

# MATLAB®による画像処理・コンピュータビジョン評価キット

R2019b

MathWorks Japan

アプリケーションエンジニアリング部

# 評価キットの利用方法

- MATLAB/Simulink®による画像処理・コンピュータービジョン・ディープラーニング関連機能のご評価を目的とした資料です。
- ページ内でオレンジの領域で囲われているものはコマンドになります。
- コマンドウィンドウに直接入力いただけかスクリプトファイルにしてご使用いただけます。  
例) `c = corner(I)`
- 本スライドおよびデモファイルは下記リンクの"Download"より入手いただけます。
  - <https://jp.mathworks.com/matlabcentral/fileexchange/68741-ipcv-eval-kit-for-japanese>
- スライドの章番号とファイル名'I'に続く番号が対応するデモファイルです。
  - 例) スライド「2.5.1 幾何学的変換（射影法等による空間変換）」
  - →`I2_05_1_tr_road.m`
- 使用方法は下記のビデオもご覧ください。
  - <https://jp.mathworks.com/videos/how-to-use-the-image-processing-and-computer-vision-evaluation-kit-1540461587600.html>

# 無料評価版：ホームページからのご依頼手順

<http://jp.mathworks.com>



<http://jp.mathworks.com/trialrequest>



# アジェンダ

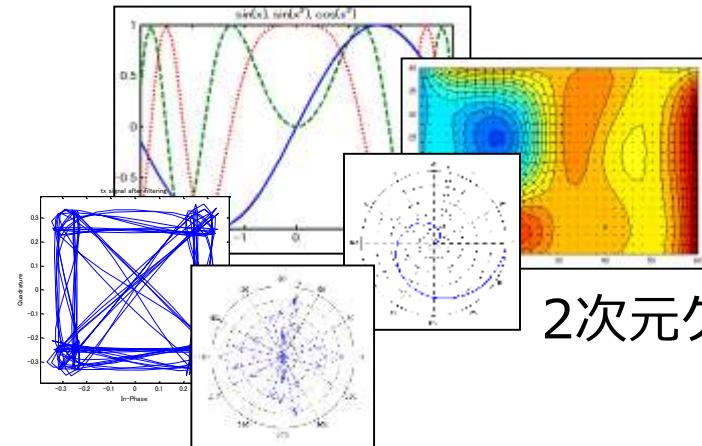
1. MATLAB/Simulinkの概要
2. 各種画像処理例
3. 連携機能
4. コンピュータービジョン処理例
5. 画像の機械学習・ディープラーニング
6. まとめ

## 1.1 画像処理デモ

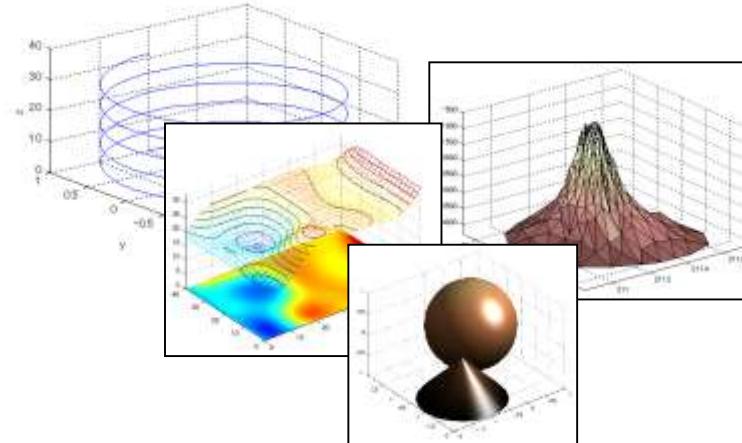
- MATLABによる画像の取扱い（行列操作等）

- インタプリターによる、簡潔な操作
- 変数が宣言・初期化なしで使用可
- 少ない記述量による処理
- 画像用可視化機能

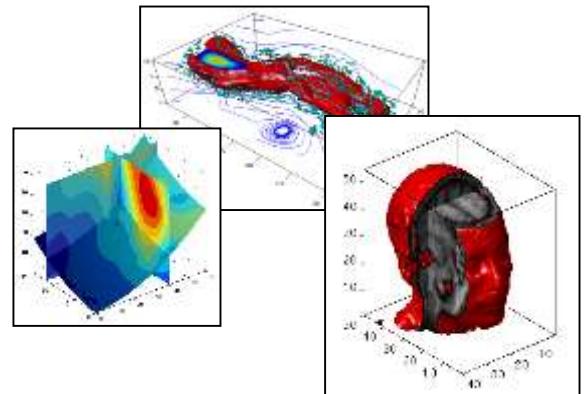
## 1.2 豊富な可視化機能：MATLAB基本関数



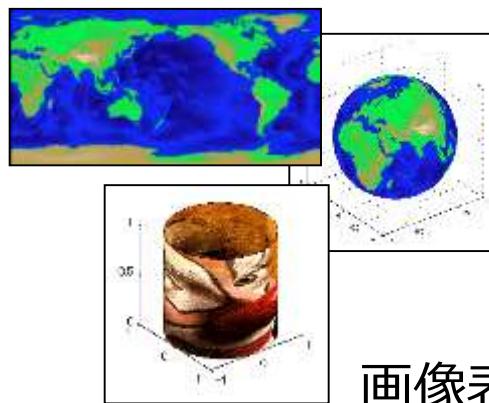
2次元グラフィックス



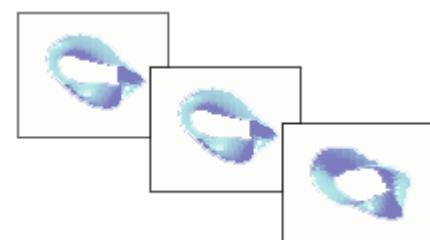
3次元グラフィックス



ボリュームデータ表示



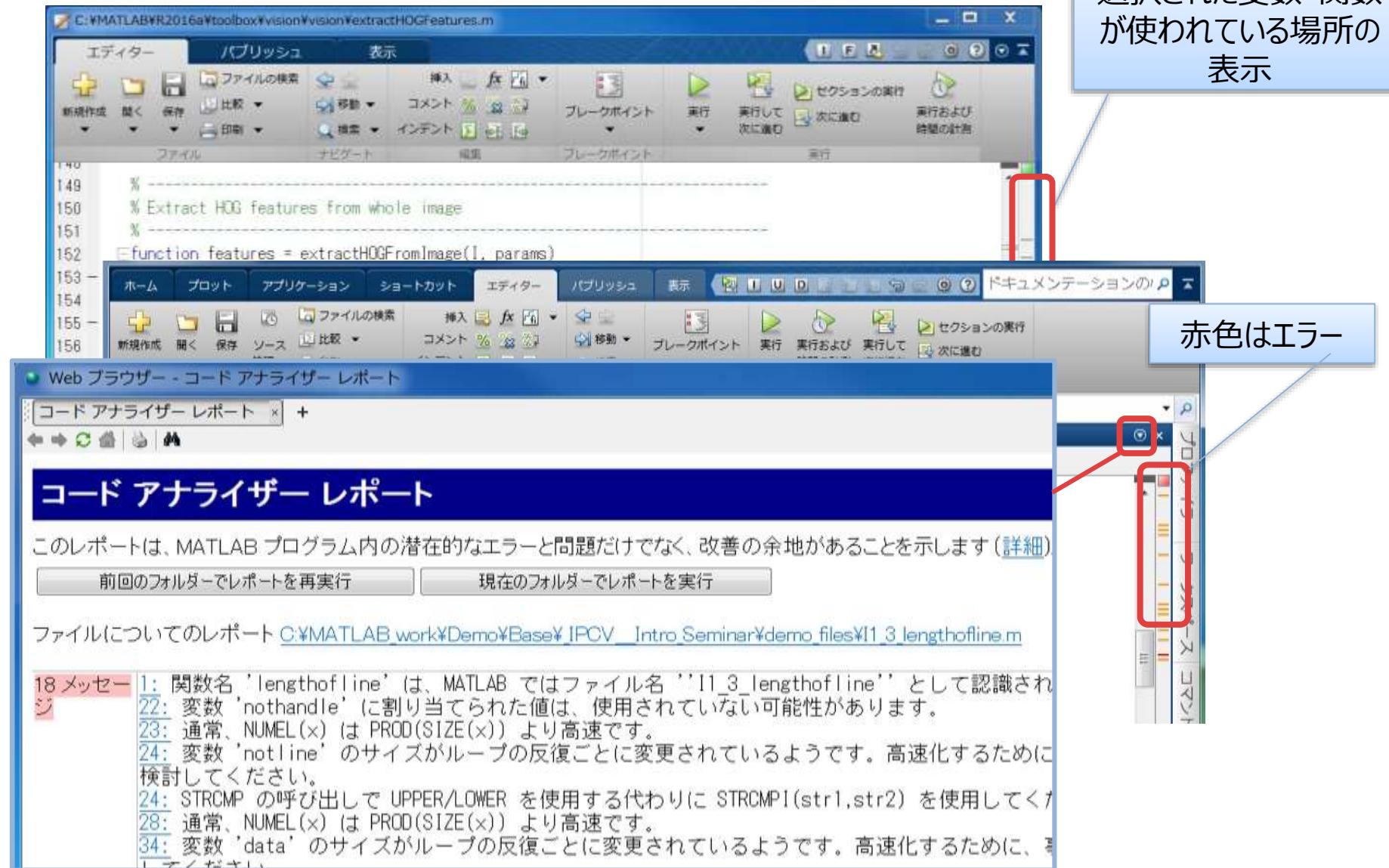
画像表示



アニメーション

高度なグラフィックスを簡単に実現  
UIによる編集も可能

# 1.3 デバッグ機能：コードアナライザ機能



## 1.4.1 デバッグ機能：ブレークポイント・プロファイラー

事前のブレークポイント設定なしでも、実行中に一時停止が可能

The screenshot shows the MATLAB Editor with the following code:

```

function profilerTest()
    I = magic(1000);
    a = 0;
    for i=1:10
        I = myMult(I, 1, 1);
        a = a + max(I(:));
    end
    if n % I.100001100000000e+06 素数
        I=magic(1000);
    end

```

A red box highlights the breakpoint at line 13, which is currently active (indicated by a red dot). A red arrow points from this box to the status bar message "実行されない行". Another red box highlights the "ステップイン" (Step Into) and "ステップアウト" (Step Out) buttons in the toolbar.

**変数の値の確認**

The variable `a` is displayed in the workspace, showing its value as `1x1 double =`. A red arrow points from this value to the status bar message "実行されない行".

**プロファイラー概要**

関数名	呼び出し
<a href="#">11_4_profilerTest</a>	1
<a href="#">11_4_profilerTest&gt;myMult</a>	10
<a href="#">magic</a>	1

**プロファイラー**

行番号	コード	呼び出し	合計時間	% 時間	時間	プロット
11	I = myMult(I, 1, 1);	10	0.084 s	57.1%	0.084	
8	I = magic(1000);	1	0.046 s	31.3%	0.046	
12	a = a + max(I(:));	10	0.015 s	10.2%	0.015	
13	end	10	0.001 s	0.7%	0.001	
18	end	1	0 s	0%	0	
他のすべての行			0.001 s	0.7%	0.001	
合計			0.147 s	100%	0.147	

**プロファイラー結果**

項目	値
関数内の行の合計	13
非コード行 (コメント、空白行)	3
コード行 (実行可能な行)	10
実行されたコード行	8
実行されなかったコード行	2
カバレッジ (実行済行/実行可能行)	80.00 %

**関数リスト**

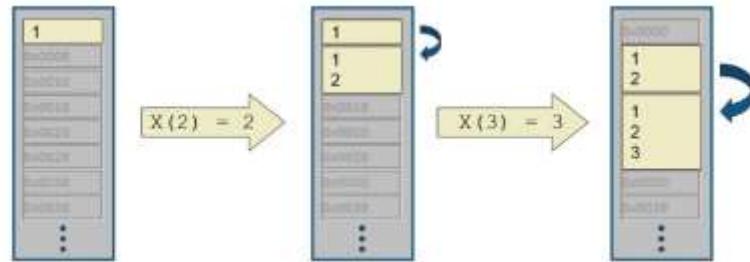
時間	呼び出し	行
0.05	1	8 I = magic(1000);
0.05	1	9 a = 0;
0.05	1	10 for i=1:10
0.08	10	11 I = myMult(I, 1, 1);
0.01	10	12 a = a + max(I(:));
< 0.01	10	13 end

## 1.4.2 処理速度の向上に関して

配列のリサイズはコストが大きい

```
>> x(1) = 1
>> x(2) = 2
>> x(3) = 3
```

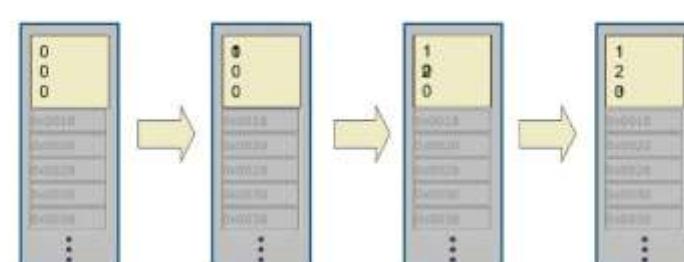
配列のリサイズは  
コストがかかる！



=> 事前割り当ての使用

```
>> x = zeros(3,1)
>> x(1) = 1
>> x(2) = 2
>> x(3) = 3
```

無駄なコピーが  
発生しない！

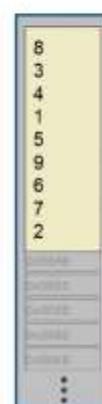


MATLAB配列のデータ格納形式

```
>> x = magic(3)
x =
    8     1     6
    3     5     7
    4     9     2
```

列優先

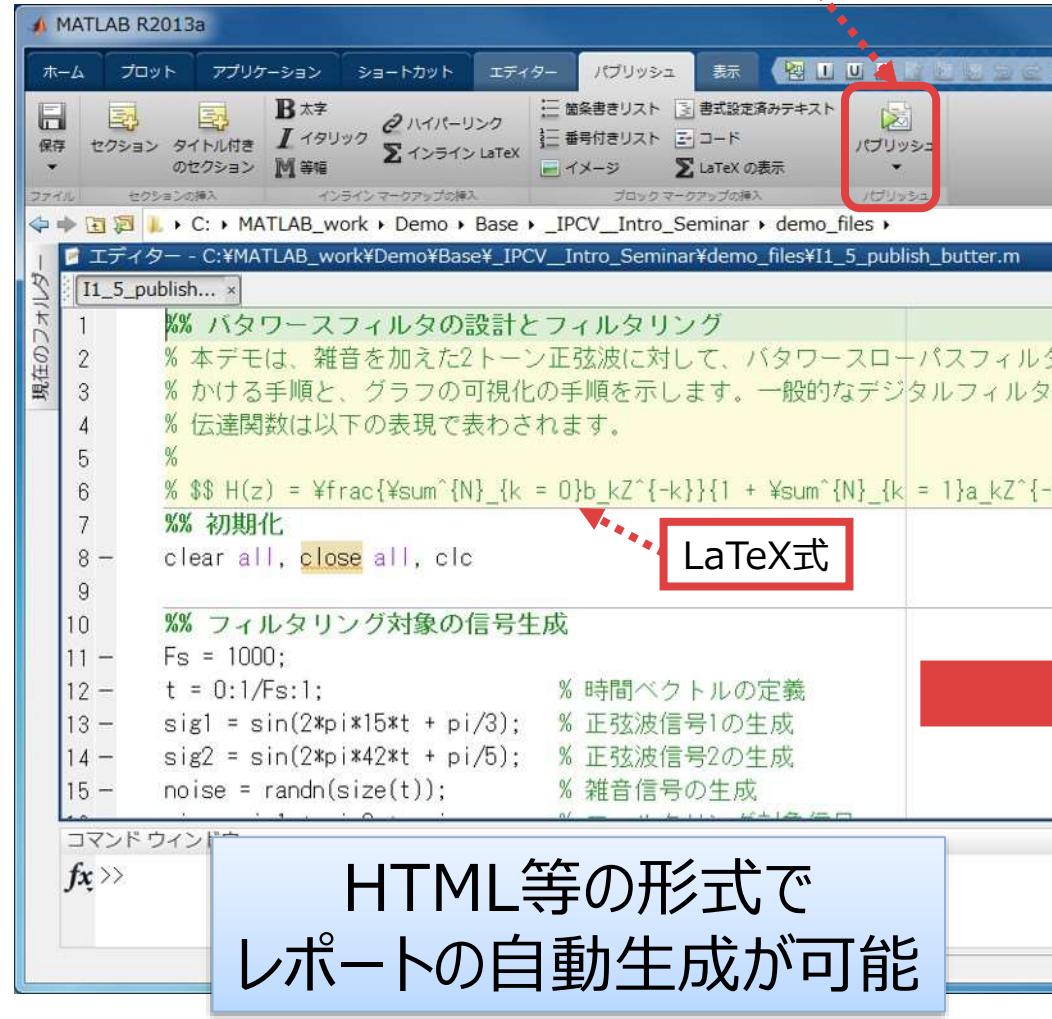
=> 縦にアクセス



パフォーマンスの改善。メモリ要件の特定と軽減  
<https://jp.mathworks.com/help/matlab/performance-and-memory.html>

# 1.5 レポート自動生成機能

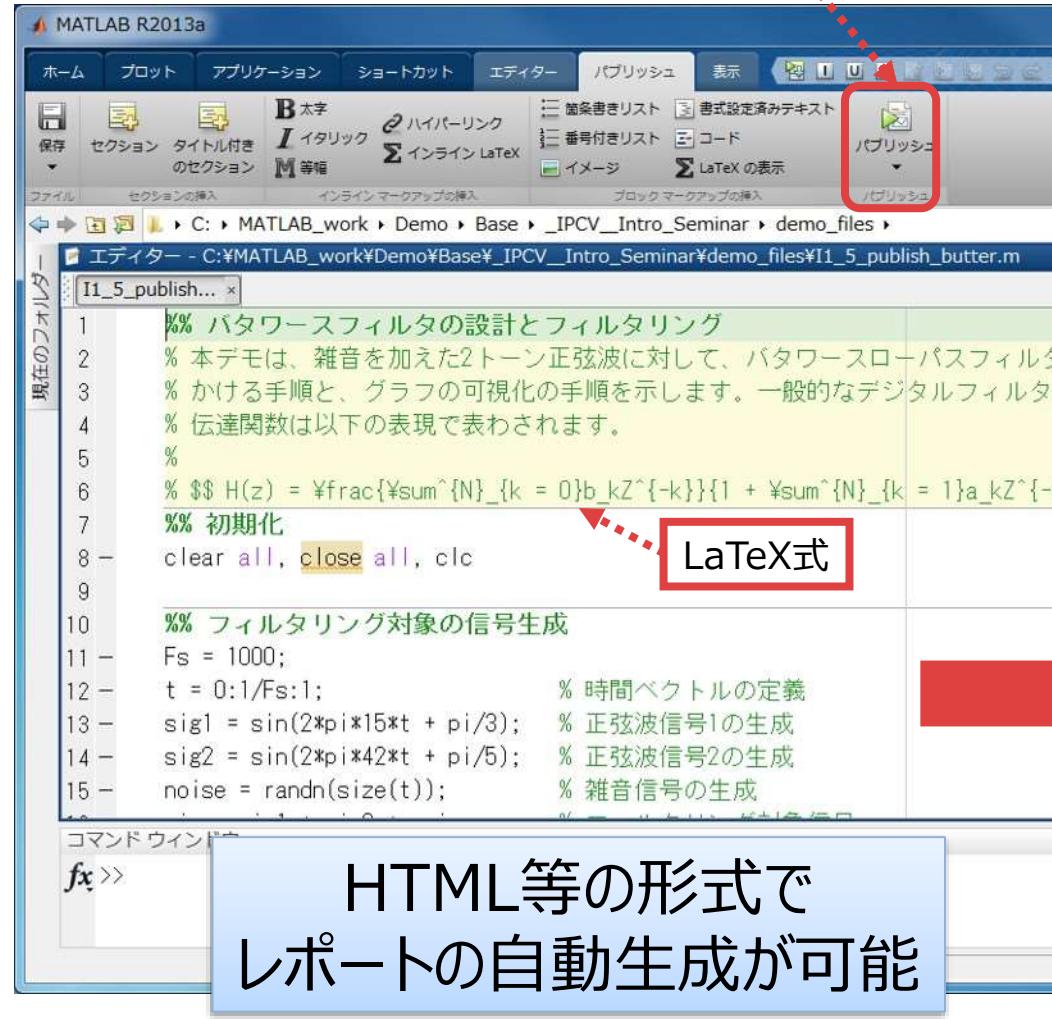
レポート生成ボタン



HTMLレポート



LaTeX式

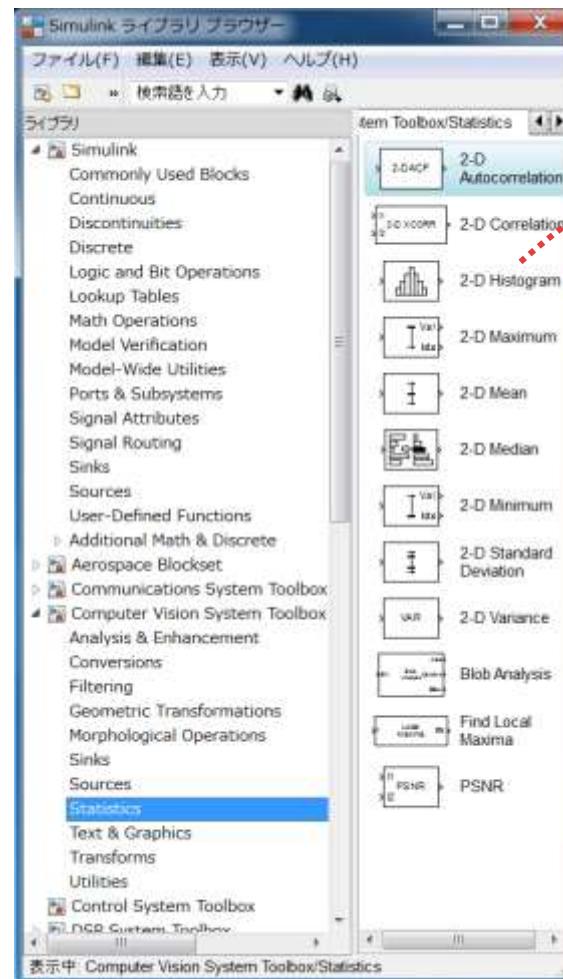


HTML等の形式で  
レポートの自動生成が可能

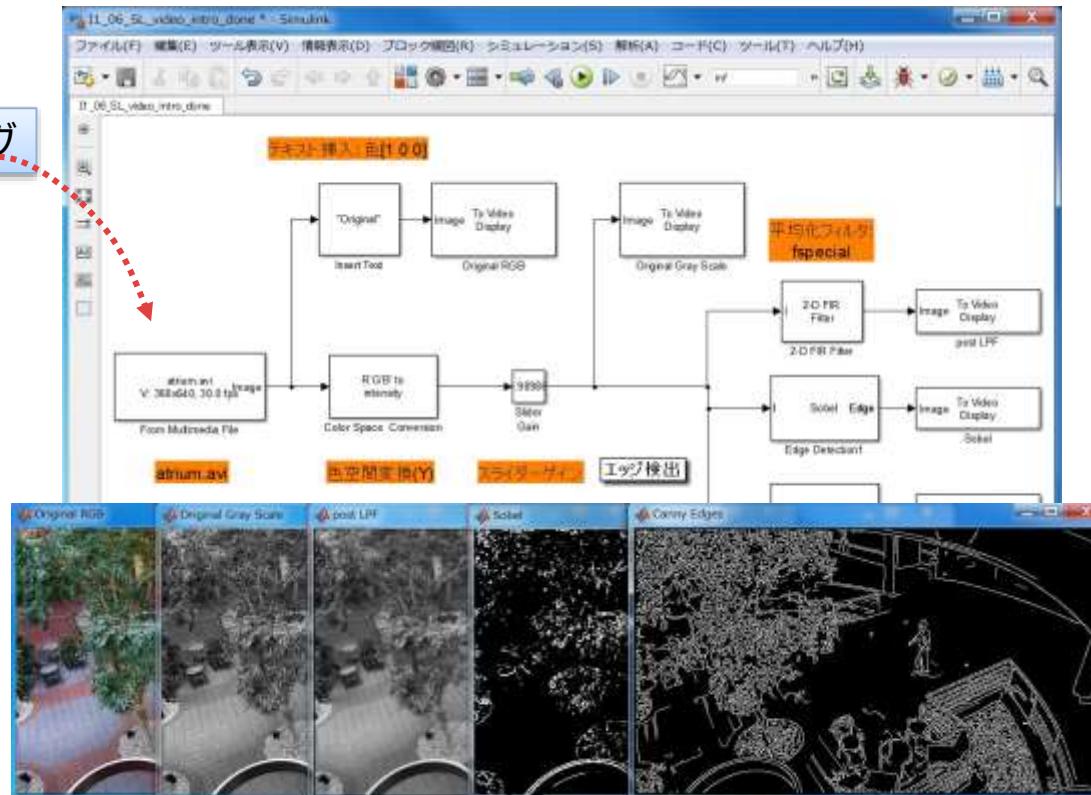
マークアップの詳細：[http://www.mathworks.co.jp/jp/help/matlab/matlab\\_prog/markup-marking-up-matlab-comments-for-publishing.html](http://www.mathworks.co.jp/jp/help/matlab/matlab_prog/markup-marking-up-matlab-comments-for-publishing.html)

# 1.6 Simulink : ブロック線図モデリング・シミュレーション環境

ライブラリブラウザ



モデルウィンドウ



- すぐに試して結果を見ることのできる環境
- 時間軸のシミュレーション（一時停止、ステップ実行）
- 各種アルゴリズムの容易な比較検討
- シミュレーション中のパラメータ変更 が容易に可能

# アジェンダ

1. MATLAB/Simulinkの概要
2. 各種画像処理例
3. 連携機能
4. コンピュータービジョン処理例
5. 画像の機械学習・ディープラーニング
6. まとめ

## 2.1.1 各種画像データフォーマットの読み込み・書き込み

### 画像の読み込み・書き込みのための関数

<code>imread()</code>	グラフィックス ファイルからイメージを読み込み ( <code>bmp, gif, jpg, png, tif, ...</code> )
<code>imwrite()</code>	イメージをグラフィックス ファイルに書き込み ( <code>bmp, gif, jpg, png, tif, ...</code> )
<code>Tiff()</code>	<code>LibTiff</code> ライブラリ ルーチンへのアクセス
<code>dicomread()</code>	DICOMイメージの読み込み
<code>dicomwrite()</code>	イメージをDICOMファイルとして書き込み
<code>nitfread()</code>	NITF(National Imagery Transmission Format)ファイルの読み込み <span style="color: orange;">R2017b</span>
<code>analyze75read()</code>	Analyze7.5 イメージ ファイルからのイメージ読み込み
<code>interfileread()</code>	Interfile形式でのイメージの読み込み
<code>hdrread()</code>	ハイダイナミックレンジ(HDR)イメージの読み込み
<code>hdrwrite()</code>	Radiance形式ハイダイナミックレンジ(HDR)イメージ ファイルの書き込み

```
A = imread('peppers.png') % ファイル拡張子によりファイル形式を判別
```

## 2.1.1 サポートされている画像読み込みデータフォーマット

`imread()` によりサポートされているフォーマット

(MATLAB基本関数)

BMP – Windows ビットマップ

CUR – Cursor ファイル

FTS – Flexible Image Transport System

GIF – Graphics Interchange Format

HDF4 – Hierarchical Data Format

ICO – Icon ファイル

JPEG – Joint Photographic Experts Group

JPEG 2000 – Joint Photographic Experts Group 2000

PBM – Portable Bitmap

PCX – Windows Paintbrush

PGM – Portable Graymap

PNG – ポータブル ネットワーク グラフ

PNM – Portable Any Map

PPM – Portable Pixmap

RAS – Sun ラスター

TIFF – Tagged Image File Format (TIFF：画像の一部の読み込みが可能)

XWD – X Window Dump

## 2.1.1 サポートされている画像書き込みデータフォーマット

`imwrite()` によりサポートされているフォーマット (MATLAB基本関数)

- BMP – Windows ビットマップ
- GIF – Graphics Interchange Format
- HDF4 – Hierarchical Data Format
- JPEG – Joint Photographic Experts Group
- JPEG 2000 – Joint Photographic Experts Group 2000
- PBM – Portable Bitmap
- PCX – Windows Paintbrush
- PGM – Portable Graymap
- PNG – ポータブル ネットワーク グラフ
- PNM – Portable Any Map
- PPM – Portable Pixmap
- RAS – Sun ラスター
- TIFF – Tagged Image File Format
- XWD – X Window Dump

詳細 : <http://www.mathworks.co.jp/jp/help/matlab/ref/imwrite.html>

## 2.1.1 バイナリ (Raw) データの読み書き

```
% バイナリデータの読み込み
fid = fopen('ファイル名', 'r', 'b'); % ファイルを開く
G = (fread(fid, [198, 135], '*uint16', 'b'))'; % ファイルの読み込み・転置
figure; imshow(G); % 表示
fclose(fid); % ファイルを閉じる
```

ディスク上のファイルの一部または全体を、アプリケーションのアドレス空間内にマッピングします。これにより、動的メモリへのアクセスと同様にディスク上のファイルにアクセスできるようになります。fread や fwrite などのIO関数を使用する場合に比べ、ファイルの読み取りと書き込みが高速化します。

```
% メモリマッピングを使った読み込み
% 大規模ファイルにランダムアクセスする場合や小さなファイルに頻繁にアクセスする場合等にも
m = memmapfile('ファイル名')
m.Format = 'uint16' % Endianは、os固有のものを使用 (Windows : Little)

% EndianをLittleからBigに変更 (m.Repeat = Infなので、全データを読み込み)
I1 = mod(m.Data, 256) * 256 + (m.Data/256);
figure; imshow(reshape(I1, 198, 135)); % 表示
```

```
% バイナリデータの書き込み
fid = fopen('ファイル名', 'w', 'b'); % ファイルを開く
fwrite(fid, g, 'uint16', 'b'); % ビッグエンディアンで書出し
fclose(fid); % ファイルを閉じる
```

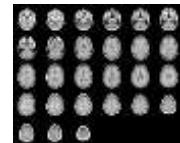
## 2.1.2 画像表示・調査用各種ツール

### 画像の表示・調査用の関数

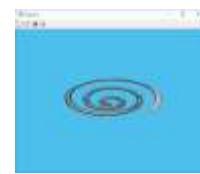
**imshow()** イメージの表示 (MATLAB基本関数) R2014b

**imshowpair()** イメージ間の差異の比較、横に並べて表示 (合成は `imfuse()`)

**montage()** 複数のイメージフレームを四角形モンタージュとして表示 (異なる画像サイズサポート R2018a)



**warp()** テクスチャ マッピングされたイメージの表示



**volshow()** 3次元ボリュームデータの表示 R2018b



**imtool()** 画像ビューアー アプリケーション

**colorThresholder()** 色の閾値 アプリケーション

**imcontrast()** コントラスト調整ツール

**imcrop()** 画像のトリミング (3次元対応: `imcrop3()`) R2019b

**imdilate()** 距離ツール

**getpts()** マウスを使用した、ピクセル位置の指定

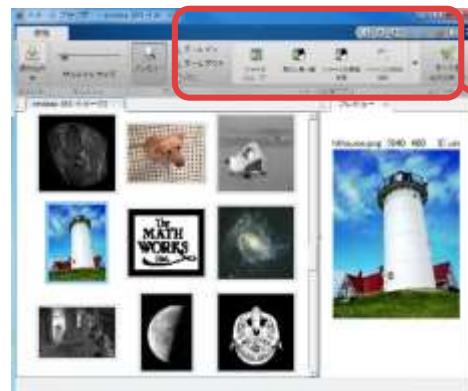
⋮

```
A = imread('peppers.png')
figure; imshow(A)
% Figureを一つ開き、画像を表示
```

## 2.1.2 画像・ボリュームデータの表示・調査用ツール



% 画像ビューアー アプリケーション  
`imtool (A)`



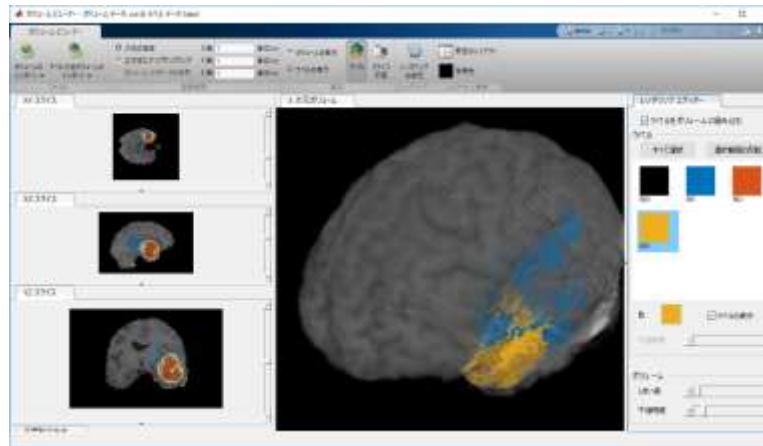
異なるサイズ、データ型の画像を一覧表示 R2016b

% イメージブラウザ  
% (異なるサイズ、データ型の画像を一覧表示)  
`imageBrowser ()`



他のアプリケーションを起動

## 2.1.2 画像・ボリュームデータの表示・調査用ツール



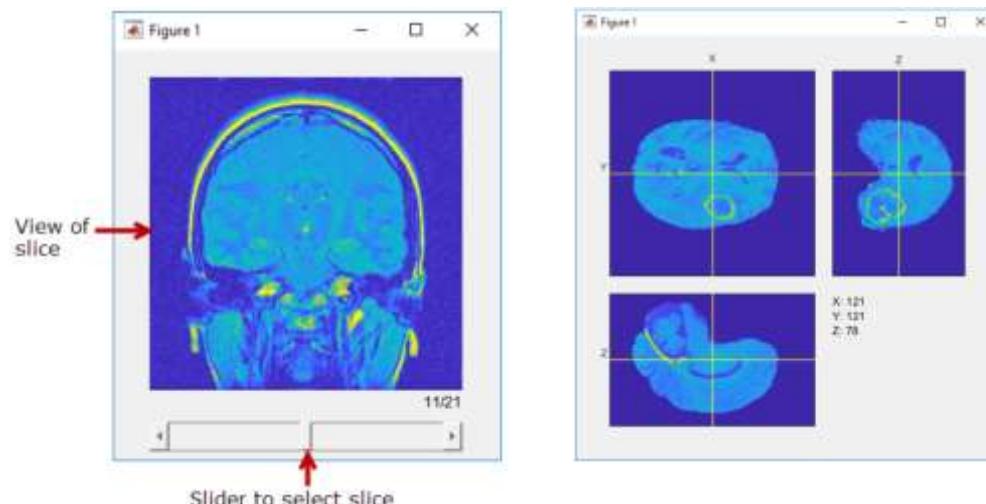
3次元ボリュームデータの可視化

R2017a

% ボリュームビューアー  
`volumeViewer()`



- ラベルボリューム表示機能 R2019a
  - オリジナルの輝度ボリュームとのオーバーレイ表示も可能
- 輝度ボリュームの表示にColormapの指定が可能
- ビューの状態を構造体にエクスポート R2019b



3次元ボリューム断面の可視化

R2019b

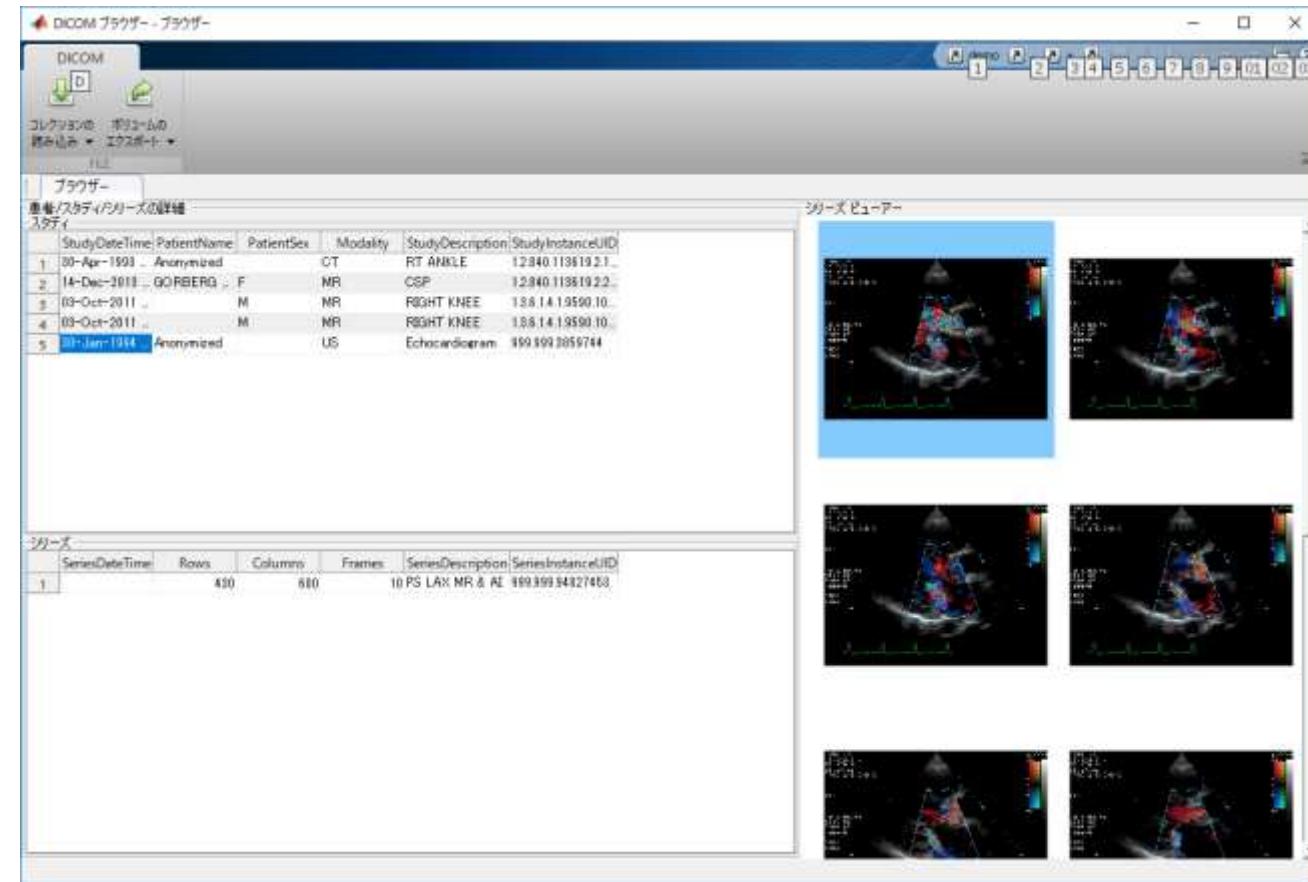
% スライスピューアー  
`sliceViewer()`  
`orthosliceViewer()`

- 表示のColormap指定が可能
- ボリュームのx,y,z方向のスケールを指定可能

## 2.1.2 医用画像の確認：DICOM/NIfTI画像

R2017b

- DICOMブラウザ R2017b
    - DICOMファイルのプレビュー
    - DICOMDIRファイルの読み取り
    - 各種ビューワーへのエクスポート機能
      - ワークスペースへエクスポート
      - ボリュームビューワーで表示
      - ビデオビューワーで表示
    - 大量のDICOMファイルの確認に便利
  
  - NIfTI画像の読み取り R2017b
    - `v = niftiread('brain.nii');`
- NIfTI-2ファイルフォーマットをサポート R2019a



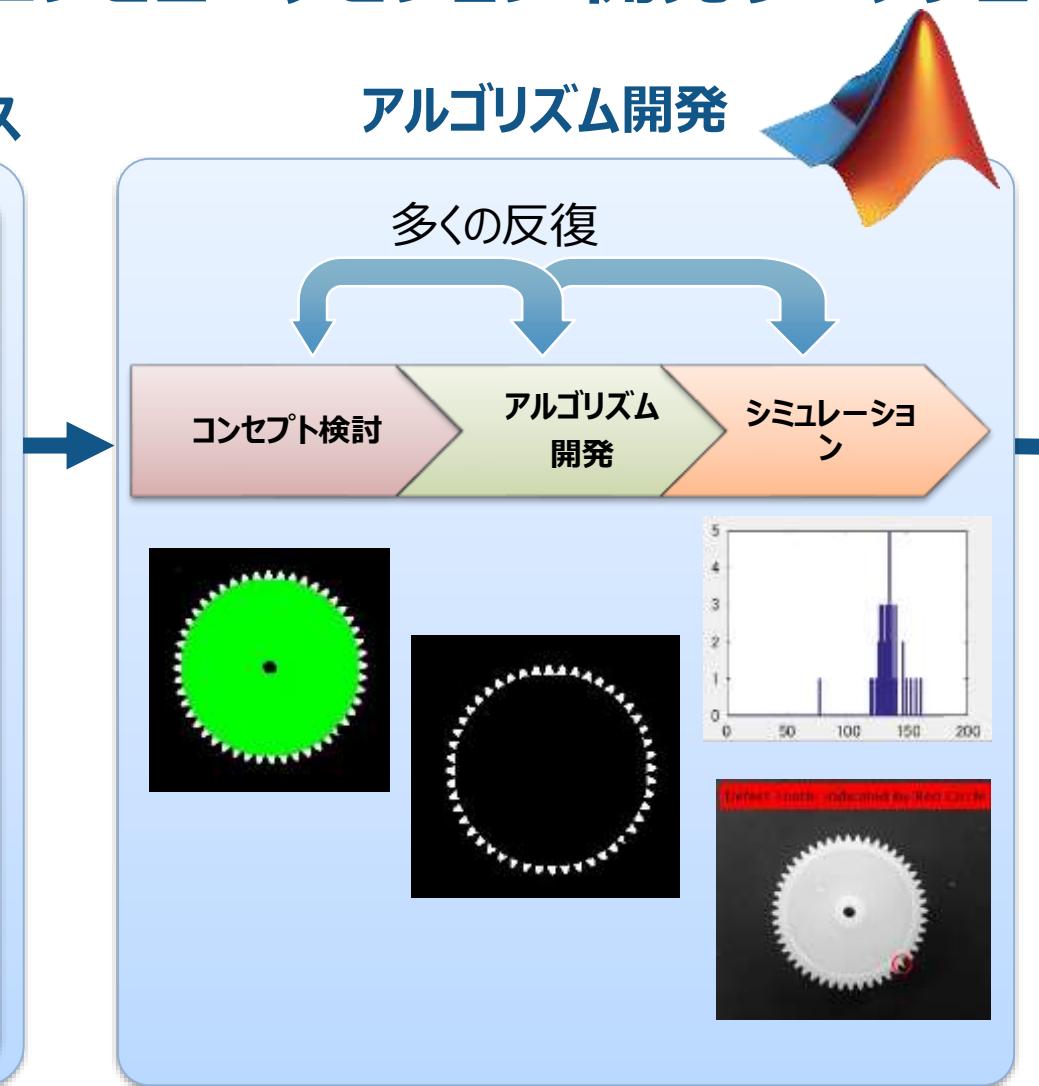
>> dicomBrowser R2017b

## 2.1.3 画像処理・コンピュータビジョン 開発ワークフロー

### 画像へのアクセス



### アルゴリズム開発



### 結果の共有/IP化



効率よいアルゴリズム開発のための、数百の関数群や各種ツール

## 2.1.3 静止画 画像処理・解析のワークフロー

%% 画像読み込み

```
A = imread('peppers.png');
```



各種画像処理 ・ 解析



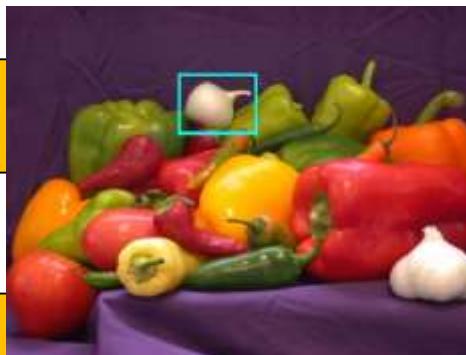
%% 結果の挿入 (例)

```
A1 = insertShape(A, 'Rectangle', loc, 'Color', 'cyan', 'LineWidth', 3);
```



%% 結果画像の表示

```
figure; imshow(A1); % Figureを一つ開き、画像を表示
```



%% 結果の書き出し

```
imwrite(A1, 'result.png');
```

## 2.1.4 グラフィックス

Computer Vision Toolbox

### 描画

#### 関数

`insertMarker()`

マーク挿入

`insertObjectAnnotation()`

注釈挿入

`insertShape()`

図形挿入

`insertText()`

テキスト挿入

R2015b  
日本語対応

#### システムオブジェクト

`vision.AlphaBlender()`

画像のブレンド

`vision.ImagePadder()`

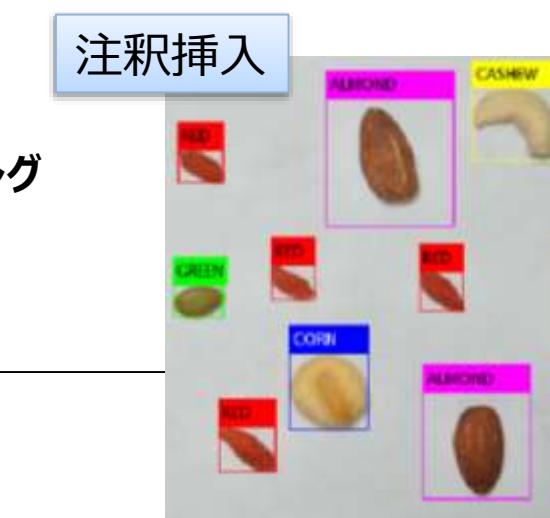
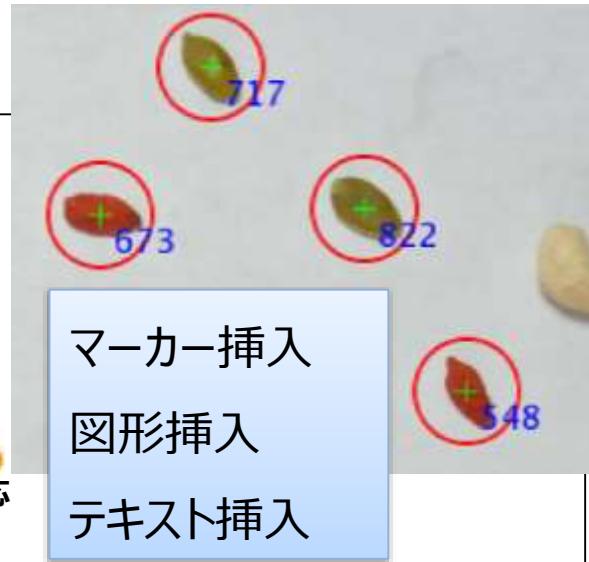
画像の切り取り、パディング

`vision.MarkerInserter()`

マーク挿入

`vision.ShapeInserter()`

図形挿入



## 2.1.5 動画 画像処理・解析のワークフロー：高速ストリーミング処理

Computer Vision Toolbox

```
%% 動画の読み込み・表示・書き出し 用のシステムオブジェクトの生成  
vidReader = vision.VideoFileReader('visiontraffic.avi');  
vidPlayer = vision.DeployableVideoPlayer;  
vidWriter = vision.VideoFileWriter('myFile.avi');
```



```
%% 1フレームずつ順に処理  
while (1)  
    I = step(vidReader); % 1フレーム 読込み
```



```
step(vidPlayer, I); % 1フレーム 表示  
step(vidWriter, I); % 1フレーム 書出し  
end
```

```
%% 生成したシステムオブジェクトをリリース  
release(vidReader);  
release(vidPlayer);  
release(vidWriter);
```

