger_auto_Callback: pfSETMODE error!');
end
error_code = pfSTART_CAMERA(handles.board_number);
if error_code ~= 0
      error('....camera_utility_trigger_auto_Callback: pfSTART_CAMERA error!')
end

% Update handles structure
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function hbin_CreateFcn(hObject, eventdata, handles)
% hObject    handle to hbin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in hbin.
function hbin_Callback(hObject, eventdata, handles)
% hObject    handle to hbin (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns hbin contents as cell array
%        contents{get(hObject,'Value')} returns selected item from hbin

handles.hbin = get(hObject,'Value')-1;
error_code = pfSTOP_CAMERA(handles.board_number);
if error_code ~= 0
    error('....camera_utility_trigger_external_Callback: pfSTOP_CAMERA error!')
end
error_code = pfSETMODE(handles.board_number, handles.mode, 0, handles.exposure,...
                  handles.hbin,handles.vbin,handles.gain, 0,handles.bit_pix,0);
if error_code ~= 0
   error('....camera_utility_trigger_auto_Callback: pfSETMODE error!');
end
error_code = pfSTART_CAMERA(handles.board_number);
if error_code ~= 0
      error('....camera_utility_trigger_auto_Callback: pfSTART_CAMERA error!')
end
[ccd_width, ccd_height, image_width, image_height, bit_pix,error_code] =
pfGETSIZES(handles.board_number);
handles.image_width = image_width;
handles.image_height = image_height;
handles.bit_pix = bit_pix;

if bit_pix==8
   handles.imagesize=image_width*image_height;
else
   handles.imagesize=image_width*image_height*2;
end
hold off
handles.image = zeros(double(handles.image_height), double(handles.image_width));
handles.image_buffer_map = imagesc(handles.image);
set(handles.display_axes, ...
  'Visible', 'off',...
  'Units', 'pixels');
hold on
if handles.hbin
   colormap(gray);
end
handles.is_camera_image = 'yes';
handles.res_str = [' resolution: ' num2str(double(handles.image_width)) ' x '
num2str(double(handles.image_height)) ' pixel '];
% Update handles structure
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function vbin_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to vbin_listbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in vbin_listbox.
function vbin_Callback(hObject, eventdata, handles)
% hObject    handle to vbin_listbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns vbin_listbox contents as cell array
%        contents{get(hObject,'Value')} returns selected item from vbin_listbox

handles.vbin = get(hObject,'Value')-1;
error_code = pfSTOP_CAMERA(handles.board_number);
if error_code ~= 0
    error('....vbin_listbox_Callback: pfSTOP_CAMERA error!')
end
error_code = pfSETMODE(handles.board_number, handles.mode, 0, handles.exposure,...
                  handles.hbin,handles.vbin,handles.gain, 0,handles.bit_pix,0);
if error_code ~= 0
     error('....vbin_listbox_Callback: pfSETMODE error!');
end
error_code = pfSTART_CAMERA(handles.board_number);
if error_code ~= 0
      error('....vbin_listbox_Callback: pfSTART_CAMERA error!')
end

[ccd_width, ccd_height, image_width, image_height, bit_pix,error_code] =
pfGETSIZES(handles.board_number);
handles.image_width = image_width;
handles.image_height = image_height;
handles.bit_pix = bit_pix;

if bit_pix==8
   handles.imagesize=image_width*image_height;
else
   handles.imagesize=image_width*image_height*2;
end
axes(handles.display_axes);
hold off
```

```
handles.image_buffer_map = 0.0;
handles.image= zeros(double(handles.image_height), double(handles.image_width));
%handles.image_buffer_map = image(handles.image, 'EraseMode', 'none');
handles.image_buffer_map = imagesc(handles.image);
set(handles.display_axes, ...
      'Visible', 'off',...
      'Units', 'pixels');
hold on
if handles.vbin
   colormap(gray);
end
handles.is_camera_image = 'yes';
handles.res_str = [' resolution: ' num2str(double(handles.image_width)) ' x '
num2str(double(handles.image_height)) ' pixel '];
% Update handles structure
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function exposure_edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to exposure_edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function exposure_edit_Callback(hObject, eventdata, handles)
% hObject    handle to exposure_edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of exposure_edit as text
%        str2double(get(hObject,'String')) returns contents of exposure_edit as a double

exposure_time = str2double(get(handles.exposure_edit,'String'));
%video mode from 1 to 10000ms, async mode from 0.01 to 10 ms
if bitand(handles.mode,hex2dec('0f0'))==hex2dec('030')
   if (exposure_time > 10000)
     exposure_time = 10000;
   end
   if (exposure_time < 1)||isnan(exposure_time)
     exposure_time = 1;
   end
   handles.exposure=cast(exposure_time,'uint32');
```

