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

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

% Last Modified by GUIDE v2.5 12-Sep-2016 13:45:41

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gui_OpeningFcn, ...
                   'gui_OutputFcn',  @gui_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
end


% --- Executes just before gui is made visible.
function gui_OpeningFcn(hObject, ~, 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 gui (see VARARGIN)

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

    % load java files for QR-Code decoding
    javaaddpath('core-1.7.jar')
    javaaddpath('javase-1.7.jar')

    % Load parameter functions
    handles.param = parameter();

    % Set axes palette
    colormap(handles.camview_infrared, 'jet');

    % load images to axes
    img = imread('QRCode.png');
    image(img, 'parent', handles.camview_webcam);
    handles.camview_webcam =
    set_camview_default(handles.camview_webcam);
    img = imread('laser.png');
    image(img, 'parent', handles.camview_laser);
    handles.camview_laser =
    set_camview_default(handles.camview_laser);
    img = imread('multispectral.jpg');
    image(img, 'parent', handles.camview_multispectral);
    handles.camview_multispectral =
    set_camview_default(handles.camview_multispectral);
    img = imread('infrared.jpg');
    image(img, 'parent', handles.camview_infrared);
    handles.camview_infrared =
    set_camview_default(handles.camview_infrared);

    % no laser images captured at start
    handles.laser_images = false;

    % disable demo mode at start
```

```matlab
    handles.demo = false;

    % Update handles structure
    guidata(hObject, handles);

    % UIWAIT makes gui wait for user response (see UIRESUME)
    % uiwait(handles.LegoDemo);
end

function camview = set_camview_default(camview)
    camview.XTick = [];
    camview.YTick = [];
    camview.CLim = [0, 255];
    camview.CLimMode = 'manual';
    camview.DataAspectRatio = [1, 1, 1];
end

function enabled = is_serial_port(handles)
    enabled = isfield(handles, 'serial') && isa(handles.serial,
    'class_serial_port');
end

function enabled = is_webcam(handles)
    enabled = isfield(handles, 'webcam') && isa(handles.webcam,
    'class_videoinput');
end

function enabled = is_laser(handles)
    enabled = isfield(handles, 'laser') && isa(handles.laser,
    'class_videoinput');
end

function enabled = is_infrared(handles)
    enabled = isfield(handles, 'infrared') &&
    isa(handles.infrared, 'class_videoinput');
end

function enabled = is_multispectral(handles)
    enabled = isfield(handles, 'multispectral') &&
    isa(handles.multispectral, 'class_gigecam');
end

% --- Executes on button press in camera_init.
function camera_init_Callback(hObject, ~, handles) %#ok<DEFNU>
% hObject    handle to camera_init (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles.camera_init.Enable = 'off';
```

```matlab

        % Update handles structure
        guidata(hObject, handles);
        drawnow();

        all_success = true;

        if isfield(handles, 'gigelist') == false || istable(handles.gigelist) == false
            try
                handles.gigelist = gigecamlist(); % speed up gigecam initialization
            catch e
                all_success = false;
                warning('Exception in gigecamlist(): %s', getReport(e));
            end
        end

        if ~is_webcam(handles)
            try
                handles.webcam = handles.param.camera_webcam();
                handles = enable_webcam(handles, 'on');
            catch e
                all_success = false;
                warning('Exception in camera_webcam(): %s', getReport(e));
            end
        end

        if ~is_laser(handles)
            try
                handles.laser = handles.param.camera_laser();
                handles = enable_laser(handles, 'on');
            catch e
                all_success = false;
                warning('Exception in camera_laser(): %s', getReport(e));
            end
        end

        if ~is_infrared(handles)
            try
                handles.infrared = handles.param.camera_infrared();
                handles = enable_infrared(handles, 'on');
            catch e
                all_success = false;
                warning('Exception in camera_infrared(): %s', getReport(e));
            end
        end

        if ~is_multispectral(handles)
            try
                handles.multispectral = handles.param.camera_multispectral(handles.gigelist);
                handles = enable_multispectral(handles, 'on');
            catch e
                all_success = false;
                warning('Exception in camera_multispectral(): %s', getReport(e));
            end
        end