## 2.1.6 高速動画ストリーミング処理

Computer Vision Toolbox

### 動画入出力

`vision.VideoFileReader()`

動画ファイルの読み込み（音声も可）

`vision.VideoPlayer()`

動画表示

`vision.DeployableVideoPlayer()`

動画表示（Windows用Cコード生成対応）

更に高速化 **R2015a**

`vision.VideoFileWriter()`

動画ファイルの書き出し（音声も可）

`vision.BinaryFileReader()`

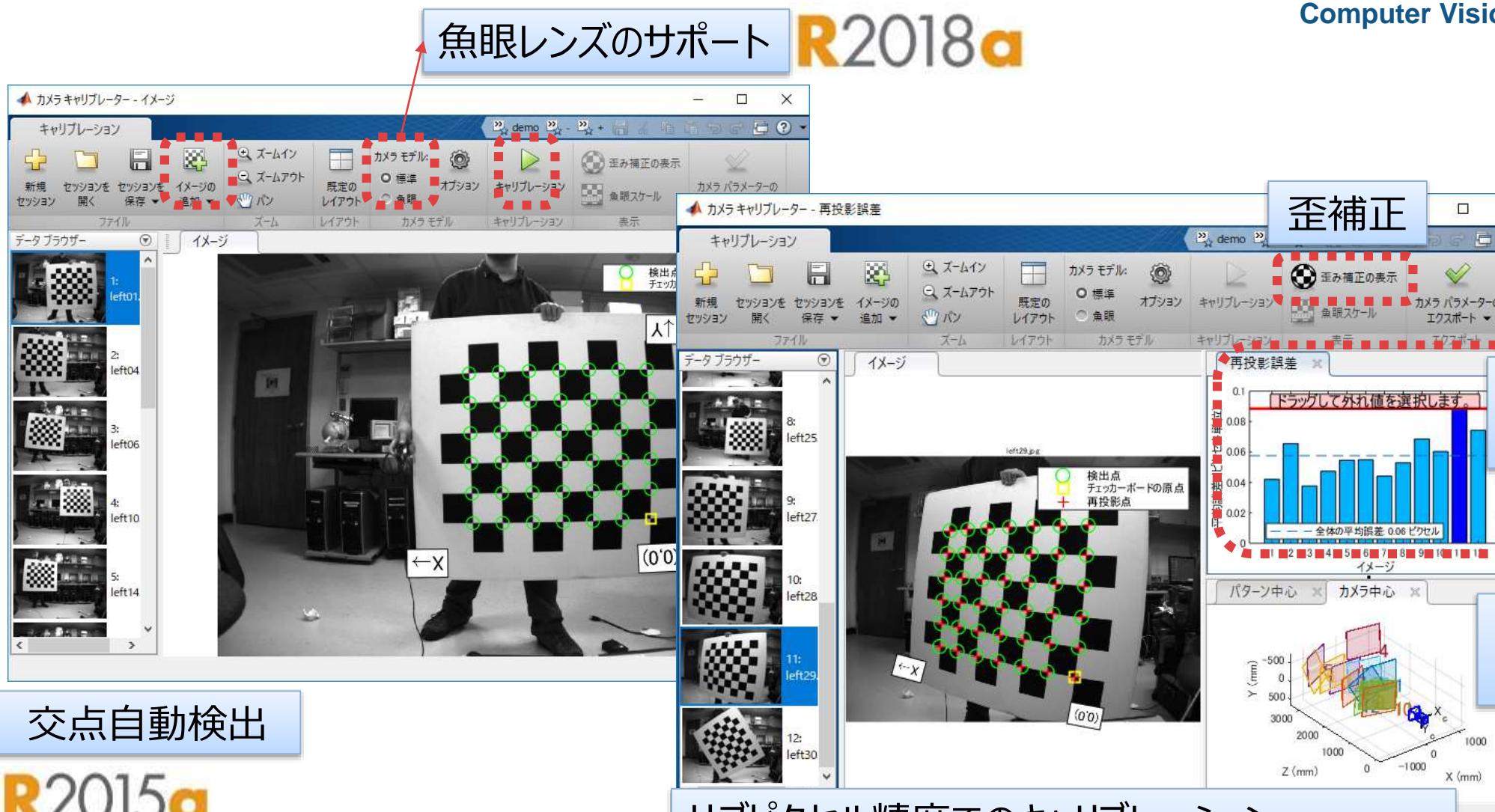
バイナリ動画ファイルの読み込み

`vision.BinaryFileWriter()`

バイナリ動画ファイルの書き出し

## 2.2.1 カメラキャリブレーション：アプリケーション

Computer Vision Toolbox



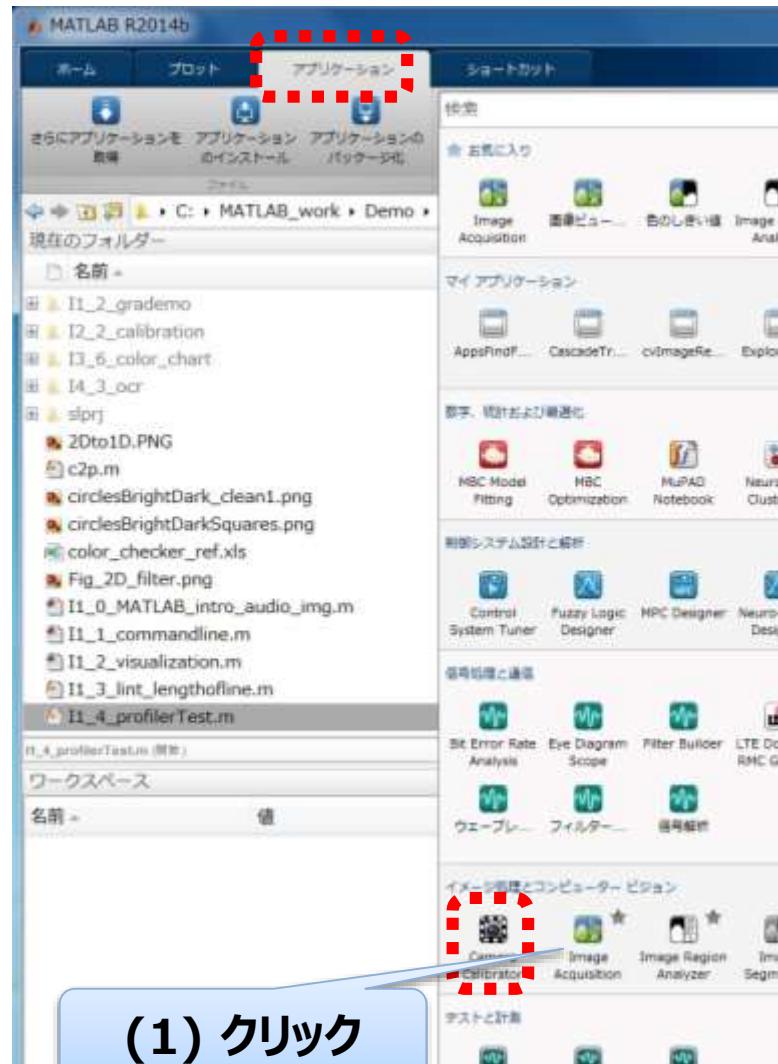
R2015a

処理速度の高速化

交点自動検出

サブピクセル精度でのキャリブレーション  
レンズ歪、カメラ内部・外部パラメータの抽出

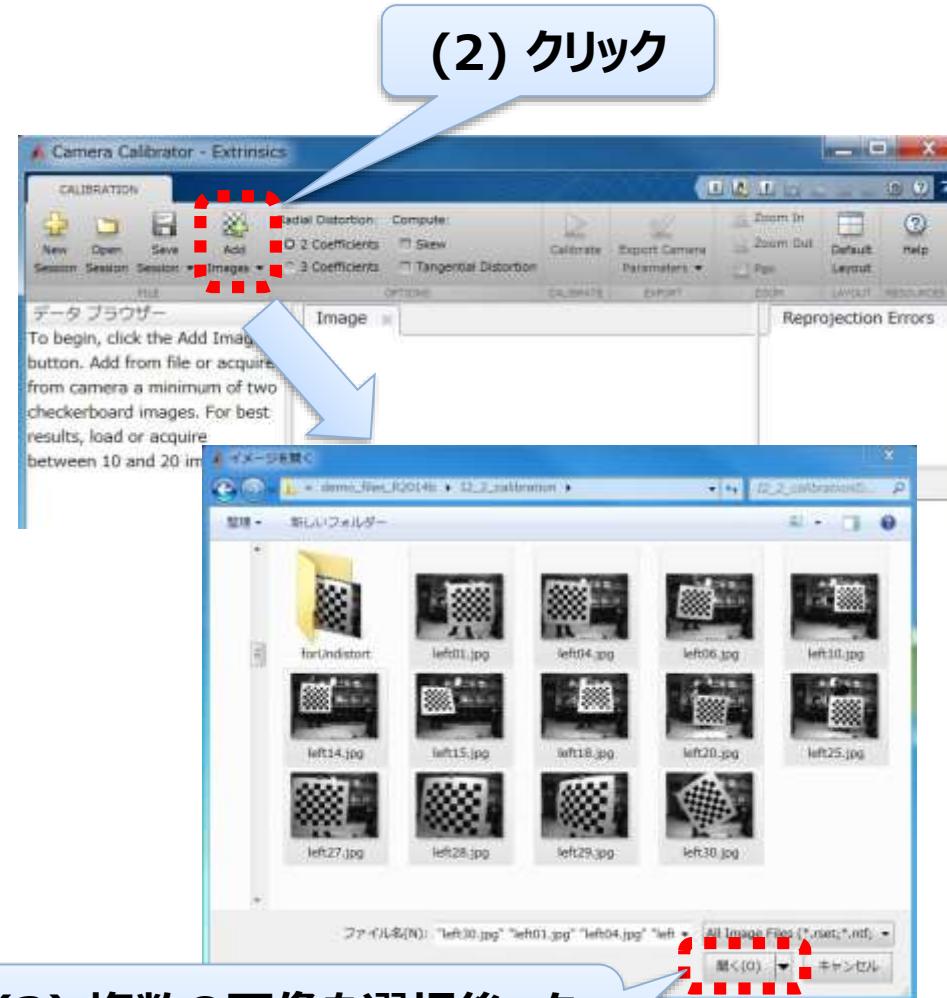
## 2.2.1 カメラキャリブレーション：手順（1）



(1) クリック

[キャリブレーション用画像準備の詳細]

<http://jp.mathworks.com/help/vision/ug/single-camera-calibrator-app.html#bt19jq-1>



(2) クリック

(3) 複数の画像を選択後、クリック（画像取込）

## 2.2.1 カメラキャリブレーション：手順（2）

(4) 定規で測定した、  
チェックバー一辺の長さを入  
力し、OKクリック

適していない画像は  
自動的に除外される

(5) OKクリック

(6) クリック

完了

最投影誤差が大きい  
画像は、右クリックし  
"削除して再実行"

## 2.2.2 カメラキャリブレーション： スクリプトでの実行

R2013b

Computer Vision Toolbox

```
% 画像ファイル名の指定
for i = [1:13]
    imFileNames{i} = fullfile('I2_3_calibration', sprintf('left%02d.jpg', i));
end

%% 画像内のチェックバードのパターンの検出 (不適切な画像は自動的に除去)
[imagePoints, boardSize, imagesUsed] = detectCheckerboardPoints(imFileNames);

%% コーナー点の 実世界での位置(world coordinates) を計算
squareSize = 150; % 単位:mm
worldPoints = generateCheckerboardPoints(boardSize, squareSize);

%% カメラパラメータの推定
cameraParams = estimateCameraParameters(imagePoints, worldPoints);
```

非常に簡潔なコードで、カメラキャリブレーションを実現

## 2.2.3 レンズ歪の補正

Computer Vision Toolbox

### レンズ歪補正用の関数

`undistortImage()`

画像の、レンズ歪の補正

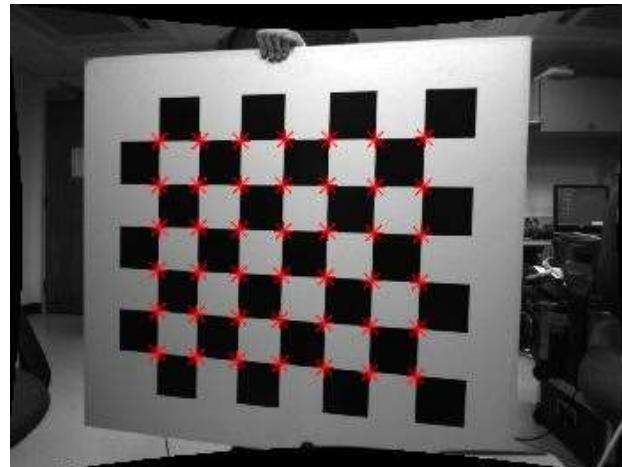
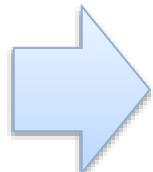
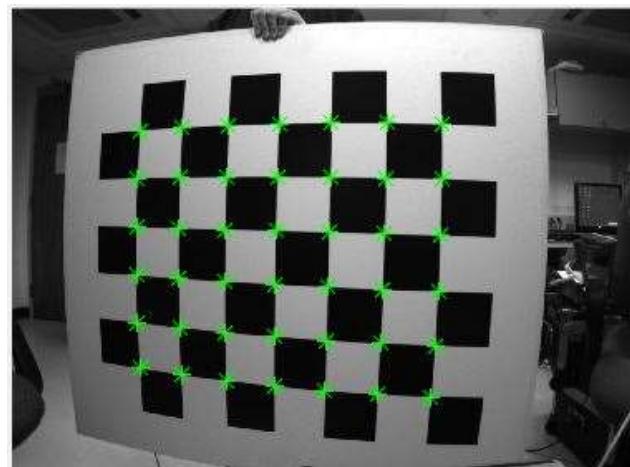
`undistortPoints()`

任意の点座標の、レンズ歪の補正

```
% 画像に対して、レンズ歪補正 (OutputViewがfullの場合は、端が切れないように中心座標は移動)
[undistI, newOrigin] = undistortImage(origI, cameraParams, 'OutputView', 'full');

% 点座標に対して、レンズ歪補正
undistPoints = undistortPoints(points, cameraParams);

% 補正後の点座標を、補正後画像へアライメント
undistPoints = [undistPoints(:,1)-newOrigin(1), undistPoints(:,2)-newOrigin(2)];
```

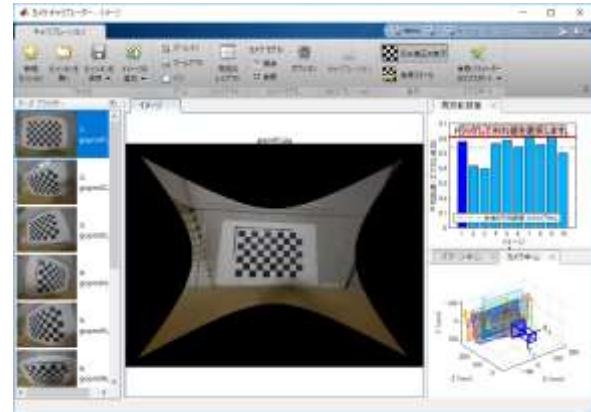


## 2.2.4 魚眼レンズの歪補正

### 魚眼レンズの歪補正用の関数

<code>fisheyeParameters</code>	魚眼レンズのカメラパラメータ用オブジェクト
<code>fisheyeIntrinsics</code>	魚眼レンズのカメラ内部パラメータ用オブジェクト
<code>estimateFisheyeParameters ()</code>	魚眼レンズのカメラパラメータの推定
<code>undistortFisheyeImage ()</code>	画像の、レンズ歪の補正
<code>undistortFisheyePoints ()</code>	任意の点座標の、レンズ歪の補正

R2017b  
Computer Vision Toolbox



カメラキャリブレーターアプリケーションでの  
魚眼レンズキャリブレーション対応 R2018a



原画像

レンズ  
歪  
補  
正



Full View  
(周囲の切り取りなし)

## 2.3 イメージタイプ変換・コントラスト調整・階調変更

### イメージタイプ変換・コントラスト調整・量子化用の関数

`rgb2gray()`      RGBイメージまたはカラーマップをグレースケールに変換 (MATLAB基本関数)

`label2rgb()`      各領域のラベル番号 (画素値) 毎に別の色の画像 (ラベル画像) へ変換

`imhist()`      イメージデータのヒストグラム表示 (イメージタイプにより、BIN数を調整)

`imcontrast()`      コントラスト調整ツール

`imadjust()`      イメージの強度値またはカラーマップの調整

`histeq()`      ヒストグラム均等化を用いたコントラストの強調

`adapthisteq()`      コントラストに制限を付けた適応ヒストグラム均等化を実行 (CLAHE)

`imhistmatch`      イメージのヒストグラムを、参照イメージのヒストグラムと一致させる

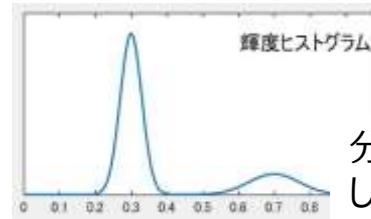
`multithresh()`      大津法による、複数レベル イメージしきい値計算

`imquantize()`      指定された量子化レベルと出力値によるイメージの量子化

⋮

## 2.3.1 二値化・適応二値化

- Otsu法（判別分析法）によるしきい値計算



分離度を最大にする  
しきい値を決定

```
t = graythresh(G);
```

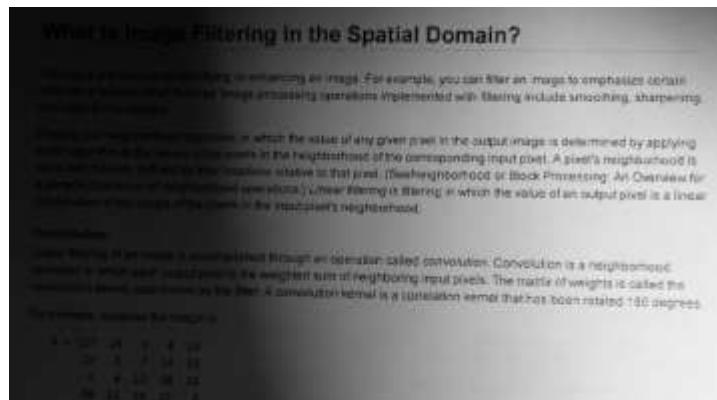
% Otsu法により、大局的しきい値を計算

```
BW = imbinarize(G);
```

% Otsu法により、画像を二値化

R2016a

- 適応しきい値計算 (近傍の輝度平均を用い、画素単位でしきい値を計算)



適応2値化

R2016a

**What Is Image Filtering in the Spatial Domain?**

Filtering is a technique for modifying or enhancing an image. For example, you can filter an image to emphasize certain features or remove other features. Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement.

Filtering is a neighborhood operation, in which the value of any given pixel in the output image is determined by applying some algorithm to the values of the pixels in the neighborhood of the corresponding input pixel. A pixel's neighborhood is the same set of pixels, defined by their locations relative to that pixel. (See Neighborhood or Block Processing: An Overview for a general discussion of neighborhood operations.) Linear filtering is filtering in which the value of an output pixel is a linear combination of the values of the pixels in the input pixel's neighborhood.

### Convolution

Linear filtering of an image is accomplished through an operation called convolution. Convolution is a neighborhood operation in which each output pixel is the weighted sum of neighboring input pixels. The matrix of weights is called the convolution kernel, also known as the filter. A convolution kernel is a correlation kernel that has been rotated 180 degrees.

For example, suppose the image is

12	24	1	8	15
23	5	7	24	16
4	6	13	20	22
18	12	19	21	3

```
T = adaptthresh(G);
BW = imbinarize(I, T);
```

% 適応しきい値計算 (画素単位のしきい値)  
% 指定されたしきい値を用い、画像を二値化

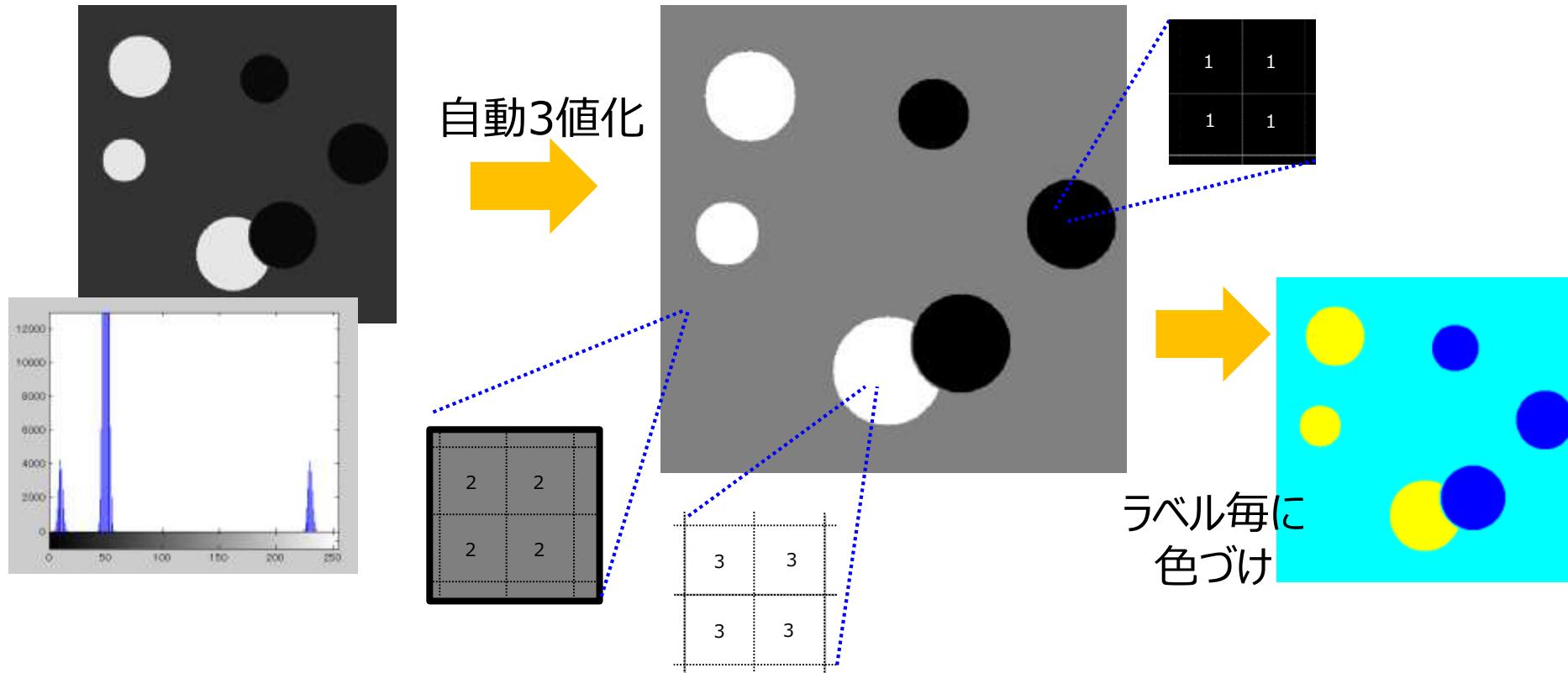
R2016a

```
BW = imbinarize(G, 'adaptive');
```

% 適応しきい値計算により、画像を二値化

R2016a

## 2.3.2 イメージタイプ変換・コントラスト調整・階調変更

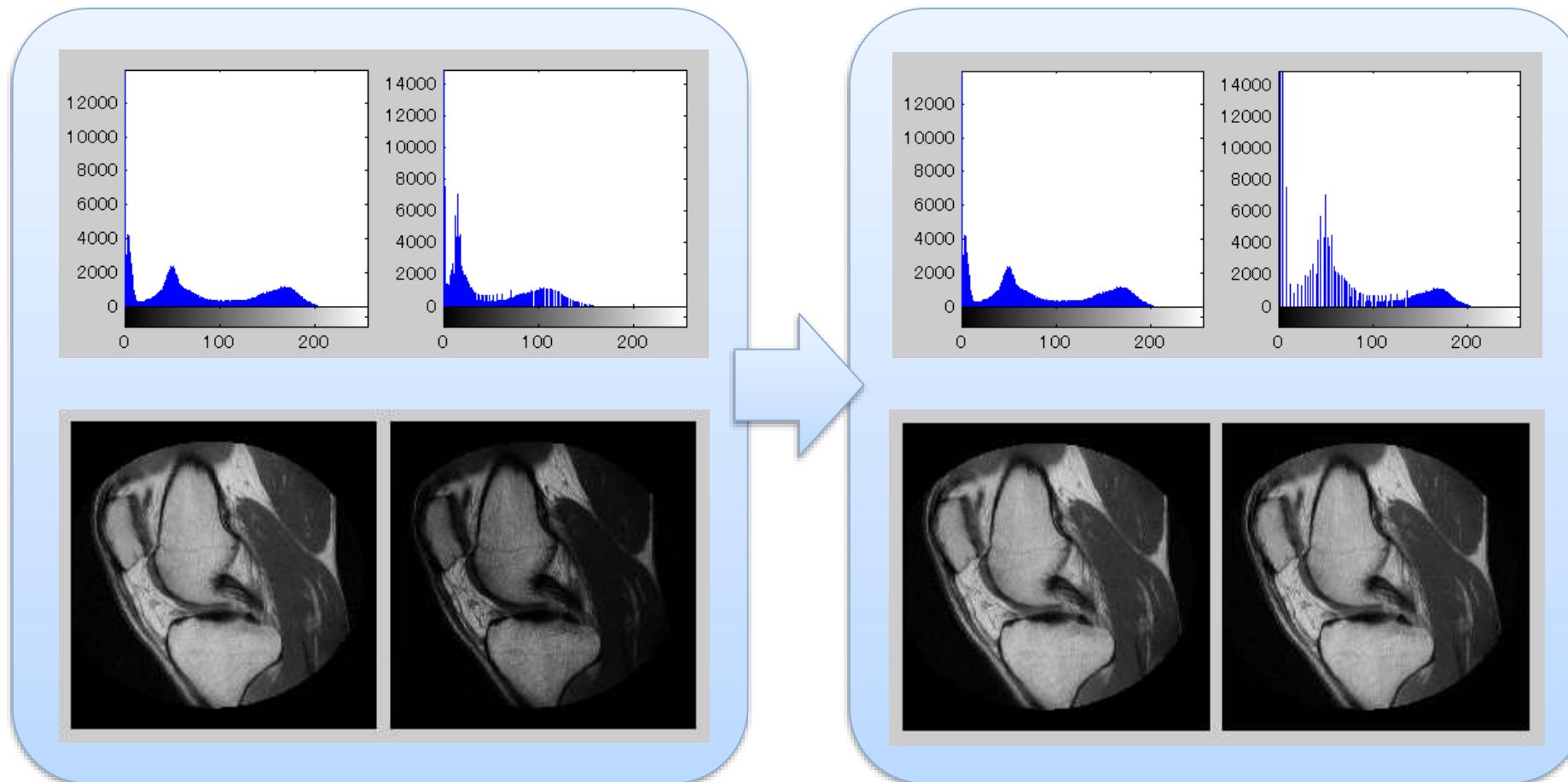


```
I = imread('circlesBrightDark_clean.png');

thresh = multithresh(I,2);      % Otsu法により、3値化用のしきい値を計算
seg_I = imquantize(I,thresh);   % 得られたしきい値により、画素値を量子化(1,2,3)

RGB = label2rgb(seg_I);         % 異なるラベル番号(画素値)を異なる色へ
imshowpair(I,RGB,'montage');    % 画像表示
```

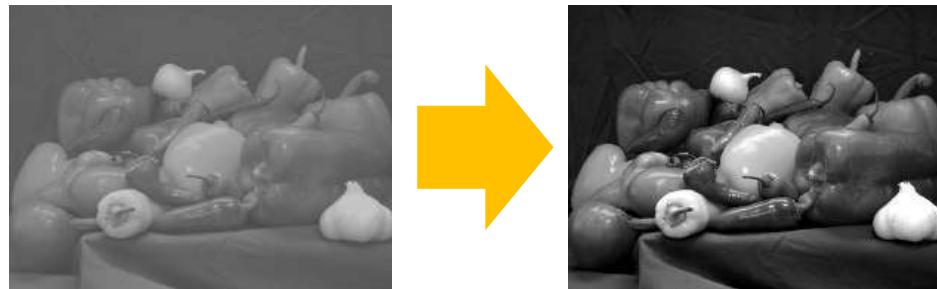
## 2.3.3 ヒストグラム（コントラスト）のマッチング



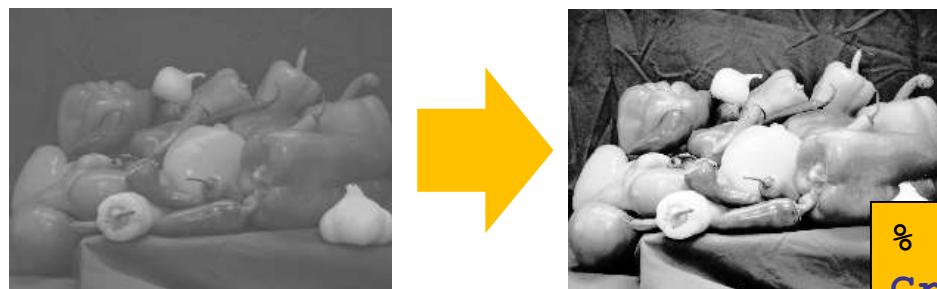
`B = imhistmatch(A, Ref, 256); % 画像Aのヒストグラムを画像Refに一致させた画像を生成`

3次元以上の任意の次元数に対応 R2017a

## 2.3.4 各種コントラスト調整

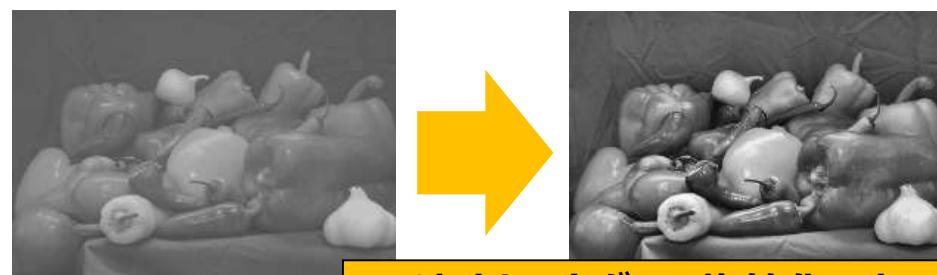


% 低・高輝度側で1%飽和するよう自動調整  
Gray1 = imadjust(Gray);



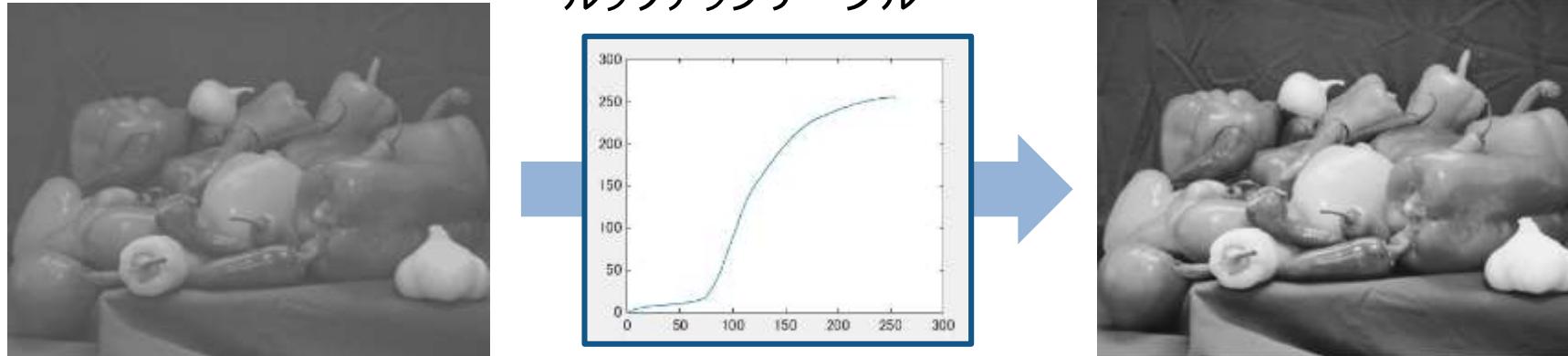
% ヒストグラム均等化を用いたコントラストの強調  
Gray2 = histeq(Gray, 256);

3次元以上の任意の次元数に対応 R2017a



% 適応ヒストグラム均等化によるコントラストの強調  
Gray5 = adapthisteq(Gray, 'Distribution', 'exponential');

## 2.3.5 ルックアップテーブルによる任意特性のコントラスト調整

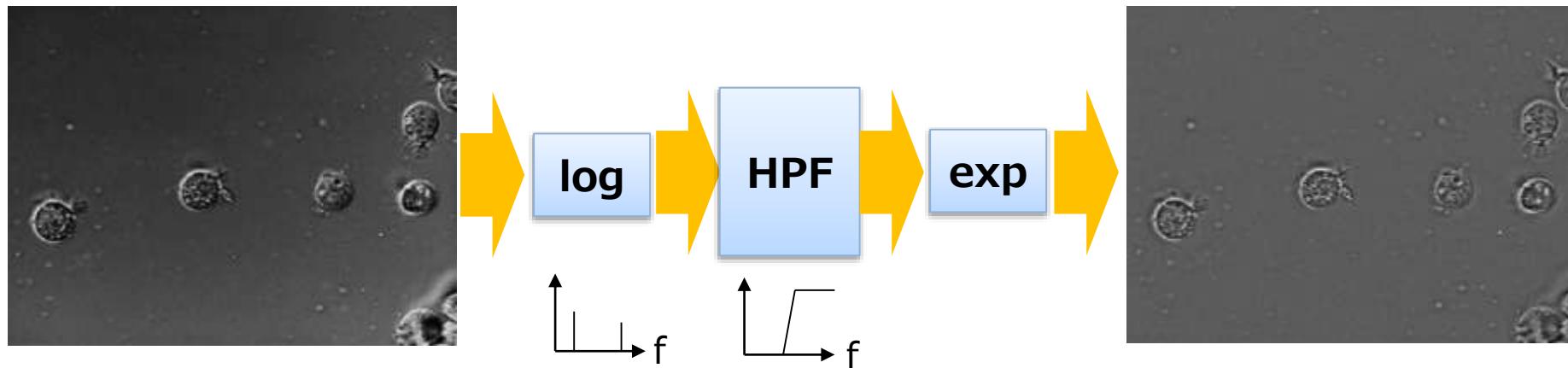


```
% ルックアップテーブルの作成
in = [0 1 50 100 200 255];
out = [0 1 15 150 250 255];
LUT = uint8(interp1(in, out, 0:255, 'pchip'));

% ルックアップテーブル適用
B = intlut(A, LUT);
```

## 2.3.6 Homomorphic Filteringによる照度不均一の除去

$I1 = I .* R$  の形式で画像が影響を受けている場合対数変換をすることで  
 $\ln(I1) = \ln(I) + \ln(R)$  の和の形式に変換



% 自然対数変換

```
I1 = log(I + 1);
```

% ハイパスフィルタ処理 : hHPFはフィルタ係数

```
I1f = imfilter(I1, hHPF, 'replicate');
```

% exp関数で、対数変換を戻す

```
If = exp(I1f) - 1;
```

詳細は

<http://blogs.mathworks.com/steve/2013/06/25/homomorphic-filtering-part-1>

<http://blogs.mathworks.com/steve/2013/07/10/homomorphic-filtering-part-2>

## 2.3.7 HDR(ハイダイナミックレンジ)画像の取り扱い

### HDR(ハイダイナミックレンジ)画像用の関数

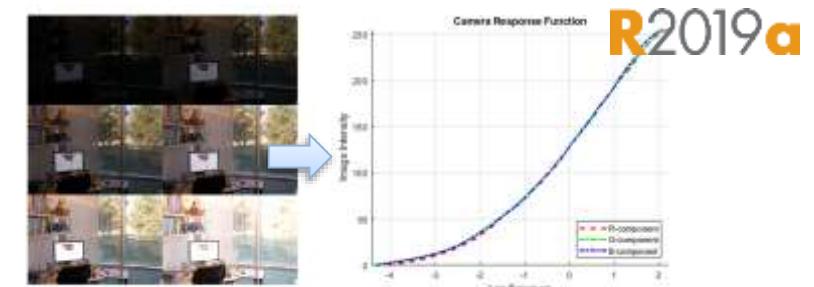
<code>makehdr</code>	HDR画像生成
<code>tonemap</code>	可視化のためにHDR画像をレンダリング
<code>tonemapfarbman</code>	HDR画像をエッジ保存マルチスケールデコンポジションで通常画像に変換 <b>R2018b</b>
<code>localtonemap</code>	局所的なコントラストを強調した可視化画像をHDR画像から生成
<code>blendexposure</code>	複数露光画像からのHDR画像合成 <b>R2018a</b>
<code>camresponse</code>	複数露光画像からのカメラ応答関数(CRF)を推定、画像のEXIFの露光時間を使用 <b>R2019a</b>

複数枚の画像からコントラストが融合された画像を作成



```
I1 = imread('car_1.jpg'); I2 = imread('car_2.jpg');
I3 = imread('car_3.jpg'); I4 = imread('car_4.jpg');
E = blendexposure(I1,I2,I3,I4);
```

複数露光画像からカメラ応答関数(CRF)を推定



```
files = [ "office_1.jpg", ...
"office_2.jpg", "office_3.jpg", ...
"office_4.jpg", "office_5.jpg", ...
"office_6.jpg" ];
crf = camresponse(files);
```

## 2.3.8 低光量画像の明るさ調整



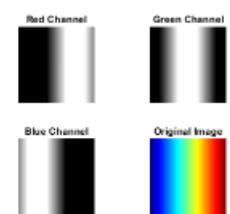
```
% 低光量画像の明るさ調整  
B = imlocalbrighten(A);
```

## 2.4 色空間変換

### 色空間変換用の関数

<code>rgb2ycbcr()</code>	RGB画像を YCbCr色空間へ変換	
<code>ycbcr2rgb()</code>	YCbCr画像を RGB色空間へ変換	
<code>rgb2HSV()</code>	RGB画像を HSV色空間へ変換	(MATLAB基本関数)
<code>HSV2RGB()</code>	HSV画像を RGB色空間へ変換	(MATLAB基本関数)
<code>RGB2LAB()</code>	RGB画像を L*a*b*色空間へ変換	
<code>RGB2XYZ()</code>	RGB画像を XYZ色空間へ変換	
⋮	⋮	
<code>ICCREAD()</code>	ICCプロファイルの読み取り	
<code>MAKECFORM()</code>	色変換構造体を作成 (CMYK/uvL/xyY 等の色空間や、変換元・先のICCプロファイルを指定)	
<code>APPLYCFORM()</code>	色空間変換を適用	
<code>IMSPLIT()</code>	色のチャネルごとに分割 <b>R2018b</b>	
<code>RGB2LIGHTNESS()</code>	RGB画像CIE 1976 L*a*b*色空間の輝度画像L*を生成 <b>R2019a</b>	

**R2014b**



```

RGB = imread('peppers.png');
HSV = rgb2HSV(RGB); % RGB画像を HSV色空間へ変換
cform = makecform('srgb2cmyk'); % 色変換構造体を作成
LAB = applycform(RGB,cform); % sRGB画像を CMYK色空間へ変換

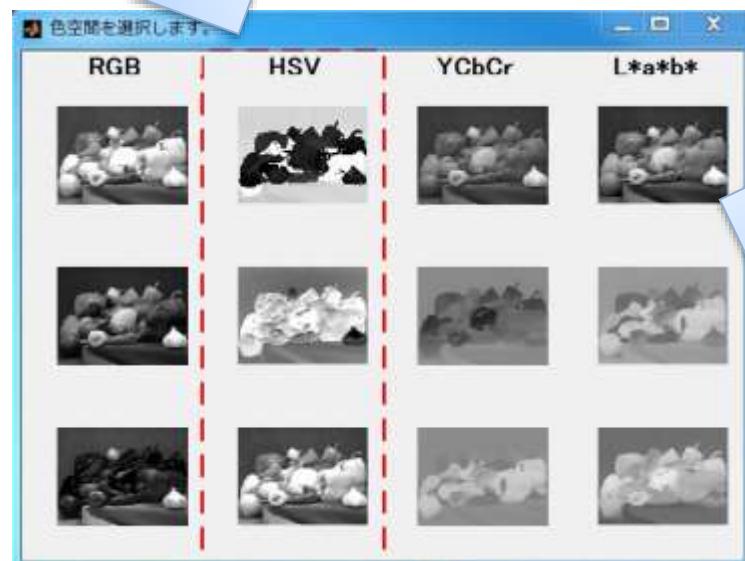
```

## 2.4.1 色による閾値処理のアプリケーション

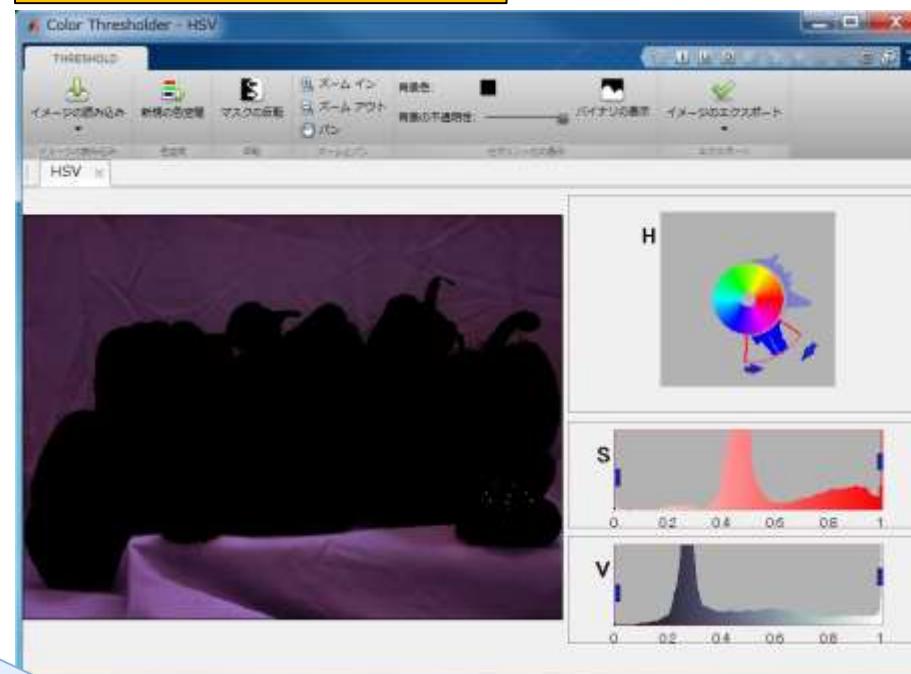
R2014a



色空間の変換



色の閾値 アプリケーション  
>> colorThresholder



閾値探索

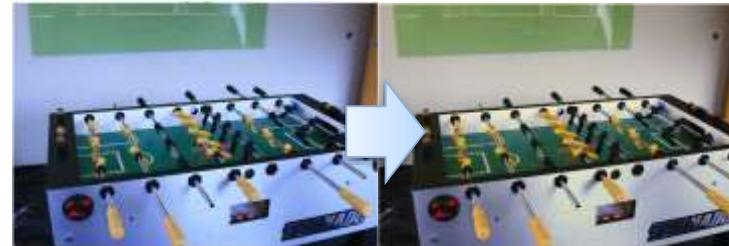
```
a = imread('peppers.png');  
colorThresholder(a);
```

## 2.4.2 カラーバランス/ホワイトバランス/ガンマ補正

R2017b

- カラーバランスの調整
  - `chromadapt`
- ホワイトバランスの調整
  - `illumgray` (グレイワールド)
  - `illumPCA` (主成分分析)
  - `illumwhite` (White Patch Retinex)
- 霧の除去
  - `imreducehaze`
- ガンマ補正/逆ガンマ補正
  - `lin2rgb`, `rgb2lin`

カラーバランス調整(`chromadapt`)



ホワイトバランス調整(`illumgray`)

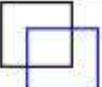
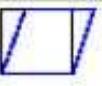


霧の除去(`imreducehaze`)



## 2.5 幾何学的変換

$$[x \ y \ 1] = [u \ v \ 1] \times T$$

アフィン変換	例	変換行列	
平行移動		$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$	$t_x$ は $x$ 軸に沿って移動を指定します。 $t_y$ は $y$ 軸に沿って移動を指定します。
スケール		$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$s_x$ は $x$ 軸に沿って倍率を指定します。 $s_y$ は $y$ 軸に沿って倍率を指定します。
せん断		$\begin{bmatrix} 1 & sh_y & 0 \\ sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$sh_x$ は $x$ 軸に沿ってせん断係数を指定し(時計の回転方向) $sh_y$ は $y$ 軸に沿ってせん断係数を指定します。
回転		$\begin{bmatrix} \cos(q) & \sin(q) & 0 \\ -\sin(q) & \cos(q) & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$q$ は回転の角度を指定します。

## 2.5.1 幾何学的変換（射影法等による空間変換）

<code>imcrop()</code>	画像のトリミング
<code>imresize()</code>	画像のサイズ変換
<code>imresize3()</code>	ボリュームデータのサイズ変換 <b>R2017a</b>
<code>imrotate()</code>	画像の中心の周りに回転
<code>imrotate3()</code>	ボリュームデータの回転 <b>R2017a</b>
<code>imtranslate()</code>	画像の平行移動 <b>R2014a</b>
<code>fitgeotrans()</code>	空間変換構造体 ( <code>tform</code> ) を作成
<code>imwarp()</code>	イメージへの幾何学変換の適用

```
T = fitgeotrans(Porig, Ppost, 'projective');  
registered = imwarp(I, T, 'OutputView', Ri); % 幾何学的変換  
figure; imshow(registered); % 画像表示
```



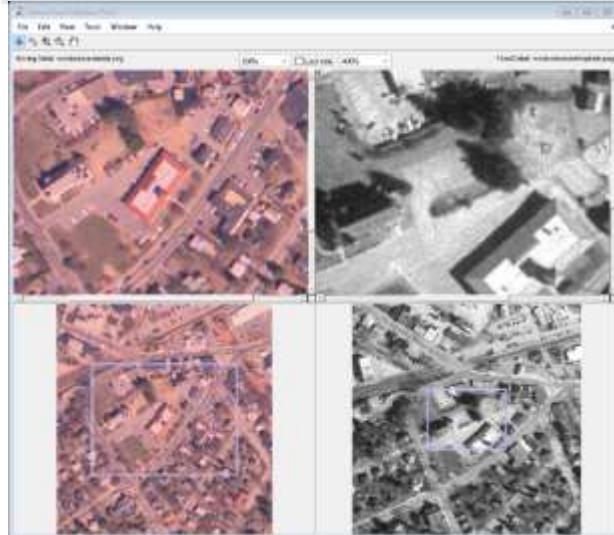
道路の画像を  
真上から見た画像に変換  
(文字、標識認識などへの応用)

変換して抽出



## 2.5.2 幾何学的変換（コントロールポイントツールを用いた位置合せ）

コントロールポイントの選択ツール



```
input_img = imread('westconcordaerial.png');
base_img = imread('westconcordorthophoto.png');

% コントロールポイント選択ツールを起動
cpselect(input_img,base_img)

T=cp2tform(input_points,base_points,'projective');
registered = imtransform(input_img,T);
```

重ね合わせた  
2つのイメージ

変換

コントロールポイントの選択



## 2.5.3 幾何学的変換（カスタム変換による画像変換）

```
%ユーザ定義の写像を定義
T = maketform('custom',2,2,@pex006,@ipex006,[]);
udata = [-pi pi]; % 入力画像に対するX軸の範囲
vdata = [0 90]; % 入力画像に対するY軸の範囲
xdata = [-90 90]; % 出力画像に対するX軸の範囲
ydata = [-90 90]; % 出力画像に対するY軸の範囲

b = imtransform(a,T,'cubic','UData',udata,...
    'VData',vdata,'XData',xdata,'YData',ydata, ...
    'Size',[91 360],'FillValues',255);
```

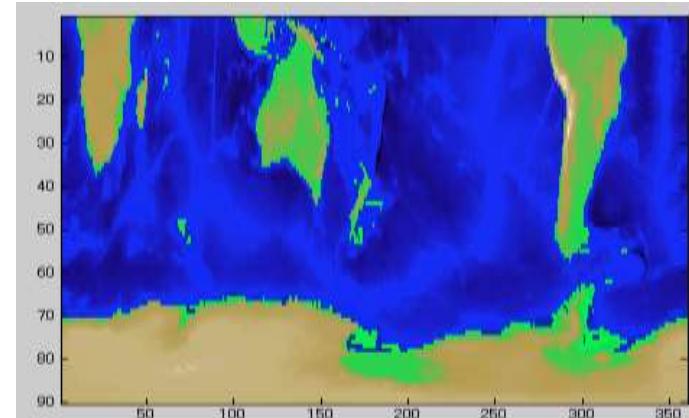
```
function X = pex006( U, t )
[th,r] = cart2pol(U(:,1),U(:,2));
X(:,1) = th;
X(:,2) = r;
```

写像  $f$  : 正規座標  $\Rightarrow$  極座標

```
function X = ipex006( U, t )
[x,y] = pol2cart(U(:,1),U(:,2));
X(:,1) = x;
X(:,2) = y;
```

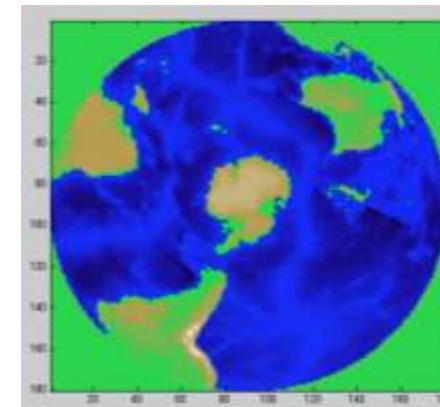
逆写像  $f^{-1}$  : 極座標  $\Rightarrow$  正規座標