```
else
    if (exposure_time > 65.535)
      exposure_time = 65.535;
    end
    if (exposure_time < 0.01)||isnan(exposure_time)
      exposure_time = 0.01;
    end
   handles.exposure=cast(exposure_time*1000,'uint32');
end
handles.exposure_str = num2str(exposure_time, '%12u');
exposure_str = num2str(exposure_time, '%8.2f');
set(handles.exposure_edit, 'String', exposure_str);
% Update handles structure
error_code = pfSTOP_CAMERA(handles.board_number);
if error_code ~= 0
    error('....exposure_edit_Callback: pfSTOP_CAMERA error!')
end
error_code = pfSETMODE(handles.board_number, handles.mode, 0, handles.exposure,...
                    handles.hbin,handles.vbin,handles.gain, 0,handles.bit_pix,0);
if error_code ~= 0
     error('....exposure_edit_Callback: pfSETMODE error!');
end
error_code = pfSTART_CAMERA(handles.board_number);
if error_code ~= 0
      error('....exposure_edit_Callback: pfSTART_CAMERA error!')
end

guidata(hObject, handles);

% --------------------------------------------------------------------

% --------------------------------------------------------------------
function file_menu_Callback(hObject, eventdata, handles)
% hObject    handle to file_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function file_load_Callback(hObject, eventdata, handles)
% hObject    handle to file_open (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[filename, pathname] = uigetfile('*.mat','load a previously saved image into workspace' );
load_name = [pathname filename];
load(load_name,'write_image');
handles.image = write_image;
image_dimensions = size(handles.image);
handles.width = image_dimensions(2);
handles.height = image_dimensions(1);
```

```
%map=colormap;
set(handles.image_buffer_map,'CData',write_image);
if size(size(write_image))<3
   colormap(gray);
else
    colormap('default');
end
handles.is_camera_image = 'no';
set(handles.display_axes, ...
    'Visible', 'off',...
    'Units', 'pixels');
handles.output_string = ['image: ' load_name '  with resolution: '...
        num2str(double(handles.width)) ' x '...
        num2str(double(handles.height)) ' pixel '];
set(handles.status_text,...
    'String', handles.output_string);
% Update handles structure
drawnow
guidata(hObject, handles);

% --------------------------------------------------------------------
function file_save_Callback(hObject, eventdata, handles)
% hObject    handle to file_save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

write_image=handles.image;
save matlab.mat write_image
% Update handles structure
guidata(hObject, handles);

% --------------------------------------------------------------------
function file_save_as_Callback(hObject, eventdata, handles)
% hObject    handle to file_save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[filename, pathname] = uiputfile('*.mat', 'Name image-file');
if isequal(filename,0) | isequal(pathname,0)
   disp('User pressed cancel')
   return;
else
   write_name = [pathname filename];
   disp(['User selected ', fullfile(pathname, filename)])
   write_image = handles.image;
   save(write_name, 'write_image');
end
% Update handles structure
guidata(hObject, handles);

%--------------------------------------------------------------------
```

```
function quit_Callback(hObject, eventdata, handles)
% hObject    handle to file_quit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

error_code = pfSTOP_CAMERA(handles.board_number);
if error_code ~= 0
    error('....quit_Callback: pfSTOP_CAMERA error!')
end
error_code = pfUNMAP_BUFFER(handles.board_number,handles.bufnr);
if error_code ~= 0
   error('....unmap buffer error!')
end
error_code = pfFREE_BUFFER(handles.board_number,handles.bufnr);
if error_code ~= 0
   error('....unmap buffer error!')
end
error_code = pfCLOSEBOARD(handles.board_number);
if error_code ~= 0
   error('....disconnected error!');
end

if (libisloaded('Pcocnv'))
  calllib('Pcocnv','DELETE_COLORLUT',handles.colorlutptr);
 %   unloadlibrary('Pcocnv');
end
if (libisloaded('hpccamvb'))
   unloadlibrary('hpccamvb');
end
delete(hObject);
close all

% --------------------------------------------------------------------
function trigger_Callback(hObject, eventdata, handles)
% hObject    handle to camera_utility_trigger_rising (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if (get(hObject,'Value') == get(hObject,'Max'))
   % then checkbox is checked-take approriate action
    handles.mode=bitand(handles.mode,254);
    handles.trig_str = ' trigger: external ';
else
   % checkbox is not checked-take approriate action
   handles.mode=bitor(handles.mode,1);
   handles.trig_str = ' trigger: auto ';
end
[temp_ccd, error_code] = pfREADTEMPERATURE(handles.board_number);
if error_code == 0
   handles.temp_ccd = temp_ccd;
```