        % Set preview data to native camera bit depth (default is 8 bit)
        imaqmex('feature', '-previewFullBitDepth', true);

        if all_success == false
            handles.camera_init.Enable = 'on';
        end

        % Update handles structure
        guidata(hObject, handles);
    end

    function handles = enable_webcam(handles, value)
        if ~is_webcam(handles) || handles.demo
            return;
        end

        handles.live_webcam.Enable = value;
        handles.stop_webcam.Enable = value;
        handles.snapshot_webcam.Enable = value;
        handles.qr_button.Enable = value;

        if is_serial_port(handles)
            handles.demomode.Enable = value;
        end
    end

    function handles = enable_laser(handles, value)
        if ~is_laser(handles) || handles.demo
            return;
        end

        handles.live_laser.Enable = value;
        handles.stop_laser.Enable = value;
```

```matlab
        handles.capture_start.Enable = value;
        handles.capture_stop.Enable = value;
        handles.capture_calc.Enable = value;

        if is_serial_port(handles)
            handles.demomode.Enable = value;
        end
    end

    function handles = enable_multispectral(handles, value)
        if ~is_multispectral(handles) || handles.demo
            return;
        end

        handles.live_multispectral.Enable = value;
        handles.stop_multispectral.Enable = value;
        handles.snapshot_multispectral.Enable = value;

        if is_serial_port(handles)
            handles.demomode.Enable = value;
        end
    end

    function handles = enable_infrared(handles, value)
        if ~is_infrared(handles) || handles.demo
            return;
        end

        handles.live_infrared.Enable = value;
        handles.stop_infrared.Enable = value;
        handles.snapshot_infrared.Enable = value;

        if is_serial_port(handles)
            handles.demomode.Enable = value;
        end
    end

    % enable control elements
    function handles = enable_serial(handles, value)
        if ~handles.demo
            handles.train_dir_left.Enable = value;
            handles.train_dir_right.Enable = value;
            handles.train_speed.Enable = value;

            handles.led0.Enable = value;
            handles.led1.Enable = value;
            handles.led2.Enable = value;
            handles.led3.Enable = value;
            handles.ledA.Enable = value;

            handles.halo0.Enable = value;
            handles.halo1.Enable = value;
            handles.haloA.Enable = value;
        end

        handles.demomode.Enable = value;
    end

    % --- Executes on button press in connect_serial_port.
    function connect_serial_port_Callback(hObject, ~, handles) %#ok<DEFNU>
    % hObject    handle to connect_serial_port (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
        handles.connect_serial_port.Enable = 'off';

        % Update handles structure
        guidata(hObject, handles);
        drawnow();

        % Initialize serial port
        if is_serial_port(handles) == false
            try
                handles.serial = handles.param.serial_port({@serial_callback, hObject});
            catch e
                warning('Exception in camera_webcam(): %s', getReport(e));
            end
        end

        % connect
        success = false;
        try
            success = handles.serial.connect();
        catch e
            warning('Exception in serial.connect: %s', getReport(e));
        end

        if success == true
            handles = enable_serial(handles, 'on');
        else
            % warn that connecting failed
            waitfor(msgbox('Verbindung konnte nicht hergestellt werden.', 'Fehler', 'warn'));
            handles.connect_serial_port.Enable = 'on';
        end

        % Update handles structure
        guidata(hObject, handles);
    end

    % --- Outputs from this function are returned to the command line.
    function varargout = gui_OutputFcn(~, ~, 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;
    end

    function img = normalize_adjust(img)
        img = double(img);
        minv = min(img(:));
        maxv = max(img(:));
        diff = maxv - minv;
        img = (img - minv) ./ diff;
    end

    function img = normalize_adjust_255(img)
        img = normalize_adjust(img) .* 255;
    end

    function img = infrared_adjust(img)
        img = normalize_adjust_255(img);
        img = imrotate(img, 90);
    end

    function preview_normalize_adjust_255(~, event, himage)
        himage.CData = normalize_adjust_255(event.Data);
    end

    function preview_gray(~, event, himage)
        himage.CData = event.Data(:,:,1);
    end

    function preview_infrared_adjust(~, event, himage)
        img = (normalize_adjust(event.Data) .* 64);
        img = imrotate(img, 90);
        himage.CData = img;
```