正規座標



逆写像  $f^{-1}$

写像  $f$



極座標

## 2.6 自動レジストレーション（位置合せ）

### 自動レジストレーション用の関数

`imregtform()` 強度ベースのレジストレーションの幾何学的変換行列の推定

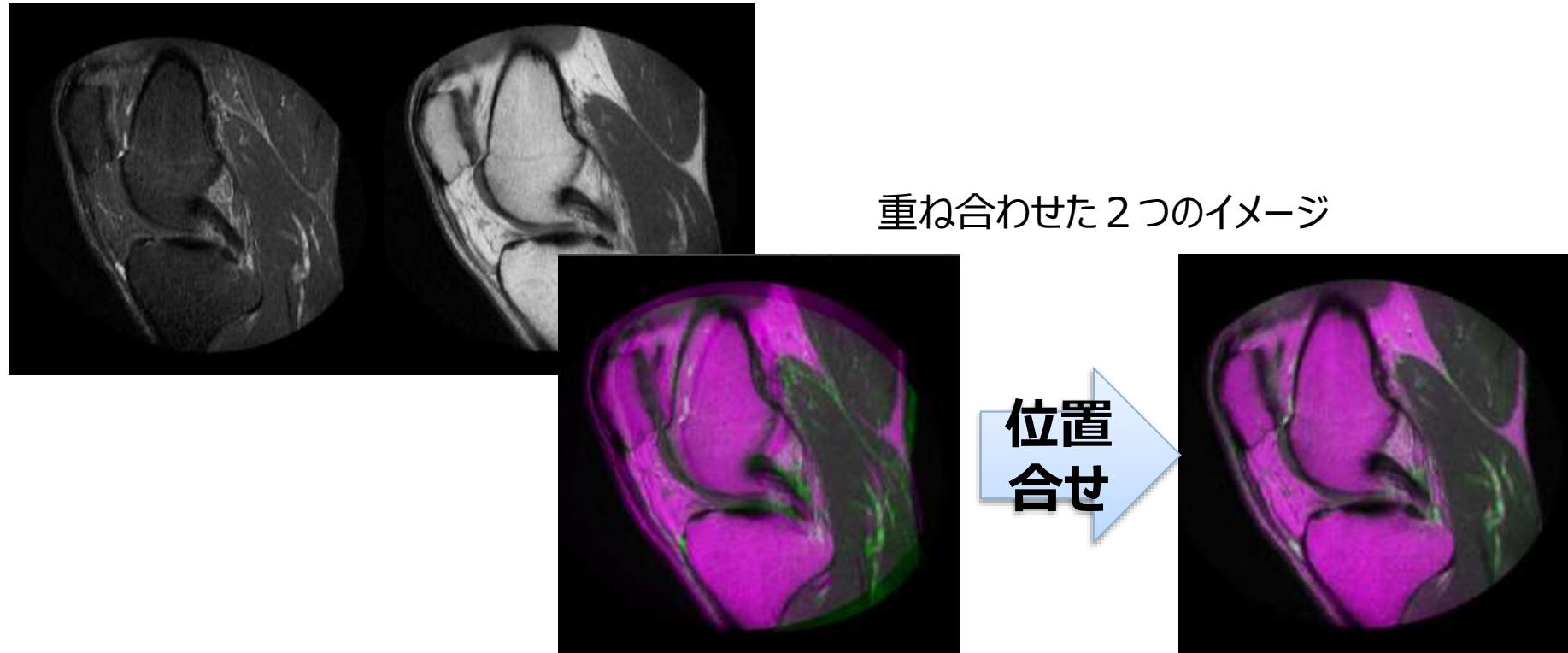
`imregister()` 強度ベースのレジストレーション（幾何学的変換まで実行する）

`imregcorr()` 位相相関を用いたレジストレーションの幾何学的変換行列の推定 **R2014a**

`imregdemons()` 非剛体(非線形)レジストレーション **R2014b**

`estimateGeometricTransform()` マッチングポイントからの幾何学的変換の推定

## 2.6.1 輝度ベースの自動レジストレーション（位置合せ）



% パラメータ設定

```
[optimizer,metric] = imregconfig('multimodal');
```

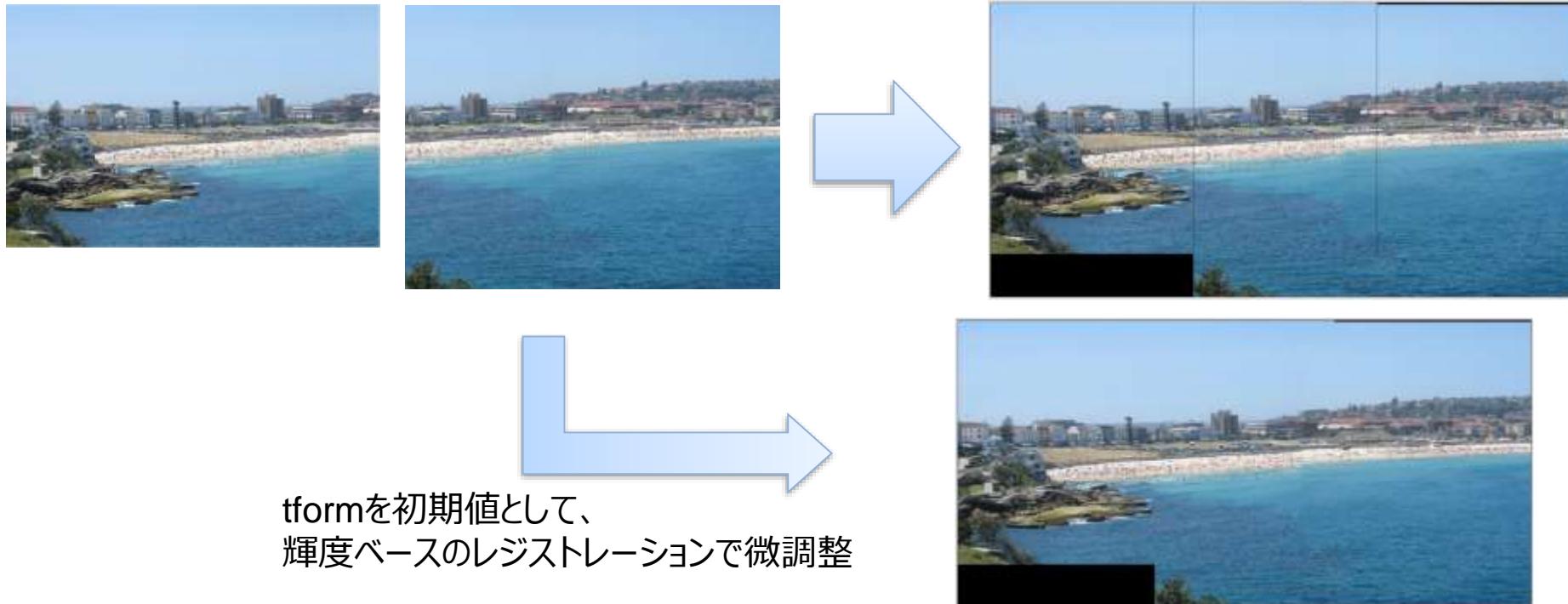
% 輝度ベースのレジストレーション実行（変換行列が必要な場合は、imregtform を使用）

```
Registered = imregister(moving, orig, 'affine', optimizer, metric);
```

**imregister**は、3次元画像にも対応

## 2.6.2 位相相関を用いた自動レジストレーション

位相相関を用いたレジストレーション  
(重ねるための幾何学的変換行列 tform)



% 位相相関を用いた、位置ずれの検出 (大きなずれに対しても安定)

```
tform = imregcorr(moving, fixed, 'translation')
```

% 輝度ベースのレジストレーションで微調整

```
tformOptim = imregtform(movingGray, fixedGray, 'translation', ...  
    optimizer, metric, 'InitialTransformation', tform);
```

## 2.6.3 非剛体自動レジストレーション（位置合せ）

R2014b



左の画像に合うように  
変形をかけて位置合せ

場所ごとに異なる変換



% 変位場行列を求める (各ピクセル毎の、x,y方向の変位)

```
D = imregdemons(moving, fixed);
```

% 幾何学的変換を行い、位置合わせをする

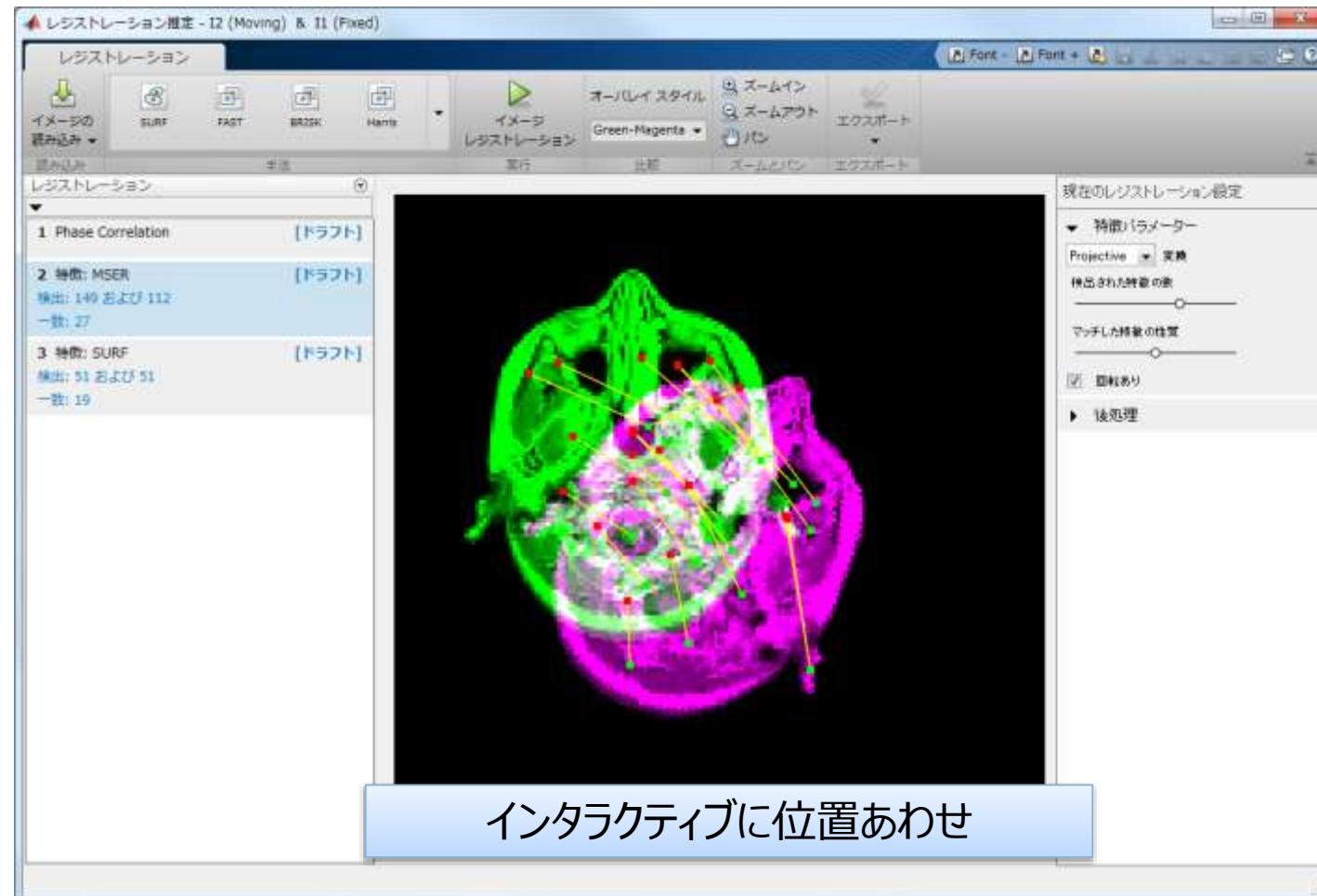
```
movingReg = imwarp(moving, D);
```

R2016b

3次元画像にも対応  
(CUDA® GPU と Parallel Computing Toolboxが必要)

## 2.6.4 レジストレーション推定アプリケーション

R2017a



```
%% 画像の読み込み・変形  
I1 = imread('mri.tif'); I2 = imrotate(I1,-30);  
%% アプリケーションの起動  
registrationEstimator
```

## 2.6.5 複数露光画像のためのレジストレーション（位置合せ）

R2018a

MTB(Mean Threshold Bitmaps)アルゴリズムによる位置合わせ  
露光度の異なる画像が多数ある場合のレジストレーションに有効



```
[R1,R2,R3,R4,R5,shift] = imregmtb(I1,I2,I3,I4,I5,I6);
```

## 2.7 画像フィルタリング

### 2次元フィルター処理用の関数

#### > フィルター係数の作成

<code>fspecial()</code>	事前定義型の 2 次元フィルター係数の作成
<code>fsamp2()</code>	周波数応答に対する2次元FIRフィルター係数の作成
<code>ftrans2()</code>	1次元FIRフィルターから、円対称の2次元FIRフィルター係数の作成
<code>fwind1()</code>	1次元ウインドウ法を使用した、円対称の2次元FIRフィルター係数の作成

#### > フィルター処理

<code>imfilter()</code>	多次元イメージの N 次元フィルター処理
<code>roifilt2()</code>	イメージの関心領域 (ROI) に対するフィルター処理

#### > その他のフィルター処理用関数

<code>imgaussfilt</code>	2次元のガウシアン フィルター処理	<b>R2015a</b>
<code>medfilt2()</code> 、 <code>medfilt3()</code>	2次元・3次元のメディアン フィルター処理	<b>R2016b</b> : 3次元
<code>imsharpen()</code>	イメージの鮮銳化	
<code>imguidedfilter()</code>	ガイド付き フィルター処理	<b>R2014a</b>
<code>normxcorr2()</code>	正規化された 2次元相互通関	
	⋮	

## 2.7.1 事前定義型の画像フィルタ処理 (fspecial関数)



原画像



平均化 : average



円状平均化 : disk



モーション : motion



ガウシアン : gaussian



ラプラシアン: laplacian



ガウスのラプラシアン: log

```
F = fspecial('average', 5)      % フィルタ パラメータ設定  
ImagAve = imfilter(I, F);    % フィルタ実行 (N次元で可能)
```

## 2.7.2.1 その他のフィルター

R2014a

```
Iguided = imguidedfilter(Inputs); % エッジ保存型の平滑化 (self-guidance)
```



元画像

平均化フィルタ

Guided Filter

```
Iguided = imguidedfilter(I, G); % ガイド画像を用いたエッジ保存型の平滑化
```



I: 高感度撮影画像  
(高ノイズ)



G: フラッシュ撮影画像  
(ガイド画像)



## 2.7.2.1 その他のフィルター

R2018b

フラットフィールドコレクション

```
Iflatfield = imflatfield(I,30); %フラットフィールドコレクション
```



元画像



影などの全体の明るさのバラツキを補正

Non-local means フィルター

```
J = imnlmfilt(I); %Non-local means フィルター
```

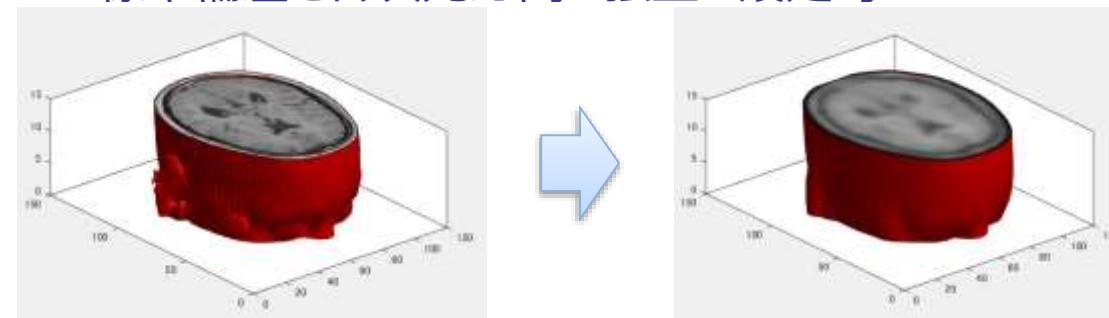


## 2.7.2.2 その他のフィルター：3次元画像

R2015a

```
I1 = imgaussfilt(I, [sigX sigY]); % 2次元のガウシアンフィルタ処理
I1 = imgaussfilt3(I, [sigX sigY sigZ]); % 3次元のガウシアンフィルタ処理
```

- 標準偏差を各次元方向に独立に設定可



```
F = ones(3,3,3)/27 % 3x3x3 のフィルタ係数定義 (平均化フィルタ)
volAve = imfilter(D, F); % 任意の3次元フィルタの適用
```

R2016a

```
[Gmag,Gazimuth,Gelevation] = imgradient3(I); % 3次元画像の勾配強度・方向
[Gx,Gy,Gz] = imgradientxyz(I); % 3次元画像の勾配
```

R2016b

```
filteredV = medfilt3(noisyV); % 3次元の中間値フィルタ
```

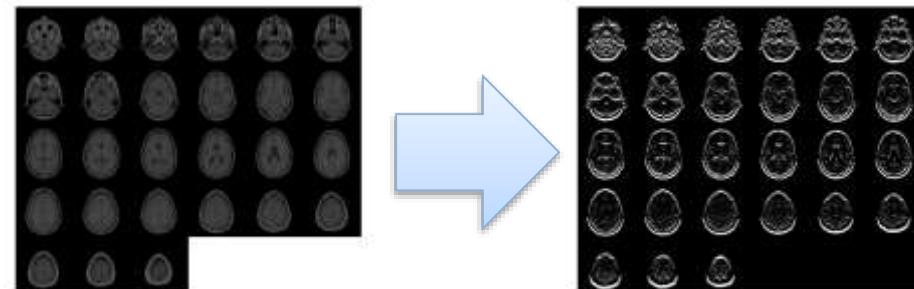
## 2.7.2.2 その他のフィルター：3次元事前定義型フィルター

R2018b

種類	説明
'average'	平均化フィルタ (非推奨、 <a href="#">imboxfilt3</a> がより高速)
'ellipsoid'	楕円平均フィルター
'gaussian'	ガウシアンローパスフィルター(非推奨、 <a href="#">imgaussfilt3</a> がより高速)
'laplacian'	ラプラシアンフィルター
'log'	ガウスのラプラシアンフィルター
'prewitt'	プレフィットエッジ強調フィルター
'sobel'	ソーベルエッジ強調フィルター



**ellipsoid**: 楕円平均フィルター



**sobel**: ソーベルエッジフィルター

```
H = fspecial3('ellipsoid',[7 7 3]);
volSmooth = imfilter(mristack,H,'replicate');
```

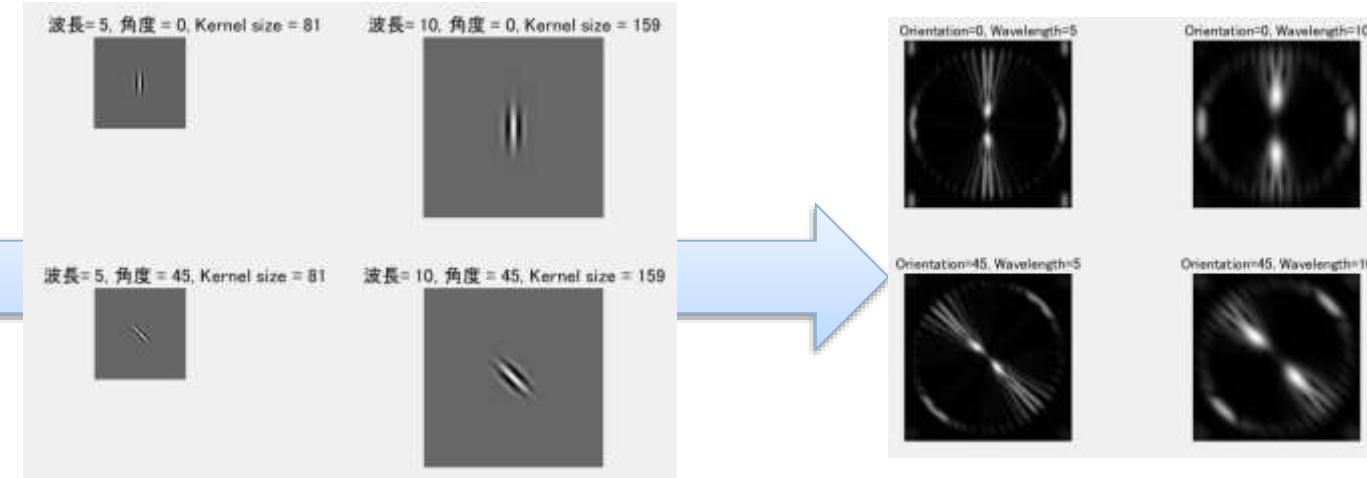
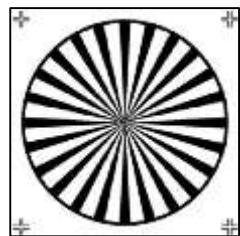
```
H = fspecial3('sobel','Y');
volSmooth = imfilter(mristack,H,'replicate');
```

## 2.7.2.3 その他のフィルター

### ガボールフィルターバンク

R2015b

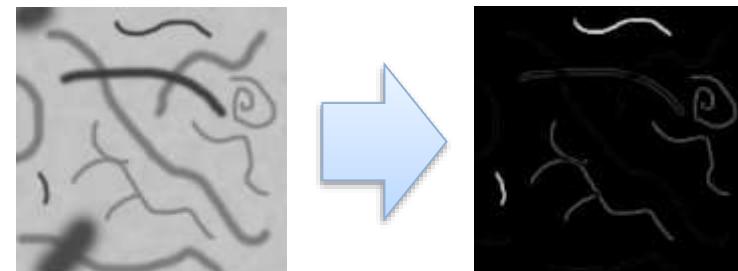
対象画像



```
g1 = gabor(wavelength, orientation); % ガボールフィルタを生成
mag1 = imgaborfilt(G, g1); % 画像にガボールフィルタを適応
```

ヘッセ行列固有値に基づく線強調フィルタ

R2017a



```
mask = fibermetric(G, 8, 'ObjectPolarity', 'dark', 'StructureSensitivity', 5);
```

## 2.7.2.4 その他のフィルター

R2018a

- 異方性拡散フィルタ

```
Idiffusion = imdiffusefilt(I);
```

Smoothing Using Anisotropic Diffusion (Left) vs. Gaussian Blurring (Right)



ガウシアンフィルタと比較して、  
エッジを保持した形でスムージング

- バイラテラルフィルタ

```
I = imread('cameraman.tif');  
patch = imcrop(I,[170, 35, 50 50]);  
patchVar = std2(patch)^2;  
DoS = 2*patchVar;  
J = imbilatfilt(I,DoS);
```

Degree of Smoothing: 51.9395



スムージング量をノイズの分散の  
推定値を基に設定

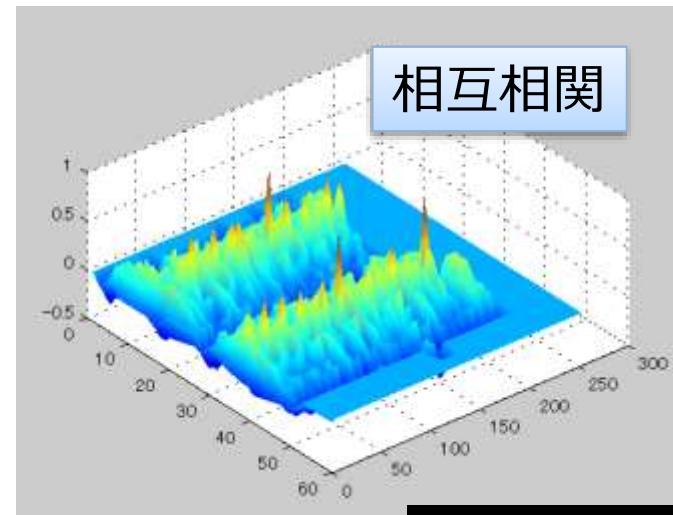
## 2.7.3 正規化された2次相互通関

The term watershed  
refers to a ridge that ...

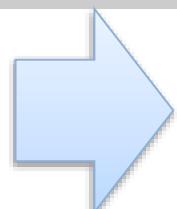
テンプレート



```
corr = normxcorr2(tPlate, BW); % 正規化された2次元相互通関
```



テンプレートとの相互通関  
を用い、'a' を検出

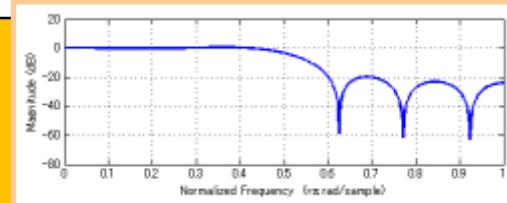


The term watershed  
refers to a ridge that ...

## 2.7.4 1次元フィルタからの二次元フィルタ設計

**Signal Processing Toolbox** 関数  
による1次元フィルタ設計

```
N = 12; % フィルタ次数
Fpass = 0.5; % 通過帯域周波数
Fstop = 0.55; % 遮断帯域周波数
Wpass = 1; % 通過帯域重み
Wstop = 1; % 遮断帯域重み
b=firls(N,[0 Fpass Fstop 1],[1 1 0 0],[Wpass Wstop]);
freqz(b,1) % 1次元フィルタ周波数応答表示
```



FDAToolによる  
1次元フィルタ設計

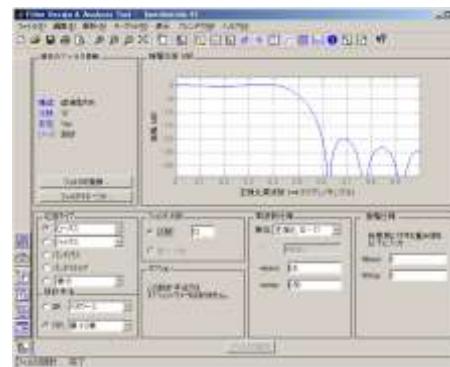
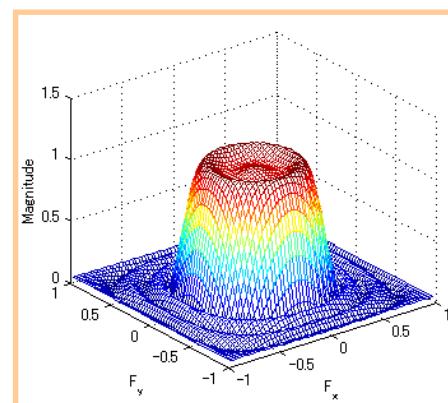
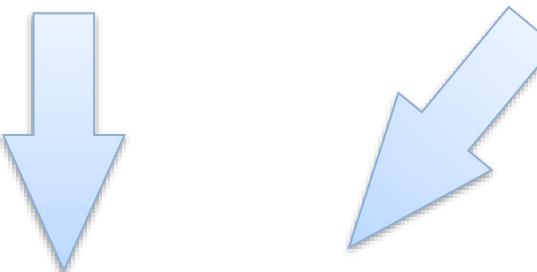


Image Processing Toolboxによる  
1次元フィルタの2次元化

```
H2 = ftrans2(b); % 周波数変換法による2次元フィルタ設計
freqz2(H2) % 2次元フィルタ周波数応答表示
If = imfilter(I,H2); % 2次元フィルタ処理
```

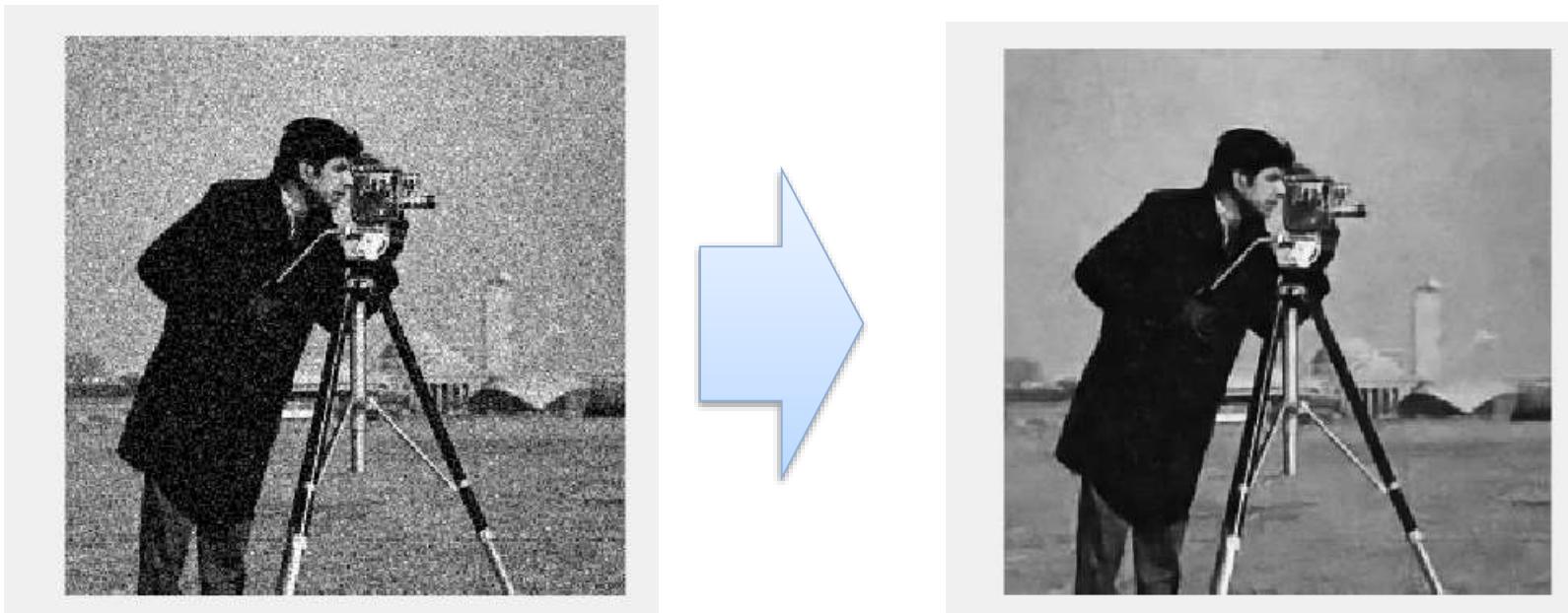


2次元フィルタ振幅応答

## 2.7.5 ディープラーニングベースのノイズ除去

R2017b

- ディープラーニングベースのノイズ除去(ガウシアンノイズ向け)  
(要: Deep Learning Toolbox)



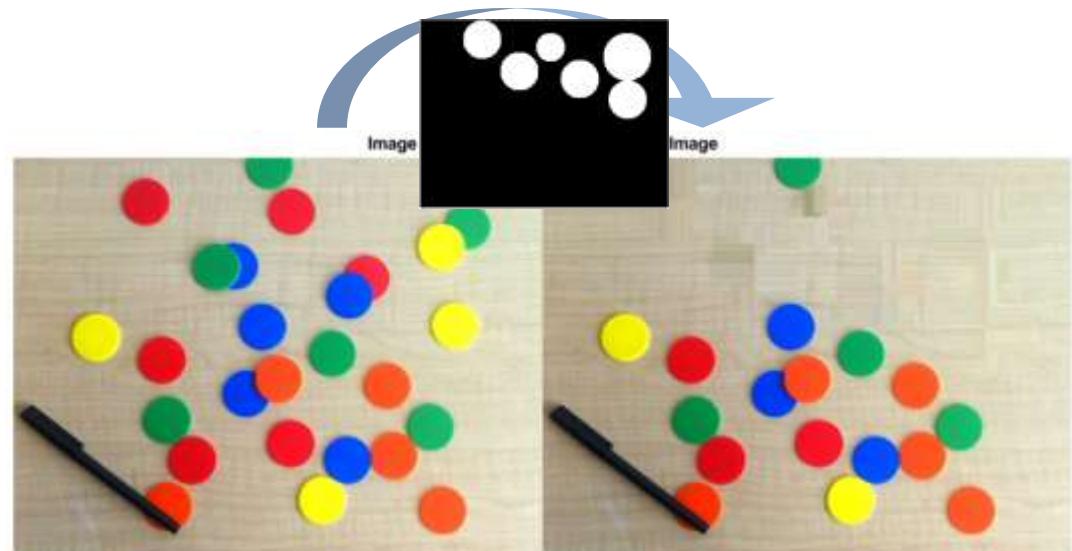
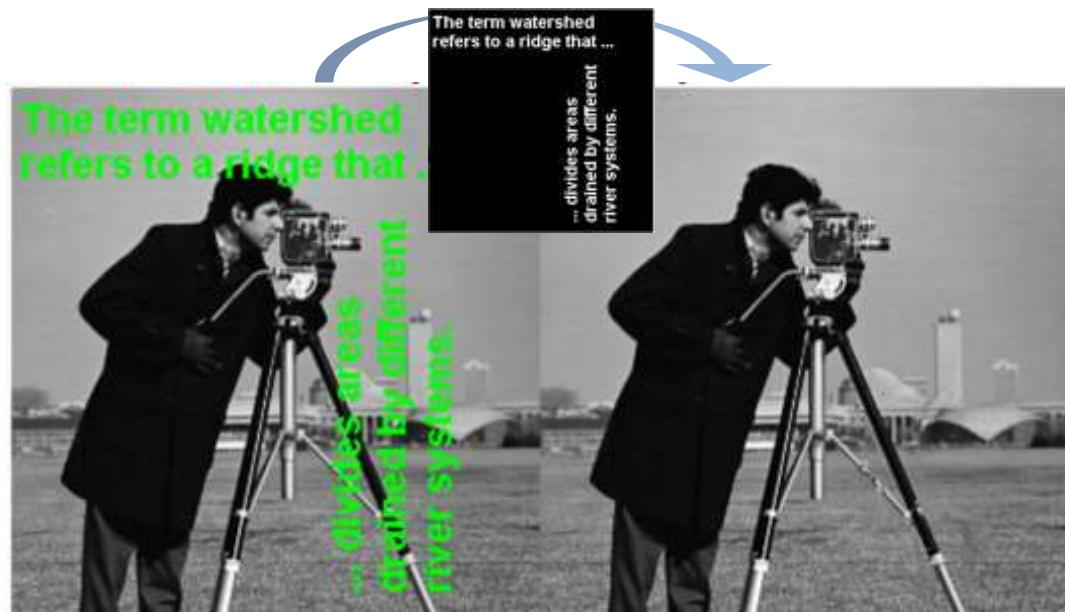
```
net = denoisingNetwork('DnCNN');  
denoisedI = denoiseImage(noisyI, net);
```

`denoisingImageDatastore`によるガウスノイズの付加をサポート R2018a

## 2.7.6 画像の修復（インペインティング・不要物の除去）

R2019a

コピー・レンストラントベースの画像修復を使用して特定の画像領域を復元



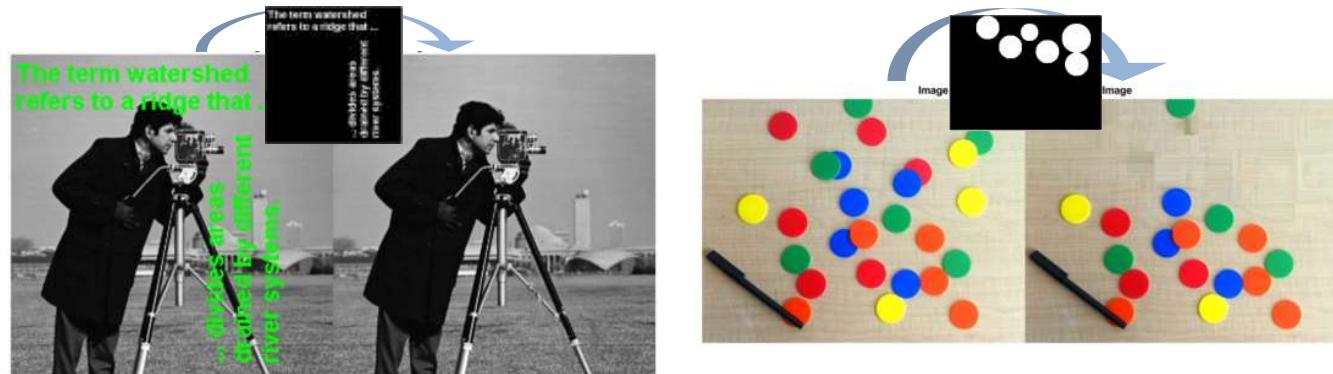
```
J = inpaintCoherent(I,mask,'SmoothingFactor',0.5,'Radius',1);
```

## 2.7.6 画像の修復（インペインティング・不要物の除去）

R2019a

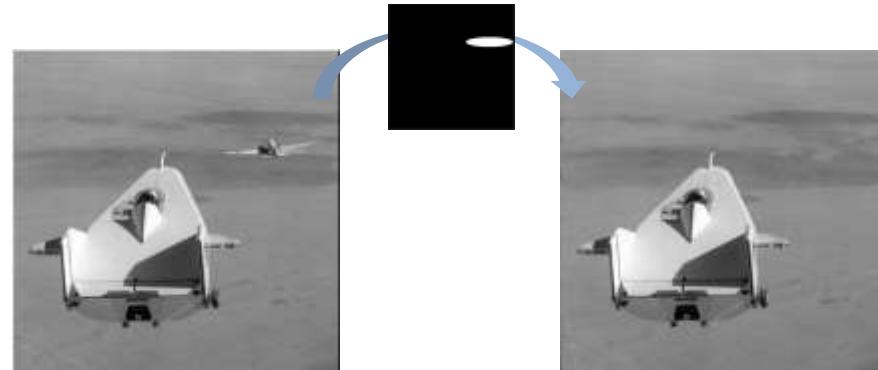
### 指定したマスク領域を修復する

コピーレンストラントベースの画像修復を使用して特定の画像領域を復元



```
J = inpaintCoherent(I,mask,'SmoothingFactor',0.5,'Radius',1);
```

エグゼンプラーベースの画像修復を使用して特定の画像領域を復元



```
J = inpaintExemplar(I,mask);
```

R2019b

## 2.7.7 連射画像から高解像度画像生成(超解像)

R2019a

- 逆距離重み付け法を使用して  
低解像度連射画像から高解像度画像を生成



## 2.8 モルフォロジー処理・ラベリング・物体の定量評価

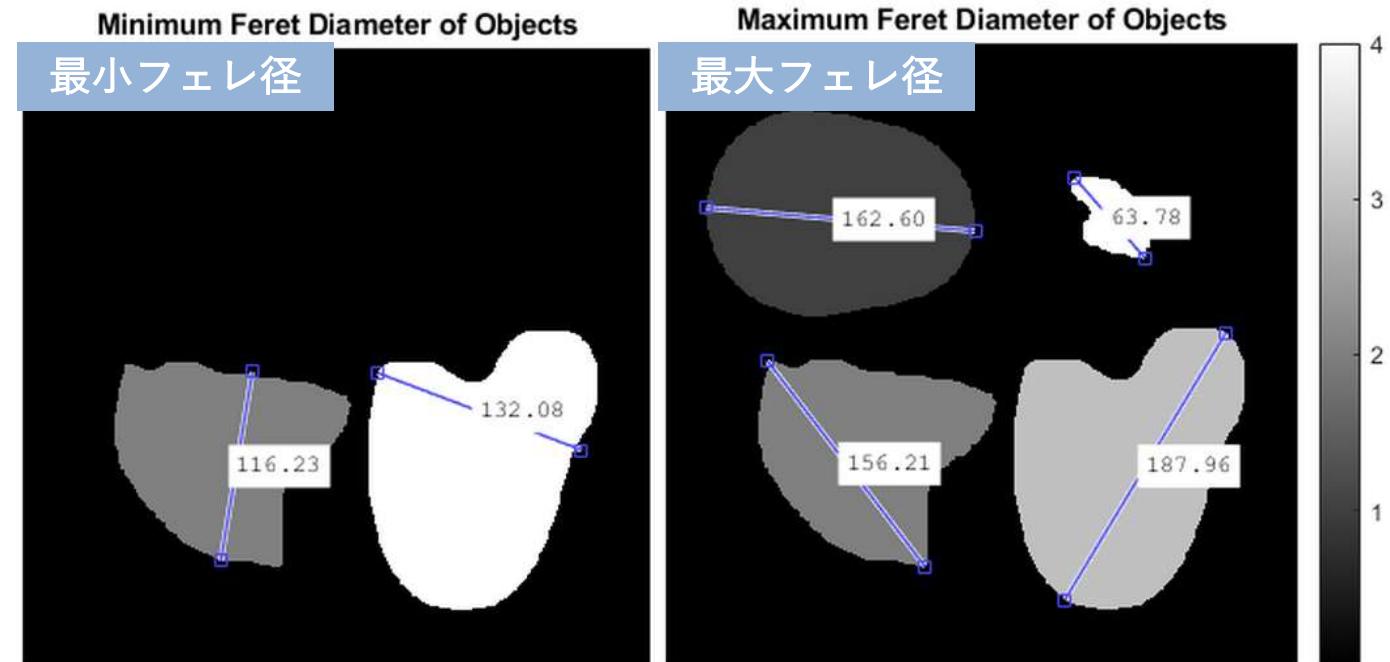
### モルフォロジー処理・ラベリング・定量評価用の関数

<code>bwmorph()</code>	二値画像のモルフォロジー演算 (2次元のみ)	
<code>bwareaopen()</code>	二値画像からの小さなオブジェクトの削除	
<code>bwareafilt()</code>	二値画像からの面積によるオブジェクト抽出	R2014b
<code>bwpropfilt()</code>	二値画像からのプロパティーによるオブジェクト抽出	R2014b
<code>imbothat()</code>	ボトム ハット フィルター処理 (クローズ画像 - 元画像)	
<code>imclearborder()</code>	イメージ境界と連結している明るい構造体を削除にする	
<code>imclose()</code> / <code>imopen()</code>	イメージのクローズ処理 (膨張 -> 収縮) / オープン処理 (収縮 -> 膨張)	
<code>imdilate()</code> / <code>imerode()</code>	イメージの膨張/収縮	
<code>imfill()</code>	イメージ領域と穴の塗りつぶし	
<code>imtophat()</code>	トップ ハット フィルター処理 (元画像 - オープン画像)	
<code>bwlabel()</code> 、 <code>bwlabeln()</code>	2値画像内の連結要素をラベル付け	
<code>bwselect()</code>	バイナリ イメージのオブジェクトを選択 (2次元のみ)	
<code>Bwconvhull()</code>	バイナリ イメージから凸包イメージの生成	
<code>regionprops()</code>	各領域のプロパティ計測	多くは、任意次元にも対応

## 2.8 モルフォロジー処理・ラベリング・物体の定量評価

R2019a

- **regionprops**のプロパティとして
  - ‘Circularity’
  - ‘MinFeretProperties’
  - ‘MaxFeretProperties’
- **bwferet**
  - フェレ径計測用関数



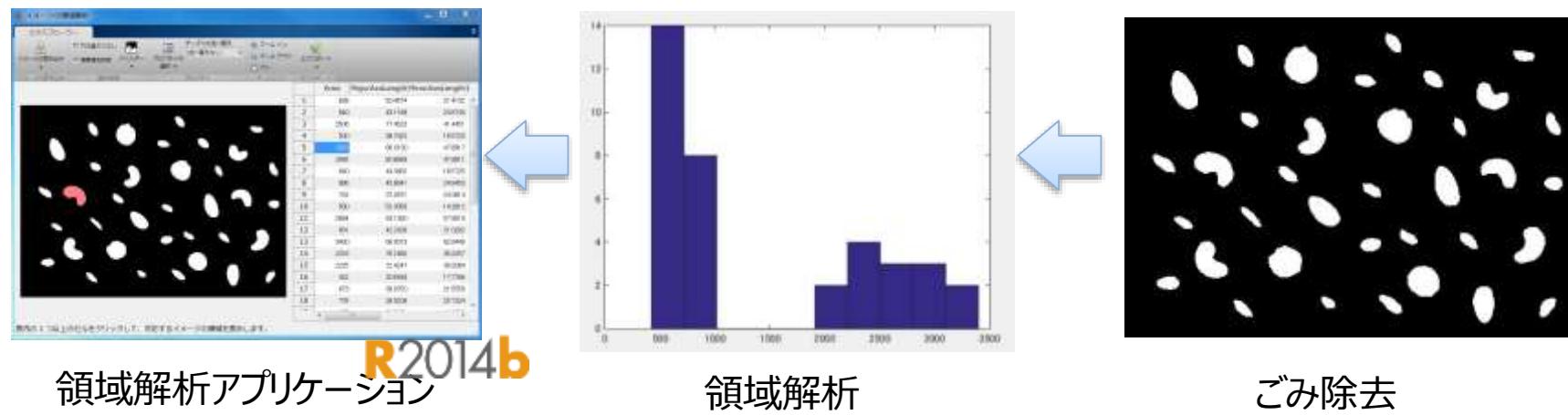
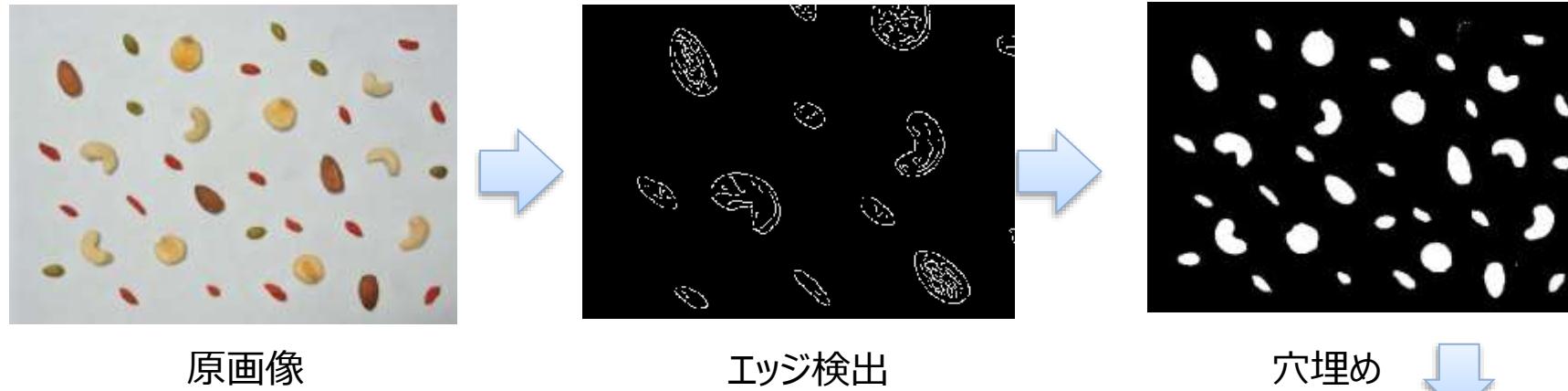
円形度：真円に近いと1、遠いと0となるパラメータ

$$\text{Circularity} = 4\pi S / L^2$$

S: 領域の面積 L:外周

## 2.8.1 モルフォロジー処理・ラベリング・定量評価

- 前処理としてモルフォロジー処理を行い、各領域の面積を求める



(面積・境界ボックス・重心・領域の平均輝度・周囲長・橿円の長軸/短軸 等)

## 2.8.2 関心領域（ROI）の指定と操作

R2018b

関心領域(ROI)の指定と操作のためのオブジェクト

橙円、円、フリーハンド、フリーハンド(境界自動検出)、直線、点、ポリゴン、折れ線、四角形をサポート

インタラクティブな操作を柔軟に作成可能

**imellipse**



**imfreehand**

**imline**

**impoint**

**impoly**

**imrect**

R2018a 以前

```
imshow('peppers.png');
h = drawassisted;
```

**drawellipse**

**drawcircle**

**drawfreehand**  
**drawassisted**

**drawline**

**drawpoint**

**drawpolygon**

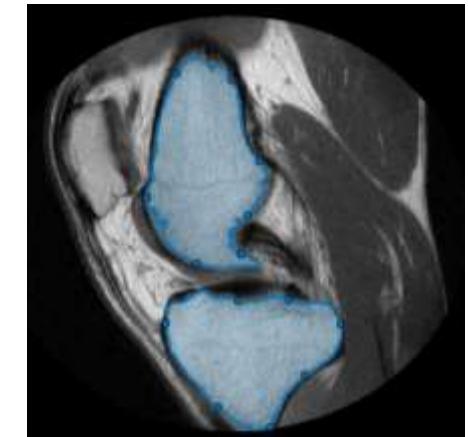
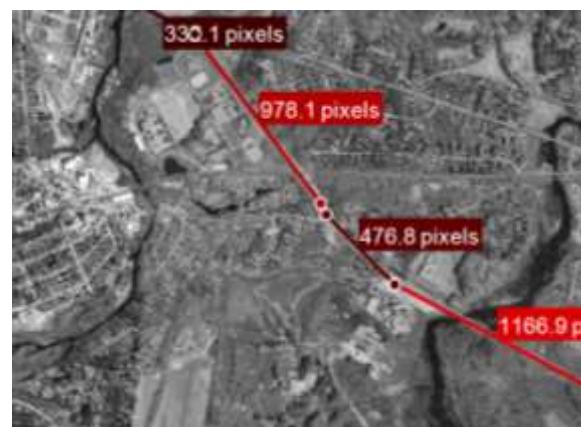
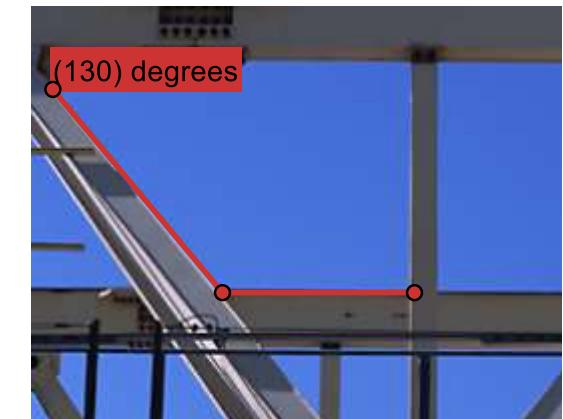
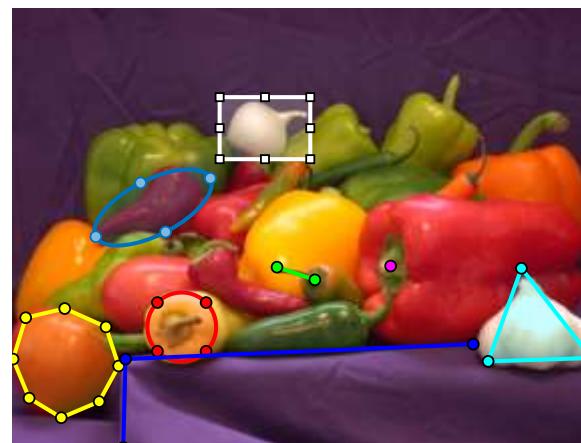
**drawpolyline**