```
else
   handles.temp_ccd = -1;
end
handles.ccd_temp_str = ['CCD temperature: ' num2str(double(handles.temp_ccd)) ' °C'];
handles.output_string = [handles.ccd_temp_str handles.res_str handles.trig_str];
set(handles.status_text,...
   'String', handles.output_string);
error_code = pfSTOP_CAMERA(handles.board_number);
if error_code ~= 0
    error('....trigger_Callback: pfSTOP_CAMERA error!')
end
error_code = pfSETMODE(handles.board_number, handles.mode, 0, handles.exposure,...
                    handles.hbin,handles.vbin,handles.gain, 0,handles.bit_pix,0);
if error_code ~= 0
     error('....trigger_Callback: pfSETMODE error!');
end
error_code = pfSTART_CAMERA(handles.board_number);
if error_code ~= 0
      error('....trigger_Callback: pfSTART_CAMERA error!')
end
% Update handles structure
guidata(hObject, handles);

% --- Executes on selection change in gain.
function gain_Callback(hObject, eventdata, handles)
% hObject    handle to gain (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns gain contents as cell array
%        contents{get(hObject,'Value')} returns selected item from gain
handles.gain=get(hObject,'Value')-1;
error_code = pfSTOP_CAMERA(handles.board_number);
if error_code ~= 0
    error('....gain_Callback: pfSTOP_CAMERA error!')
end
error_code = pfSETMODE(handles.board_number, handles.mode, 0, handles.exposure,...
                    handles.hbin,handles.vbin,handles.gain, 0,handles.bit_pix,0);
if error_code ~= 0
     error('....gain_Callback: pfSETMODE error!');
end
error_code = pfSTART_CAMERA(handles.board_number);
if error_code ~= 0
      error('....gain_Callback: pfSTART_CAMERA error!')
end
% Update handles structure
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function preview_togglebutton_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to preview_togglebutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
if ispc
  set(hObject,'BackgroundColor',[0.9,0.9,0.9]);
else
  set(hObject,'BackgroundColor','white');
end

% --------------------------------------------------------------------
function uipanel13_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to uipanel13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
switch get(hObject,'Tag')   % Get Tag of selected object
   case 'radiobutton8'
      handles.bit_pix=8;
      error_code = pfSTOP_CAMERA(handles.board_number);
      if error_code ~= 0
          error('....radiobutton8_Callback: pfSTOP_CAMERA error!')
      end
      error_code = pfSETMODE(handles.board_number, handles.mode, 0,
handles.exposure,...
                          handles.hbin,handles.vbin,handles.gain, 0,handles.bit_pix,0);
      if error_code ~= 0
          error('....radiobutton8_Callback: pfSETMODE error!');
      end
      error_code = pfSTART_CAMERA(handles.board_number);
      if error_code ~= 0
          error('....radiobutton8_Callback: pfSTART_CAMERA error!')
      end
      handles.depth = 1;
      handles.imagesize=handles.image_width*handles.image_height;
      colormap(gray);

   case 'radiobutton12'
      if handles.bit_pix~=12;
         handles.bit_pix=12;
         error_code = pfSTOP_CAMERA(handles.board_number);
         if error_code ~= 0
             error('....radiobutton12_Callback: pfSTOP_CAMERA error!')
         end
         error_code = pfSETMODE(handles.board_number, handles.mode, 0,
handles.exposure,...
                          handles.hbin,handles.vbin,handles.gain, 0,handles.bit_pix,0);
         if error_code ~= 0
             error('....radiobutton12_Callback: pfSETMODE error!');
         end
         error_code = pfSTART_CAMERA(handles.board_number);
         if error_code ~= 0
```

```
            error('....radiobutton12_Callback: pfSTART_CAMERA error!')
        end
        handles.imagesize=handles.image_width*handles.image_height*2;
    end
    handles.depth = 2;
    colormap(gray);

  case 'radiobuttonrgb'
    if handles.bit_pix~=12
        handles.bit_pix=12;
        error_code = pfSTOP_CAMERA(handles.board_number);
        if error_code ~= 0
            error('....radiobuttonrgb_Callback: pfSTOP_CAMERA error!')
        end
        error_code = pfSETMODE(handles.board_number, handles.mode, 0,
handles.exposure,...
                               handles.hbin,handles.vbin,handles.gain, 0,handles.bit_pix,0);
        if error_code ~= 0
            error('....radiobuttonrgb_Callback: pfSETMODE error!');
        end
        error_code = pfSTART_CAMERA(handles.board_number);
        if error_code ~= 0
            error('....radiobuttonrgb_Callback: pfSTART_CAMERA error!')
        end
        handles.imagesize=handles.image_width*handles.image_height*2;
    end
    handles.image= zeros(double(handles.image_height), double(handles.image_width),3);
    handles.image_buffer_map = image(handles.image);
    drawnow
    if handles.color==1
        handles.depth = 3;
    else
        handles.depth=2;
    end
end
% Update handles structure
guidata(hObject, handles);
```