```matlab
end

% --- Executes on button press in live_webcam.
function live_webcam_Callback(hObject, ~, handles) %#ok<DEFNU>
% hObject    handle to live_webcam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles = enable_webcam(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    if ~handles.webcam.preview(false, handles.camview_webcam)
        waitfor(msgbox('Livebild konnte nicht geladen werden.', 'Fehler', 'warn'));
    end

    handles = enable_webcam(handles, 'on');
    guidata(hObject, handles);
end

% --- Executes on button press in live_laser.
function live_laser_Callback(hObject, ~, handles) %#ok<DEFNU>
% hObject    handle to live_laser (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles = enable_laser(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    colormap(handles.camview_laser, 'gray');
    if ~handles.laser.preview(@preview_normalize_adjust_255, handles.camview_laser)
        waitfor(msgbox('Livebild konnte nicht geladen werden.', 'Fehler', 'warn'));
    end

    handles = enable_laser(handles, 'on');
    guidata(hObject, handles);
end

% --- Executes on button press in live_multispectral.
function live_multispectral_Callback(hObject, ~, handles) %#ok<DEFNU>
% hObject    handle to live_multispectral (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```matlab
    handles = enable_multispectral(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    colormap(handles.camview_multispectral, 'hot');
    if ~handles.multispectral.preview(@preview_gray, handles.camview_multispectral)
        waitfor(msgbox('Livebild konnte nicht geladen werden.', 'Fehler', 'warn'));
    end

    handles = enable_multispectral(handles, 'on');
    guidata(hObject, handles);
end

% --- Executes on button press in live_infrared.
function live_infrared_Callback(hObject, ~, handles)
% hObject    handle to live_infrared (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles = enable_infrared(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    if ~handles.infrared.preview(@preview_infrared_adjust, handles.camview_infrared, true)
        waitfor(msgbox('Livebild konnte nicht geladen werden.', 'Fehler', 'warn'));
    end

    handles = enable_infrared(handles, 'on');
    guidata(hObject, handles);
end

% --- Executes on button press in stop_webcam.
function stop_webcam_Callback(hObject, ~, handles) %#ok<DEFNU>
% hObject    handle to stop_webcam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles = enable_webcam(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    if ~handles.webcam.stoppreview()
        waitfor(msgbox('Livebild konnte gestoppt werden.', 'Fehler', 'warn'));
    end

    handles = enable_webcam(handles, 'on');
    guidata(hObject, handles);
end

% --- Executes on button press in stop_laser.
function stop_laser_Callback(hObject, ~, handles) %#ok<DEFNU>
% hObject    handle to stop_laser (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles = enable_laser(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    if ~handles.laser.stoppreview()
        waitfor(msgbox('Livebild konnte gestoppt werden.', 'Fehler', 'warn'));
    end

    handles = enable_laser(handles, 'on');
    guidata(hObject, handles);
end

% --- Executes on button press in stop_multispectral.
function stop_multispectral_Callback(hObject, ~, handles) %#ok<DEFNU>
% hObject    handle to stop_multispectral (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles = enable_multispectral(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    if ~handles.multispectral.stoppreview()
        waitfor(msgbox('Livebild konnte gestoppt werden.', 'Fehler', 'warn'));
    end

    handles = enable_multispectral(handles, 'on');
    guidata(hObject, handles);
end

% --- Executes on button press in stop_infrared.
function stop_infrared_Callback(hObject, ~, handles)
% hObject    handle to stop_infrared (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles = enable_infrared(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    if ~handles.infrared.stoppreview()
        waitfor(msgbox('Livebild konnte gestoppt werden.', 'Fehler', 'warn'));
    end

    handles = enable_infrared(handles, 'on');
    guidata(hObject, handles);
end

% --- Executes on button press in snapshot_webcam.
function snapshot_webcam_Callback(hObject, ~, handles)
% hObject    handle to snapshot_webcam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles = enable_webcam(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    fprintf('Webcam start\n');
    if ~handles.webcam.snapshot(false, handles.camview_webcam)
        waitfor(msgbox('Einzelbild konnte nicht geladen werden.', 'Fehler', 'warn'));
    end
    fprintf('Webcam end\n');

    handles = enable_webcam(handles, 'on');
    guidata(hObject, handles);
end

% --- Executes on button press in snapshot_multispectral.
function snapshot_multispectral_Callback(hObject, ~, handles)
% hObject    handle to snapshot_multispectral (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles = enable_multispectral(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    fprintf('Multispectral start\n');
    colormap(handles.camview_multispectral, 'hot');
    if ~handles.multispectral.snapshot(false, handles.camview_multispectral)
```