**drawrectangle**

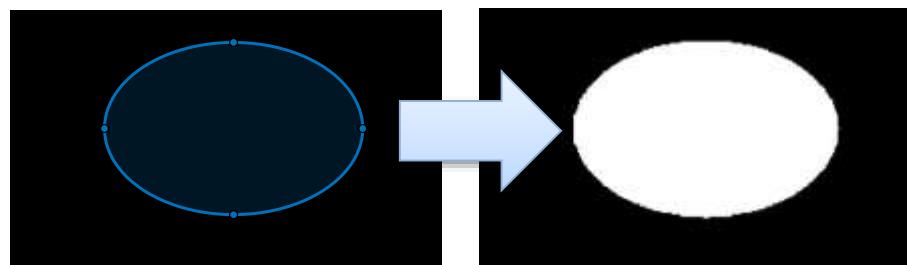
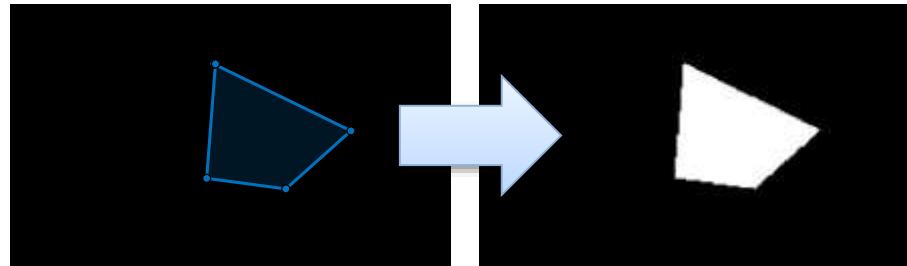
**R2018b**

**drawcrosshair**

**R2019b**



## 2.8.2 関心領域（ROI）の指定と操作



### 多角形のROIマスク(バイナリ画像)の指定

```
x = [116 194 157 112 117];
y = [ 34 72 105 99 34];
drawpolygon('Position',[x' y']);
BW = poly2mask(x, y, 150, 250);
```

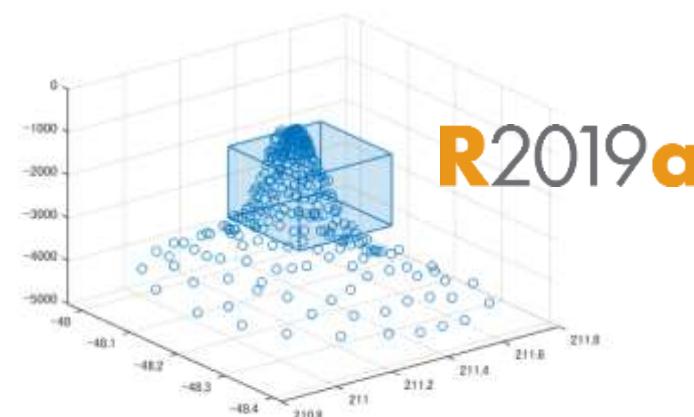
生成するマスクの高さ

マスクの幅

### 橿円形のROIマスクの指定

生成するマスクのサイズ

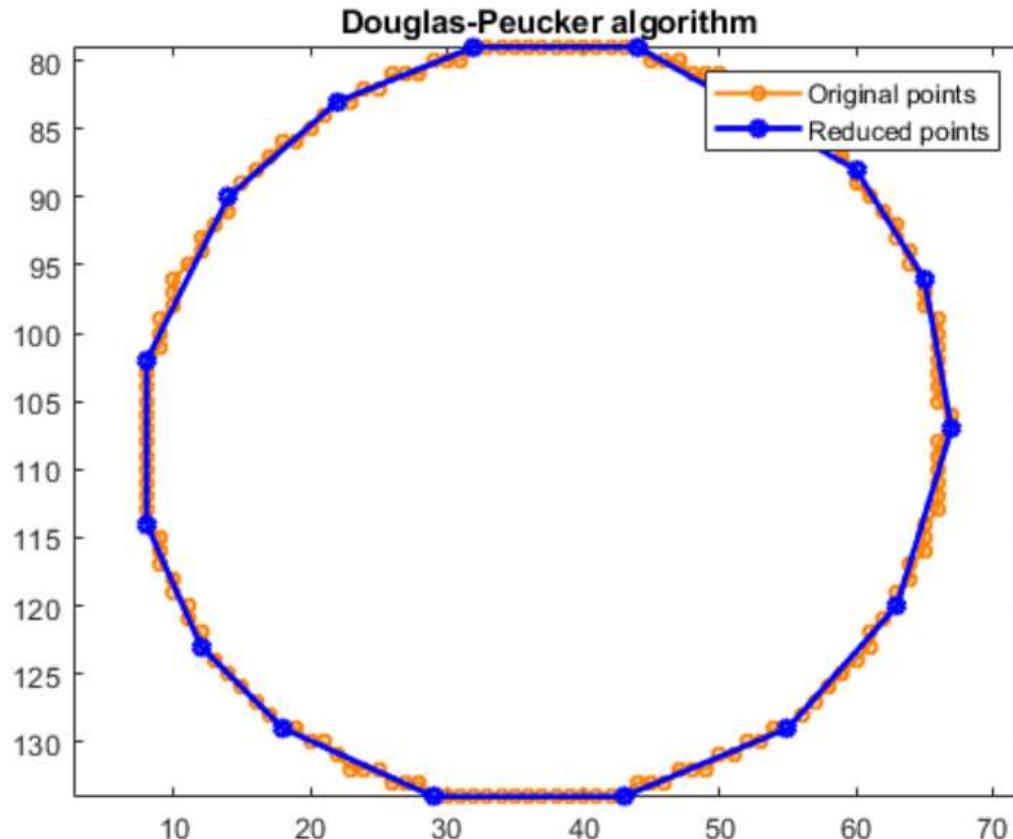
```
img = false(150, 250);
figure; h_im = imshow(img);
e = drawellipse(gca,...,
    'Center',[130 70],...           中心座標
    'SemiAxes',[75 50]);           長軸短軸の長さ
BW = createMask(e, h_im);
```



3次元空間におけるROIの設定  
面をドラッグ&ドロップして範囲を調整・回転も可能

```
load seamount
hScatter = scatter3(x,y,z)
drawcuboid(hScatter);
```

## 2.8.3 関心領域（ROI）の指定と操作



### ROI点の効率的な削減

```
p = [boundary(:,2) boundary(:,1)]; % ROI境界座標  
tolerance = 0.02; %  
p_reduced = reducepoly(p,tolerance); % ROI点の削減
```

Douglas-Peuckerのアルゴリズムを使って、直線部分を削減。  
カーブ部分は保存。

## 2.9 画像の解析(エッジ・コーナー・オブジェクト検出、セグメンテーション)

### 画像解析用の関数

`edge()` エッジ検出 (ソーベル法、プレウイット法、キャニー法 等)

`detectHarrisFeatures()` コーナー検出 (ハリス)

`detectMinEigenFeatures` コーナー検出 (最小固有値法)

`hough()` ハフ変換

`houghpeaks()` ハフ変換のピークの特定

`houghlines()` ハフ変換に基づく線分の抽出

`imfindcircles()` 円のハフ変換を利用した円の検索

`imgradient()` イメージの勾配の大きさと方向

`qtdecomp()` 四分木分割

`boundarymask()` 全ての境界を前景とする2値画像を生成 (2値・ラベル画像)

`bwperim()` 2値画像内の全ての領域境界をトレース

⋮



R2016a

## 2.9 画像の解析： 領域解析

### 領域解析用の関数

`bwareafilt()` 面積による2値画像からのオブジェクトの抽出 **R2014b**

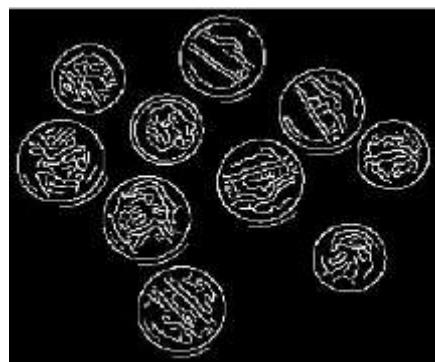
`bwpropfilt()` 2値画像からプロパティを使用してオブジェクトを抽出 **R2014b**

`bwselect()` 2値画像内のオブジェクトを選択

`mean2()` 多次元配列の全要素の平均

⋮

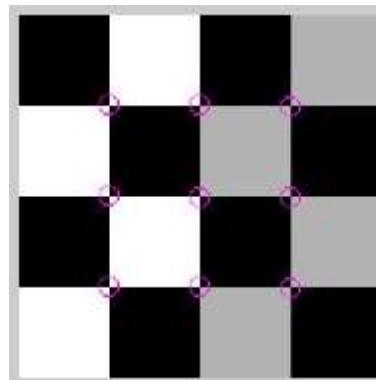
## 2.9.1 エッジ検出・コーナー検出・円検出



### エッジ検出

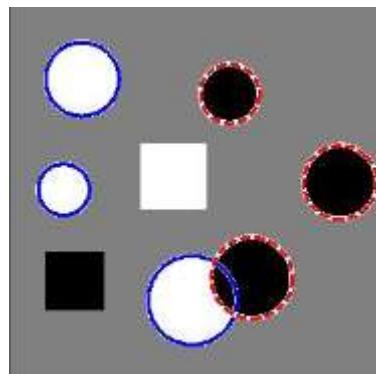
```
BWsobel = edge(I, 'sobel')          % ソーベル法
BWcanny = edge(I, 'canny')          % キヤニー法
Bwacanny = edge(I, 'approxcanny')   % 高速キヤニー法
```

R2016b



### コーナー検出

```
C = detectHarrisFeatures(I)        % Harrisコーナー検出器
C = detectMinEigenFeatures(I)      % 最小固有値法
```



### 円検出

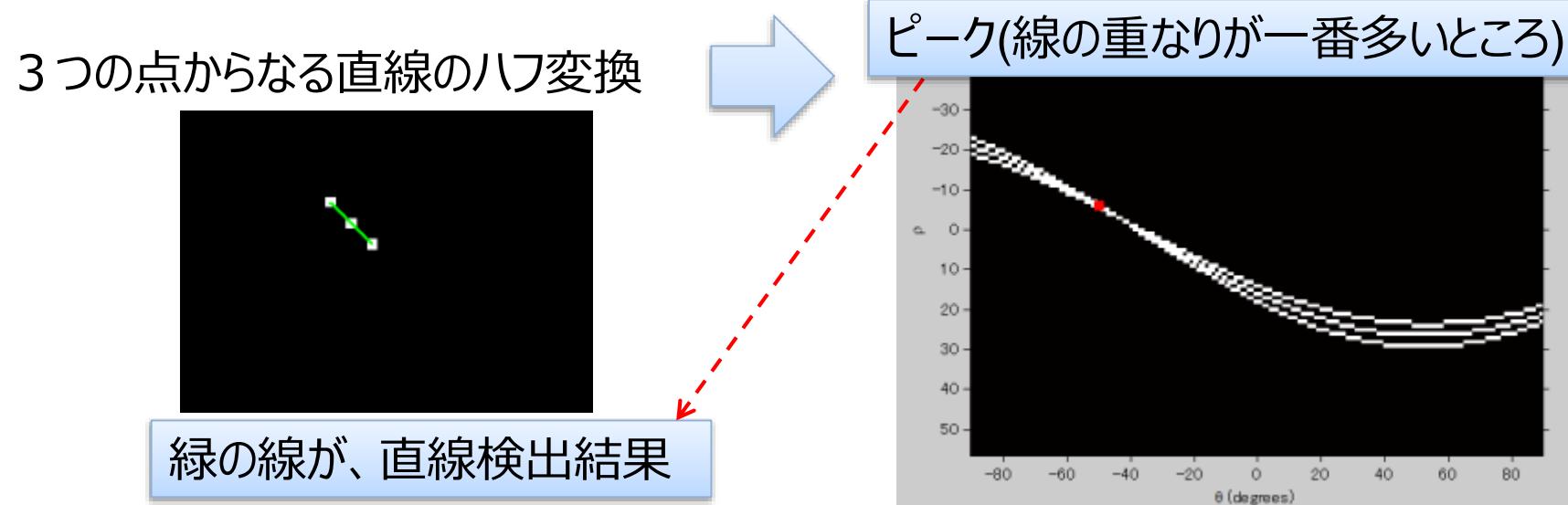
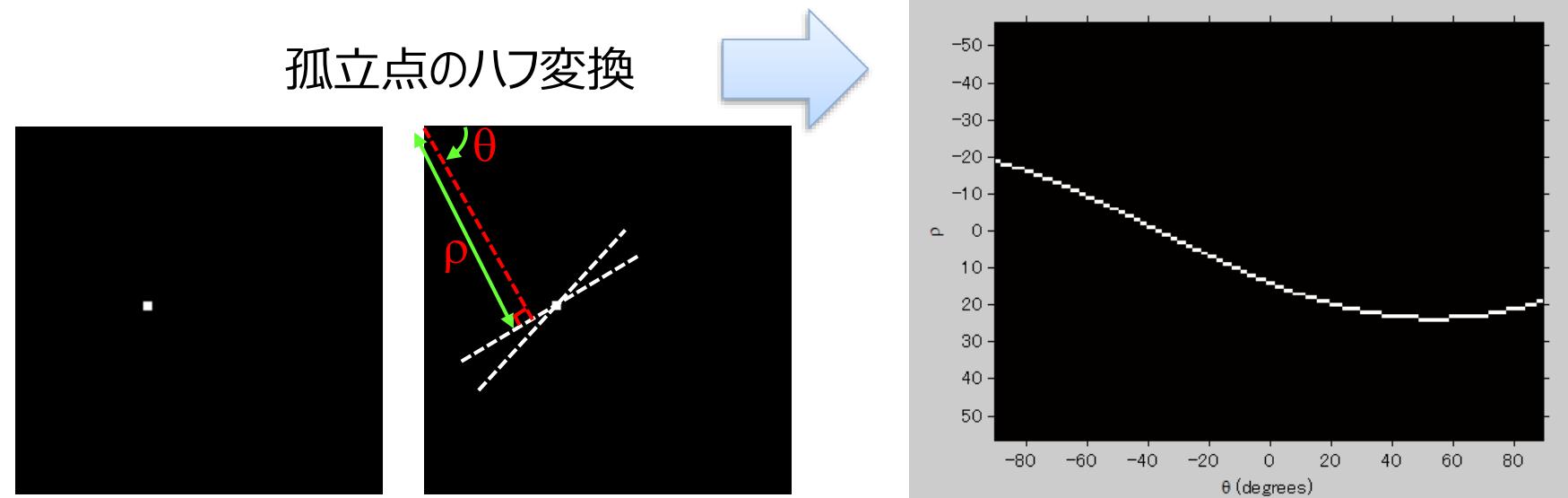
```
[cDark, rDark] = imfindcircles(G, [30 65], ...
    'ObjectPolarity', 'dark');
```

各円の中心座標

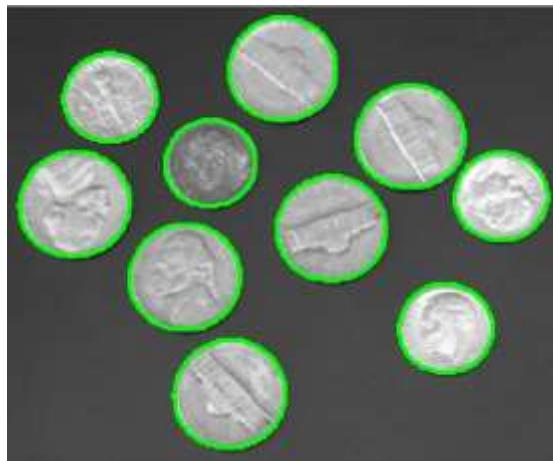
半径

背景より暗い円を検出

## 2.9.2 直線検出（ハフ変換）



## 2.9.3 境界のトレース



画像内全領域の境界をトレース (bはマスク画像)

R2016a

```
b = boundarymask(BW); % 境界をマスクとして生成  
Ib = imoverlay(G, b, 'g'); % マスクを画像へ上書き
```

2値画像内全領域の境界をトレース (bはXY座標リスト)

R2015a

```
b = bwboundaries(BW, 'noholes');  
visboundaries(b, 'color', 'g'); % Fig上へ上書き
```



2値画像内の1つの領域境界をトレース

```
b = bwtraceboundary(BW, [row, col], 'N');
```

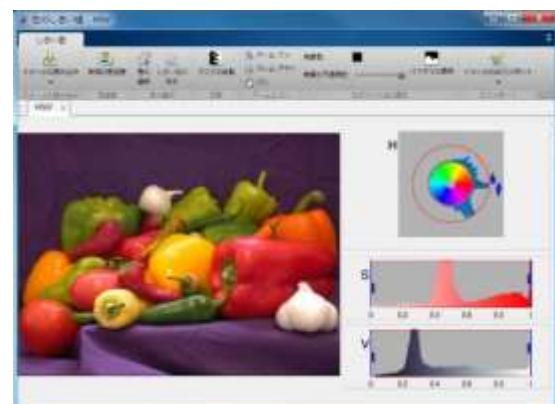
トレース開始の境界上座標点

## 2.9.4 セグメンテーション（領域分割）

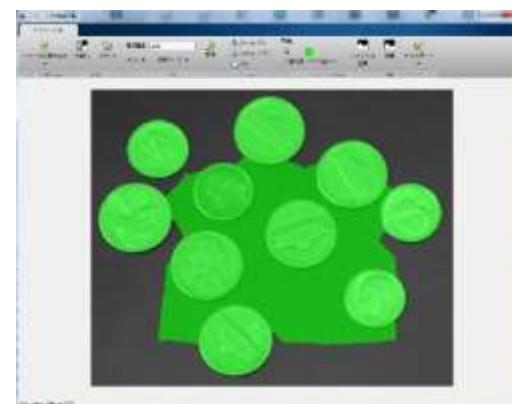
### セグメンテーション（領域分割）用の関数

<code>activecontour()</code>	動的輪郭を使用したセグメンテーション	R2014b
<code>imsegfmm()</code>	高速マーチング法を用いたバイナリイメージのセグメンテーション	
<code>imseggeodesic()</code>	色の測地線距離ベースのセグメンテーション	R2015a
<code>grayconnected()</code>	シードピクセルに類似した輝度値によるセグメンテーション	R2015b
<code>colorThresholder()</code>	色の閾値アプリケーション	R2014a
<code>imageSegmenter()</code>	イメージの領域分割アプリケーション	R2014b
<code>superpixels()</code>	スーパーピクセルの生成	R2016a
<code>imsegkmeans()</code>	k-meansベースのセグメンテーション	R2018b

色の閾値 アプリケーション



イメージの領域分割 アプリケーション

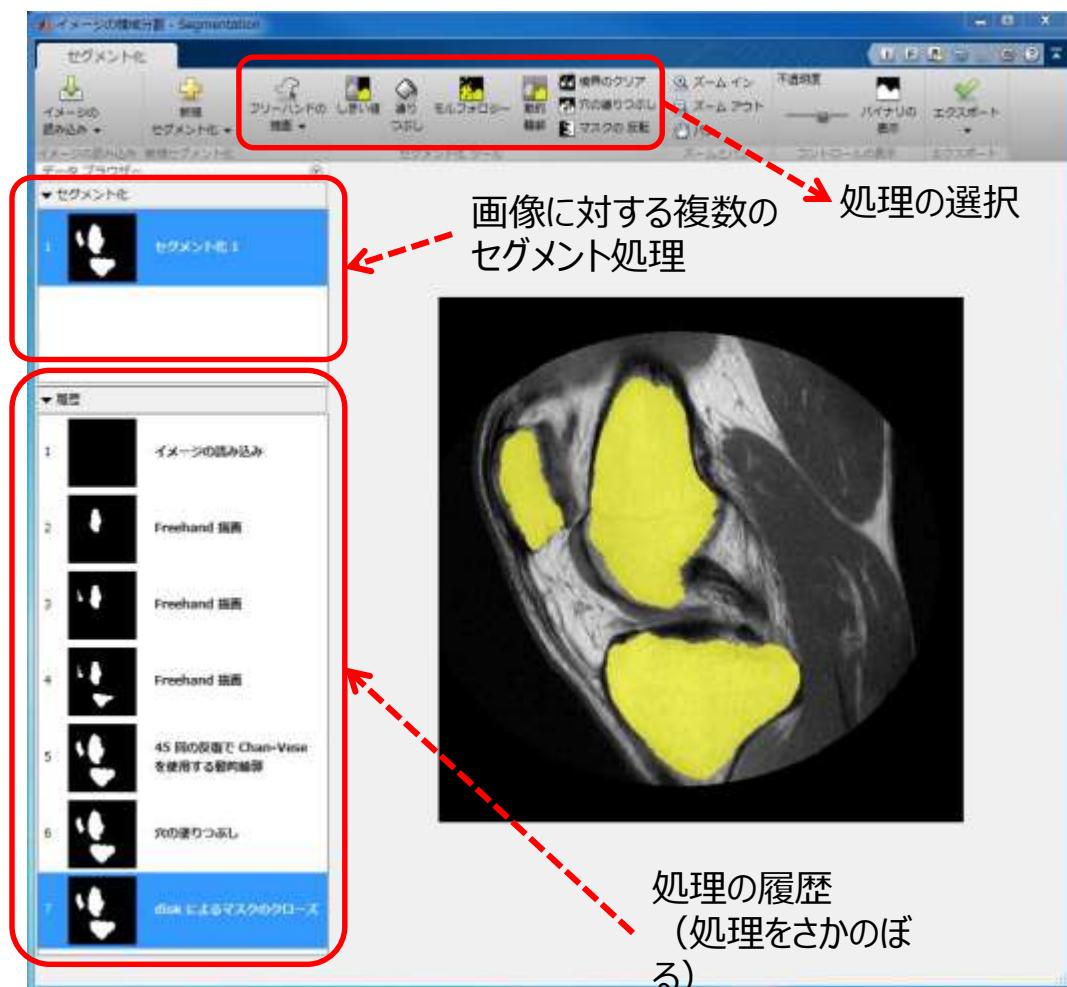


## 2.9.4.1 イメージの領域分割アプリケーション

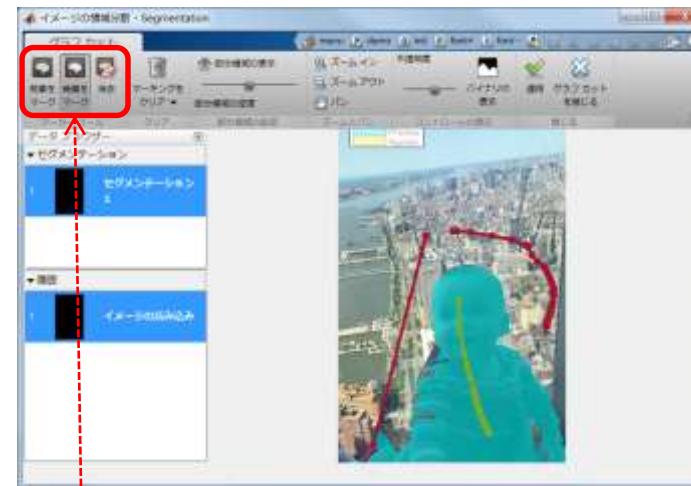
イメージの領域分割アプリケーション  
 >> `imageSegmenter`



R2014b  
 R2016a  
 R2017a



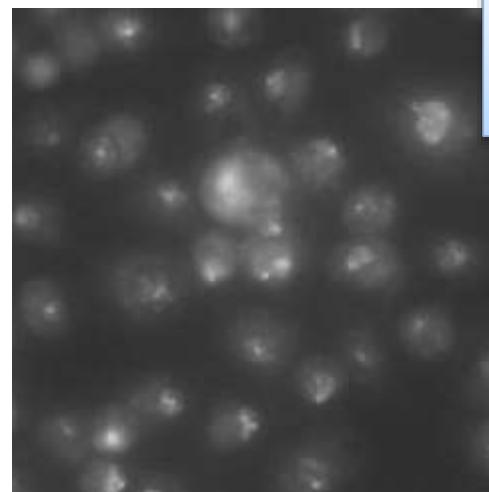
グラフカットに対応 R2017a ローカルグラフカット  
 (grabcut)に対応 R2018a



前景や背景を指定して切り抜き  
 ガボールフィルタによる  
 テクスチャ解析の追加  
 K-meansによる自動クラスタリング  
 (Statistics and Machine Learning Toolbox 必要)

R2017b

## 2.9.4.2 動的輪郭を用いたセグメンテーション

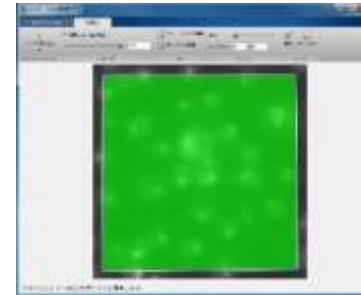


輪郭の不鮮明な物体の輪郭抽出・  
セグメンテーション

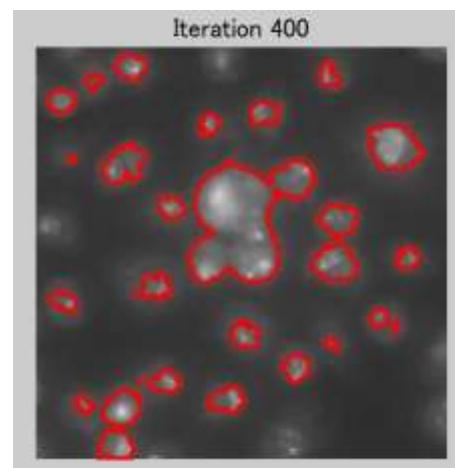
R2014b

イメージの領域分割 アプリケーション

`imageSegmenter (G)`

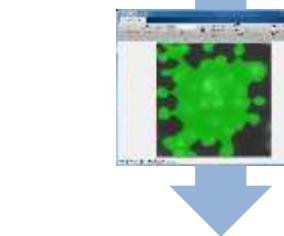


初期輪郭の指定

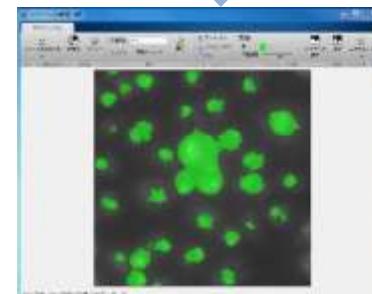


`BW = activecontour (G, mask, 400);`

初期輪郭 反復回数の上限

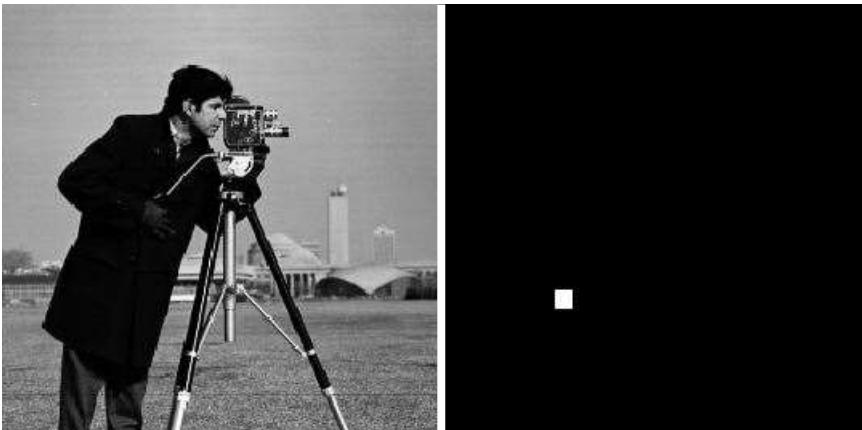


動的輪郭モードを選択



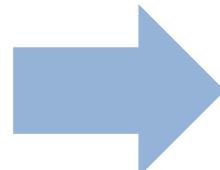
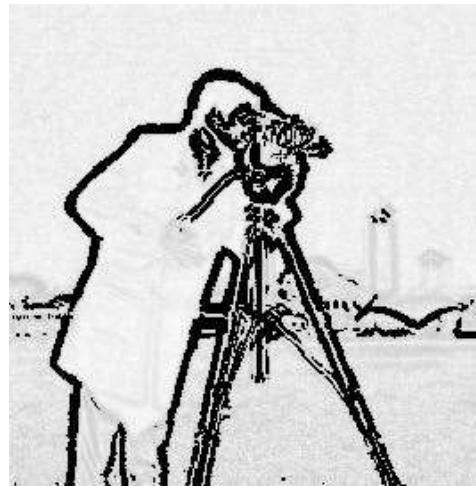
## 2.9.4.3 高速マーチング法を用いたセグメンテーション

mask (シード位置)



重み配列の計算

- 各ピクセル毎
- 勾配小で大きくなる
- もしくは
- 基準点との輝度絶対差

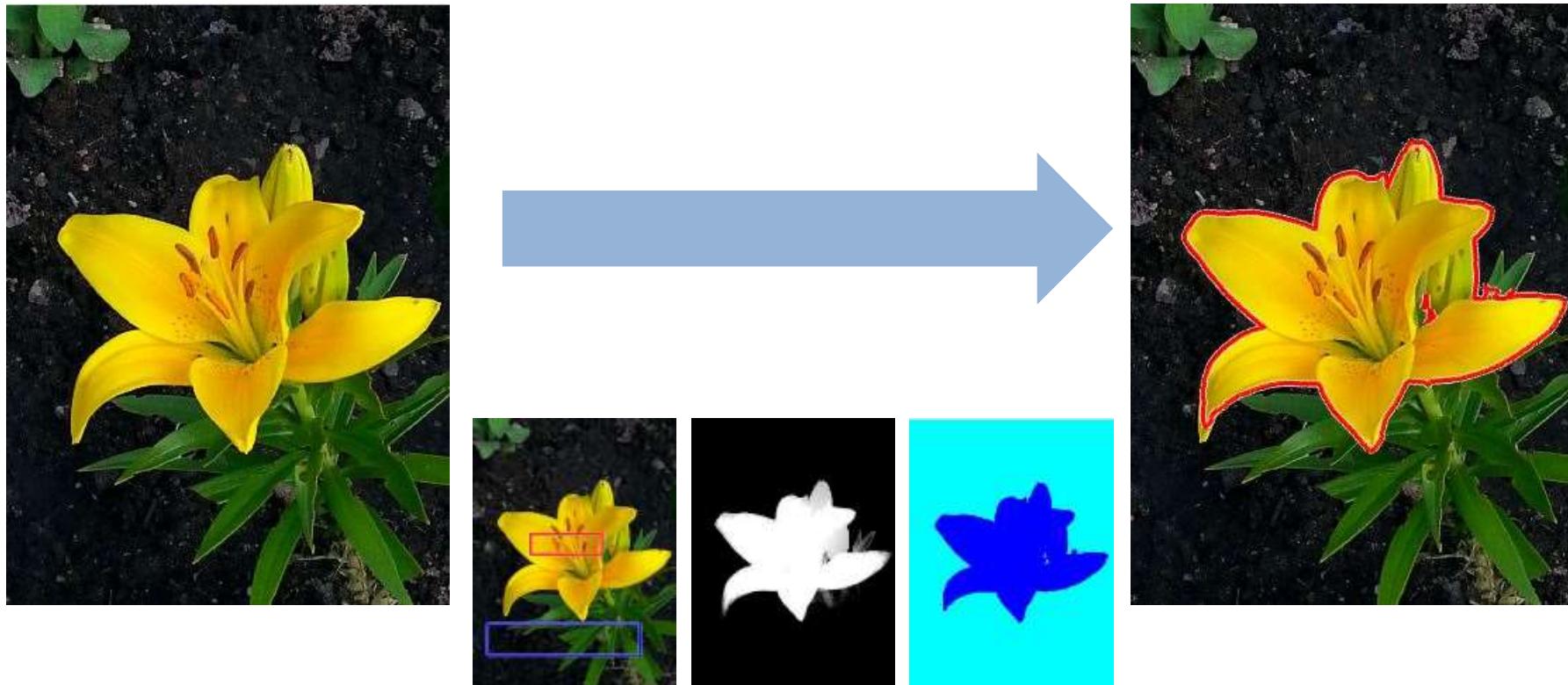


```
W = gradientweight(G, sigma);  
or W = graydiffweight(G, mask);
```

```
BW = imsegfmm(W,mask,thresh);
```

## 2.9.4.4 色の測地線距離ベースのセグメンテーション

```
[L,P] = imseggeodesic(RGB,BW1,BW2);
```



[1] A. Protiere and G. Sapiro, Interactive Image Segmentation via Adaptive Weighted Distances, IEEE Transactions on Image Processing, Volume 16, Issue 4, 2007

## 2.9.4.5 スーパーピクセルを用いたセグメンテーション R2016a

R2016b  
(3次元対応)

スーパーピクセル：処理を、画素単位から、輝度や色が似ている領域単位にすることで処理の単純化・処理量の軽減



SLICスーパーピクセル生成



色の類似性でクラスタリング



対象の抽出

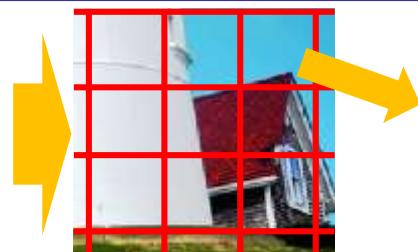


imsegkmeans

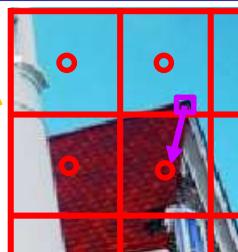
R2018b

```
[Ls, N] = superpixels(I, 600); % ~600個のスーパーピクセルを生成 (ラベル画像)
Bmask = boundarymask(Ls); % ラベル境界をトレース (2値画像)
I1 = imoverlay(I, Bmask, 'cyan'); % 画像中に、2値画像を指定色で上書き
```

### SLIC (simple linear iterative clustering) スーパーピクセル



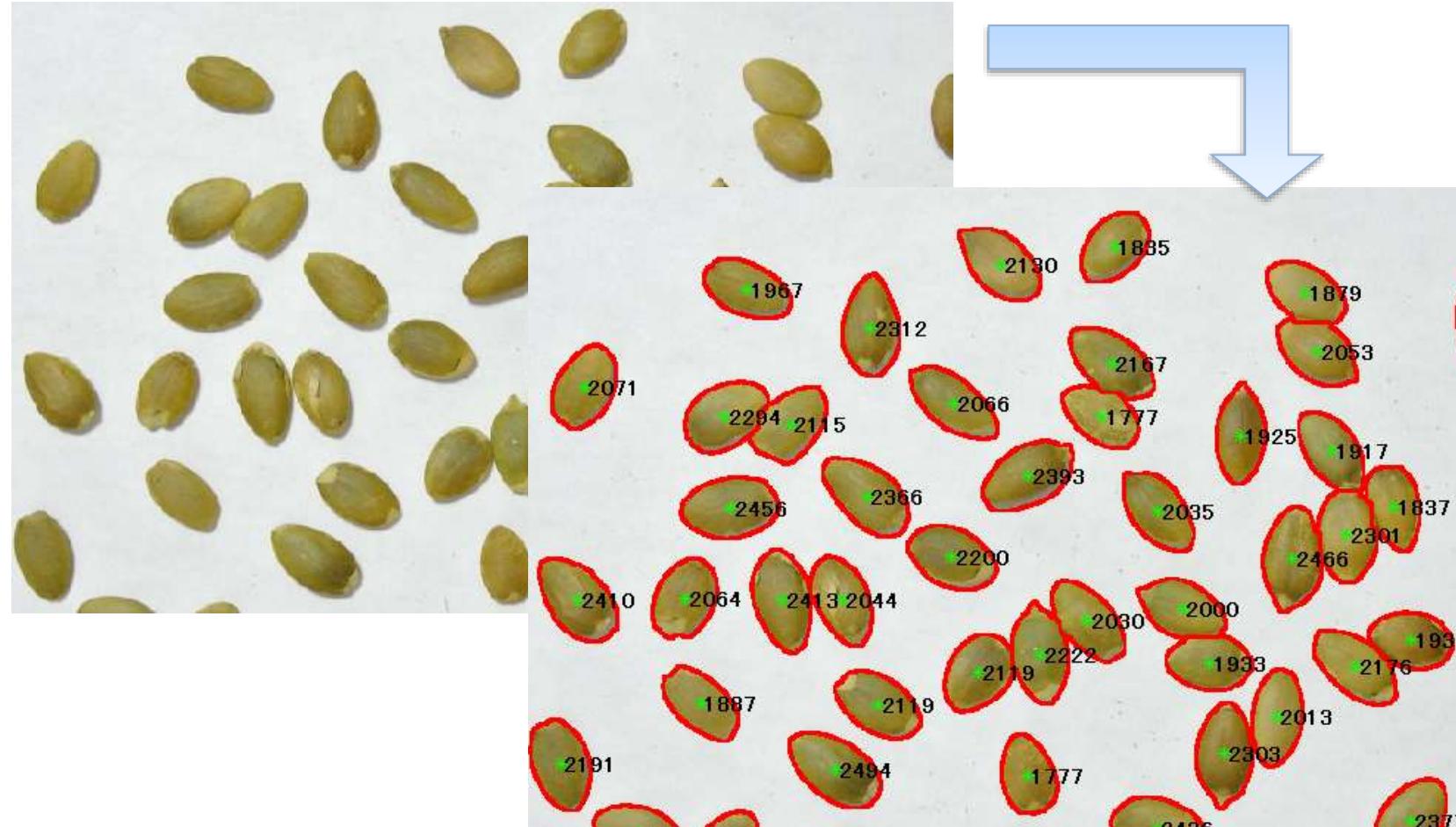
初期 矩形スーパーピクセル



各画素と、周囲のスーパーピクセル中心の  
"色差 + 距離" を計算し  
一番近いスーパーピクセルへ割当て  
=>スーパーピクセル中心を再計算し、  
各画素の再割当 (繰り返し)

## 2.9.4.6 モルフォロジー処理を用いたセグメンテーション例： - 粒子が接触している場合

MATLABによる処理結果

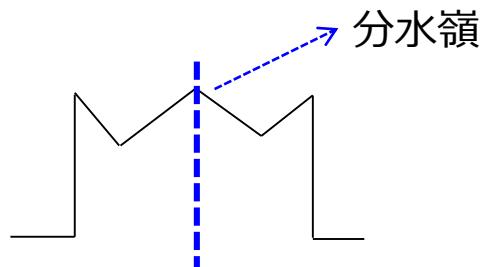
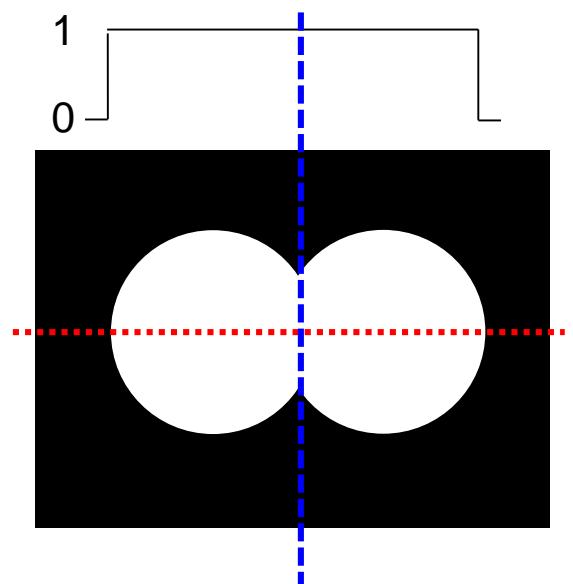


くつについていても別々と認識し、各粒の輪郭を囲む・中心点と面積を表示

## 2.9.4.6 モルフォロジー処理を用いたセグメンテーション例： - 粒子が接触している場合

watershed (分水嶺) セグメンテーションによる領域分離

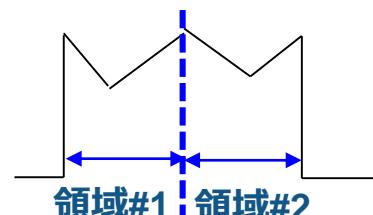
このようなプロファイルへ変換できれば



極小点毎に領域分割

% 分水嶺変換

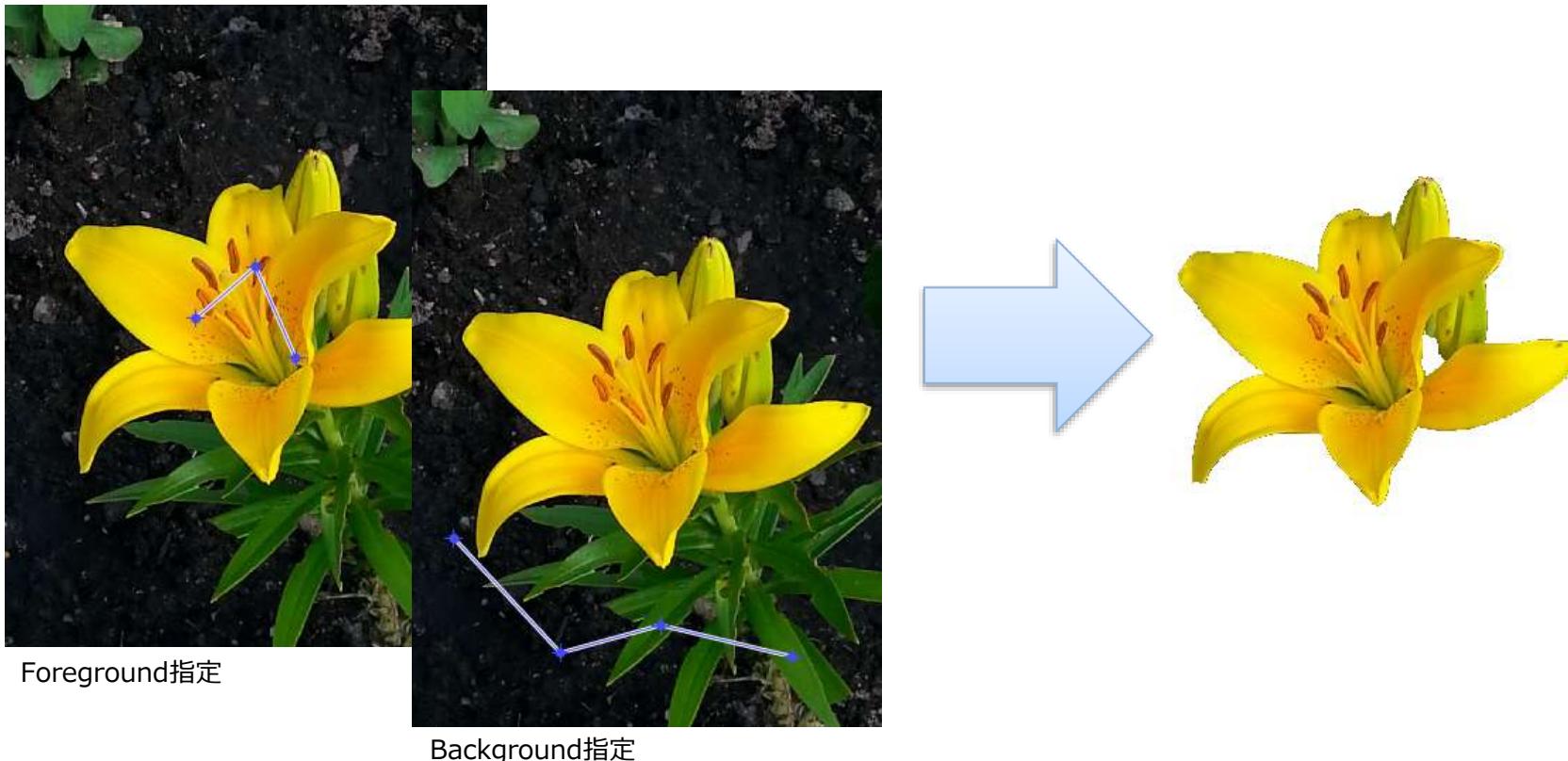
**$BWshed = watershed(BWhmin)$**



領域#1 領域#2

## 2.9.4.7 グラフベースのセグメンテーション

R2017a

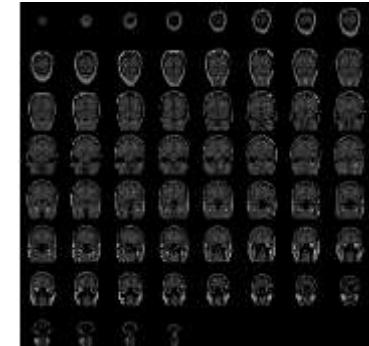


```
BW = lazysnapping(RGB, L, foregroundInd, backgroundInd);  
maskedImage = RGB;  
maskedImage(repmat(~BW, [1 1 3])) = 255;
```

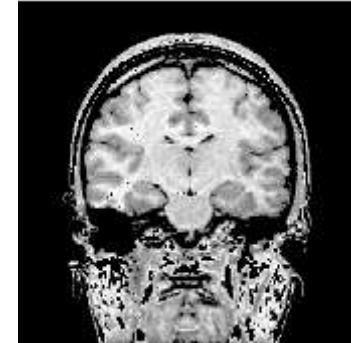
## 2.9.4.8 3次元画像処理



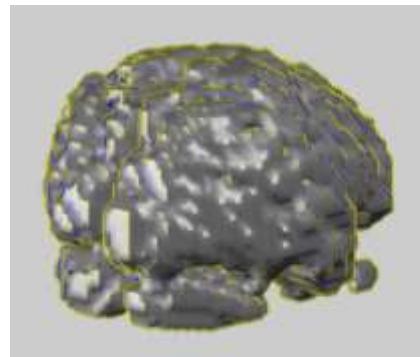
DICOM画像確認  
**dicomBrowser R2017b**



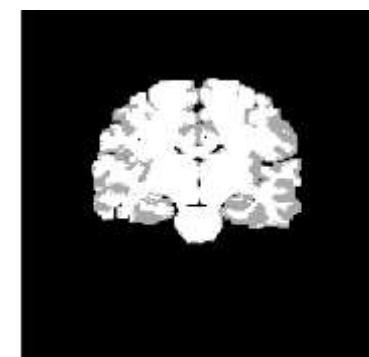
DICOMからデータ読み込み  
**dicominfo, dicomread,**  
**dicomreadVolume R2017b**



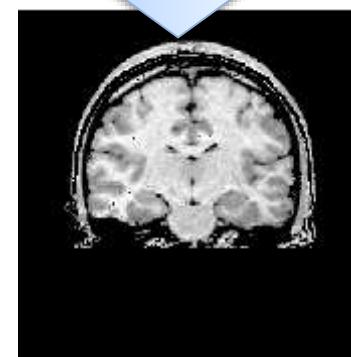
輝度値ベースの  
ノイズ除去



3次元画像の構築・容積計算  
**regionprops3 R2017b**



モルフォロジー処理  
&面積による脳抽出  
(**imopen, regionprops**)



脳下部分離

## 2.9.4.8 3次元画像処理

### 3次元画像の幾何学変換・2値化・セグメンテーション・定量評価

<code>imresize3()</code>	リサイズ	R2017a
<code>imrotate3()</code>	回転	R2017a
<code>activecontour()</code>	動的輪郭を使用したセグメンテーション	R2017a
<code>edge3()</code>	エッジ検出	R2017b
<code>bwselect3()</code>	3次元バイナリ画像の選択	R2017b
<code>regionprops3</code>	定量評価	R2017b
<code>imadjustn</code>	画像の調整	R2017b
<code>imbinarize</code>	2値化	R2017b
<code>adaptthresh</code>	適応しきい値	R2017b
<code>bwmorph3</code>	モルフォロジー	R2018a
<code>bwskel</code>	スケルトン化	R2018a
<code>imsegkmeans3</code>	k-meansクラスタリング	R2018b



## 2.10 イメージ変換・近傍処理

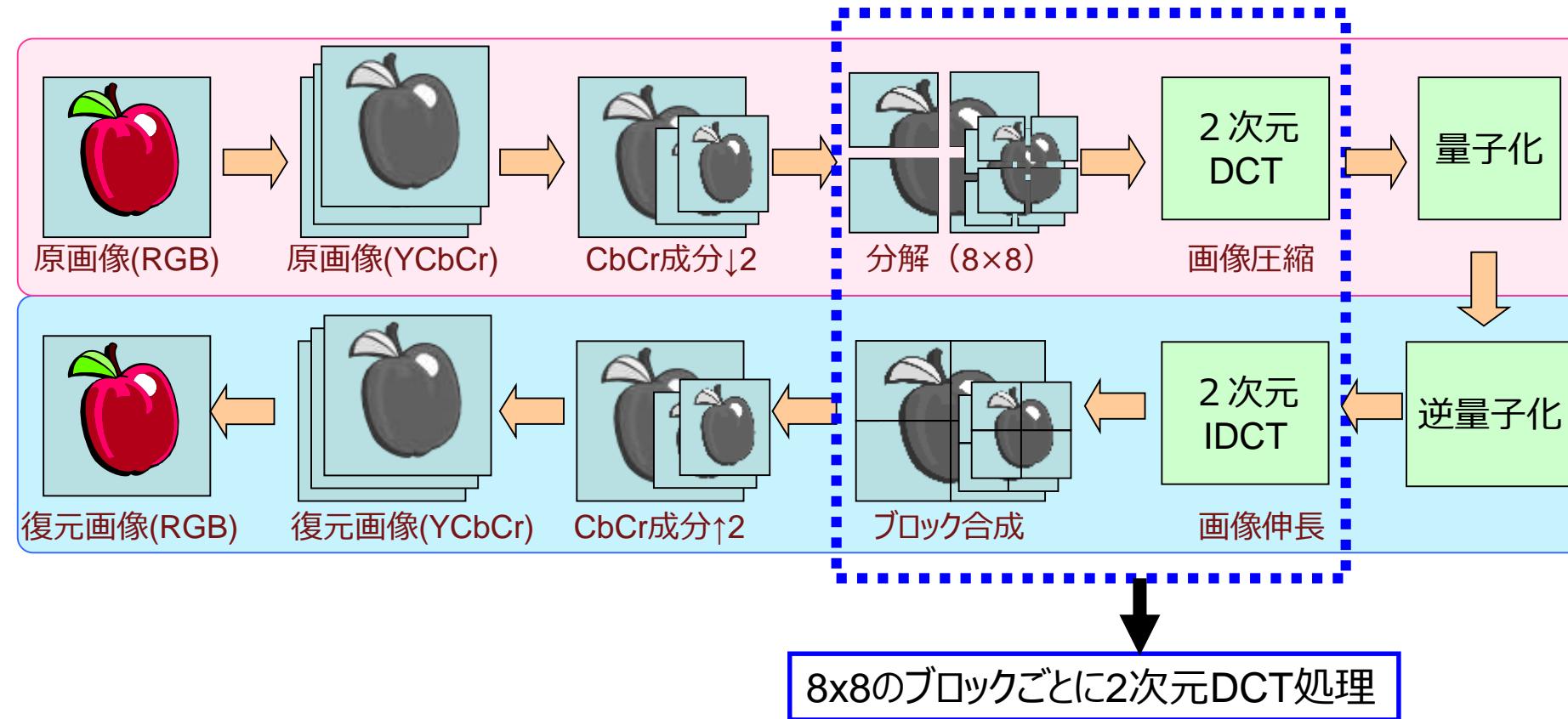
### イメージ変換用の関数

<code>dct2()</code>	2次元の離散コサイン変換
<code>idct2()</code>	2次元の逆離散コサイン変換
<code>fft2()</code>	2次元の高速フーリエ変換 (MATLAB基本関数)
<code>fftshift()</code>	DC成分をスペクトルの中心へ移動 (MATLAB基本関数)
<code>ifft2()</code>	2次元の逆高速フーリエ変換 (MATLAB基本関数)
<code>radon()</code>	ラドン変換
<code>bwdist()</code>	2値画像の距離変換 (最も近いTrue画素までの距離)
<code>bwdistgeodesic()</code>	2値画像の測地線距離変換 (各True画素からシード画素まで、False画素を通らない最短距離)
:	

### 近傍処理用の関数

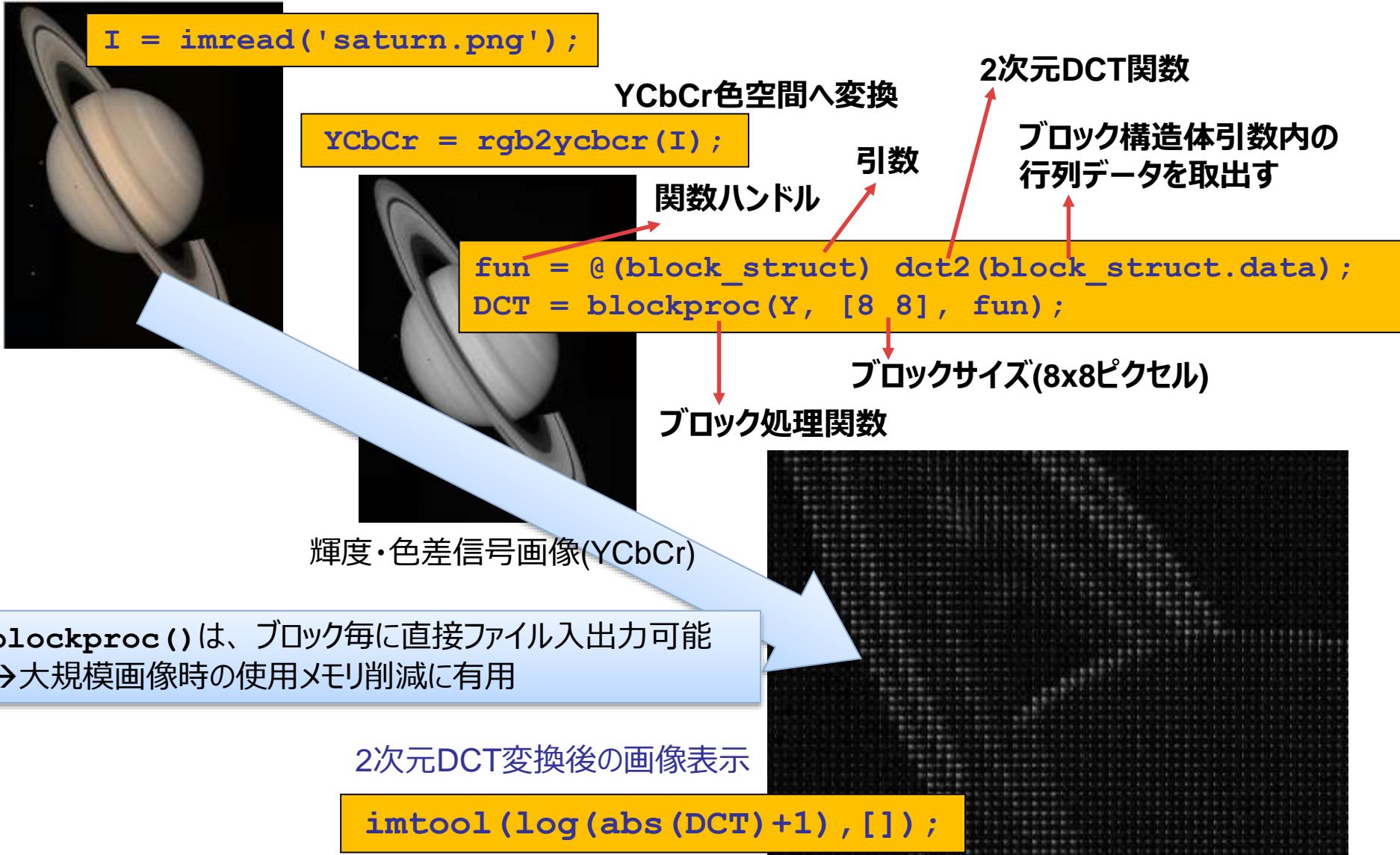
<code>blockproc()</code>	イメージの個別ブロック処理
<code>nlfilt()</code>	一般的なスライディング近傍演算
<code>colfilt()</code>	列方向の近傍演算
:	

## 2.10.1 ブロック処理例：DCT（離散コサイン変換）符号化



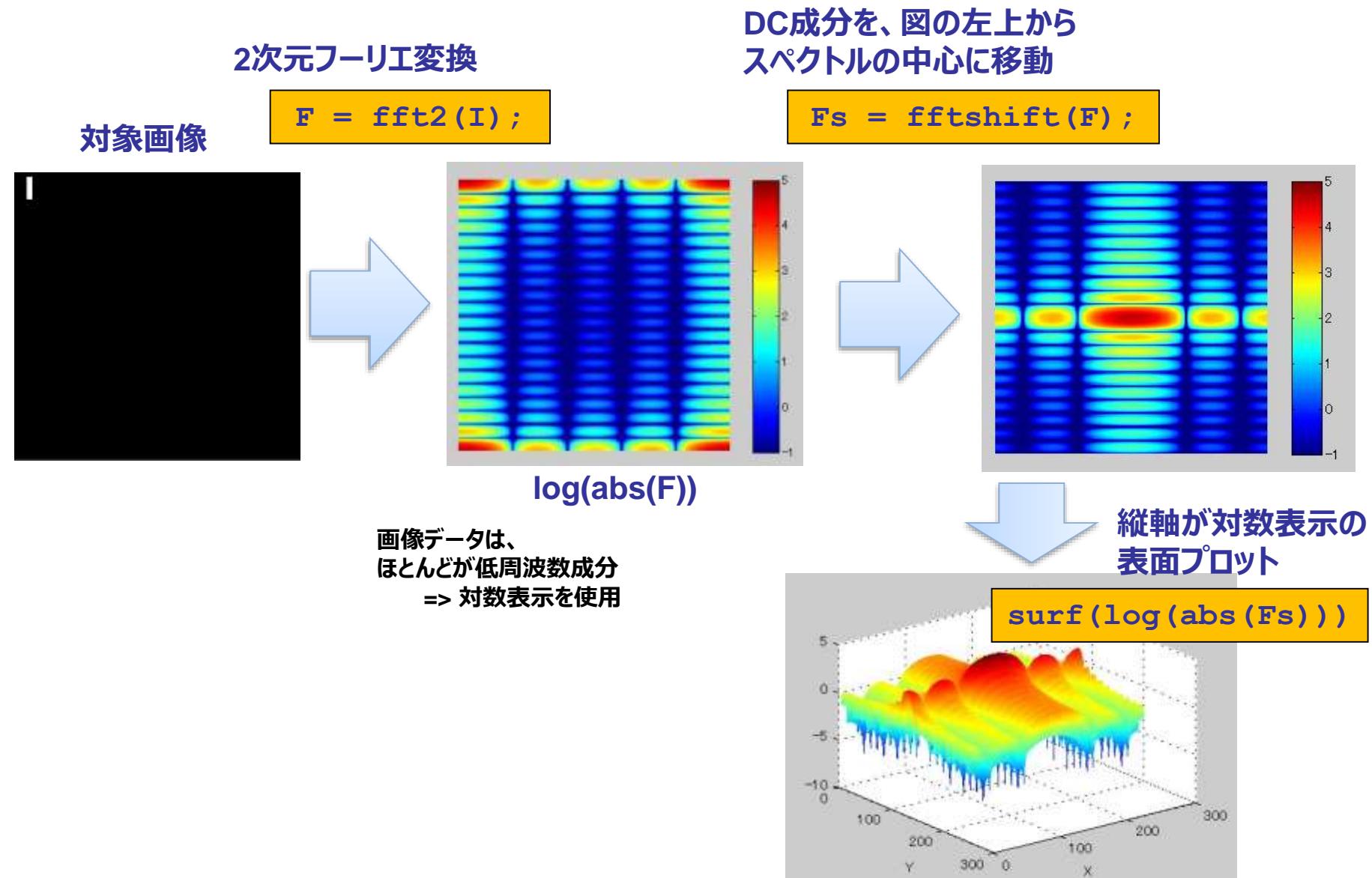
## 2.10.1 ブロック処理例：DCT（離散コサイン変換）符号化

原画像 (RGB)



`blockproc()`は、ブロック毎に直接ファイル入出力可能  
→大規模画像時の使用メモリ削減に有用

## 2.10.2a 2次元高速フーリエ変換 (ML基本関数)

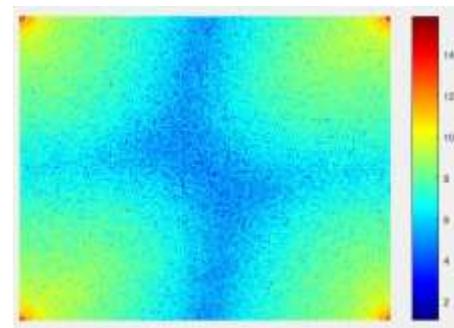


## 2.10.2b 位相画像

2次元フーリエ変換

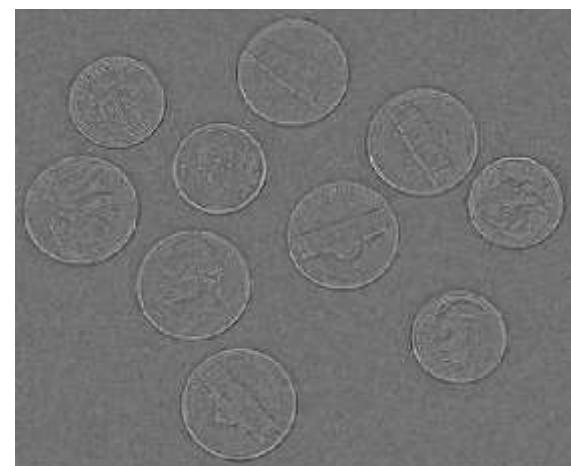
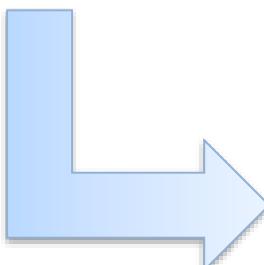
対象画像

`F = fft2(G);`



`log(abs(F))`

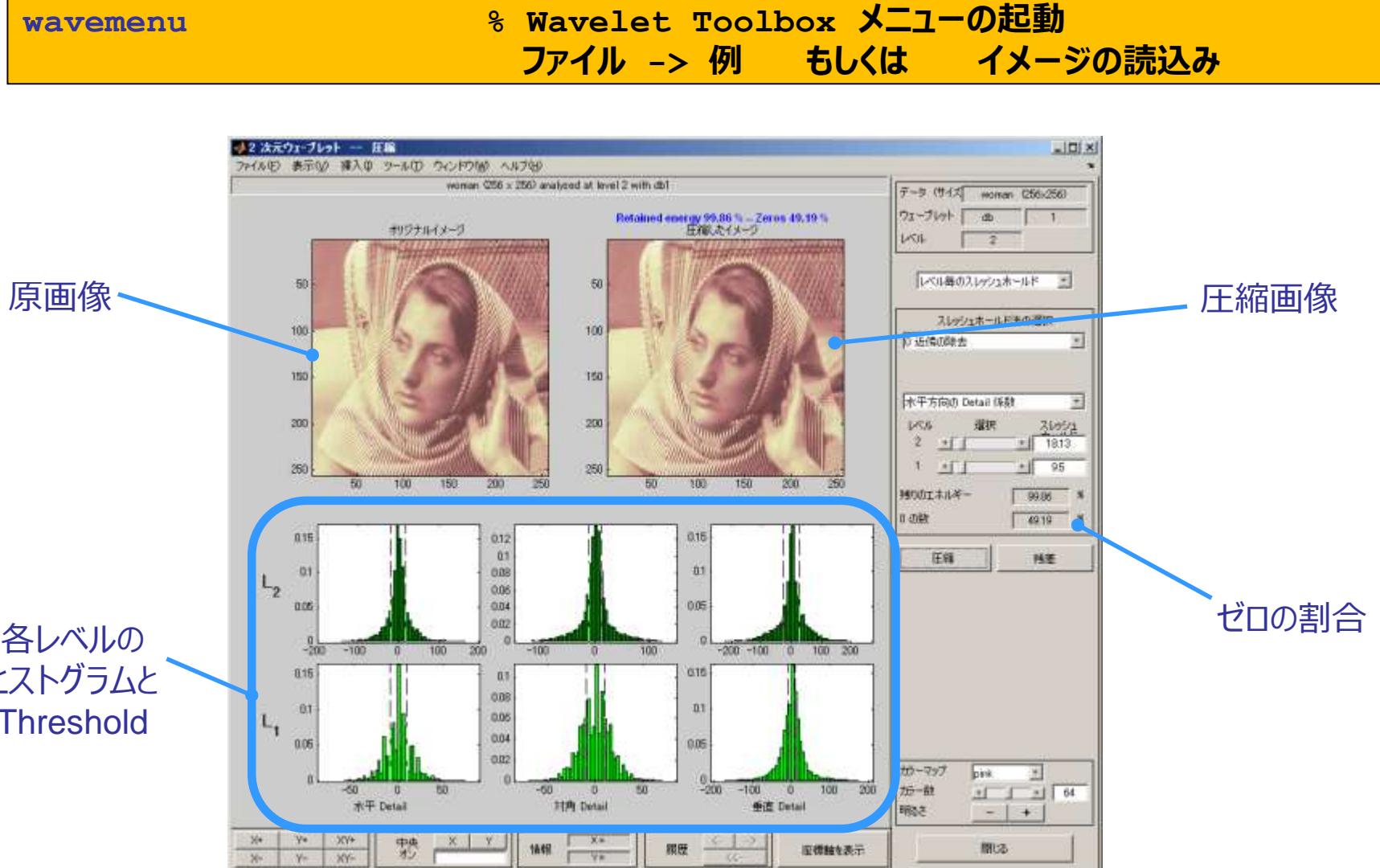
位相画像



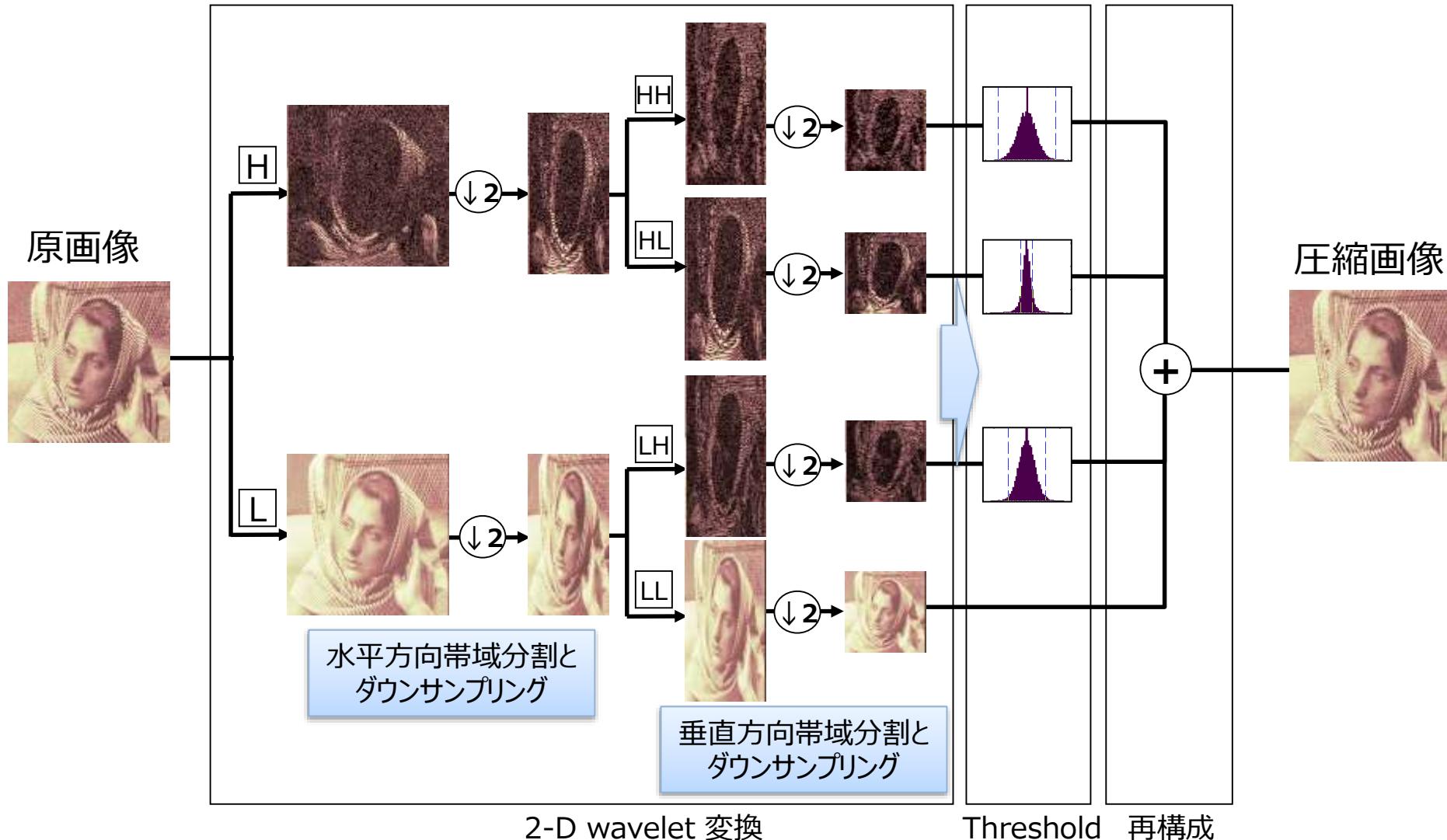
`Fn = F ./ abs(F); % 各成分値を振幅で正規化`  
`Gr = ifft2(Fn); % 逆フーリエ変換`

## 2.10.3 2次元Wavelet変換による圧縮

Wavelet Toolbox



## 2.10.3 2次元Wavelet変換による圧縮例



## 2.11 テクスチャ解析

### テクスチャ解析用の関数

`entropy()` グレースケール イメージのエントロピーを計算

`stdfilt()` イメージの局所的な標準偏差を計算

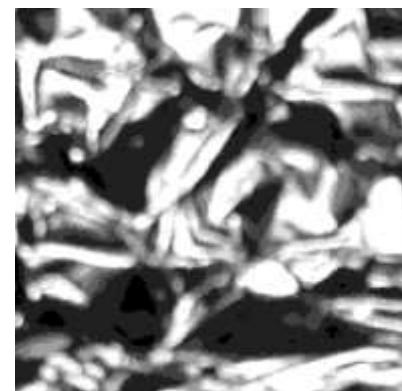
`graycomatrix()` イメージからグレーレベルの同時生起行列を作成

⋮

### 同時生起行列

(GLCM: Gray Level Co-occurrence Matrix)

例)



画素の値として1~8を持つ  
グレースケール画像

(隣り合う画素の明るさの関係  
テクスチャの違いの表現に適している)

GLCM

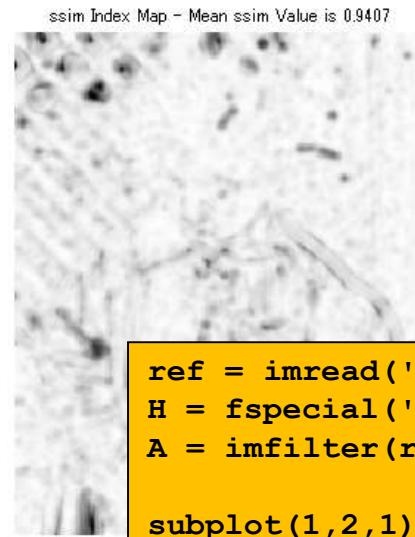
	1	2	3	4	5	6	7	8
1	1	2	0	0	1	0	0	0
2	0	0	1	0	1	0	0	0
3	0	0	0	0	1	0	0	0
4	0	0	0	0	1	0	0	0
5	1	0	0	0	0	1	2	0
6	0	0	0	0	0	0	0	1
7	2	0	0	0	0	0	0	0
8	0	0	0	0	1	0	0	0

水平方向に隣接する  
画素値が1-2の組が2つ

## 2.12 画質の定量評価

**ssim / psnr / immse**

R2014a  
R2014b



2枚の画像の平均二乗誤差を算出

```
err = immse(B, ref);
fprintf('The mean-squared error is %0.4f\n', err);

The mean-squared error is 348.6405
```

```
B = imnoise(ref, 'salt & pepper', 0.02);
figure, imshow(B);
```

2枚の画像の構造的類似性を算出

```
ref = imread('pout.tif');
H = fspecial('Gaussian',[11 11],1.5);
A = imfilter(ref,H,'replicate');

subplot(1,2,1); imshow(ref); title('Reference Image');
subplot(1,2,2); imshow(A); title('Blurred Image');

%Structural Similarity Index
[ssimval, ssimmap] = ssim(A,ref);
```

## 2.13 画像の一括（バッチ）処理

同一の処理を、多くの画像に適応

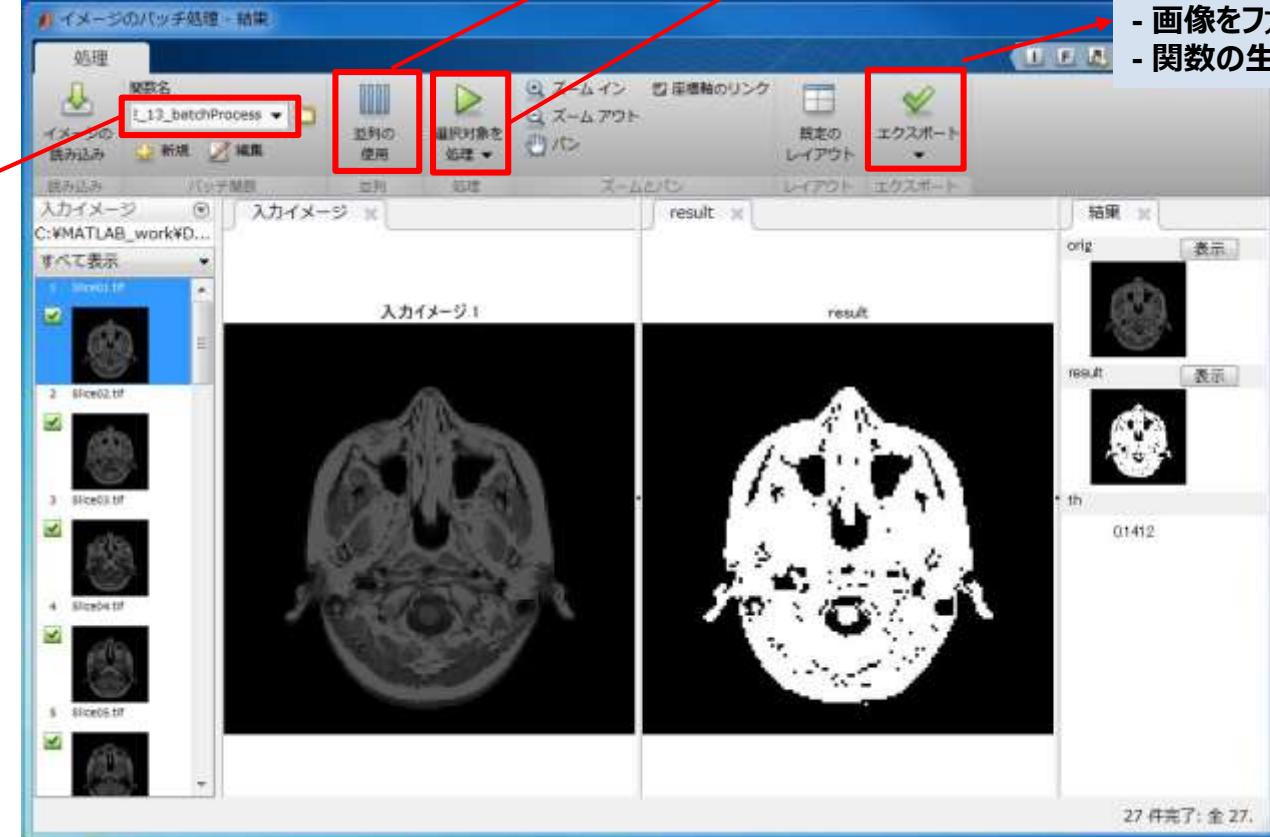
イメージのバッチ処理アプリケーション  
 >> `imageBatchProcessor`

並列処理  
 (Parallel Computing Toolbox必要)

選択対象を処理  
 もしくは  
 全ての画像を処理

R2015a  
 R2016a

適用する処理の  
 関数名



- ワークスペースへエクスポート
- 画像をファイルへエクスポート
- 関数の生成

## 2.14 画像品質の評価・解析

R2017b



**brisque (I)** R2017b

BRISQUE score is

20.6586.

52.6074

47.7553

BRISQUE

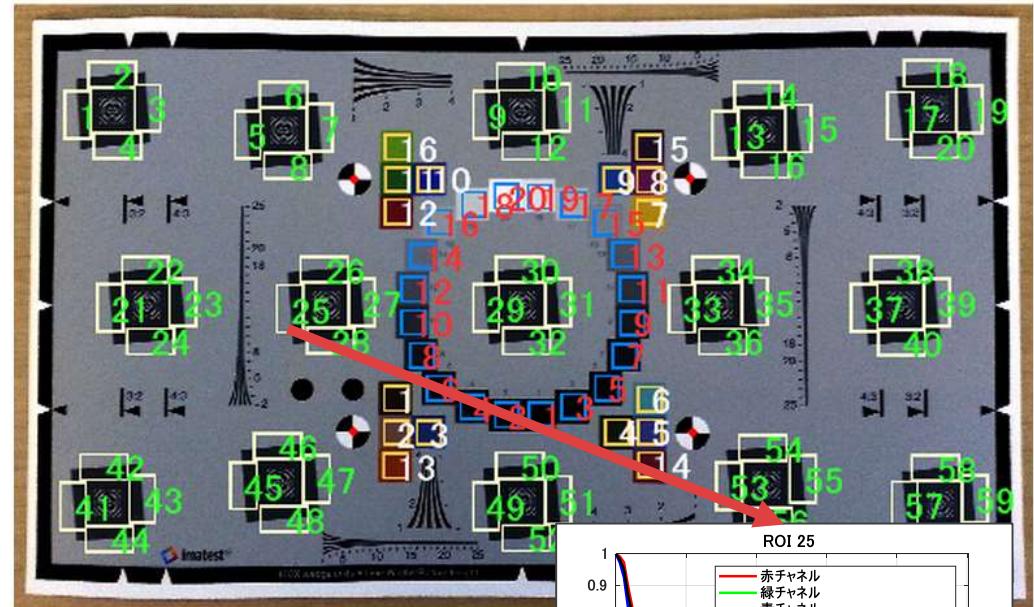
= Blind/Referenceless Image Spatial Quality Evaluator

画像単体で画質を評価する手法

**piqe** R2018b

Perception based Image Quality Evaluator

画像単体で画質を評価する手法



**esfrChart** R2017b

Imatest® edge spatial frequency response  
のチャート表示と評価項目の測定

## 2.15.1 大規模画像(bigimage)の操作

### メモリに納まらない大規模画像のための処理関数群

- 解像度によってレベル分け
- 各レベルで、ブロックごとに画像を分割

#### bigimage用の処理関数

`bigimage()` bigimageオブジェクトの作成

`bigimageshow()` bigimageの表示

`apply()` ブロックごとの関数の適用

`isequal()` 2つのbigimageが等価かどうかを調べる

`getBlock()` 指定した位置、レベルのブロック画像を取得

`getRegion()` 指定した範囲、レベルの画像を取得

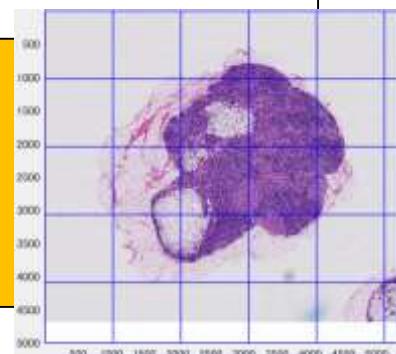
`getFullLevel()` 指定したレベルの画像を取得

`setBlock()` 指定したブロックにデータを挿入

`write()` bigimageのファイルへの書き込み

- ブロックごとの逐次読み込みによる処理
- 素早い表示
- マスクによる条件付き処理の実行

```
%% bigimageオブジェクトの作成
bim=bigimage('tumor_091R.tif');
%% 表示
Bshow = bigimageshow(bim, ...
'GridVisible','on','GridLevel',1);
```



## 2.15.2 bigimageDatastore

- 1枚ないし複数枚のbigimageについて、ブロックごとに画像を管理

### bigimageDatastore用の処理関数

<code>combine()</code>	複数のデータストアを結合	<code>readRelative()</code>	ブロック間の相対位置でデータを読み込み
<code>numpartitions()</code>	データストアの区画数	<code>reset()</code>	初期状態にリセット
<code>partitions()</code>	データストアを分割	<code>shuffle()</code>	シャッフル
<code>preview()</code>	サブセットを取得	<code>hasdata()</code>	read関数で読めるデータがあるかどうかを調べる
<code>read()</code>	データの読み込み	<code>transform()</code>	データストア内のブロック画像ごとに関数を適用

### %% bigimageデータストアの利用

```
Bimds=bigimageDatastore(bim, 2, 'BlockSize', [512 512]);
```

### %% 読み込むブロック数の指定

```
bimds.ReadSize = 4;
```

### %% 読み込み

```
blocks = read(bimds);
```

### %% 表示

```
montage(blocks, 'Size', [1, bimds.ReadSize], 'BorderSize', 5, 'BackgroundColor', 'k');
```



## 2.16 水増し用(data augmentation)の各種関数

### 深層学習時に使える様々な画像変換



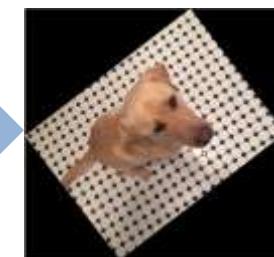
%% 色のランダムな変換

```
J = jitterColorHSV(I, 'Contrast', 0.4, ...
'Hue', 0.1, 'Saturation', 0.2, 'Brightness', 0.3);
```



%% ランダムなアフィン変換(2次元/3次元)

```
tform1 = randomAffine2d('Rotation', range);
J = imwarp(I, tform1);
```



%% ランダムな切り抜き窓の生成(2次元/3次元)

```
win = randomCropWindow2d(size(I), targetSize);
J = imcrop(I, win);
```



%% 画像中心の切り抜き窓の生成(2次元/3次元)

```
win = centerCropWindow2d(size(I), targetSize);
J = imcrop(I, win);
```



# アジェンダ

1. MATLAB/Simulinkの概要
2. 各種画像処理例
3. 連携機能
4. コンピュータービジョン処理例
5. 画像の機械学習・ディープラーニング
6. まとめ

## 3.1 並列処理・GPGPUによる高速・大規模画像処理 Parallel Computing Toolbox

- 並列・分散処理による高速化
- マルチコア、マルチCPUによる並列高速処理
- MATLAB 関数のGPUによる高速処理
  - 2次元FFT等 150以上のMATLAB基本関数
  - Image Processing Toolboxの50以上の関数

```
imrotate(), imfilter(), imdilate(), imerode(), imopen(), imclose(),
imtophat(), imbothat(), imshow().....
```
- 既存CUDA®コードの取込み
- MATLAB Distributed Computing Serverとの併用によりコンピュータークラスタの使用が可能

サポートされるGPU (CUDAに対応した NVIDIA® GPU) の要件：  
<http://www.mathworks.com/products/parallel-computing/requirements.html>

### 3.1.1 マルチコア/プロセッサでの並列処理

Parallel Computing Toolbox

- 独立したタスクの並列処理

```
%>> 2次元離散コサイン変換 (Y成分とCb成分)
```

```
for i = 1:2  
    DCT(:,:,i) = blockproc(YCbCr(:,:,i), [8 8], fun);  
end
```

forをparforに変更して処理の並列化

```
%>> 2次元離散コサイン変換 (Y成分とCb成分)
```

```
parfor i = 1:2  
    DCT(:,:,i) = blockproc(YCbCr(:,:,i), [8 8], fun);  
end
```

例  
11.5sec  
↓  
6.5sec  
(-44%)  
に短縮

2コアCPUの例

- 各種組込み関数でのサポート (PCT オプション要)：  
<http://www.mathworks.co.jp/products/parallel-computing/builtin-parallel-support.html>

### 3.1.1 マルチコア/プロセッサでの並列処理

Parallel Computing Toolbox

- 各種組込み関数でのサポート (PCT オプション要) :

<http://www.mathworks.co.jp/products/parallel-computing/builtin-parallel-support.html>

```
%% 2次元 DCT 処理を、関数ハンドル'fun'として定義
fun = @(block_struct) dct2(block_struct.data);

%% 画像 Y を 8x8 ピクセルに分割し、それぞれに対し処理 'fun' を実行
%      'UseParallel' オプションを使うことで処理を並列化
DCT = blockproc(Y, [8 8],fun, 'UseParallel', true);
```

I2\_11\_1\_dct\_blockproc.m

## 3.1.2 GPGPU処理

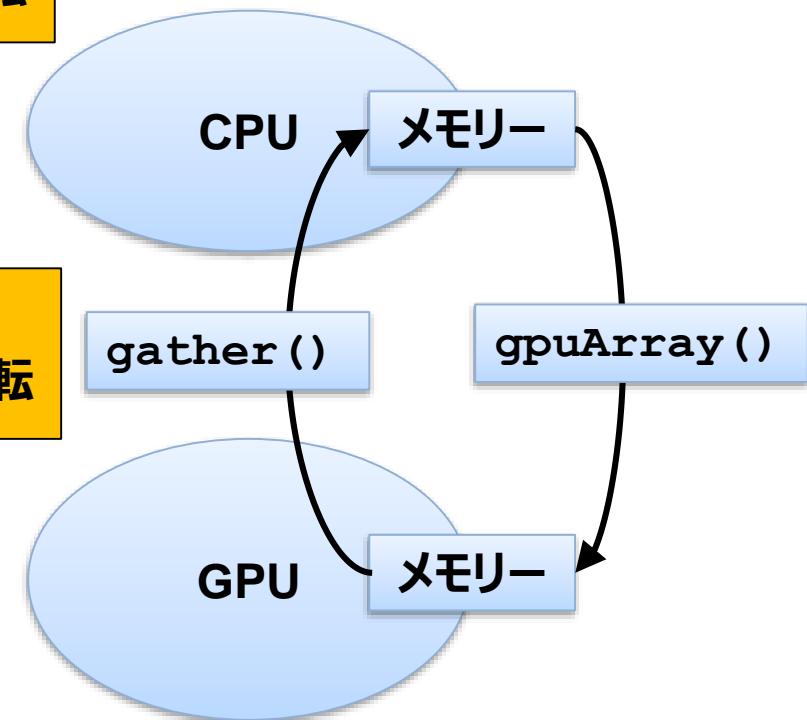
Parallel Computing Toolbox

各種関数を、GPU上で実行

```
I = imrotate(I, 75, 'bicubic'); % 画像を75°回転
```



```
I = gpuArray(I);  
I = imrotate(I, 75, 'bicubic'); % 画像を75°回転
```



わずかなコード変更で、高速化が可能

### 3.1.3 基本関数での実装

Parallel Computing Toolbox

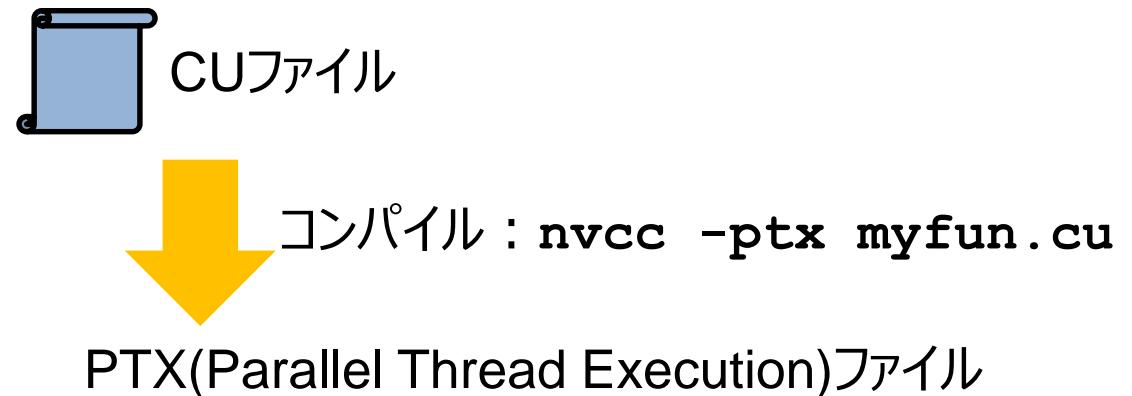
GPU用のarrayfun、bsxfunを用い、入力引数の各要素に関数を適応  
(スカラー計算をGPUで並列化)

```
hornerFcn = @horner;  
  
data1 = rand( 2000, 'single' );  
gdata1 = gpuArray( data1 );  
  
gresult1 = arrayfun( hornerFcn, gdata1 );
```

pagefun : 2次元の行列の演算をバッチで処理

### 3.1.4 既存CUDAコードの取り込み

Parallel Computing Toolbox



% カーネルオブジェクト作成

```
k = parallel.gpu.CUDAKernel('myfun.ptx', 'myfun.cu');
```

```
i1 = gpuArray(rand(100, 1, 'single'));  
o1 = feval(k, i1, i2);
```

% 結果をMATLABワークスペースへ転送

```
r1 = gather(o1);
```

**mexcuda**関数を使う方法もあり

## 3.1.5 CUDA MEXの統合

```

2    %% 画像データ読み込み
3    imCPU = imread('concordaerial.png');
4
5    %% データをGPUに転送
6    imGPU = gpuArray(imCPU);
7
8    %% グレースケール変換
9    imGPUgray = rgb2gray(imGPU);
10
11   %% 2値化
12   imWaterGPU = imGPUgray<70;
13
14   %% 細かいノイズ除去
15   imWaterMask = imopen(imWaterGPU,strel('disk',4));
16   imWaterMask = bwmorph(imWaterMask,'erode',3);
17   imshow(imWaterMask)
18
19   %% ガウシアンフィルタで画像をぼかす
20   blurH = fspecial('gaussian',20,5);
21   imWaterMask = imfilter(single(imWaterMask)*10, blurH);
22
23   %% 青色の要素を強調
24   blueChannel = imGPU(:,:,3);
25   blueChannel = imlincomb(1, blueChannel,6, uint8(imWaterMask));
26   imGPU(:,:,3) = blueChannel;
27
28   %% CPUにデータを転送し、結果を表示
29   outCPU = gather(imGPU);
30   imshow(outCPU)

```

GPU対応関数で処理  
(Parallel Computing Toolbox)

CUDA MEXを生成して処理  
(GPU Coder)

GPU対応関数で処理  
(Parallel Computing Toolbox)

Parallel Computing Toolbox  
GPU Coder

R2018b

gpuArray型を関数入力として指定することで  
無駄なデータ転送(CPU<>GPU)を回避し高速化

```

cfg = coder.gpuConfig('mex');
cfg.TargetLang = 'C++';

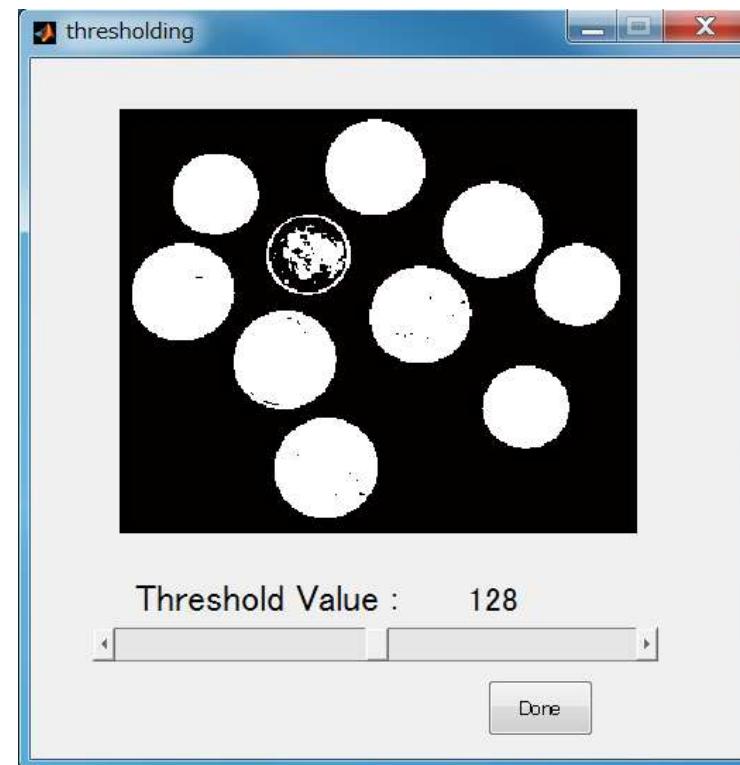
t = coder.typeof(gpuArray(false), [2036 3060]);

codegen -args {t} -config cfg myfilter

```

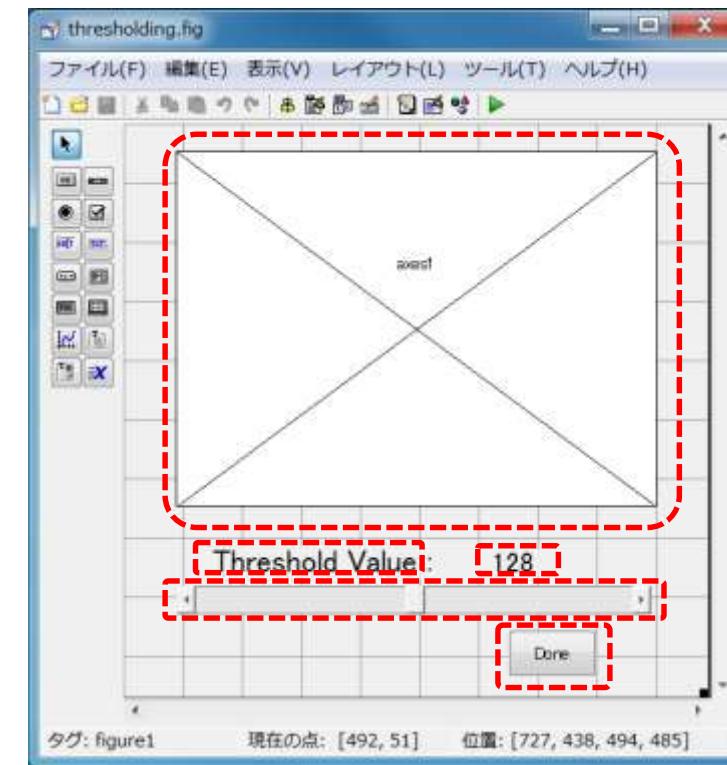
## 3.2 ユーザーアンターフェース構築機能 (MATLAB基本機能)

ユーザーインターフェース  
構築例 (2値化)



ユーザーインターフェース  
設計環境 (GUIDE)

>> guide



thresholding.fig

## 3.2 ユーザーアンターフェース構築機能

thresholding.m

```
function thresholding_OpeningFcn(hObject, eventdata, handles, varargin)
```

.....