```matlab
        waitfor(msgbox('Einzelbild konnte nicht geladen werden.', 'Fehler', 'warn'));
    end
    fprintf('Multispectral end\n');

    handles = enable_multispectral(handles, 'on');
    guidata(hObject, handles);
end

% --- Executes on button press in snapshot_infrared.
function snapshot_infrared_Callback(hObject, ~, handles) %#ok<DEFNU>
% hObject    handle to snapshot_infrared (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles = enable_infrared(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    fprintf('Infrared start\n');
    if ~handles.infrared.snapshot(@infrared_adjust, handles.camview_infrared)
        waitfor(msgbox('Einzelbild konnte nicht geladen werden.', 'Fehler', 'warn'));
    end
    handles.camview_infrared = set_camview_default(handles.camview_infrared);
    fprintf('Infrared end\n');

    handles = enable_infrared(handles, 'on');
    guidata(hObject, handles);
end

% --- Executes during object creation, after setting all properties.
function train_speed_CreateFcn(hObject, ~, ~) %#ok<DEFNU>
% hObject    handle to train_speed (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

    % Hint: slider controls usually have a light gray background.
    if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end
end
```

```matlab

% --- Executes on slider movement.
function train_speed_Callback(hObject, ~, handles)
% hObject    handle to train_speed (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,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
    handles = enable_serial(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    try
        val = round(hObject.Value);
        hObject.Value = val;

        handles.train_speed_label.String = sprintf('%d', val);
        handles.serial.setTrainSpeed(val, handles.train_dir_left.Value == 0);
    catch e
        warning('Train exception: %s', getReport(e));
        waitfor(msgbox('Interner Fehler während der SerialPort Kommunikation.', 'Fehler', 'warn'));
    end

    % Update handles structure
    handles = enable_serial(handles, 'on');
    guidata(hObject, handles);
end

% --- Executes on button press in train_dir_XXX.
function train_dir_Callback(hObject, ~, handles) %#ok<DEFNU>
% hObject    handle to train_dir_left (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles = enable_serial(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    try
        speed = str2double(handles.train_speed_label.String);
        handles.serial.setTrainSpeed(speed, handles.train_dir_left.Value == 0);
    catch e
        warning('Train exception: %s', getReport(e));
        waitfor(msgbox('Interner Fehler während der SerialPort Kommunikation.', 'Fehler', 'warn'));
    end

    % Update handles structure
    handles = enable_serial(handles, 'on');
    guidata(hObject, handles);
end

% --- Executes on button press in ledX.
function led_Callback(hObject, ~, handles) %#ok<DEFNU>
% hObject    handle to led0 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles = enable_serial(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    try
        switch(1)
            case handles.led0.Value
                val = 0;
            case handles.led1.Value
                val = 1;
            case handles.led2.Value
                val = 2;
            case handles.led3.Value
                val = 3;
            case handles.ledA.Value
                val = 4;
            otherwise
                error('Unknown LED Radio Button checked.');
        end
        handles.serial.setLed(val);
    catch e
        warning('LED exception: %s', getReport(e));
        waitfor(msgbox('Interner Fehler während der SerialPort Kommunikation.', 'Fehler', 'warn'));
    end

    % Update handles structure
    handles = enable_serial(handles, 'on');
    guidata(hObject, handles);
end

% --- Executes on button press in haloX.
function halo_Callback(hObject, ~, handles) %#ok<DEFNU>
% hObject    handle to halo1 (see GCBO)
```

```matlab
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles = enable_serial(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    try
        switch(1)
            case handles.halo0.Value
                val = 0;
            case handles.halo1.Value
                val = 1;
            case handles.haloA.Value
                val = 4;
            otherwise
                error('Unknown Halogen Radio Button checked.');
        end
        handles.serial.setHalogen(val);
    catch e
        warning('Halogen exception: %s', getReport(e));
        waitfor(msgbox('Interner Fehler während der SerialPort
        Kommunikation.', 'Fehler', 'warn'));
    end