%% ユーザーアンターフェースが立ち上がった時の処理を記述

```
handles.in = varargin{1};
```

% 関数コールの引数から入力画像を取得

```
imshow(handles.in > 128);
```

% 初期閾値(128)で2値化画像表示

```
function slider1_Callback(hObject, eventdata, handles)
```

%% スライダーを動かした際の動作を記述

```
svalue = get(hObject, 'Value');
```

% スライダーの値を取得

```
set(handles.text2, 'string', svalue);
```

% スライダー値を表示

```
handles.I = handles.in > svalue;
```

% 2値画像を生成

```
imshow(handles.I);
```

% 2値画像を表示

```
guidata(hObject, handles);
```

% 保持している生成画像をUpdate

```
function pushbutton1_Callback(hObject, eventdata, handles)
```

%% Done ボタンを押したときの動作を記述

% MATLABのワークスペース ('base') へ、OUTという変数名で保存

```
assignin('base', 'OUT', handles.I);
```

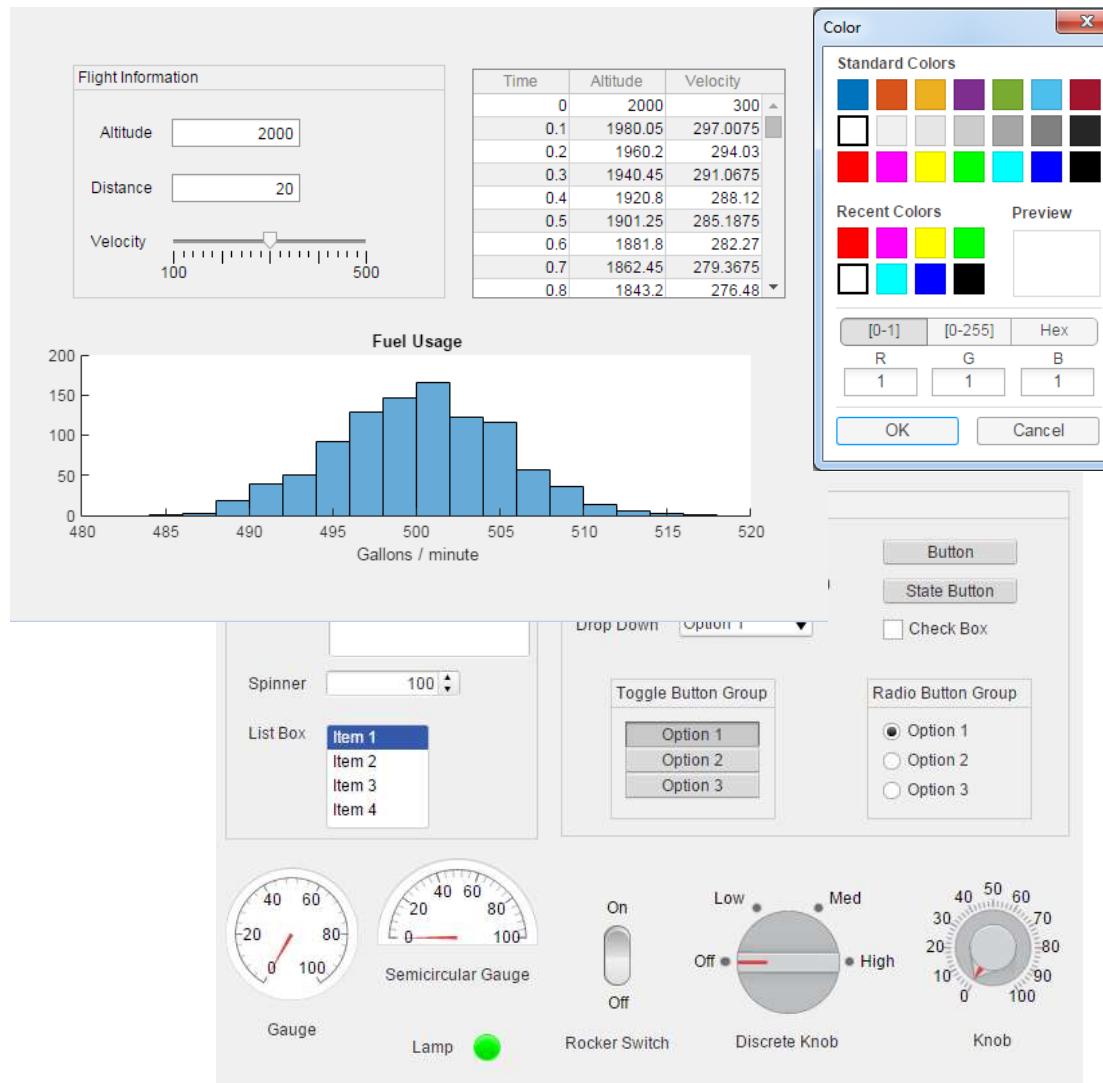
マニュアル : <http://jp.mathworks.com/help/matlab/gui-development.html>

## 3.2 ユーザーアンターフェース構築機能(AppDesigner)

R2016a

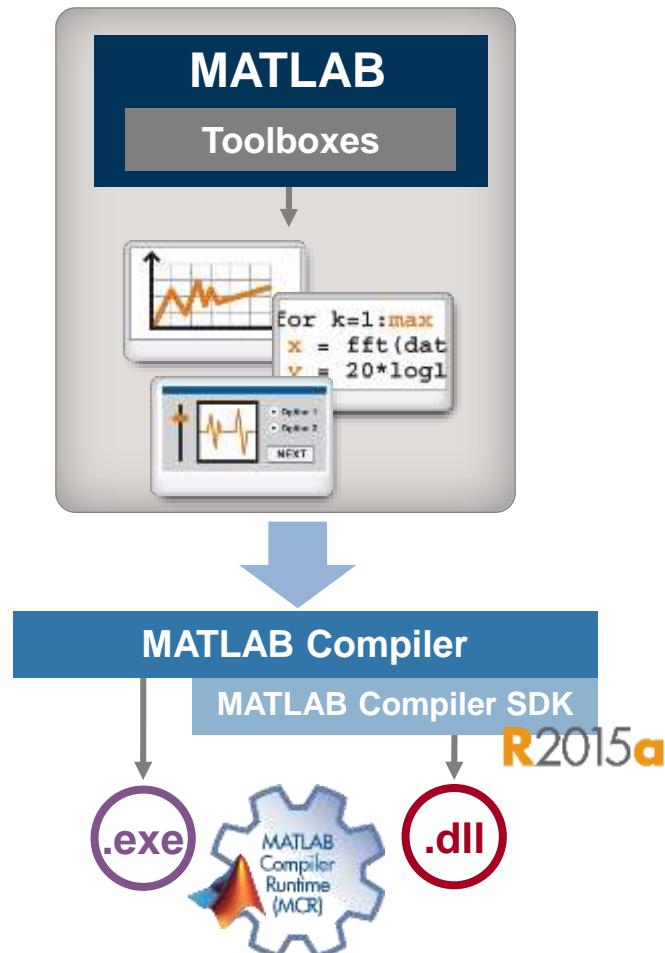
>> appdesigner

- Guide機能に比べて、選べるコンポーネントが増加
- スマートガイド機能
  - 位置合わせが容易に
- ツールストリップ上で選べる豊富な編集機能
- GUIレイアウトとプログラミングが同一環境で可能
- Classをベースにしたコード生成



### 3.3 実行形式プログラムの配布

MATLAB Compiler



ほぼすべての関数をサポート・GUIやFigureも対応

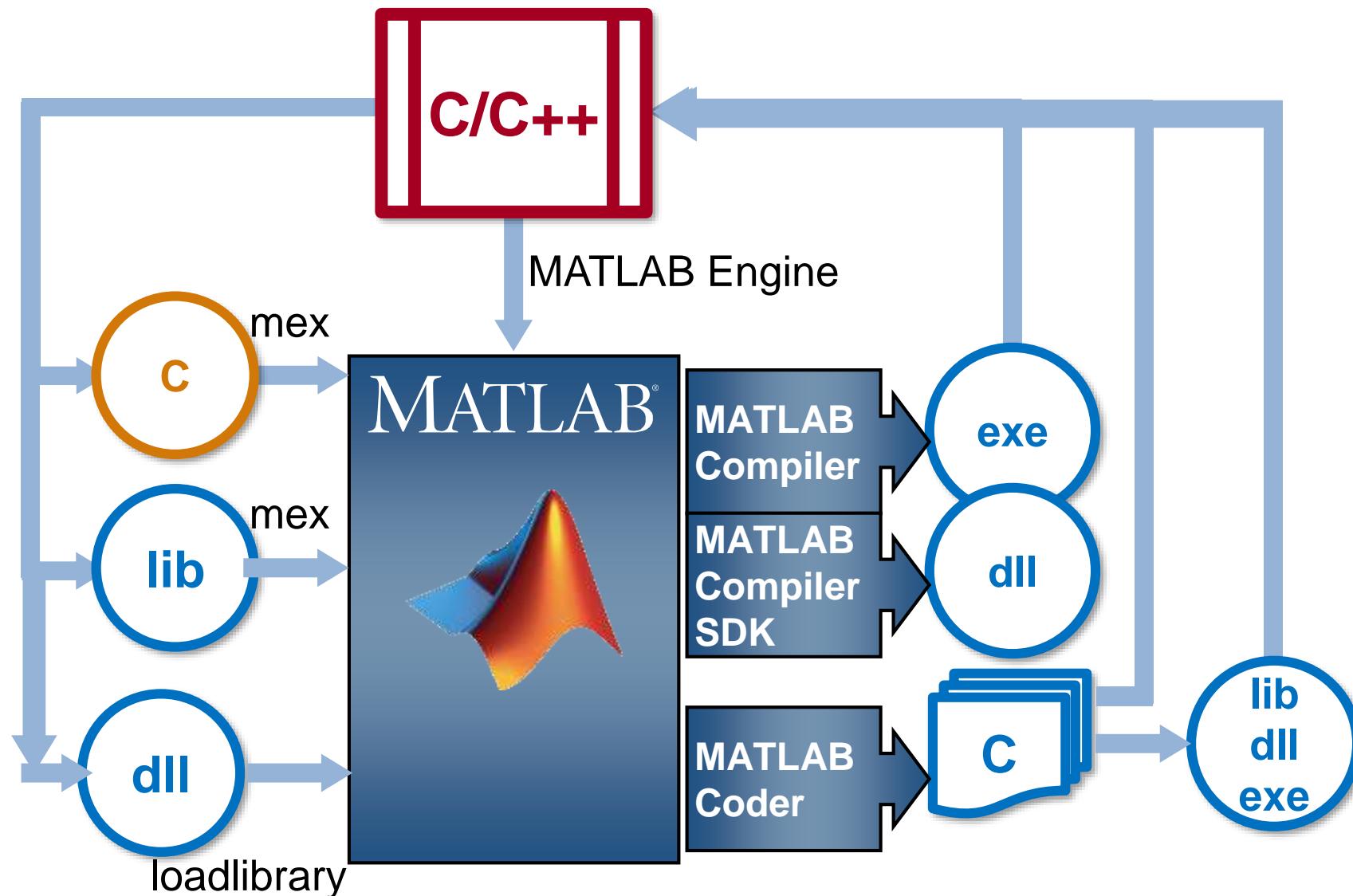
制限：<http://www.mathworks.co.jp/help/compiler/limitations-about-what-may-be-compiled.html>

対応Toolbox関数：[http://www.mathworks.co.jp/products/compiler/supported/compiler\\_support.html](http://www.mathworks.co.jp/products/compiler/supported/compiler_support.html)

R2016aからは、64bitのみサポート



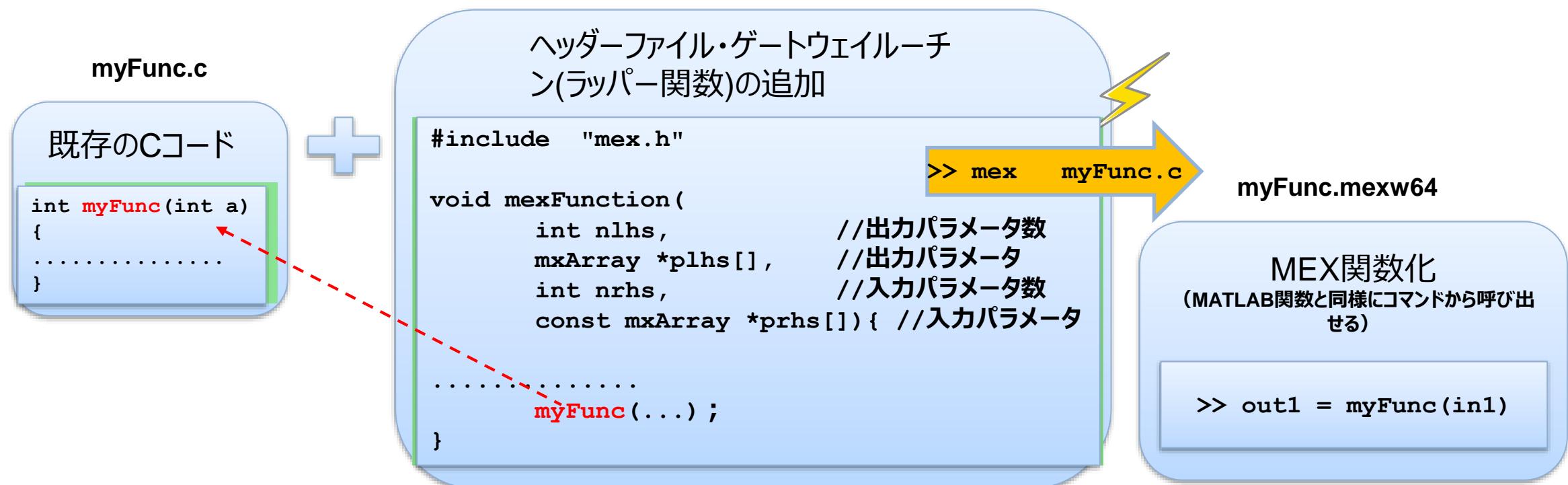
## 3.4 Cコードとの相互連携



## 3.4.1 Cコードの取り込み (MATLAB基本機能)

外部Cコードの使用：

MEX (MATLAB Executable)機能を使い、既存のC関数やFortran関数などをMATLABの関数と同様にコマンドラインから呼び出し、実行可。MATLAB本体、ツールボックスでサポートされていない機能・関数を既存やオープンソース・コード等で補うことが可能。



マニュアル：<http://www.mathworks.com/help/matlab/build-cc-mex-files.html>

## 3.4.1 Cコードの読み込み例 (MATLAB基本機能)

### ラッパー関数

```
void mexFunction(int nlhs,          //出力パラメータ数
                  mxArray *plhs[],    //出力パラメータへのポインタ配列
                  int nrhs,           //入力パラメータ数
                  const mxArray *prhs[]) { //入力パラメータへのポインタ配列

    char *fileName;                      //第1引数
    double *format;                     //第2引数
    unsigned short height;              //C関数からの戻り値
    unsigned short width;               //C関数からの戻り値
    unsigned short *pOutRaw;            //C関数からの戻り値

    fileName = mxArrayToString(prhs[0]); //第1引数を取り込み
    format = mxGetPr(prhs[1]);         //第2引数を取り込み

    // C関数を呼ぶ *****
    read_raw_main(fileName, (int)*format, &height, &width, &pOutRaw);
}
```

MATLAB独自のデータ構造体  
と  
Cのデータ型  
の変換用のAPI関数を提供

## 3.4.2 OpenCVで作られた関数へのアクセス

R2014b

Computer Vision Toolbox

mexOpenCV matchTemplateOCV.cpp % OpenCVを用いた関数からのMEX生成

呼び出す関数にあわせて記述したラッパー

用意されている、MATLABのデータ型と、OpenCVの型 を相互に変換するAPI例

```
// Convert mxArray inputs into OpenCV types
cv::Ptr<cv::Mat> templateImgCV = ocvMxArrayToImage_uint8(prhs[0], true);
```

説明のショートビデオ（約9分）：<http://jp.mathworks.com/videos/using-opencv-with-matlab-106409.html>

OpenCV の対応バージョンが 3.1.0 へ変更

R2017a

OpenCV の対応バージョンが 3.4.0 へ変更

R2018b

## 3.5 MATLAB Engine : C開発環境からのMATLAB呼び出し

Cコード例

```
#include "engine.h"                                     // MATLAB Engine用のヘッダファイル
|
|
Engine *ep;                                         // MATLAB Engineへのポインタの宣言
mxArray *Igray = NULL;                             // MATLAB型の変数へのポインタの宣言

ep = engOpen(NULL);                                // MATLAB Engine のスタート

engEvalString(ep, "cd C:/MATLAB_work/demo");      // MATLABのワークディレクトへ移動
engEvalString(ep, "img = imread('Image.jpg');");    // 画像の読み込み
engEvalString(ep, "Igray = rgb2gray(img);");        // グレースケールへ変換
engEvalString(ep, "figure; imshow(Igray);");         // MATLAB上で画像の表示

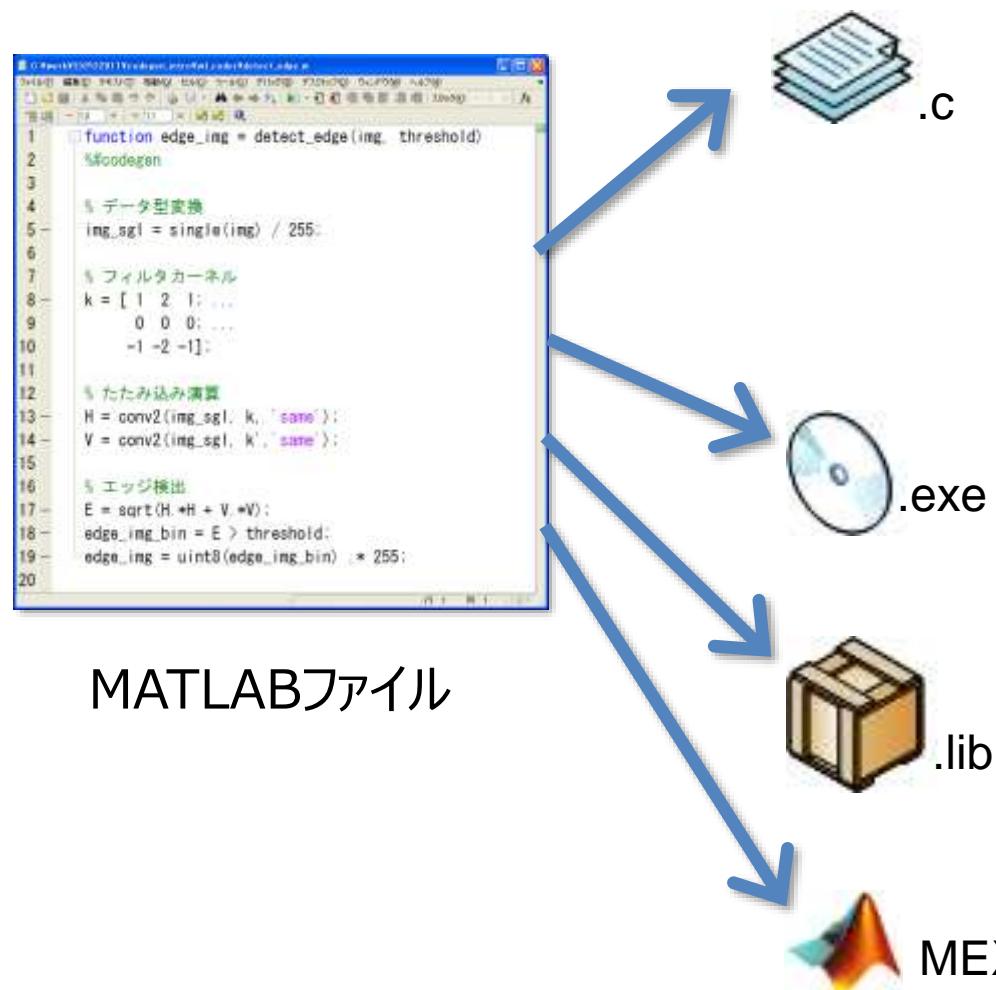
Igray = engGetVariable(ep, "Igray");                // MATLAB内の変数"Igray"へのポインタの取得
const uint8_t* Ip = static_cast<uint8_t*>(mxGetData(Igray)); // C変数へのポインタに変換
|
|
engPutVariable(ep, "output", output);               // MATLABへデータを送り、変数"output"とする
```

Cから、別プロセスのMATLABへ実行コマンドを指示

MATLABとCの間でのデータの受け渡し

## 3.5 MATLABコードからのCコード生成

MATLAB Coder



### コード生成/実装・アルゴリズム仕の リファレンスマル

- 実装(C/HDL)技術者へハンドオフ
- 整数・浮動小数点・  
固定小数点演算対応  
(Fixed-Point Designerが必要)

### スタンドアロンのアプリケーション作成 (プロトタイプ・配布)

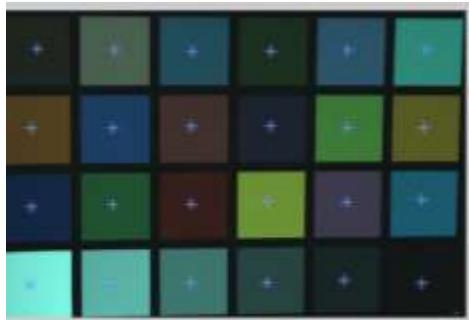
### 既存のCソフトウェアへ MATLABアルゴリズムの統合

### MATLABファイル実行の高速化

## 3.6.1 Excelとの連携 / カーブフィティング

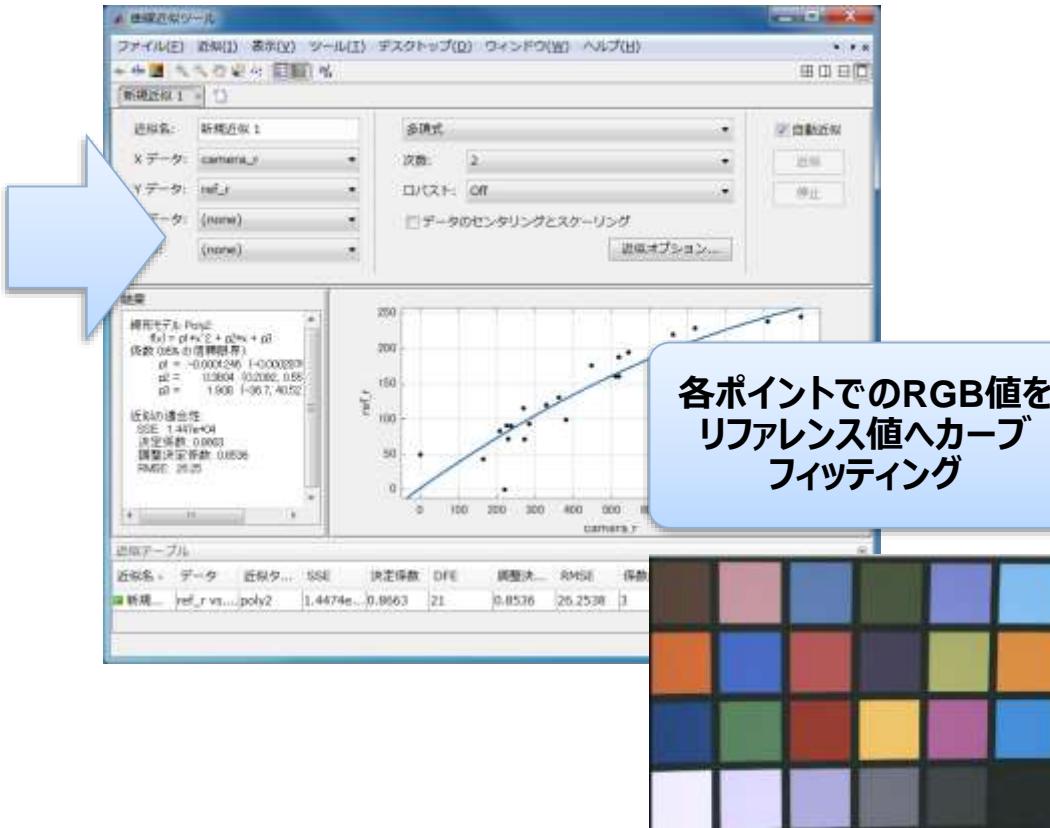
Spreadsheet Link EX  
Curve Fitting Toolbox

原画像



Excelデータ(リファレンスデータ)取り込み

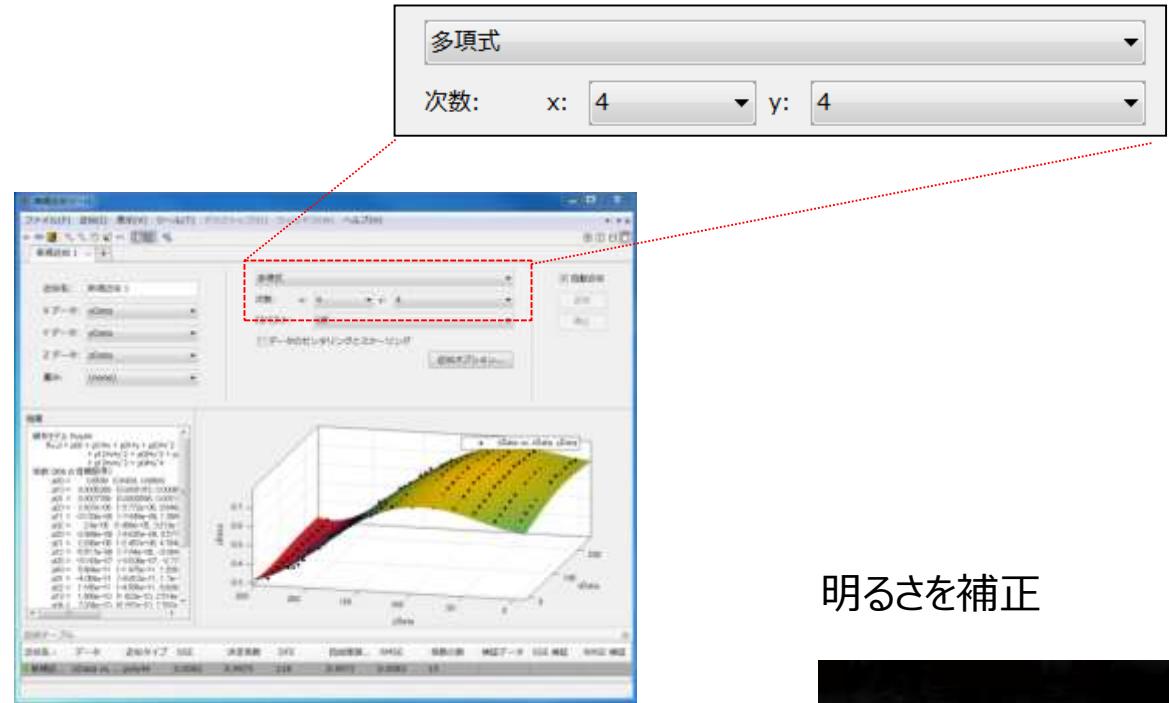
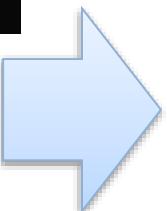
- 曲線近似GUIツール
- 曲線・曲面への線形回帰、非線形回帰
- スプライン 補間
- 平滑化 等



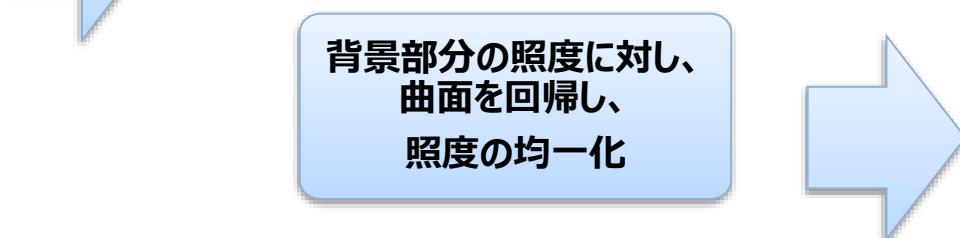
## 3.6.2 曲面へのフィティング

Curve Fitting Toolbox

不均一な照度が足しあわ  
されている場合



明るさを補正



例) 線形モデル Poly44:

$$\begin{aligned} f(x,y) = & p_{00} + p_{10}x + p_{01}y + p_{20}x^2 + p_{11}xy + p_{02}y^2 + p_{30}x^3 + p_{21}x^2y \\ & + p_{12}xy^2 + p_{03}y^3 + p_{40}x^4 + p_{31}x^3y + p_{22}x^2y^2 \\ & + p_{13}xy^3 + p_{04}y^4 \end{aligned}$$

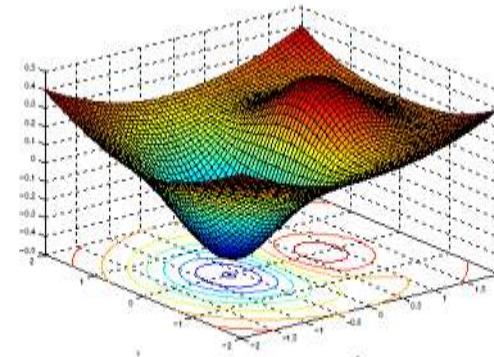
面の最大点等を求めるには最適化ツール

## 3.7 最適化ツール

### Optimization Toolbox

#### - 標準最適化問題および大規模最適化問題の解法

- 線形計画法
- 二次計画法
- 二値整数計画法
- 非線形最適化
- 非線形最小二乗
- 連立非線形方程式
- 多目的最適化

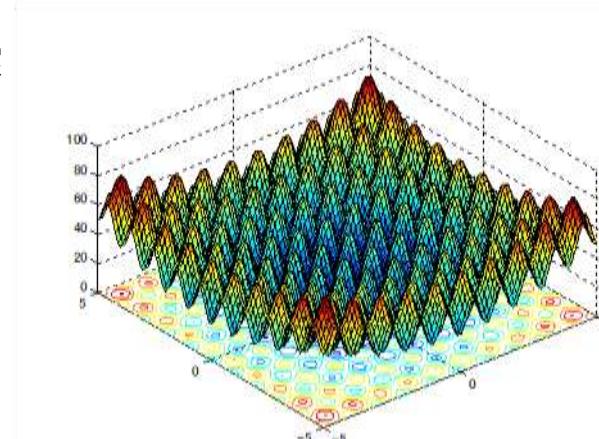


### Global Optimization Toolbox

#### - 大域的最適化問題

複数の極小値/極大値を持つ関数から最小値/最大値を探索

- グローバルサーチ
- マルチスタート
- パターンサーチ
- 遺伝的アルゴリズム
- 焼きなまし法



## 3.8 外部ハードウェアとのインターフェース

(カメラとの接続には Image Acquisition Toolbox を使用)

### Data Acquisition Toolbox

(Windowsのみ)

データ収集ボード・計測デバイス

アナログ入力(A/D), アナログ出力(D/A), ディジタルI/O(DIO)

サポートされるボードベンダ National Instruments, Analog Devices etc...

サポートされるハードウェアの詳細 : <http://www.mathworks.co.jp/hardware-support/data-acquisition-software.html>

### Instrument Control Toolbox

オシロスコープ、信号発生器その他計測機器との接続

他のPCや機器との接続

TCP/IP、UDP、Bluetooth (SPP: Serial Port Profile)、GPIB、SPI、I2C、RS-232 等

サポートされるハードウェアの詳細 : <http://www.mathworks.co.jp/hardware-support/instrument-control-software.html>

### Vehicle Network Toolbox

CANインターフェースハードウェア(PCI, USB接続等) のサポート。

MATLAB関数・Simulinkブロック

サポートされるハードウェアの詳細 : <http://www.mathworks.co.jp/hardware-support/can-bus-software.html>

### MATLAB

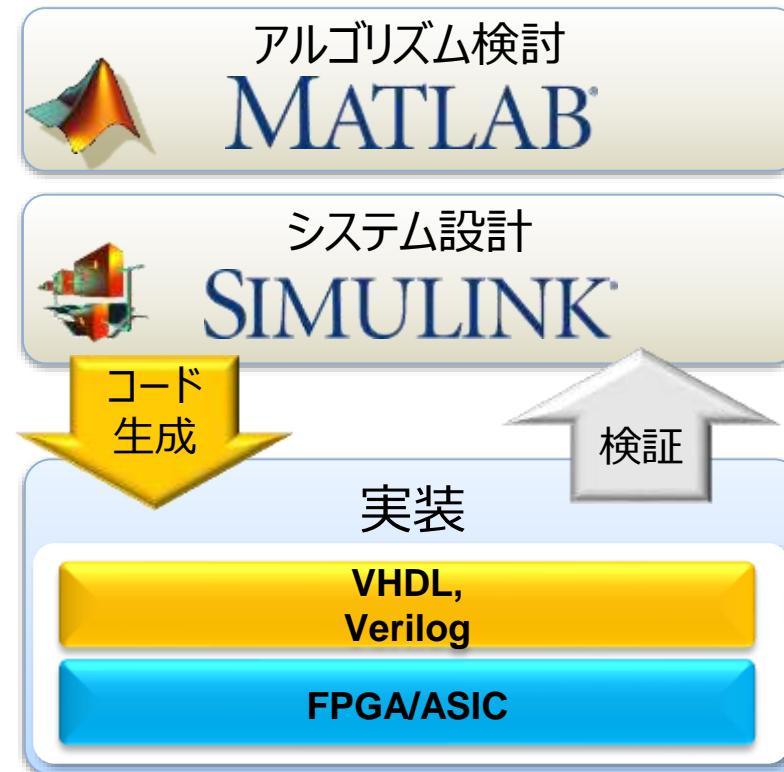
シリアルポート、上記以外のハードウェア

COM, WindowsドライバAPI 経由等

## 3.10.1 MATLABやSimulinkからのHDL自動生成

HDL Coder

- ターゲット依存しないVHDL/Verilog生成
  - VHDL(IEEE1076-1993)
  - Verilog(IEEE1364-2001)
- 業界標準コーディングルールチェック機能
- テストベンチ生成
  - VHDL/Verilogテストベンチ
  - Simulinkテストベンチモデル
- FPGAベンダツールとの連携
  - Xilinx® ISE®, Altera® Quartus® IIと連携
  - クリティカルパスの表示
- トレーサビリティレポートの生成
  - モデル-HDL-要求仕様間のリンク



## 3.10.2 固定小数点ツールによるオートスケーリング

Fixed-Point Designer

固定小数点化に要する工数を短縮

- モデルを自動的に固定小数点化・最適化
  - 浮動 $\Rightarrow$ 固定小数点化 : オーバーフロー、冗長ビットの削減
- 固定小数点最適化の方法
  - 1: 各信号の、必要な数値レンジを決定
    - 最大/最小値設定の下流への派生
    - シミュレーションによる最小/最大値を計測
  - 2: オーバーフローしない最短の整数部Bit長に調整
    - 語長は固定のBit数で、小数点の位置を調整（小数部ができるだけ長く：量子化誤差を最小化）
    - 小数部の長さは固定（固定の量子化誤差の条件で、語長を最短化）

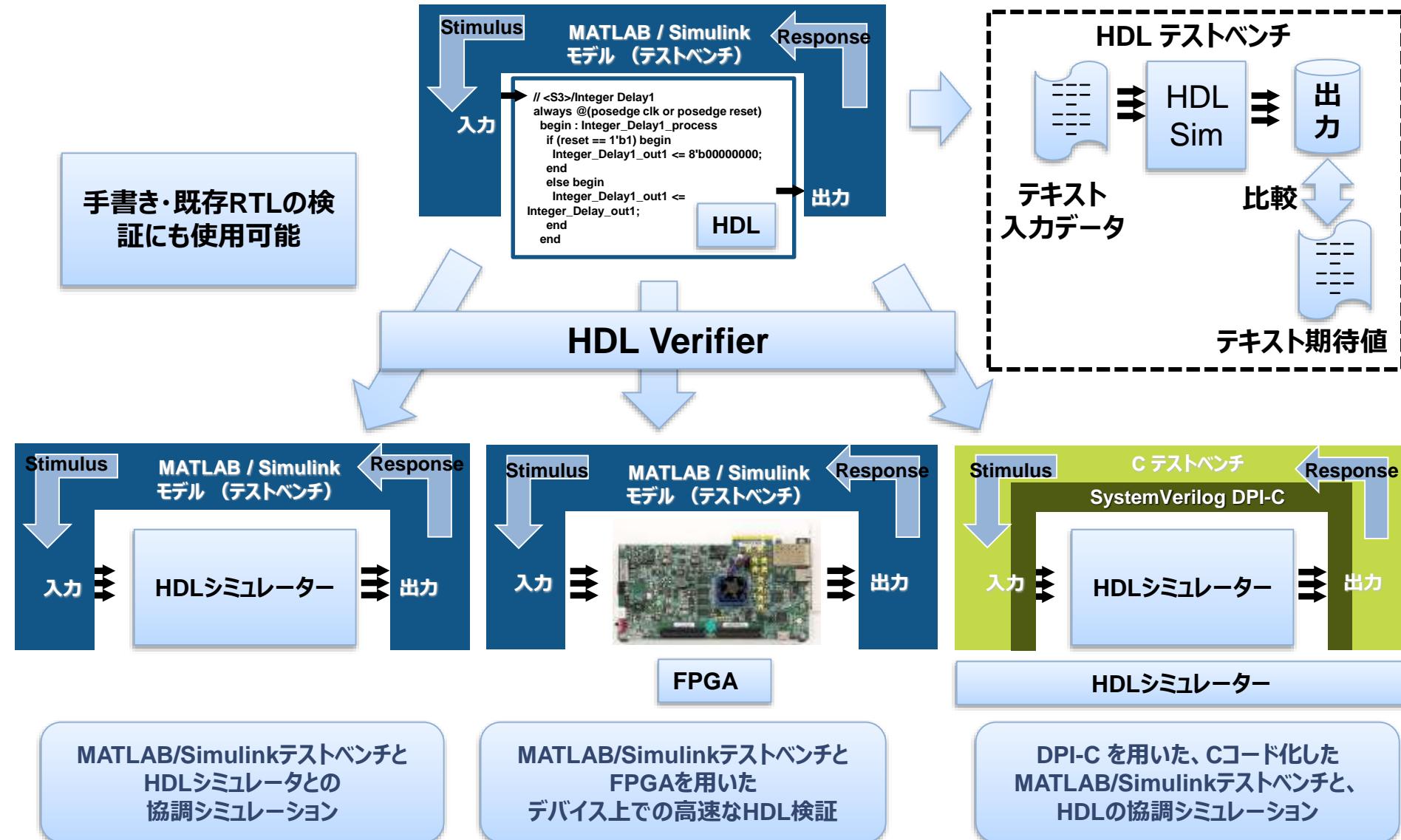
Name	Accept	ProposedDT	SpecifiedDT	DerivedMin	DerivedMax	SimMin	SimMax	Co
b(2)	<input checked="" type="checkbox"/>	fixdt(1,31,32)	fixdt(1,32,32)	-0.14850553...	0.1485806772...	-0.14859553...	0.14859100...	
b(1)	<input checked="" type="checkbox"/>	fixdt(1,29,32)	fixdt(1,32,32)	-0.04953184...	0.0495268924...	-0.04953184...	0.04953033...	
BodyLSum2 : Output	<input checked="" type="checkbox"/>	fixdt(1,31,32)	fixdt(1,40,32)	-0.19812738...	0.1981075696...	-0.19792482...	0.19796564...	
b(3)	<input checked="" type="checkbox"/>	fixdt(1,31,32)	fixdt(1,32,32)	-0.14859553...	0.1485806772...	-0.14859553...	0.14859100...	
BodyLSum3 : Output	<input checked="" type="checkbox"/>	fixdt(1,32,32)	fixdt(1,40,32)	-0.34672201...	0.3466802468...	-0.34614095...	0.34629816...	
H(4)	<input checked="" type="checkbox"/>	fixdt(1,30,32)	fixdt(1,31,32)	-0.0000000000...	0.0000000000...	-0.0000000000...	0.0000000000...	

オーバーフローの解消  
冗長ビットの削減

各信号名  
ツールが推奨した固定小数点型  
元の型  
解析的に求めた信号値のレンジ  
シミュレーションで求めた信号値のレンジ

### 3.10.3 HDL検証ソリューション

HDL Verifier



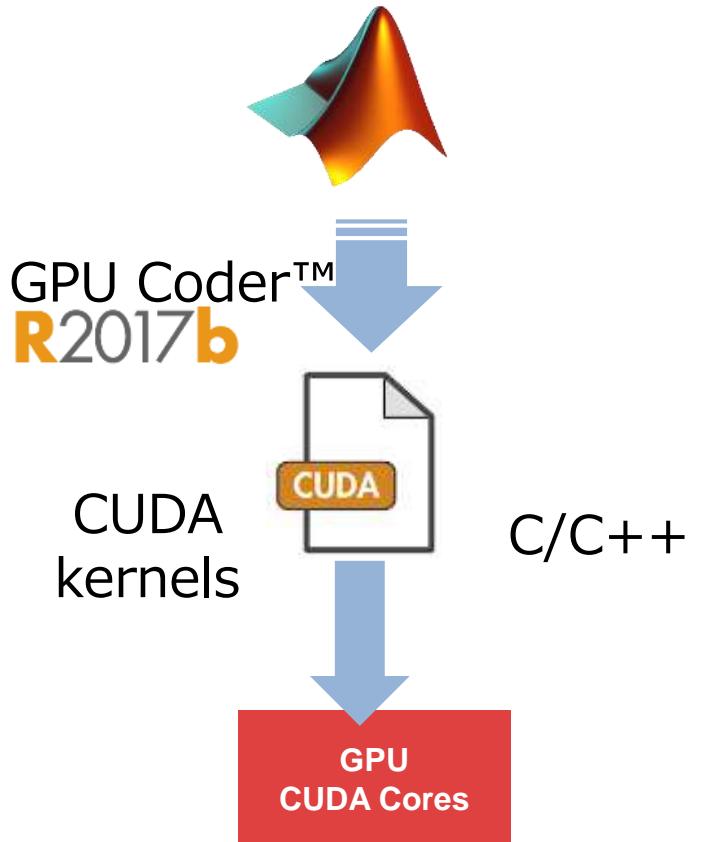
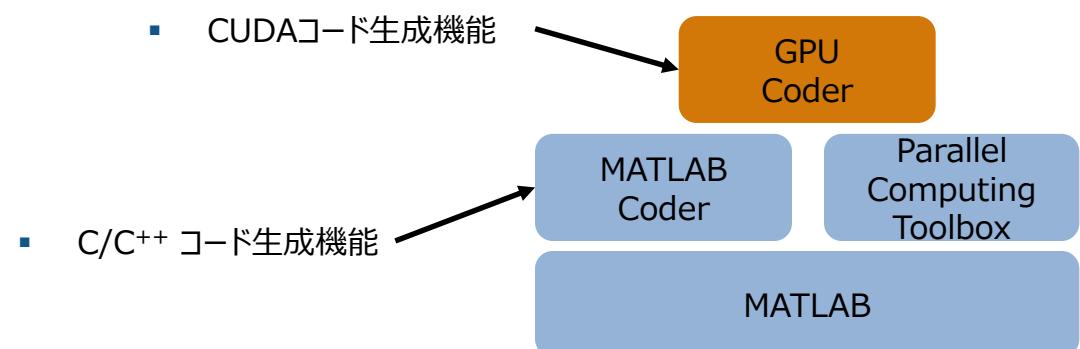
DPI-C : SystemVerilogによるC言語とのインターフェース

## 3.11 MATLABからのCUDA®コード生成

R2017b

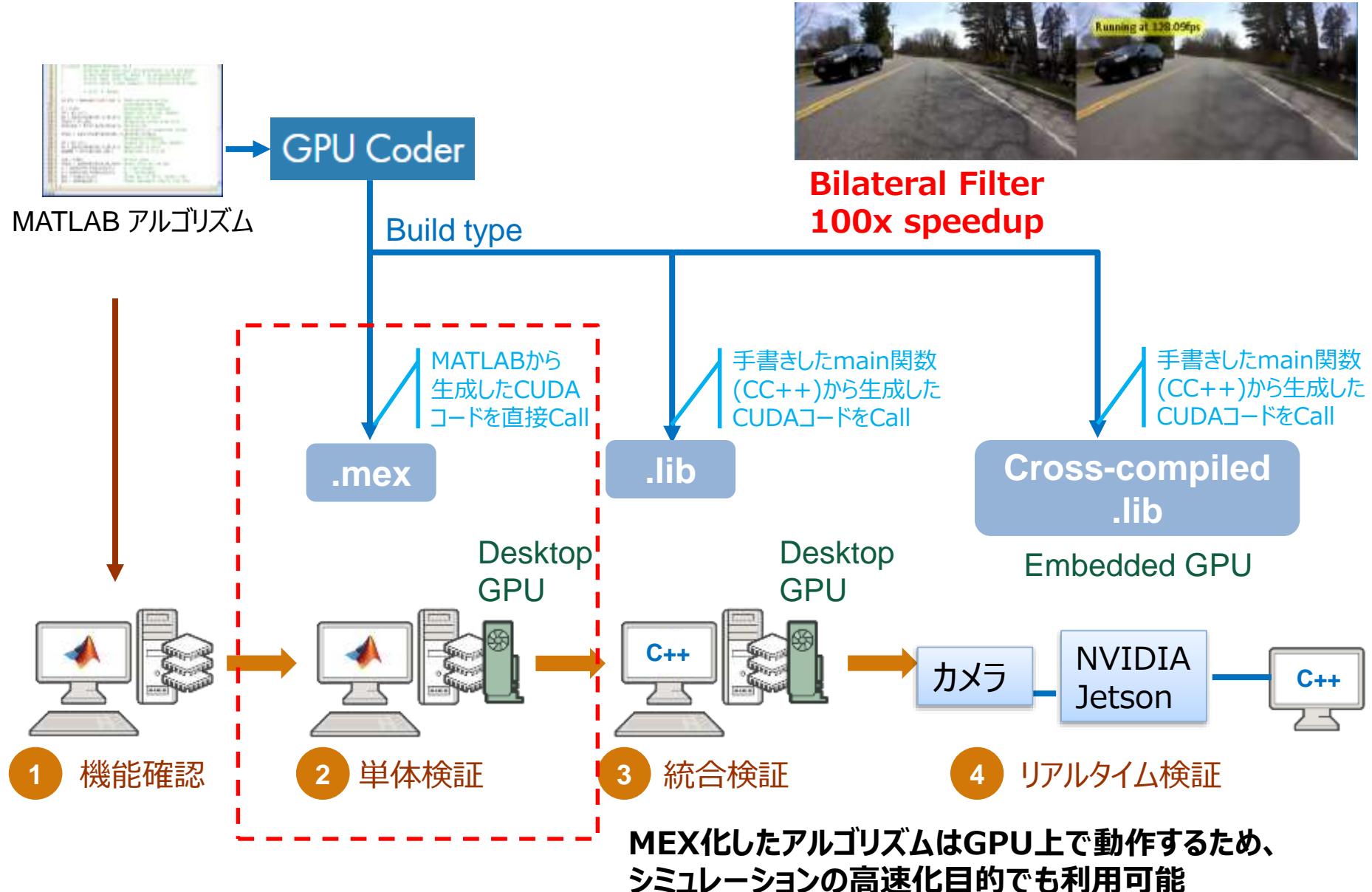
GPU Coder

- CNNを含む、画像処理アルゴリズムからの CUDA C生成
- CUDAの文法を知らなくても利用できる、専用プラグマによる関数解析
- 初めてでも使いやすい、専用GUIを提供
- DLLを生成しSimulinkへ統合可能

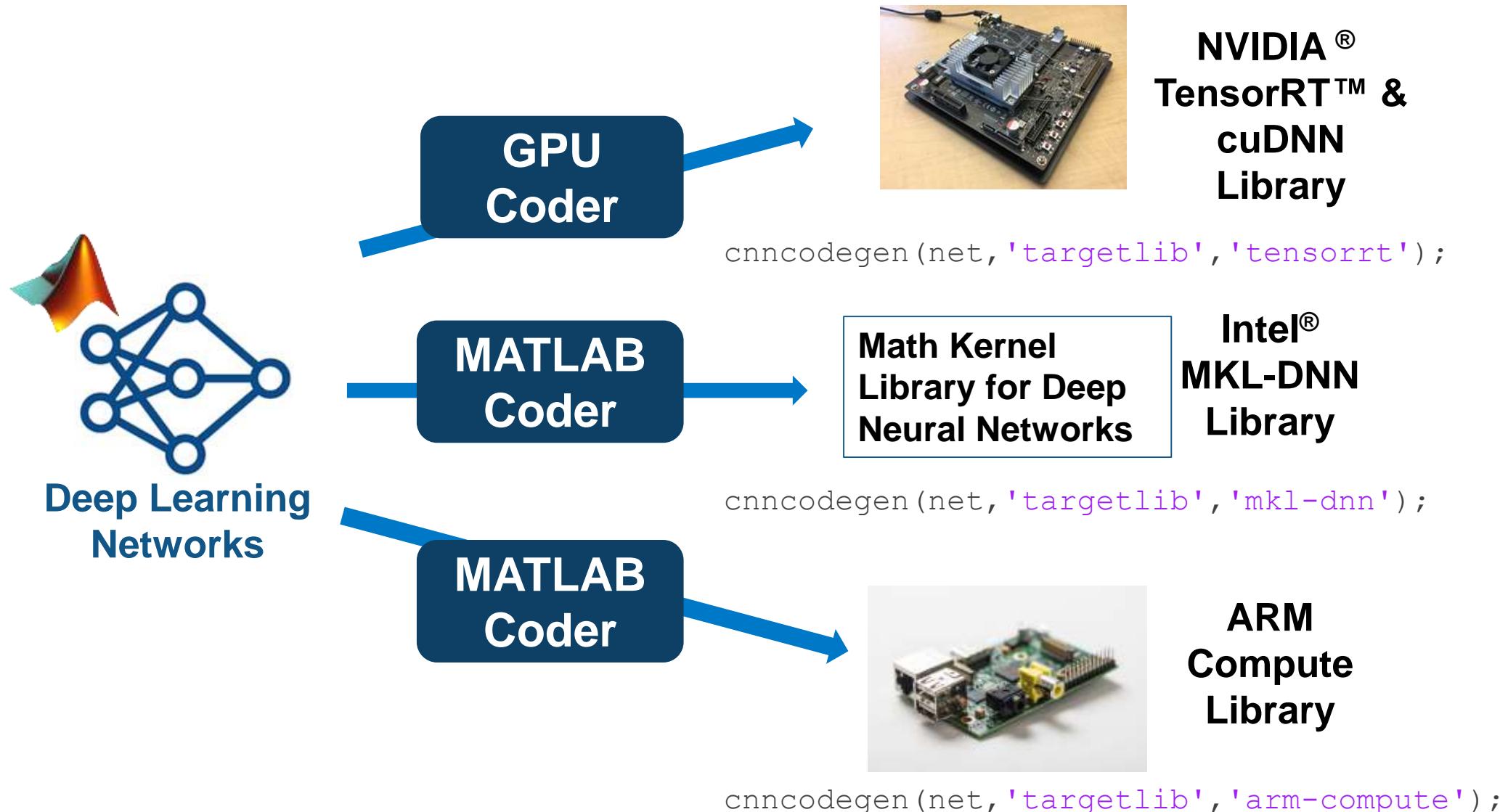


## 3.11.1 画像処理アルゴリズムのGPU実装

GPU Coder



## 3.11.2 ディープラーニングのGPU/CPU実装



MATLAB Coder  
GPU Coder

NVIDIA®  
TensorRT™ &  
cuDNN  
Library

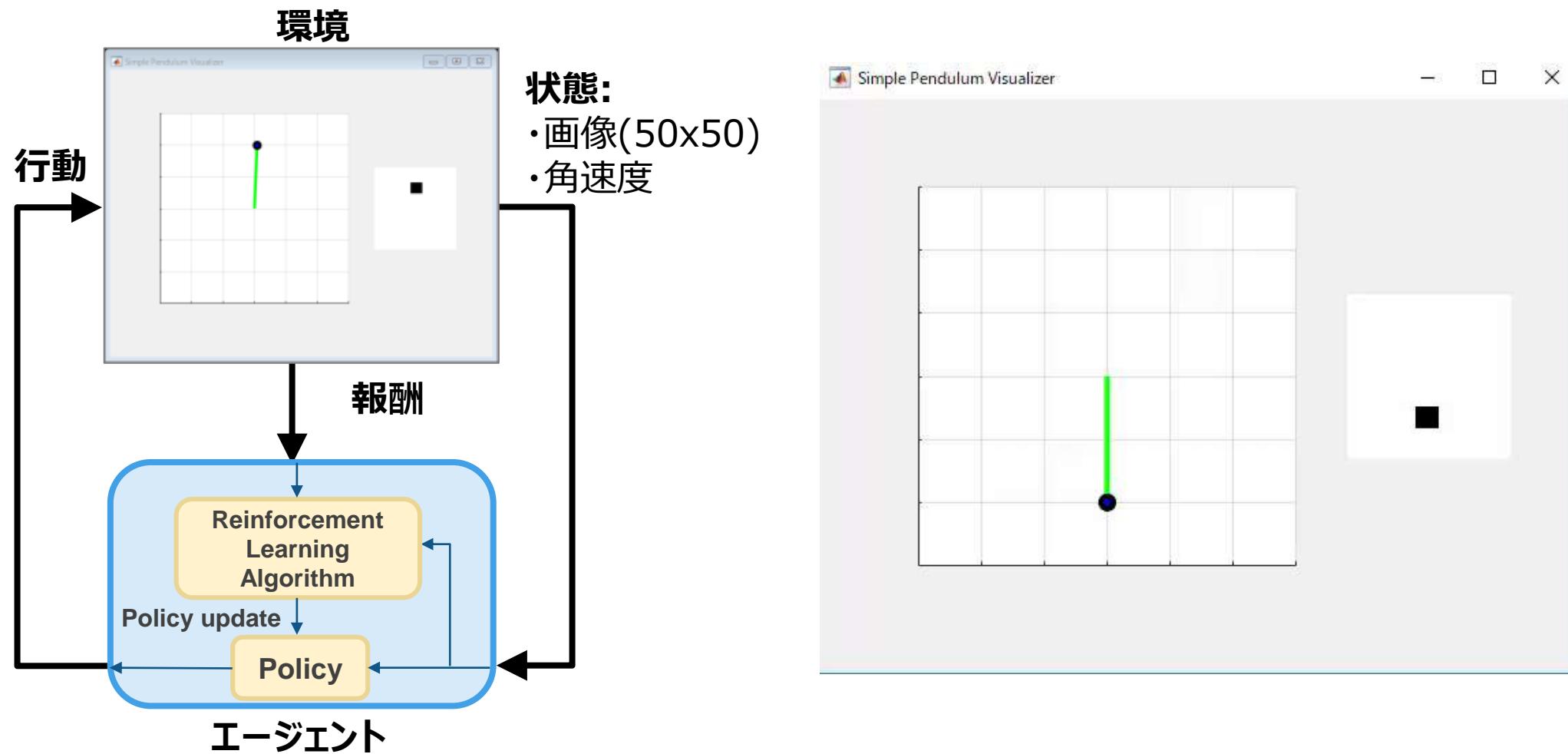
Intel®  
MKL-DNN  
Library

ARM  
Compute  
Library

`cnncodegen(net, 'targetlib', 'arm-compute');`

### 3.12.1 深層強化学習：画像入力による振り子の振り上げ

# Reinforcement Learning Toolbox™ Deep Learning Toolbox™



画像などの高次元の観測情報を状態を推定  
(この例では画像から振り子の位置が推定されている)

# アジェンダ

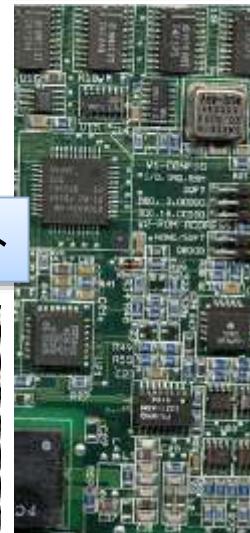
1. MATLAB/Simulinkの概要
2. 各種画像処理例
3. 連携機能
4. コンピュータービジョン処理例
5. 画像の機械学習・ディープラーニング
6. まとめ

## 4.1 パターンマッチング (領域ベースのマッチング)

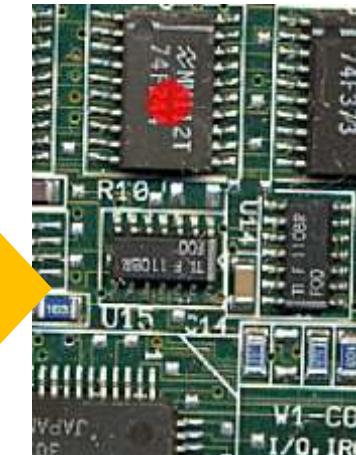
### テンプレートマッチング

```
htm=vision.TemplateMatcher;          % オブジェクト定義
Loc=step(htm,Igray,T);            % 処理実行
```

テンプレート



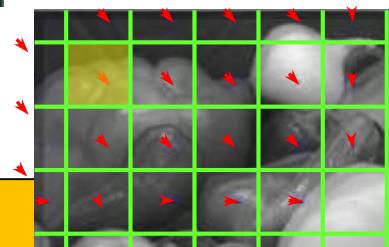
ターゲット



### ブロックマッチング

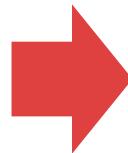
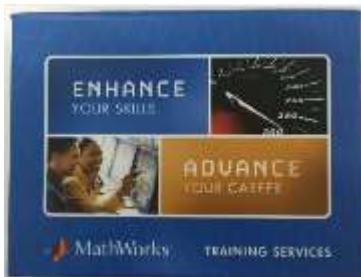
```
% オブジェクト定義
hbm = vision.BlockMatcher( ...
    'ReferenceFrameSource','Input port','BlockSize',[35 35]);
% 処理実行
motion = step(hbm, img1, img2);
```

ブロックサイズ(35x35ピクセル)



## 4.2.1 特徴点検出・特徴量抽出（1）

### マッチング・検出 対象画像



#### > 特徴点の検出

```

points = detectSURFFeatures(G) %SURF特徴点
detectHarrisFeatures()      %Harris (コーナー)
detectFASTFeatures()         %FAST (コーナー)
detectMinEigenFeatures()     %最小固有値 (コーナー)
detectBRISKFeatures()        %BRISK (コーナー)
detectMSERFeatures()         %MSER特徴点
detectKAZEFeatures()         %KAZE (コーナー) R2017b
detectORBFeatures()          %ORB (コーナー) R2019a

```

#### > 特徴量の抽出 (SURF, FREAK, BLOCK, BRISK, HOG, KAZE)

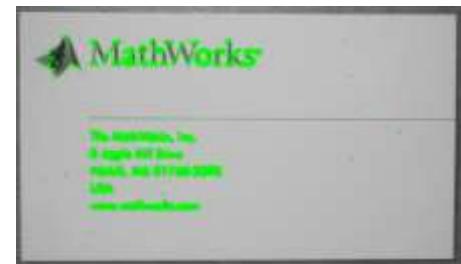
R2014a R2013b R2017b

```
[f, vpoints] = extractFeatures(G, points)
```

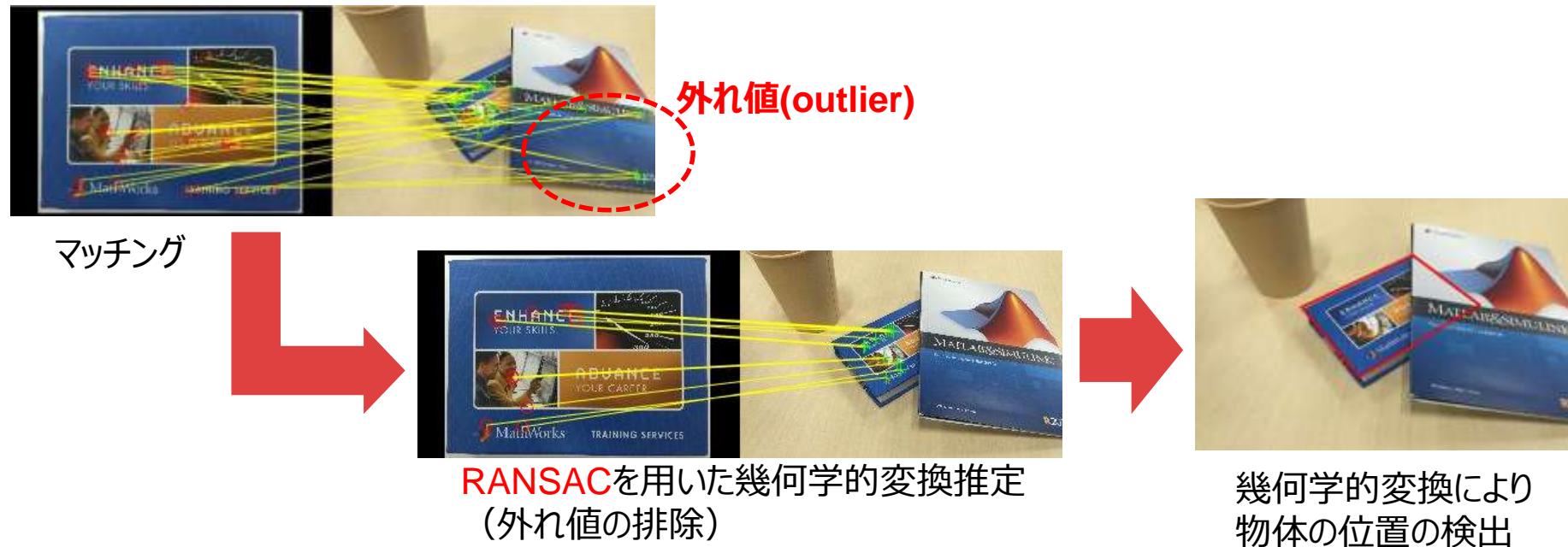
```
[f, vpoints] = extractHOGFeatures(I, points)
```

R2019a

Oriented FASTとRotated BRIEF(ORB)による特徴点検出

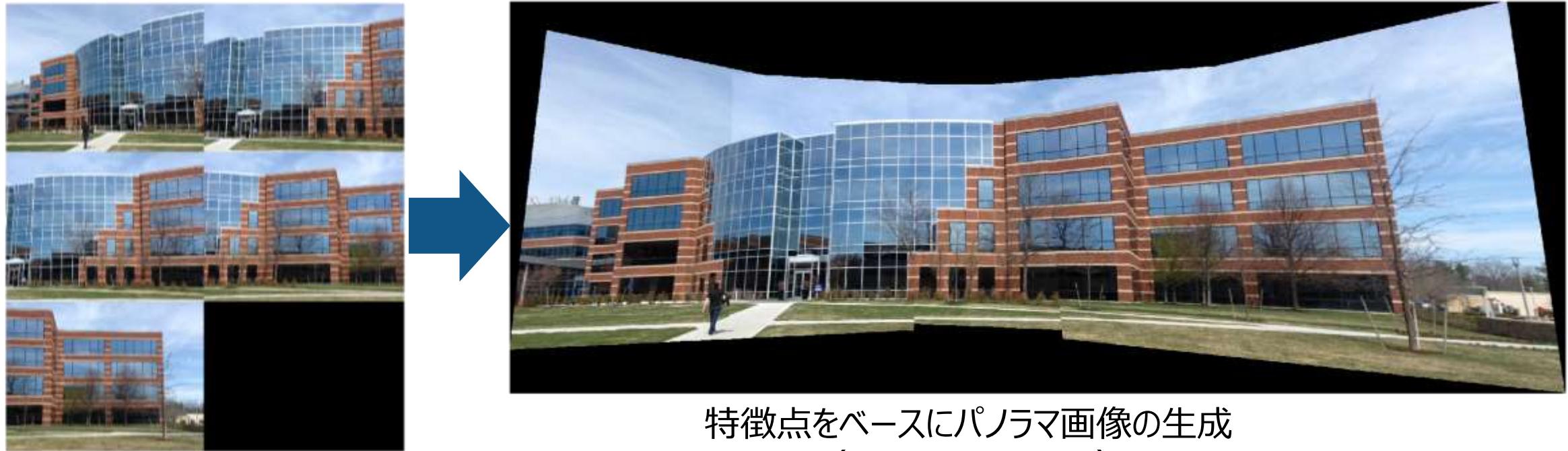


## 4.2.1 マッチング・物体検出（2）



```
% マッチング (SSD、SAD、NCC、ハミング距離)
index_pairs = matchFeatures(f1,f2)
% 幾何学的変換推定 (相似変換、アフィン変換、射影変換)
[tform, f_pts1, f_pts2] =
    estimateGeometricTransform(m_pts1, m_pts2) R2013a
% 幾何学的変換
result = imwarp(Orig, tform, 'OutputView', imref(size(I)))
```

## 4.2.2 特徴点ベースのレジストレーション



特徴点をベースにパノラマ画像の生成  
(Image Stitching)

```
[features, points] = extractFeatures(grayImage, points); % 特徴抽出
indexPairs = matchFeatures(features, featuresPrevious, 'Unique', true); % マッチング
matchedPoints = points(indexPairs(:,1), :);
matchedPointsPrev = pointsPrevious(indexPairs(:,2), :);
% 幾何学変換行列の推定(枚数分繰り返す)
tforms(n) = estimateGeometricTransform(matchedPoints, matchedPointsPrev, ...
    'projective', 'Confidence', 99.9, 'MaxNumTrials', 2000);
```

## 4.7.1 動き検出・高速動画ストリーミング処理

Image Acquisition Toolbox

- カメラからの画像の取り込み

`imaq.VideoCapture()`

カメラからの画像の取り込み

- 動きの検出

`opticalFlowHS`

Horn-Schunck (HS) オプティカルフロー

(オプティカルフロー：物体の動きの方向と速さをベクトルで表示)

`opticalFlowLK`

Lucas-Kanade (LK) オプティカルフロー

`opticalFlowLKDAG`

DoGフィルタを用いた、LK オプティカルフロー

`opticalFlowFarneback`

Farnebackオプティカルフロー

`vision.ForegroundDetector()`

混合ガウス分布モデルによる動体検出

- カメラからの簡単な動画像取り込み・ストリーミング処理
- 簡潔なMATLABコードによるオプティカルフロー検出
- 高速なインタープリター処理



## 4.7.2 動き検出を用いた物体検出



定点カメラの  
入力動画像

動きの速さ・  
方向の検出

動きを基にした  
車の抽出

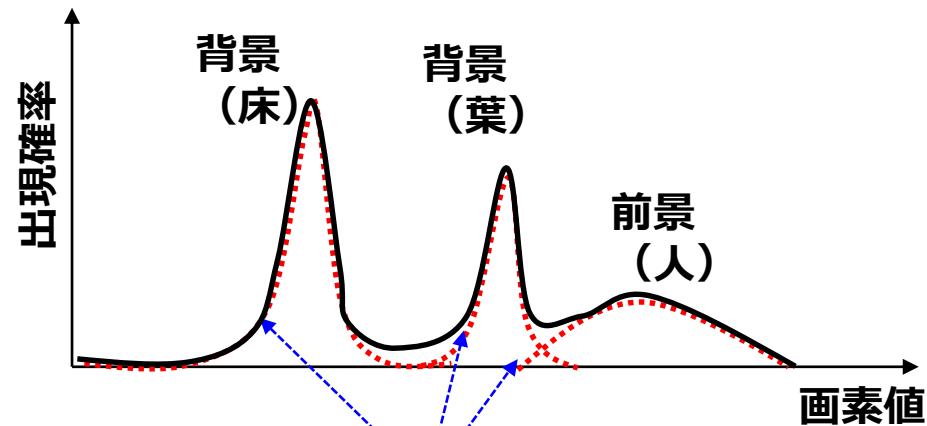
トラッキング中の車  
に対し、  
- 枠を表示  
- 台数の表示

動画からの動きや物体の検出  
(色や形等では、単純に物体と背景を区別することが難しいとき等)

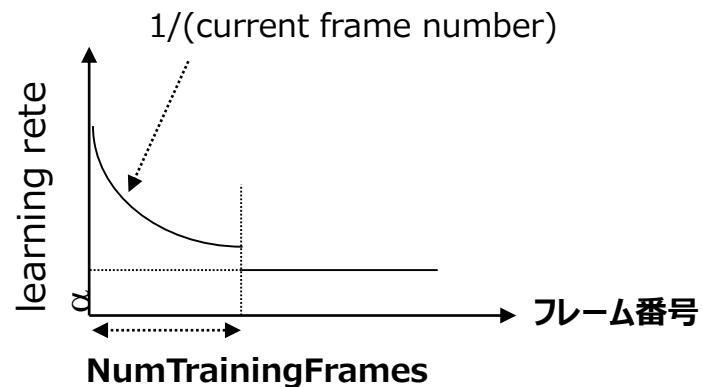
### 4.7.3 混合ガウス モデルを使用した前景の検出



動画中の各ピクセル毎に、混合ガウスモデル  
Gaussian Mixture Model (GMM) を作成



複数のガウス分布でフィッティング  
(各重み $w_i$ ) (各標準偏差 $\sigma_i$ )



ガウス分布の中で、 $\frac{w_i}{\sigma_i}$ の大きなものを背景とする。その値の大きいものから  
 $w_i$ を順に積算し、'MinimumBackgroundRatio' を越すまでのガウス分布を背景  
フレームが進むに従い、 $1/a$  の時定数（直近どの位のフレーム数から背景を抽出するか）で背景を学習（パラメータを更新）

## 4.8 物体のトラッキング

### ● ベイジアンフィルタ

`vision.KalmanFilter()`

`robotics.ParticleFilter()`

物体トラッキング用カルマンフィルター

パーティクルフィルタ

Robotics System Toolbox

R2016a

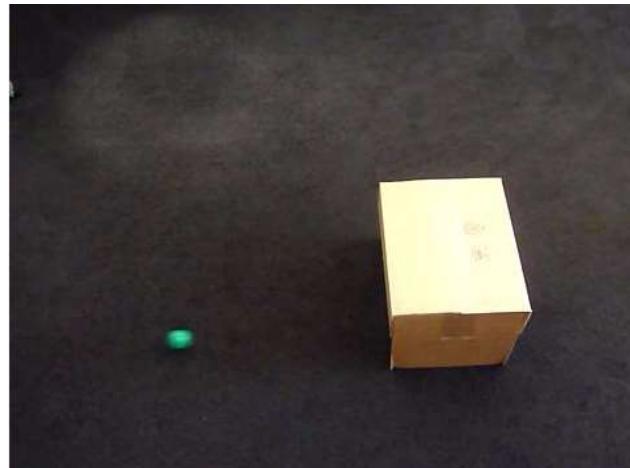
### ● その他

`vision.HistogramBasedTracker()`

`vision.PointTracker()`

ヒストグラムを基にしたトラッキング (CAMShift)

点のトラッキング (KLTトラッカー)

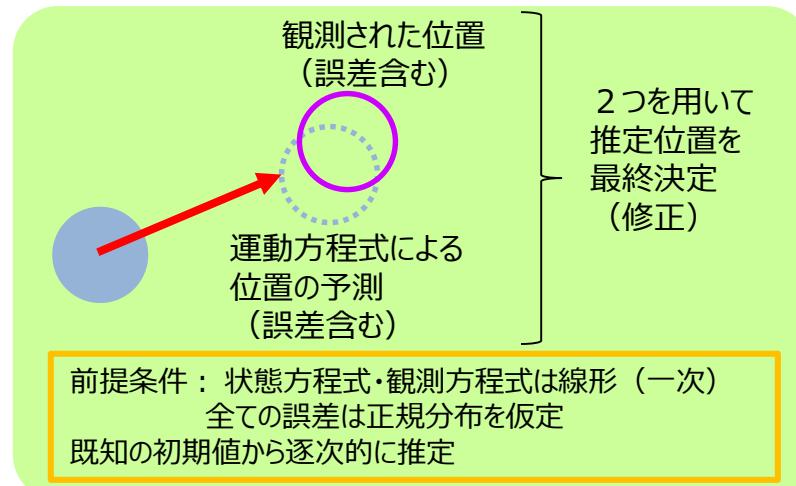


物体の動き予測  
(オクルージョン (隠れ) 含む)



複数物体の検出やトラッキング

## 4.8.1 カルマンフィルタ



(例) 時刻  $t$  での状態ベクトル： $X_t = [x_t, vx_t, y_t, vy_t]^T$

状態方程式:  $X_{t+1} = F X_t + B U_k + w$   
制御入力  
(ここでは不使用)

$$\begin{bmatrix} x_{t+1} \\ vx_{t+1} \\ y_{t+1} \\ vy_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ vx_t \\ y_t \\ vy_t \end{bmatrix} + \begin{bmatrix} w_x \\ w_{vx} \\ w_y \\ w_{vy} \end{bmatrix}$$

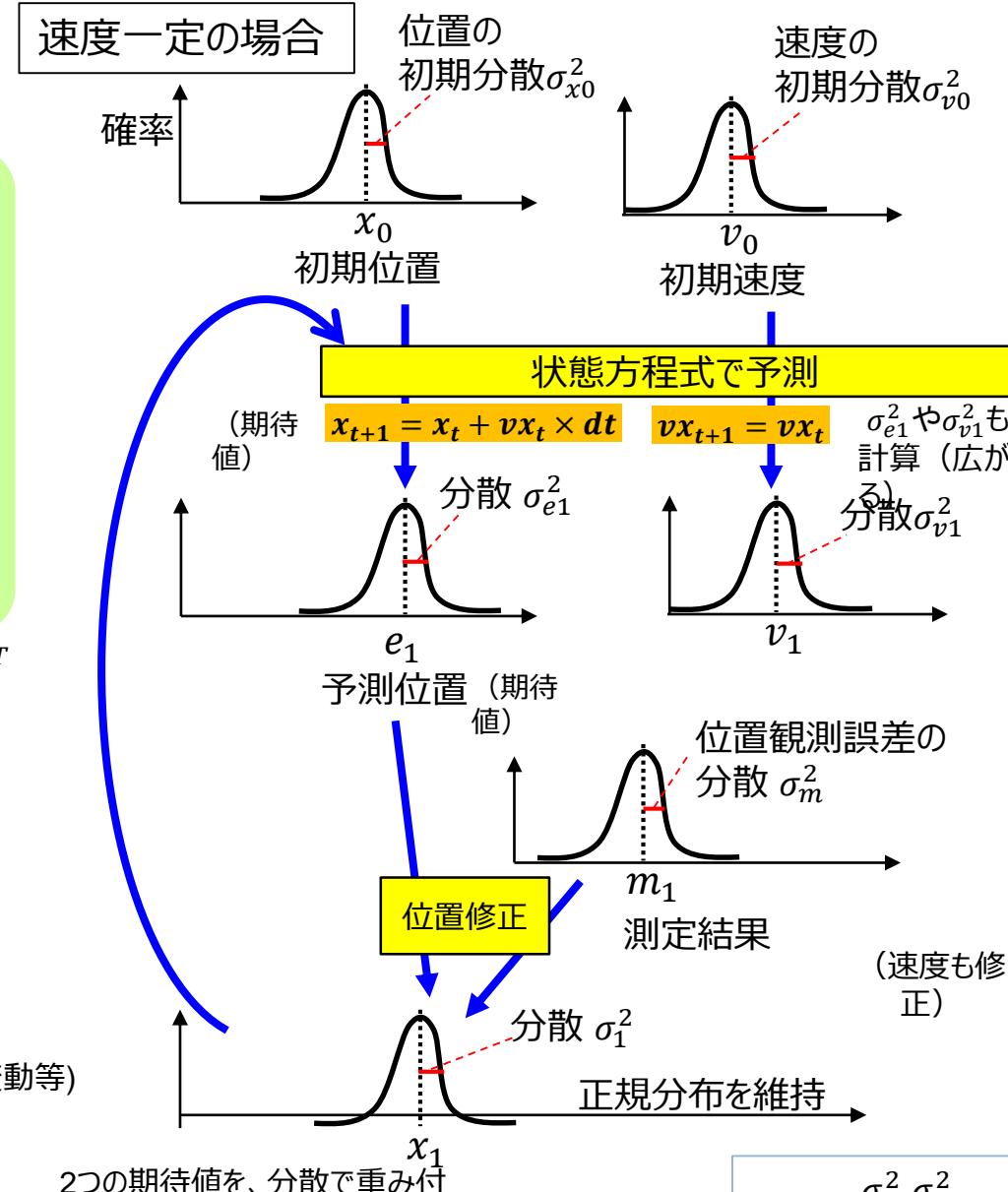
状態遷移行列  
ノイズ  
(速度変動等)

時刻  $t$  での観測ベクトル： $Z_t = [zx_t, zy_t]^T$

観測方程式:  $Z_t = H X_t + u$

$$\begin{bmatrix} zx_t \\ zy_t \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_t \\ vx_t \\ y_t \\ vy_t \end{bmatrix} + \begin{bmatrix} u_{zx} \\ u_{zy} \end{bmatrix}$$

観測行列  
観測誤差



$$x_1 = \left( \frac{\sigma_m^2}{\sigma_e^2 + \sigma_m^2} \right) e_1 + \left( \frac{\sigma_e^2}{\sigma_e^2 + \sigma_m^2} \right) m_1$$

$$\sigma_1^2 = \frac{\sigma_e^2 \sigma_m^2}{\sigma_e^2 + \sigma_m^2}$$

## 4.8.3 パーティクル フィルタ

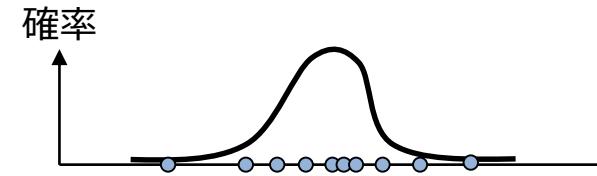
- 確率分布を、多数の粒子の密度で近似
- 非線形運動方程式も可能
- 多峰性分布も可能  
(例：等速度運動)

時刻 $t$ での状態ベクトル： $X_t = [x_t, vx_t, y_t, vy_t]^T$

時刻 $t$ での観測ベクトル： $Z_t = [zx_t, zy_t]^T$

予測ステップ#1

Robotics System Toolbox R2016a



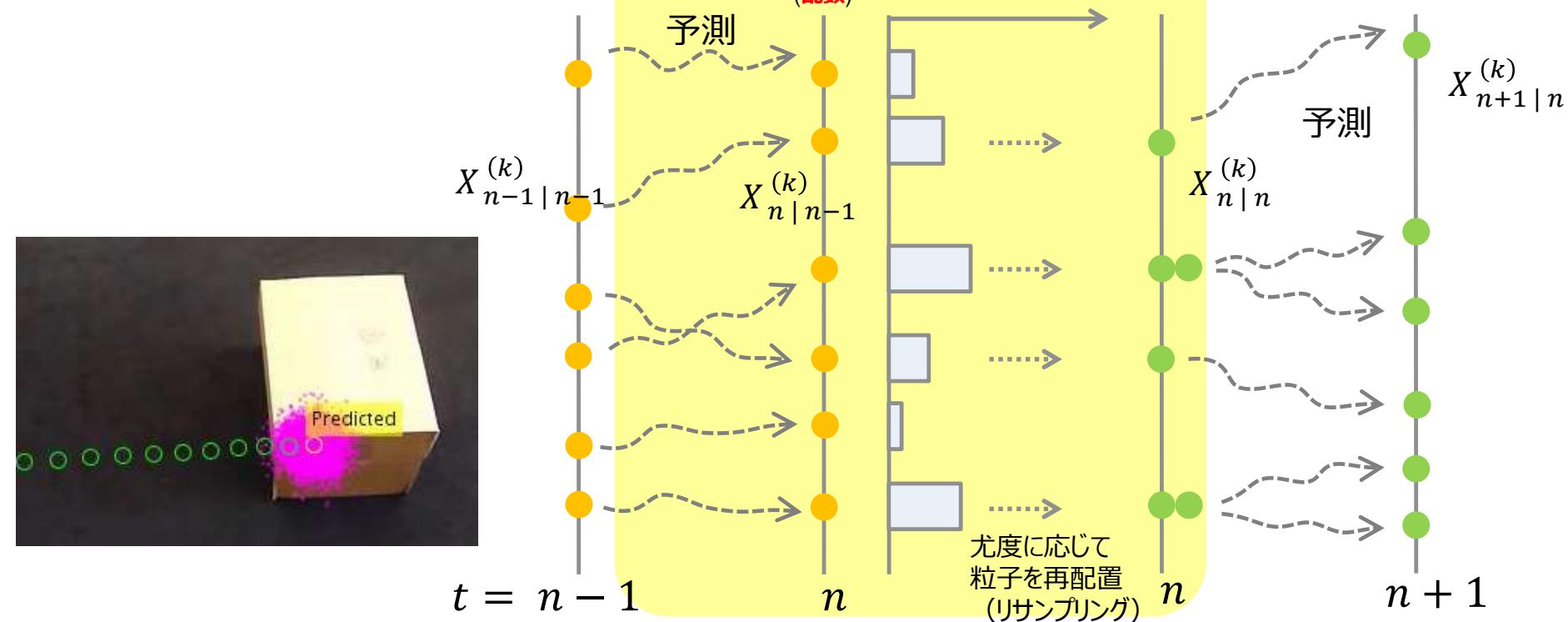
フィルタリング#1

予測ステップ#2

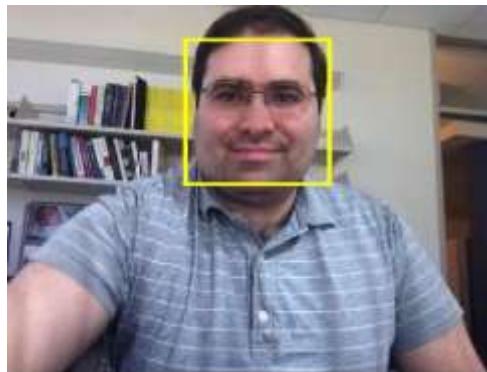
状態方程式： $X_{t+1} = FX_t + w$

予測誤差  
(乱数)

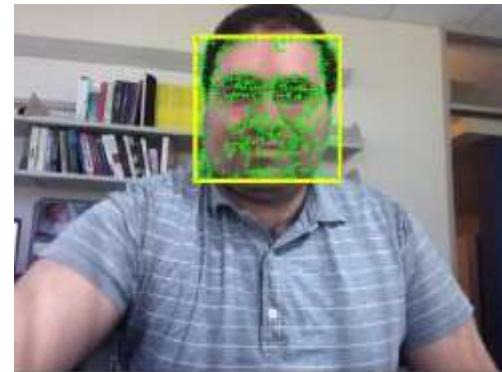
観測結果と観測誤差定  
数から、各粒子の尤度  
(もともらしさ) 決定



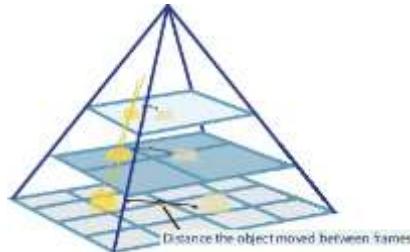
## 4.8.4 KLT ポイント トラッカー



トラッキングしたい  
領域（ROI）を検出



ROI中でコーナー等の  
特徴点を検出



画像ピラミッドを生成し、低解像度画像から順にLT法によるオプティカルフローを検出することで、大きな動きを検出

### トラッカーの初期化

```
pointTracker = vision.PointTracker;           % トラッカーのオブジェクトの生成
initialize(pointTracker, points, frame1);      % 検出した点で、トラッカーを初期化
```



### 点を次フレーム上でトラッキング

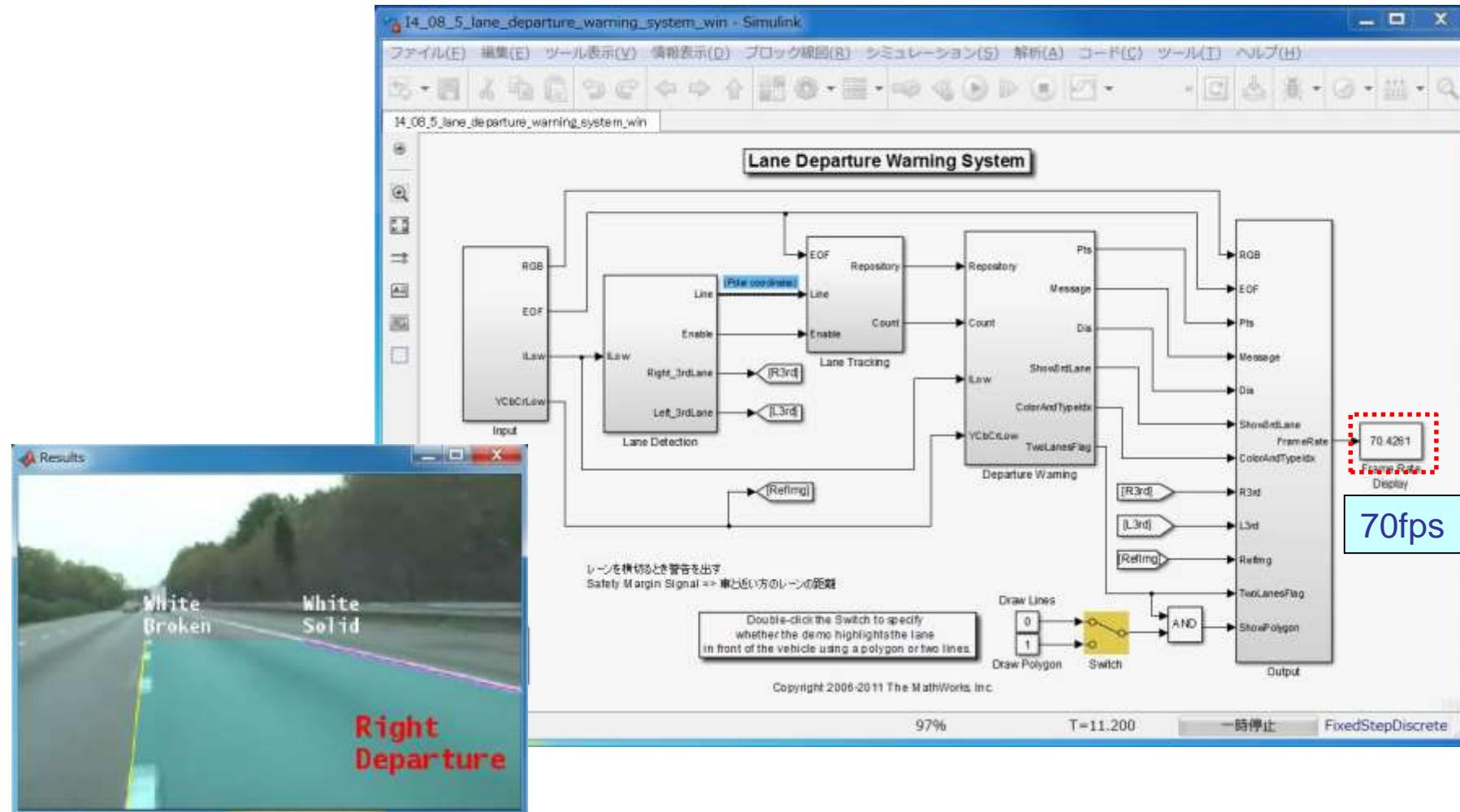
```
[points, isFound] = step(pointTracker, frame2);
```

(isFound : 現フレームで検出できた点のフラグ)



## 4.8.6 白線トラッキング

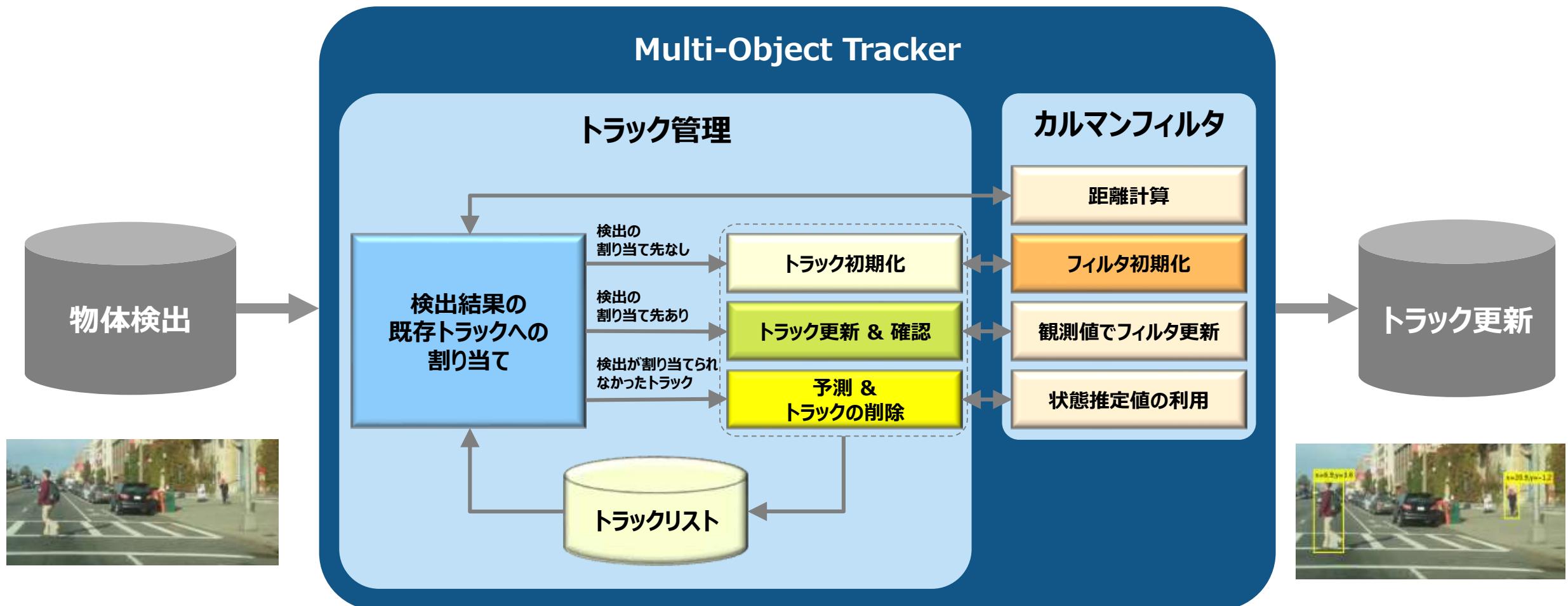
Simulink  
DSP System Toolbox



ブロック線図設計環境を用いた、リアルタイムの高速動画処理シミュレーション

## 4.8.7 マルチオブジェクトトラッカー

Automated Driving Toolbox



## 4.9 画像からカメラ位置・姿勢推定、3次元再構築

<code>bundleAdjustment</code>	カメラ位置・姿勢と3次元点群の最適化 
<code>cameraMatrix</code>	カメラ行列の生成：カメラ内部パラメータと外部パラメータ（回転・並進）から算出
<code>relativeCameraPose</code>	2台のカメラ間の関係：基礎行列・カメラパラメータ・2枚の画像上の対応点から算出
<code>disparity</code>	視差画像の生成：平行化された2枚の画像から算出
<code>epipolarLine</code>	指定された点に対応するエピポーラ線の計算：基礎行列から計算
<code>estimateCameraParameters</code>	カメラパラメータの推定
<code>estimateFundamentalMatrix</code>	基礎行列の推定：2枚の画像上の対応点から推定
<code>estimateUncalibratedRectification</code>	カメラキャリブレーション無しステレオ平行化：基礎行列と対応点から
<code>extrinsics</code>	カメラの外部パラメータ推定：内部パラメータ・画像上とワールド上の対応点から推定
<code>rectifyStereoImages</code>	ステレオ平行化：ステレオパラメータと、左右の画像から生成
<code>reconstructScene</code>	視差画像から3次元点群を生成：ステレオパラメータと、視差画像から生成
<code>triangulate</code>	点の3次元位置推定：ステレオパラメータと、左右画像上の対応点から推定 もしくは 左右のカメラ行列と、左右画像上の対応点から推定
<code>triangulateMultiview</code>	複数画像から点 3 次元位置推定：全 点対応関係・カメラポーズ・カメラパラメータから

### 専用のクラス

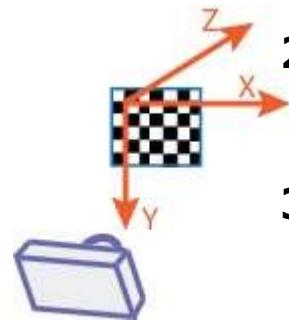
<code>viewSet</code>	Structure from Motion の為の、データ用管理クラス
<code>pointTrack</code>	複数画像間の点対応関係を格納

## 4.9.1 キャリブレーション済み単眼カメラによるカメラ位置・姿勢

例)

1 : 使用する単眼カメラをカメラキャリブレーション（内部パラメタの推定）

- 推定 / ワールド座標Z=0  
面上の物体位置推定  
(ワールド座標と画像座標  
が既知の複数の点を使用  
→ 外部パラメータ決定)



2 : チェッカーボードを単眼カメラで撮影 : チェッカーボードの面がワールド座標系 Z=0  
チェッカー端の交点がワールド座標の原点

3 : extrinsics 関数で、外部パラメータを推定  
(ワールド座標値とカメラ座標値の関係)

4 a : 外部パラメータから、カメラのワールド座標位置・姿勢を計算

4 b : 内部・外部パラメータと、測定対象点の画像座標から、  
点のZ=0ワールド座標上の位置の推定

% 外部パラメータの推定（レンズ歪の無い画像上の点座標を使用）

```
[R, t] = extrinsics(imagePoints, worldPoints, cameraParams)
```

% カメラのワールド座標位置・姿勢を計算

```
Location = -t * R' % [1x3]
```

```
Orientation = R' % [3x3]
```

%% レンズ歪の無い画像上の座標から、ワールド座標を推定

% (ワールド座標系z=0平面上のxy座標)

```
worldPoints1 = pointsToWorld(cameraParams, R, t, imagePoints1)
```

## 4.9.2 ステレオビジョン向けワークフロー

R2014a

R2014b

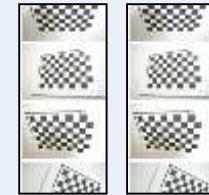
事前 キャリブレーション

キャリブレーション用パターンの撮影



ステレオカメラ キャリブレーション

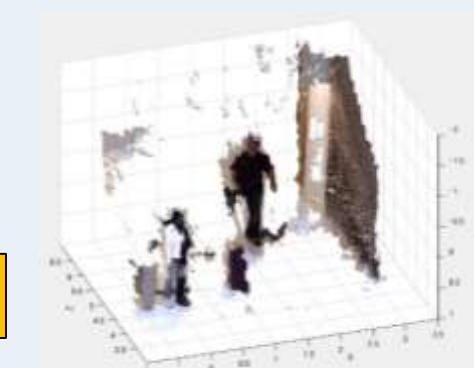
パラメータ



ステレオ画像の並行化



左右の画像のずれから、視差の計算



R2015a

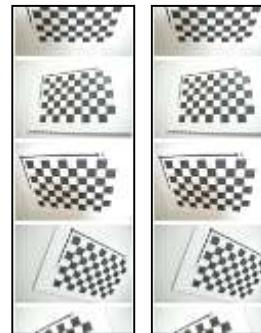
処理速度の高速化

三次元空間の再構築

## 4.9.2 ステレオビジョン向けワークフロー：キャリブレーション

### ステレオカメラキャリブレーション

撮影されたキャリブレーション用のパターン

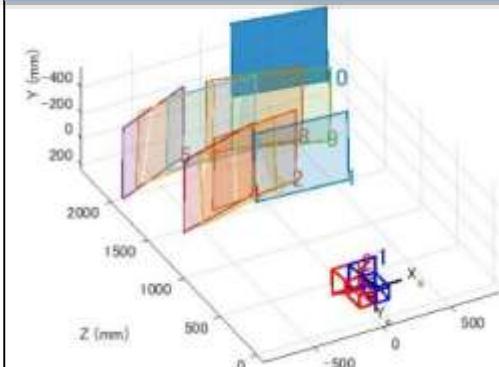


R2014a

キャリブレーション

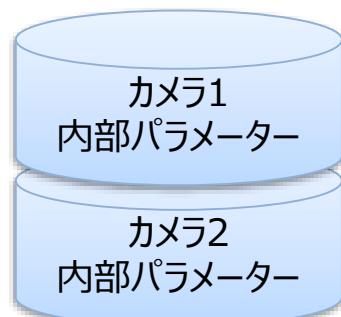
`detectCheckerboardPoints()`  
`estimateCameraParameters()`

ステレオカメラシステムのパラメータ



### ステレオベースライン推定

キャリブレーション済のパラメーター

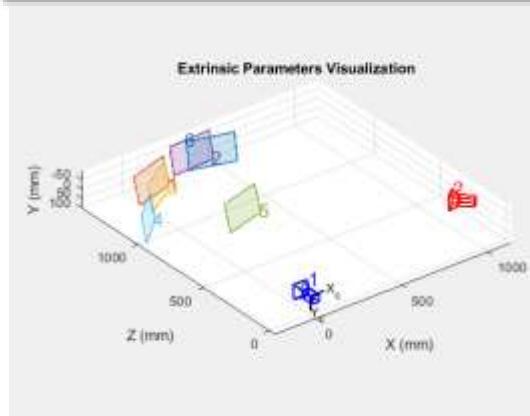


R2018a

ベースライン(外部パラメータ)推定

`estimateStereoBaseline()`

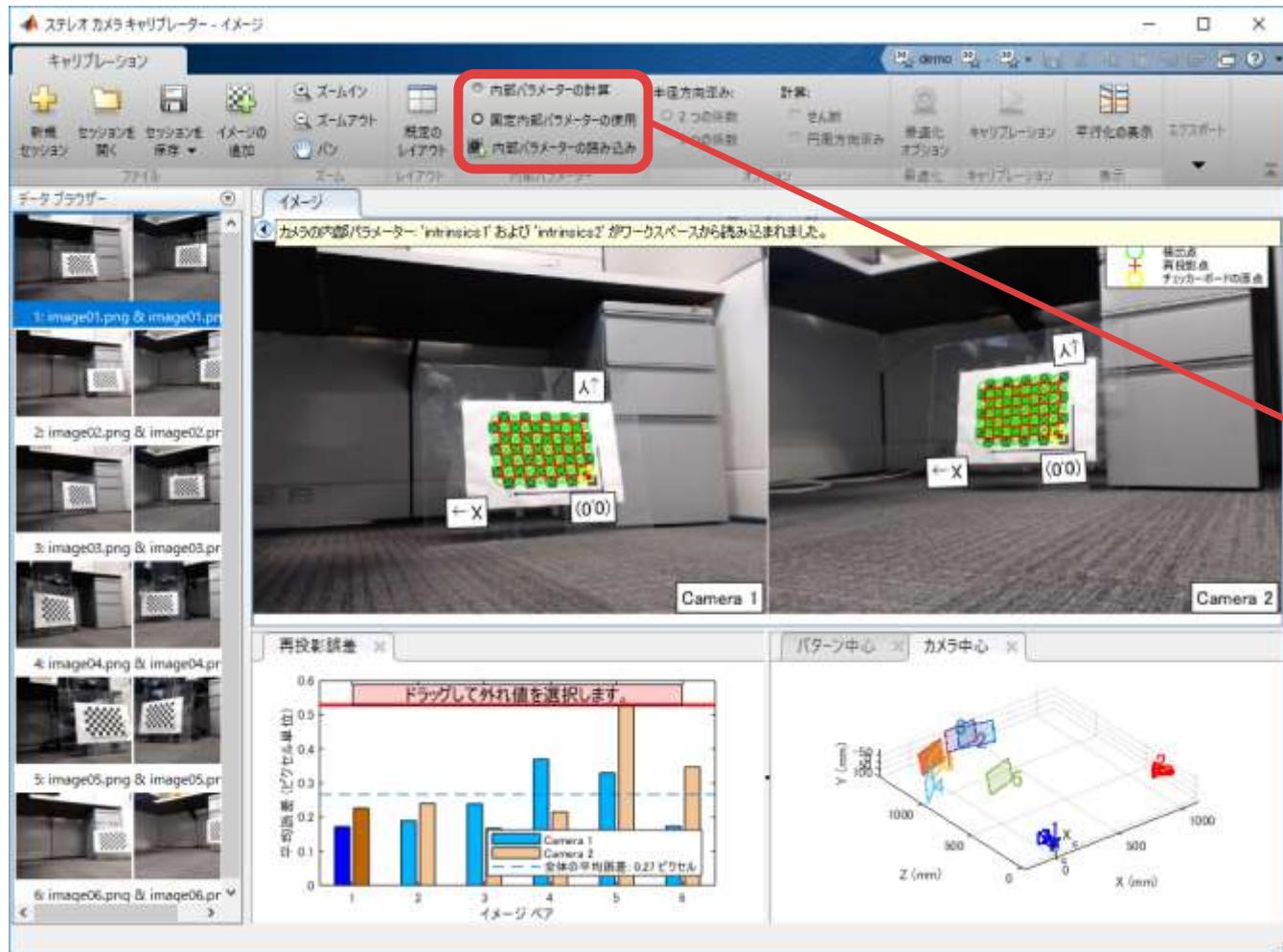
ステレオカメラシステムのパラメータ



## 4.9.2 ステレオビジョン向けワークフロー：キャリブレーション

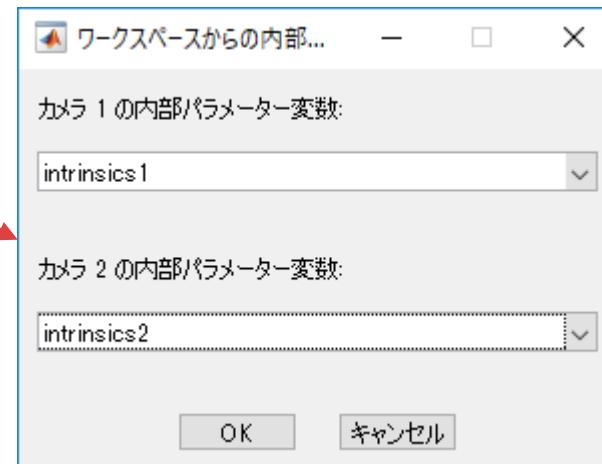
R2014b

### ステレオカメラキャリブレーション用APPS



R2018a

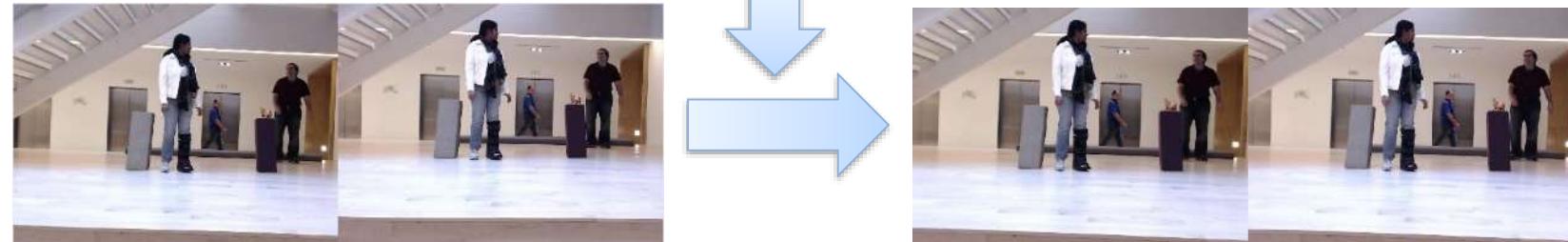
個別の内部パラメータを取り込み  
ベースラインの計算する関数をサポート



## 4.9.2 ステレオビジョン向けワークフロー：ステレオ平行化

ステレオ画像の平行化

ステレオカメラシステムのパラメータ



```
[J1, J2] = rectifyStereoImages(I1, I2, stereoParams)
```

ステレオ画像表示

R2014a

```
imshow(stereoAnaglyph(I1, I2));
```

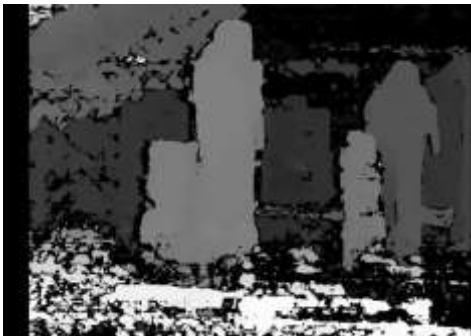


3Dメガネ



## 4.9.2 ステレオビジョン向けワークフロー：視差画像からの3次元再構築

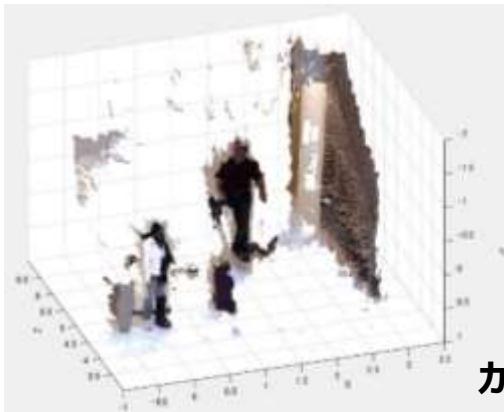
視差画像の計算



```
disparityMap = disparitySGM (G1, G2, 'DisparityRange', [0 64])
```