    % Update handles structure
    handles = enable_serial(handles, 'on');
    guidata(hObject, handles);
end

% --- Executes on button press in qr_button.
function qr_button_Callback(hObject, ~, handles)
% hObject    handle to qr_button (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles = enable_webcam(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    try
        img = getimage(handles.camview_webcam);

        text = decode_qr(img);
        if isempty(text)
            color = [1, 0, 0];
            text = 'Nichts erkannt ...';
        else
            color = [0, 1, 0];
        end
        handles.qr_text.String = text;
        handles.qr_text.ForegroundColor = color;
    catch e
        warning('QR-Code exception: %s', getReport(e));
        waitfor(msgbox('Interner Fehler während der QR-Code-Erkennung.', 'Fehler', 'warn'));
    end

    % Update handles structure
    handles = enable_webcam(handles, 'on');
    guidata(hObject, handles);
end

function handle = config_laser_start(handle, demomode)
    triggerconfig(handle, 'hardware', 'DeviceSpecific', 'DeviceSpecific');

    % remove all images in cache
    if handle.FramesAvailable > 0
        handle.FramesPerTrigger = handle.FramesAvailable;
        getdata(handle);
    end

    if demomode
        handle.FramesPerTrigger = 500;
    else
        handle.FramesPerTrigger = 500;
    end

    src = getselectedsource(handle);
    src.TriggerMode = 'On';

    start(handle);
end

% --- Executes on button press in capture_start.
function capture_start_Callback(hObject, ~, handles)
% hObject    handle to capture_start (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles = enable_laser(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    if handles.laser.stoppreview() == false || handles.laser.config({@config_laser_start, handles.demo}) == false
```

```matlab
        waitfor(msgbox('Interner Fehler während der Laserlinienbild-Aufnahmekonfiguration.', 'Fehler', 'warn'));

        handles = enable_laser(handles, 'on');
        guidata(hObject, handles);
        return;
    end

    if ~handles.demo
        handles.capture_stop.Enable = 'on';
    end

    % Update handles structure
    guidata(hObject, handles);
end

function [handle, images] = get_images(handle)
    stop(handle);
    count = handle.FramesAvailable;

    handle.FramesPerTrigger = count;
    images = getdata(handle);

    triggerconfig(handle, 'manual');

    src = getselectedsource(handle);
    src.TriggerMode = 'Off';
    handle.FramesPerTrigger = 1;
end

% --- Executes on button press in capture_stop.
function capture_stop_Callback(hObject, ~, handles)
% hObject    handle to capture_stop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles = enable_laser(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    [success, images] = handles.laser.config(@get_images);
    if success && ~islogical(images)
        n = size(images, 4);
        handles.laser_images = images;

        colormap(handles.camview_laser, 'gray');
        handles.image_slider.Enable = 'on';
        handles.image_slider.Min = 1;
        handles.image_slider.Max = n;
        handles.image_slider.SliderStep = [1 / (n - 1), 1 / (n - 1)];
        for i = 1:n
            if handles.demo && mod(i, 5) ~= 0
                continue;
            end
            capture_img = handles.laser_images(:, :, 1, i);
            image(capture_img, 'parent', handles.camview_laser);
            handles.camview_laser = set_camview_default(handles.camview_laser);
            handles.image_slider.Value = i;
            handles.img_count.String = sprintf('%d von %d', i, n);
            guidata(hObject, handles);
            drawnow();
            pause(0.001);
        end

        handles.cut_begin.Enable = 'on';
        handles.cut_end.Enable = 'on';
    else
        n = 0;
        handles.laser_images = false;
        handles.image_slider.Enable = 'off';
        handles.cut_begin.Enable = 'off';
        handles.cut_end.Enable = 'off';
    end

    handles.img_count.String = sprintf('Bilder: %d', n);

    handles = enable_laser(handles, 'on');
    guidata(hObject, handles);
end

% --- Executes on button press in capture_calc.
function capture_calc_Callback(hObject, ~, handles)
% hObject    handle to capture_calc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    handles = enable_laser(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    if ~islogical(handles.laser_images)
        n = size(handles.laser_images, 4);
    else
        n = 0;
    end

    if n < 10
        if ~handles.demo
            waitfor(msgbox('Zu wenige Bilder für 3D-Bild Berechnung.', 'Fehler', 'warn'));
        end

        handles = enable_laser(handles, 'on');
        guidata(hObject, handles);
        return;
    end

    try
        Threshold = 100;             % Gray value threshold for backgound extraction
        % pre calibatrion data - use the program laserschnittverfahren3 to calibrate your system
        ps = 0.054381;              % Pixel relative size (mm/pixel)
        alpha = 13.844982;          % Triangulation angle in (Â°)
        LinCoef = 0;                % Linear coefficient 105
        AngCoef = 0;                % Angular coefficient
        data3d = get3D(handles.laser_images, Threshold, ps, alpha, LinCoef, AngCoef);

        handles.img_count.String = sprintf('Bilder: %d', n);

        colormap(handles.camview_laser, 'jet');
        mesh(data3d, 'parent', handles.camview_laser);
        handles.camview_laser.YTick = [];
        handles.camview_laser.XTick = [];
        handles.camview_laser.ZTick = [];
        min3d = min(min(data3d));
        handles.camview_laser.DataAspectRatio = [15, 4.5, 1.5];
        handles.camview_laser.CLim = [min3d, 0];
        handles.camview_laser.CLimMode = 'manual';
        rotate3d(handles.camview_laser, 'on');
        guidata(hObject, handles);
        drawnow();
    catch e
        warning('3D calc exception: %s', getReport(e));
        waitfor(msgbox('Interner Fehler während der 3D-Bild Berechnung.', 'Fehler', 'warn'));
    end

    handles = enable_laser(handles, 'on');
    guidata(hObject, handles);
end
```

```matlab

% --- Executes on slider movement.
function image_slider_Callback(hObject, ~, handles) %#ok<DEFNU>
% hObject    handle to image_slider (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,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
    try
        val = round(hObject.Value);
        hObject.Value = val;

        if ~islogical(handles.laser_images)
            handles.img_count.String = sprintf('%d von %d', val, size(handles.laser_images, 4));

            colormap(handles.camview_laser, 'gray');
            image(handles.laser_images(:, :, 1, val), 'parent', handles.camview_laser);
            handles.camview_laser = set_camview_default(handles.camview_laser);
        end
    catch e
        warning('Laser silder exception: %s', getReport(e));
    end

    guidata(hObject, handles);
end

% --- Executes during object creation, after setting all properties.
function image_slider_CreateFcn(hObject, ~, ~) %#ok<DEFNU>
% hObject    handle to image_slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

    % Hint: slider controls usually have a light gray background.
    if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end
end


```

```matlab
% --- Executes on button press in cut_begin.
function cut_begin_Callback(hObject, ~, handles) %#ok<DEFNU>
% hObject    handle to cut_begin (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
    n = size(handles.laser_images, 4);
    first = handles.image_slider.Value;
    handles.laser_images = handles.laser_images(:, :, 1,
    first:n);
    n = size(handles.laser_images, 4);

    handles.image_slider.Min = 1;
    handles.image_slider.Max = n;
    if n > 1
        step = 1 / (n - 1);
    else
        step = 1;
    end
    handles.image_slider.Value = 1;
    handles.image_slider.SliderStep = [step, step];
    handles.image_slider.Value = 1;
    handles.img_count.String = sprintf('%d von %d',
    handles.image_slider.Value, n);

    guidata(hObject, handles);
end

% --- Executes on button press in cut_end.
function cut_end_Callback(hObject, ~, handles) %#ok<DEFNU>
% hObject    handle to cut_end (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
    last = handles.image_slider.Value;
    handles.laser_images = handles.laser_images(:, :, 1, 1:last);
    n = size(handles.laser_images, 4);

    handles.image_slider.Min = 1;
    handles.image_slider.Max = n;
    if n > 1
        step = 1 / (n - 1);
    else
        step = 1;
    end
    handles.image_slider.Value = 1;
    handles.image_slider.SliderStep = [step, step];
    handles.image_slider.Value = n;
    handles.img_count.String = sprintf('%d von %d',
        handles.image_slider.Value, n);

    guidata(hObject, handles);
end

% --- Executes on button press in demomode.
function demomode_Callback(hObject, ~, handles) %#ok<DEFNU>
% hObject    handle to demomode (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    % Set train speed
    if handles.demomode.Value == 0
        handles.train_speed.Value = 0;
    else
        handles.train_speed.Value = 9;
        handles.train_dir_left.Value = 1;
    end
    train_speed_Callback(handles.train_speed, [], handles);
    handles = guidata(hObject);

    handles = enable_serial(handles, 'off');
    guidata(hObject, handles);
    drawnow();

    try
        if handles.demomode.Value == 0
            handles.serial.setDemoMode(0);

            stop_infrared_Callback(handles.snapshot_infrared,
            [], handles);
            handles = guidata(hObject);

            handles.demo = false;

            handles = enable_webcam(handles, 'on');
            handles = enable_laser(handles, 'on');
            handles = enable_infrared(handles, 'on');
            handles = enable_multispectral(handles, 'on');
        else
            handles.serial.setDemoMode(1);

            live_infrared_Callback(handles.snapshot_infrared,
            [], handles);
            handles = guidata(hObject);

            handles = enable_webcam(handles, 'off');
            handles = enable_laser(handles, 'off');
            handles = enable_infrared(handles, 'off');
```

```matlab
            handles = enable_multispectral(handles, 'off');

            handles.demo = true;
        end
    catch e
        warning('DemoMode exception: %s', getReport(e));
        waitfor(msgbox('Interner Fehler während der SerialPort Kommunikation.', 'Fehler', 'warn'));
    end

    % Update handles structure
    handles = enable_serial(handles, 'on');
    guidata(hObject, handles);
end

% --- Handles messages from COM-Port.
function serial_callback(type, parameter, hObject)
% hObject   handle to figure
% type      type of the message as string
% parameter depends on the type
%           if type is 'bat': train battery charging state
    handles = guidata(hObject);

    switch(type)
        case 'prelap1'
            if handles.demo
                if is_laser(handles)
                    capture_start_Callback(handles.capture_start, [], handles);
                    handles = guidata(hObject);

                    pause(12);

                    capture_stop_Callback(handles.capture_stop, [], handles);
                    handles = guidata(hObject);

                    capture_calc_Callback(handles.capture_calc, [], handles);
                    handles = guidata(hObject);
                end
            end
        case 'lap1'
        case 'prelap0'
        case 'lap0'
            if handles.demo
                % Camera often crashes Matlab
                if is_multispectral(handles) &&
                handles.haloA.Value ~= 1
                    snapshot_multispectral_Callback(handles.snapshot_multispectral, [], handles);
                    handles = guidata(hObject);
                end

                if is_webcam(handles) && handles.ledA.Value ~= 1
                    snapshot_webcam_Callback(handles.snapshot_webcam, [], handles);
                    handles = guidata(hObject);

                    qr_button_Callback(handles.qr_button, [], handles);
                    handles = guidata(hObject);
                end
            end
        case 'halo'
            if handles.demo
                % Camera often crashes Matlab
                if is_multispectral(handles)
                    pause(0.2);
                    snapshot_multispectral_Callback(handles.snapshot_multispectral, [], handles);
                    handles = guidata(hObject);
                end
            end
        case 'led'
            if handles.demo
                if is_webcam(handles)
                    snapshot_webcam_Callback(handles.snapshot_webcam, [], handles);
                    handles = guidata(hObject);

                    qr_button_Callback(handles.qr_button, [], handles);
                    handles = guidata(hObject);
                end
            end
        case 'bat'
            handles.battery_label.String = sprintf('Akku: %s', parameter);
        otherwise
            error('Unknown SerialPort Callback "%s".', type);
    end

    % Update handles structure
    guidata(hObject, handles);
end

% --- Executes when user attempts to close LegoDemo.
function LegoDemo_CloseRequestFcn(hObject, ~, handles)
%#ok<DEFNU>
% hObject    handle to LegoDemo (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
    if is_serial_port(handles)
        delete(handles.serial);
    end

    if is_webcam(handles)
        delete(handles.webcam);
    end

    if is_laser(handles)
        delete(handles.laser);
    end

    if is_infrared(handles)
        delete(handles.infrared);
    end

    if is_multispectral(handles)
        delete(handles.multispectral);
    end

    delete(hObject);
end
```