**disparitySGM, disparityBM:** GPU実行可能な視差計算関数追加(Cコード生成可) **R2019a**

3次元空間の再構築・表示



**R2014a**  
**R2014b**

カメラ 1 の光学中心が原点

```
pointCloud = reconstructScene(disparityMap, stereoParams)  
showPointCloud(pointCloud, J1);
```

### 4.9.3 ステレオビジョン向けワークフロー： 距離測定への応用



画像の平行化



視差画像



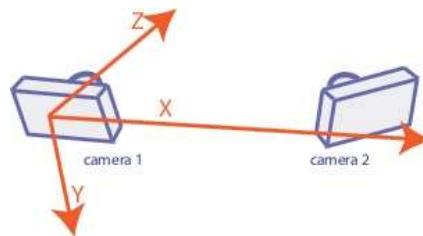
対象を検出し、距離計算



ステレオカメラにより、人物までの距離を測定し、3m以内になつたら  
赤で警告 => 衝突防止等への応用

## 4.9.4 2台のカメラで対象点の3次元位置を求める別法

(疎な3次元再構築：ステレオ平行化・視差計算不要)



- ステレオパラメータと、左右の画像上の対応点座標が既知の場合  
(画像のレンズ歪を除去の後に、対応点座標を取得)

```
% カメラ1の光学中心から、指定された点までx・y・z方向の距離を計算
```

```
point3d = triangulate(matchedPts1, matchedPts2, stereoParams);
```

- 左右のカメラの内部・外部（並進・回転）パラメータもしくは左右のカメラ行列と、左右の画像上の対応点座標が既知の場合  
(カメラの外部パラメータは、 extrinsics 関数で推定可)  
(画像のレンズ歪を除去の後に、対応点座標を取得)

```
% カメラ内部・外部パラメータから、カメラ1・2のカメラ行列を計算
```

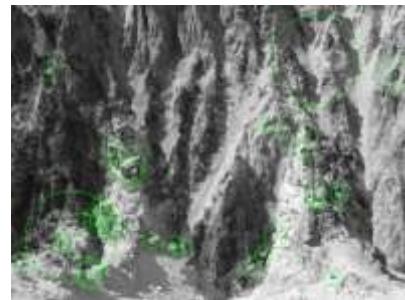
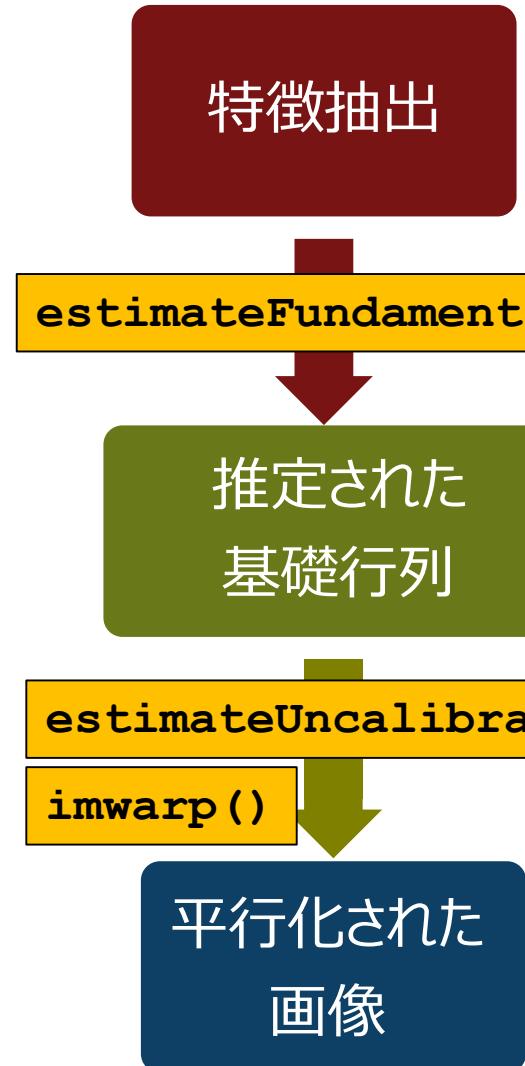
```
camMatrix1 = cameraMatrix(cameraParams1, cam1R, cam1T);
```

```
camMatrix2 = cameraMatrix(cameraParams2, cam2R, cam2T);
```

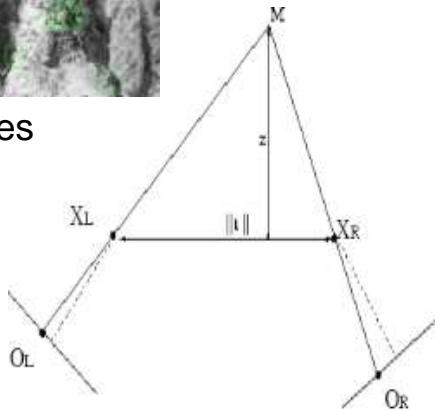
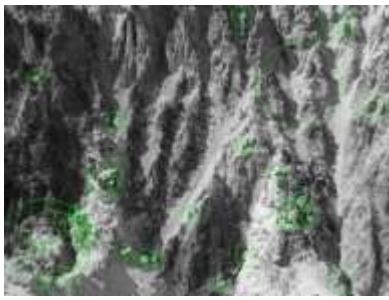
```
% 指定された点の3次元位置を計算 (外部パラメータで用いたワールド座標を使用)
```

```
point3d = triangulate(matchedPts1, matchedPts2, camMatrix1, camMatrix2);
```

## 4.9.5 ステレオビジョン向けワークフロー：キャリブレーションなしの平行化



Find Features from pair of images



Rectified image



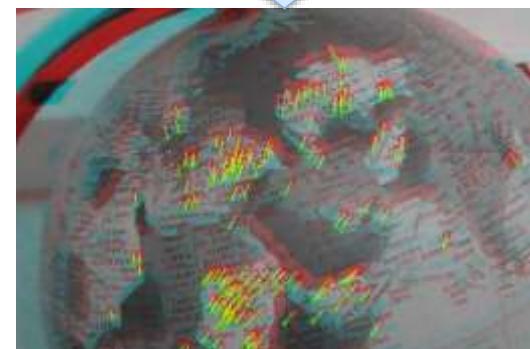
3Dメガネ

## 4.9.6.1 2枚の画像によるStructure from Motion

R2015b



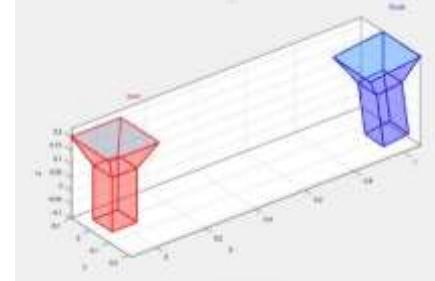
2枚の画像上の疎な対応点を探索  
`vision.PointTracker`



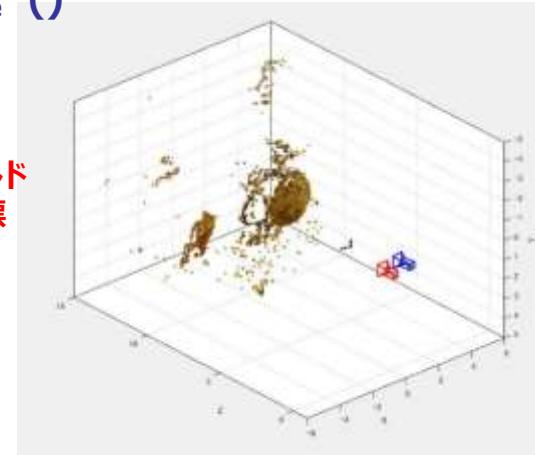
基本行列（2つの画像の対応関係） $E$  の推定  
`estimateEssentialMatrix()`

 $E$ 

$E$ と対応点情報から、カメラ1と2の位置・姿勢関係を計算  
`relativeCameraPose()`



カメラ行列（World座標と画像上の座標の関係）を計算  
`cameraMatrix()`  
密な対応点の探索  
対応点座標とカメラ行列から、点の3次元マップを計算  
`triangulate()`



## 4.9.6.2 複数画像によるStructure from Motion R2016a



複数のカメラからの画像に対応  
R2019b



SfM の為の、データ管理用クラス

```
vSet = viewSet; % viewSetクラスの生成
% vSet.Views : 各画像での特徴点座標とカメラの位置・姿勢
% vSet.Connections : 各画像間の特徴点の対応
```

2組の画像間で対応点を探索 : `matchFeatures`

基礎行列（2つの画像上の点の対応関係）を推定 :

`estimateFundamentalMatrix`

各2組の画像の相対カメラ位置・姿勢の推定 : `relativeCameraPose`

複数画像上の点対応関係から、各点の3次元位置を推定 :

`triangulateMultiview`

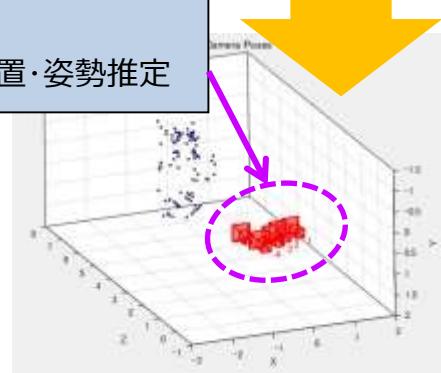
複数Viewにまたがる、点の全対応関係情報 : `findTracks`

全ての点とカメラ位置・姿勢を用い、

点群位置とカメラ位置・姿勢を最適化 : `bundleAdjustment`

Step#1

カメラの自己位置・姿勢推定



Step#2

密な3次元点群（マップ）再構成



一枚目の画像上で密な特徴点を検出

ポイントトラッカーで、2組の画像毎に点の対応関係を検出

複数Viewにまたがる、点の全対応関係情報 : `findTracks`

複数画像上の点対応関係から、各点の3次元位置を推定 :

`triangulateMultiview`

全ての点とカメラ位置・姿勢を用い、

点群位置とカメラ位置・姿勢を最適化 : `bundleAdjustment`

## 4.10.1 3次元点群：読み込み・表示

R2015a R2015b

### 3次元点群用のクラス・入出力・表示

\*サポートされている型番:VLP-16, VLP-32C, HDL-32E, HDL-64E, Puck LITE, Puck Hi-Res, VLS-128 R2019b

`pointCloud`

3次元点群用クラス

`pcread()`

PCD・PLYファイルの読み込み

`pcwrite()`

PCD・PLYファイルへのデータ書き出し

`pcfrokene()`

Microsoft Kinectからの3次元点群の読み込み（画像にアライメント）

`velodyneFileReader()`

Velodyne LiDAR\*のパケットキャプチャファイル(.pcap)の読み込み R2018b

`pcshow()`

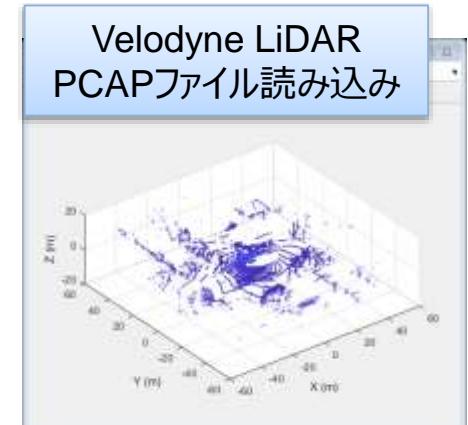
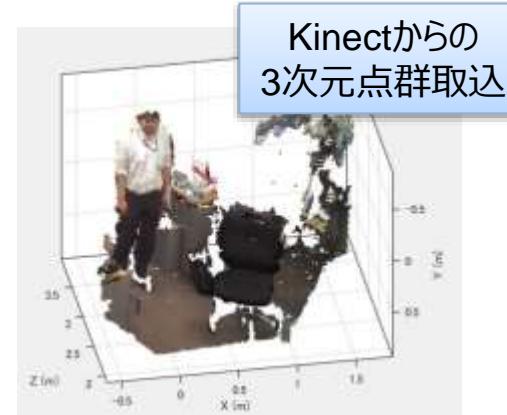
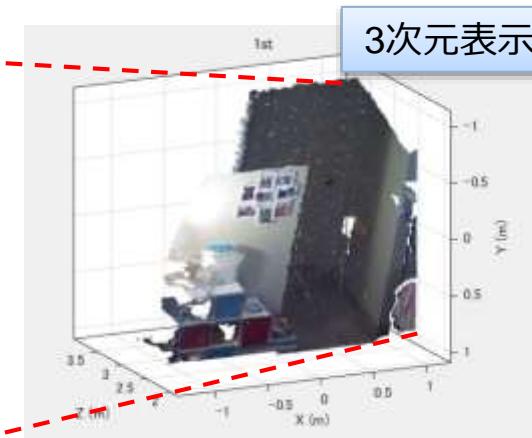
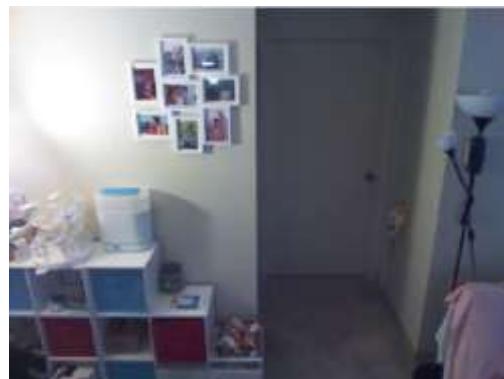
3次元点群の表示（任意の点での回転に対応 R2018a）

`pcshowpair()`

3次元点群の差分表示（任意の点での回転に対応 R2018a）

`pcplayer`

3次元点群のストリーミング表示（任意の点での回転に対応 R2018a）



## 4.10.2 3次元点群：各種処理

R2015a R2015b

### 各種3次元点群処理

<code>pcmerge () *</code>	2つの3次元点群データを統合
<code>pcdenoise () *</code>	3次元点群からのノイズ除去
<code>pcdownsample () *</code>	3次元点群の間引き
<code>pcregistericp ()</code>	2つの3次元点群のレジストレーション (ICPによる位置合わせ)
<code>pcregisterndt ()</code>	2つの3次元点群のレジストレーション (NDTによる位置合わせ)
<code>pcregistercpd () *</code>	2つの3次元点群のレジストレーション (CPDによる位置合わせ)
<code>pctransform () *</code>	3次元点群の幾何学変換 <span style="color: orange;">R2018b</span>
<code>pcfithplane () *</code>	3次元点群への面のフィッティング
<code>pcfithsphere () *</code>	3次元点群への球のフィッティング
<code>pcfithcylinder () *</code>	3次元点群への円柱のフィッティング
<code>pcsegdist () *</code>	3次元点群のセグメンテーション(クラスタリング) <span style="color: orange;">R2018a</span>
<code>segmentGroundFromLidarData ()</code>	3次元点群から路面検出 <span style="color: orange;">R2018b</span>

### [estimateCameraMatrix](#)

3次元から2次元への投影行列をDLT(Direct Linear Transform)アルゴリズムで推定  
RGB-Dカメラでの[findNearestNeighbors](#)による点群の近傍点探索を高速に

R2019a

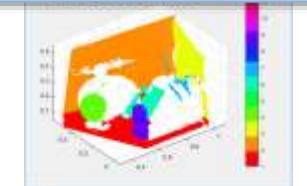
\*コード生成サポート R2019a

R2018b

複数の点群データからの  
3次元マップ生成

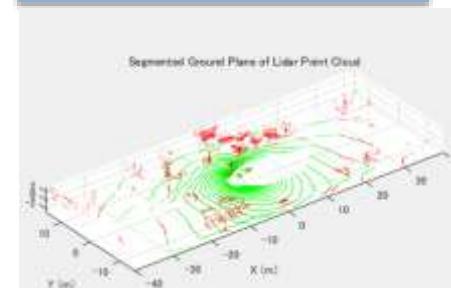


セグメンテーション



R2018a

路面検出

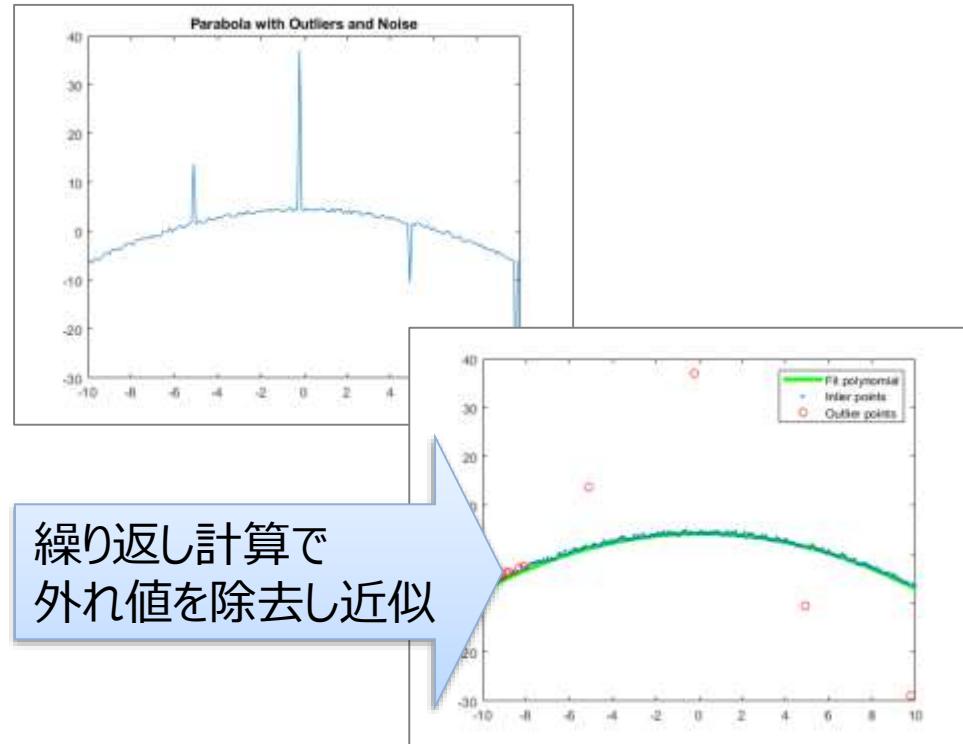


## 4.10.3 RANSACによるフィッティング

RANSAC(RAnom SAmple Consensus)は外れ値に強いロバスト推定手法

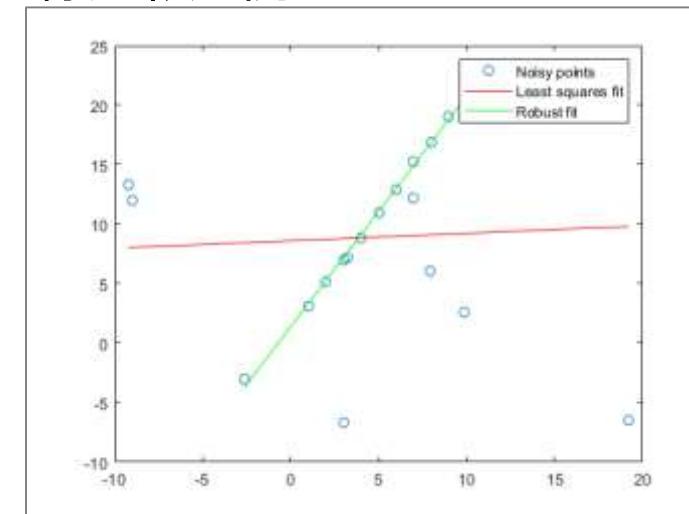
MSAC<sup>[1]</sup>(M-estimator Sample Consensus)はRANSACの一つで重付最小二乗法をベースにランダムサンプリングを繰り返す

MSACによる多項式近似



```
[P, inlierIdx] = ...
fitPolynomialRANSAC([x,y], N, maxDistance);
```

MSACによる任意モデルのフィッティング  
直線近似の例：



```
fitLineFcn = @(points) polyfit(points(:,1), points(:,2), 1); % モデル
evalLineFcn = ... % 評価用の関数
@(model, points) sum((points(:, 2) - polyval(model,
points(:, 1))).^2, 2);
% MSACでロバスト推定
[modelRANSAC, inlierIdx] =
ransac(points, fitLineFcn, evalLineFcn, ...
sampleSize, maxDistance);
```

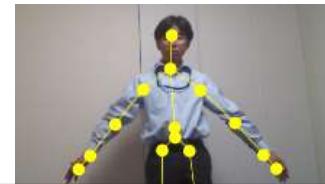
## 4.11 各種カメラからの画像データ直接取込み

Image Acquisition Toolbox

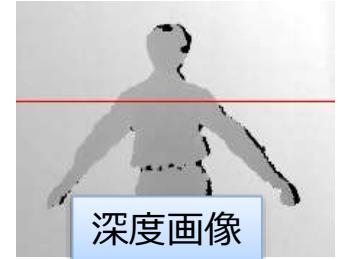
- 業界標準のHWからの動画像取込み機能を提供
  - フレームグラバ (画像入力ボード)
    - Analog 入力
    - Camera Link 入力
  - DCAM 互換 FireWire (IICC 1394)
  - GigE Vision (GenICamに対応 **R2019a**)
  - USB3 Vision **R2016a**
  - 一般的なUSB Webカメラ → 簡易的取込みはMATLABの関数で可能
  - IPカメラ (MATLAB基本関数)
- RGBD-Sensor
  - Microsoft Kinect for Windows v1 **R2015a**
  - Microsoft Xbox One Kinect センサー (Kinect v2)
- Velodyne社 LiDARデバイスサポートの拡充 **R2019a**
  - HDL-32E/VLP-32C Ultra Puck
  - VLP-16 Puck/VLP-16 Puck Lite/VLP-16 Puck Hi-Res
- カスタムアダプター開発キット
- Simulink ブロック



**R2014a**



RGB画像 + 骨格座標



深度画像

**R2016a**



**R2019a**



## 4.11.1 USBカメラからの動画像の取り込み

Image Acquisition Toolboxの  
System objectによる取り込み

R2012a

```
%% 画像を取り込むためのオブジェクトの生成
hCamera = imaq.VideoDevice('winvideo', 1)
hCamera.ReturnedDataType = 'uint8';

%% ビデオを表示するためのオブジェクトの生成
viewer = vision.DeployableVideoPlayer;

%% 1フレーム毎に処理するためのループ処理
for i=1:200
    I = step(hCamera); %1フレーム取り込み
    % 各種処理
    step(viewer,Itxt); %1フレーム表示
end

%%
release(hCamera);
release(viewer);
```

必要なハードウェアサポートパッケージ：  
Image Acquisition Toolbox Support Package for  
OS Generic Video Interface

MATLABの関数による取り込み

注) USB Video Class (UVC) compliant Webcam の場合のみ

R2014a

```
%% 画像を取り込むためのオブジェクトの生成
camera = webcam(1);

%% ビデオを表示するためのオブジェクトの生成
viewer = vision.DeployableVideoPlayer;

%% 1フレーム毎に処理するためのループ処理
for i=1:200
    I = snapshot(camera); %1フレーム取り込み
    % 各種処理
    step(viewer,Itxt); %1フレーム表示
end

%%
clear('camera');
release(viewer);
```

必要なハードウェアサポートパッケージ：  
USB Webcams

## 4.11.1 USBカメラからの動画像の取り込み

Image Acquisition Toolboxの関数による取り込み

```
%> 画像を取り込むためのオブジェクトの生成  
vidobj = videoinput('winvideo', 1)  
% マニュアルトリガでgetsnapshotのオーバーヘッドを削減  
triggerconfig(vidobj, 'manual')  
start(vidobj);  
  
%% ビデオを表示するためのオブジェクトの生成  
viewer = vision.DeployableVideoPlayer;  
  
%% 1フレーム毎に処理するためのループ処理  
for i=1:200  
    I = getsnapshot(vidobj); %1フレーム読み込み  
    % 各種処理  
    step(viewer,I); %1フレーム表示  
end  
  
%%  
stop(vidobj);  
delete(vidobj);  
release(viewer);
```

Image Acquisition Toolbox  
が提供する  
Simulink 用のブロック



必要なハードウェアサポートパッケージ：  
Image Acquisition Toolbox Support Package for  
OS Generic Video Interface

## 4.11.2 Microsoft Kinect for Windows

R2013a



Microsoft Kinectとの連携  
ハードウェアからの容易なデータ取り込み  
• 3次元座標・奥行き取得  
• ジェスチャ認識 etc



RGB画像 : 1280 x 960 / 12 fps or 640 x 480 / 30 fps  
Depth : 640x480, 320x240, 80x60 30fps  
Depth Mode: Default (range of 50 to 400 cm)  
Near (range of 40 to 300 cm).

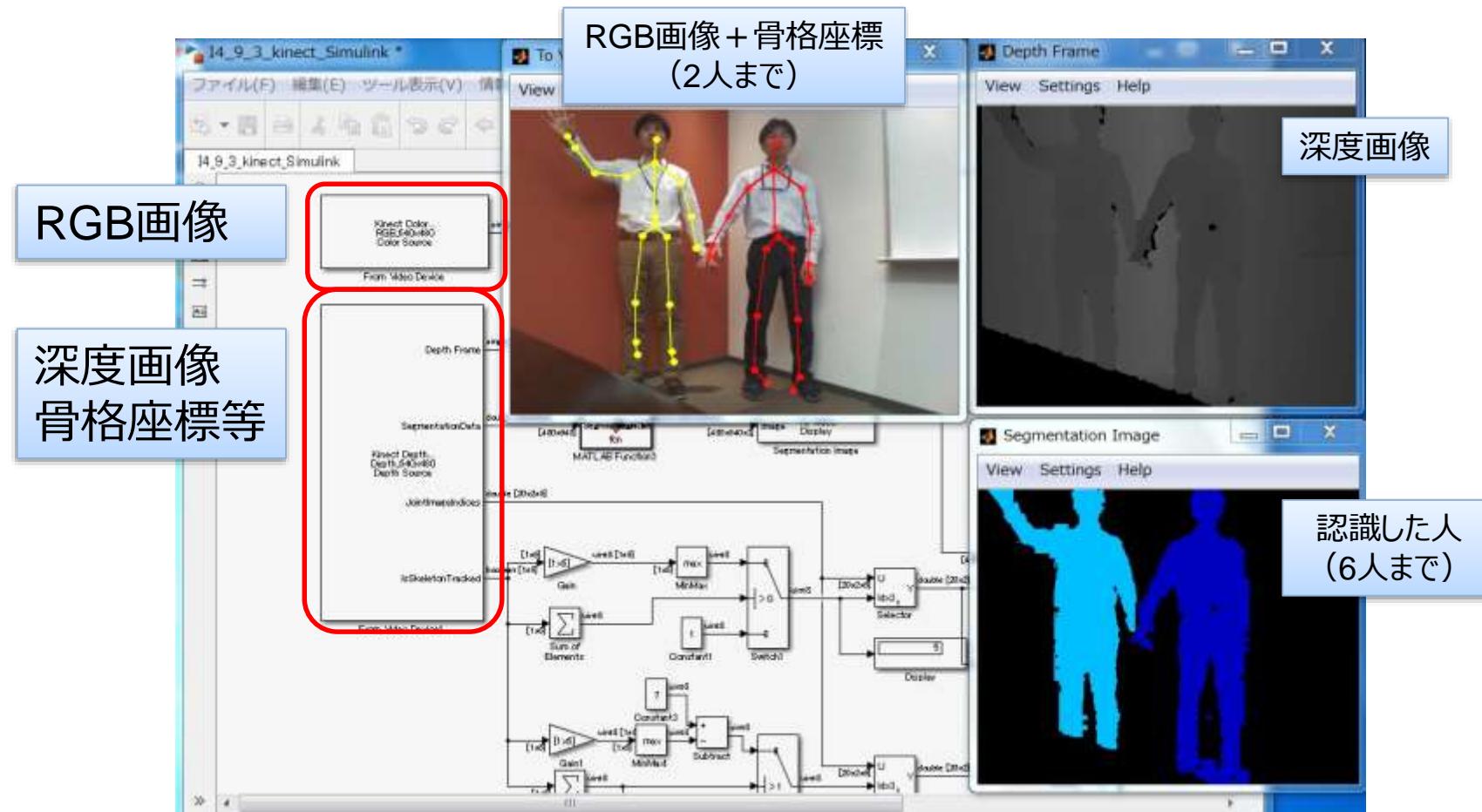
詳細 :

<http://www.mathworks.com/help/supportpkg/kinectforwindowsruntime/index.html>



## 4.11.3 Microsoft Kinect for Windows : - Simulink block と System objectでのサポート

R2013b



## 4.11.4 Xbox One Kinectセンサーサポート

R2016a

RGB画像 : 1920 x 1080 / 30 fps

Depth : 512x424 30fps (高精度レンジ : 0.5 ~ 4.5m)

人物領域 : 6人

Skelton : 6人

各Skeltonの関節 : 25関節



詳細 :

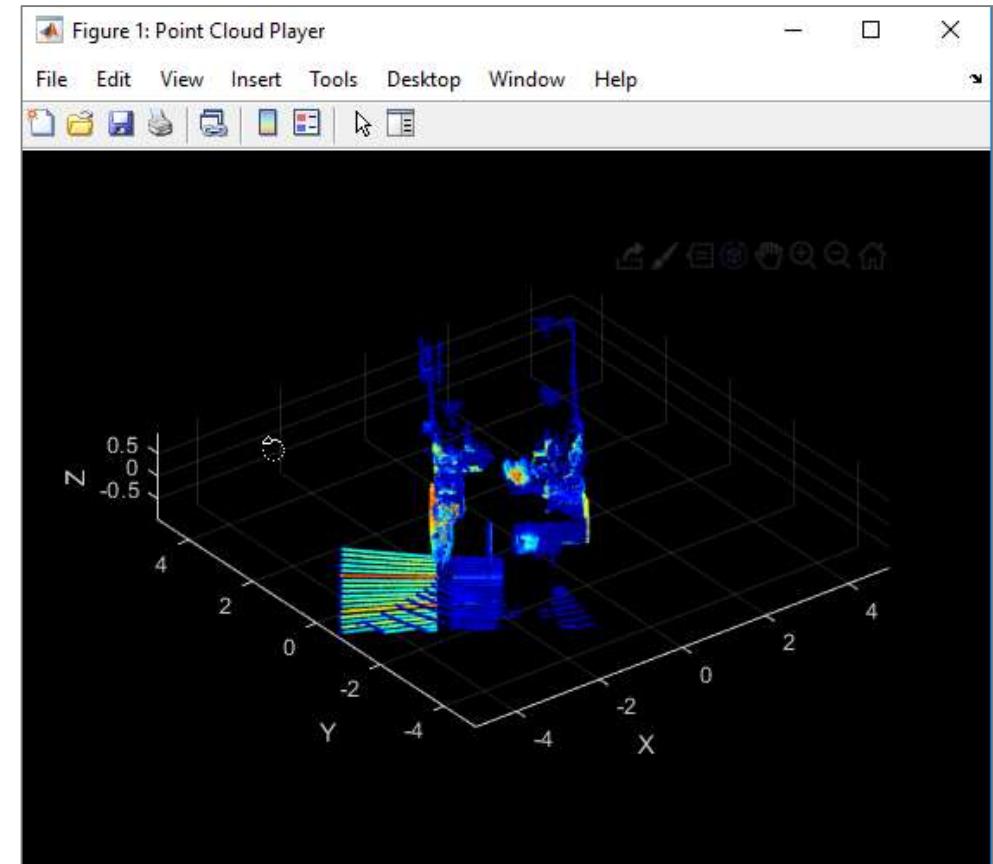
<http://www.mathworks.com/help/supportpkg/kinectforwindowsruntime/index.html>

## 4.11.5 Velodyne LiDAR®デバイスのサポート

- Velodyne LiDARデバイスサポートの拡充
  - HDL-32E/VLP-32C Ultra Puck
  - VLP-16 Puck/VLP-16 Puck Lite/VLP-16 Puck Hi-Res



```
% プレビュー  
lidar = velodynelidar('VLP16');  
preview(lidar)  
pause(10)  
closePreview(lidar)  
  
% 取得  
start(lidar)  
[pcloud, timestamp] = read(lidar, 'latest');  
pcshow(pcloud);
```



詳細：

<https://www.mathworks.com/hardware-support/velodyne-lidar.html>

# アジェンダ

1. MATLAB/Simulinkの概要
2. 各種画像処理例
3. 連携機能
4. コンピュータービジョン処理例
5. 画像の機械学習・ディープラーニング
6. まとめ

# 5.1 物体認識（顔認識・人物認識）（機械学習済）

% 顔認識 用のオブジェクト (Viola-Jonesアルゴリズム)

```
vision.CascadeObjectDetector()
```

% 人物認識 用のオブジェクト

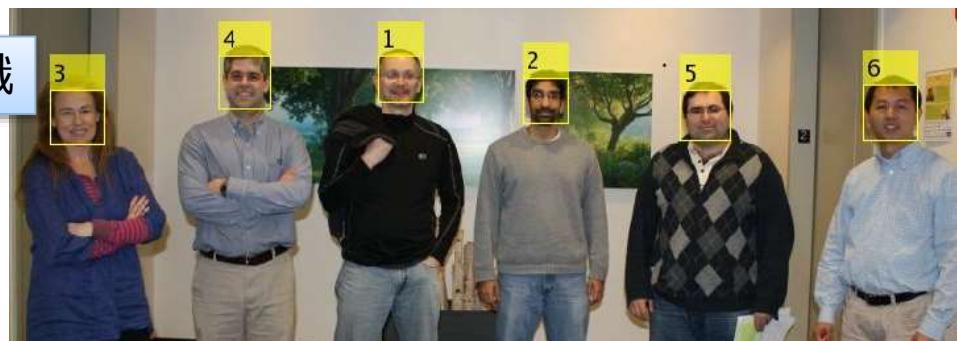
```
vision.PeopleDetector()
```

```
detectPeopleACF()
```

HOG 特徴量

ACFアルゴリズム R2016a

顔認識



人物認識



```
I = imread('visionteam.jpg');
detector = vision.CascadeObjectDetector();
faces = step(detector, I)
I2 = insertObjectAnnotation(I, 'rectangle', faces, 'Face'); % 枠描画
figure; imshow(I2); % 表示
```

% 画像の読み込み  
% 顔検出オブジェクト定義  
% 顔検出  
% 枠描画  
% 表示

わずか数行のMATLABコードで、人の顔認識

## 5.2 機械学習：静止画のラベリングアプリケーション

imageLabeler

%静止画ラベリング アプリケーション

サブラベル、属性のサポート

R2019b

複数種のラベリングに対応

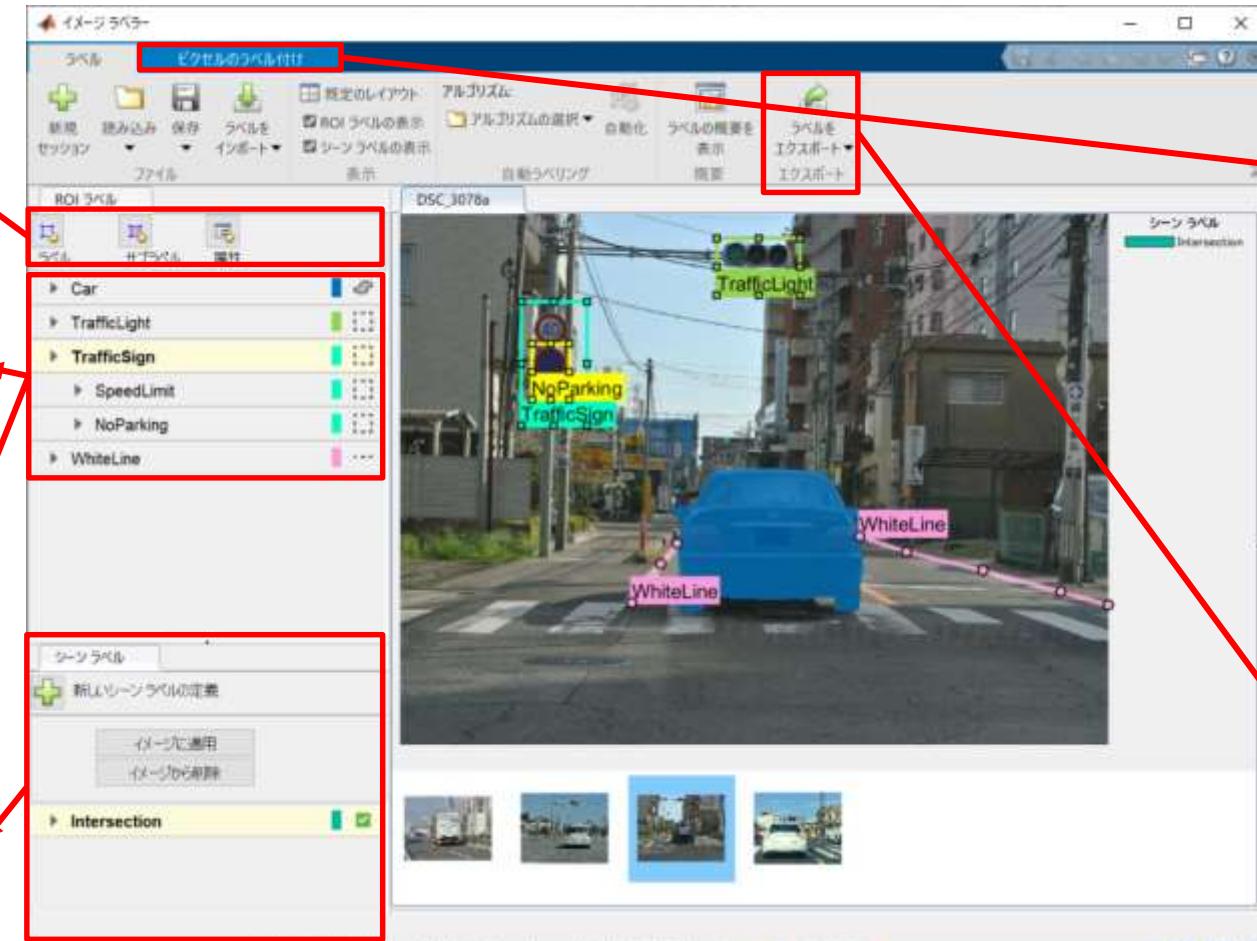
- 領域
- ピクセル
- 線

R2016b  
R2019bラベル定義のグループ化、  
リネーム、並び替えサポート

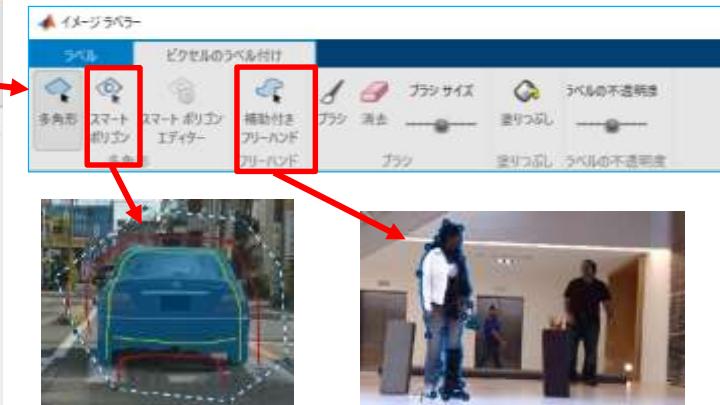
R2019a

シーンのラベリングに対応

R2017b



ピクセルレベルのラベル付け



Grab Cut/エッジ吸着を用いた半自動化

R2018a R2019a

ラベリング結果を保存する  
groundTruthオブジェクトを生成

生成したgroundTruthオブジェクトのフィルタリング

- [selectLabelsByName](#)
- [selectLabelsByType](#)
- [selectLabelsByGroup](#)

R2019a

## 5.2 機械学習：動画用のラベリングアプリケーション

R2018b

`videoLabeler('visiontraffic.avi')` %動画ラベリング アプリケーション

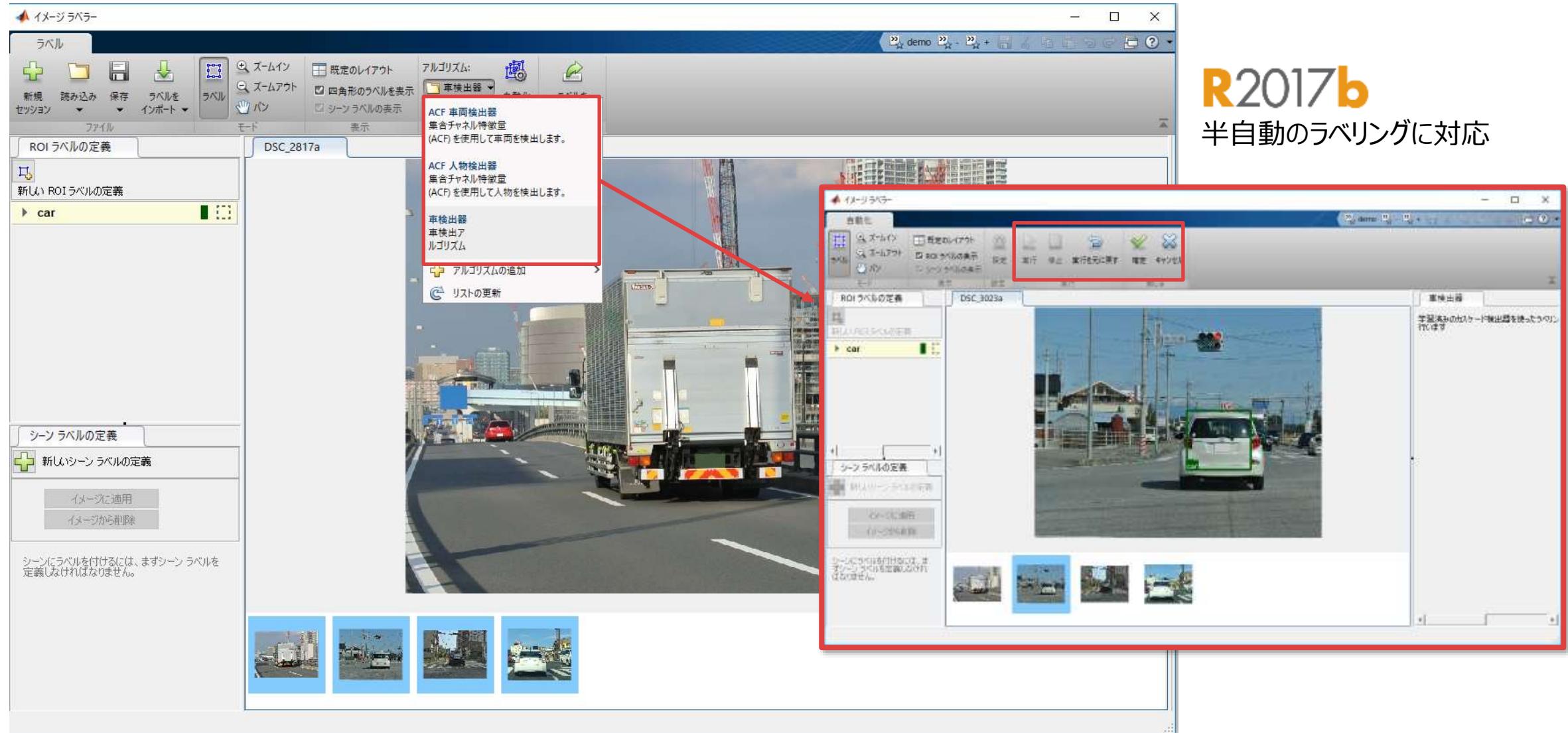


`changeFilePaths`

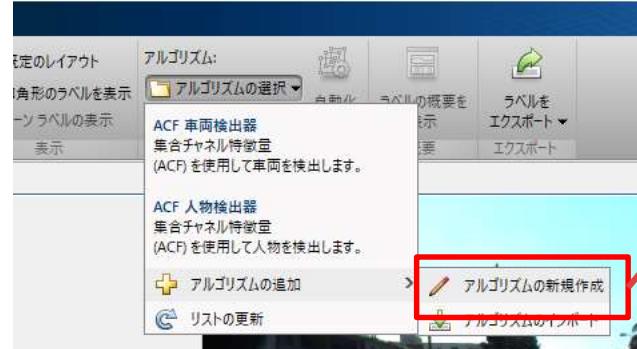
エクスポートしたgroudTruthオブジェクトのソースのパスを容易に変更可能

## 5.2 機械学習：ラベリング自動化アルゴリズム

imageLabeler, videoLabeler 共通



## 5.2 機械学習：ラベリング自動化アルゴリズムの追加



任意の自動アルゴリズムを追加可能  
R2017b



カスタムアルゴリズムの追加例：

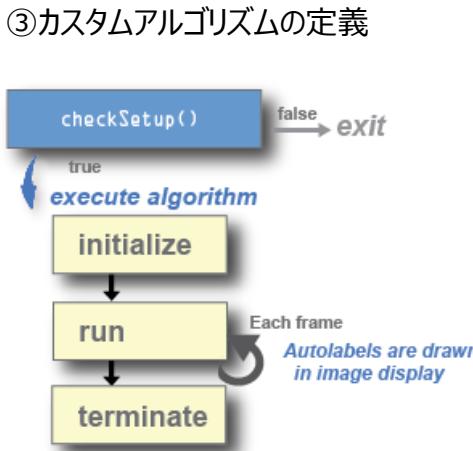
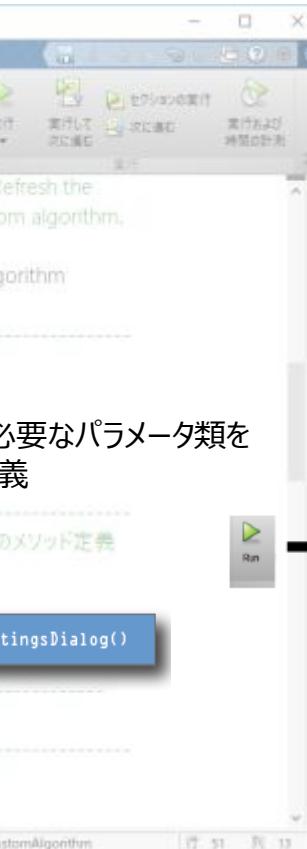
- Superpixelとk-meansによる候補領域抽出

<https://jp.mathworks.com/matlabcentral/fileexchange/67528-superpixel-and-k-means-based-semi-automation-algorithm-for-pixel-level-labeling>

- 高ダイナミックレンジ画像のためのコントラスト補正

<https://jp.mathworks.com/matlabcentral/fileexchange/68973-interactive-contrast-adjustment-tool-running-on-labeler-apps>

imageLabeler, videoLabeler 共通



## 5.2 機械学習用の関数

% 機械学習用の関数 [Haar-like/HOG/LBP/ACF]

`trainCascadeObjectDetector()`

Haar/HOG/LBP特徴による検出器の学習

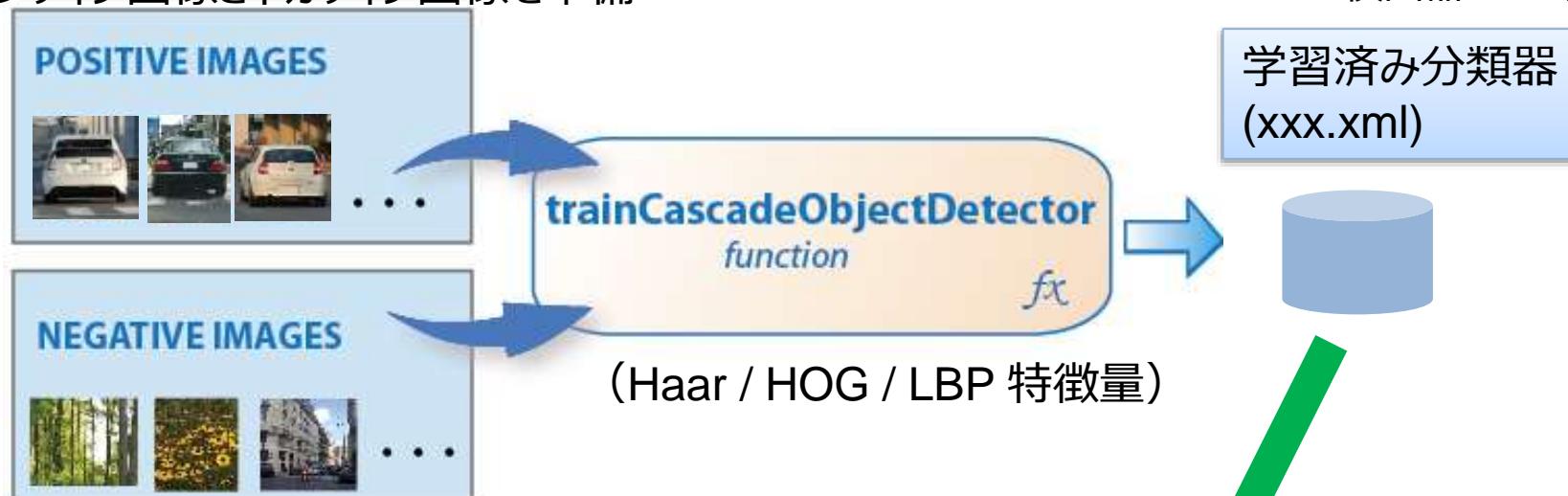
`trainACFOBJECTDetector()`

ACF特徴による検出器の学習

R2017a

ポジティブ画像とネガティブ画像を準備

ACF検出器のコード生成サポート R2019a



```
I = imread('IMG_123.jpg');
detector = vision.CascadeObjectDetector('xxx.xml');
cars = step(detector, I)
```

% 未知の画像の読み込み  
% 検出オブジェクト定義  
% 検出



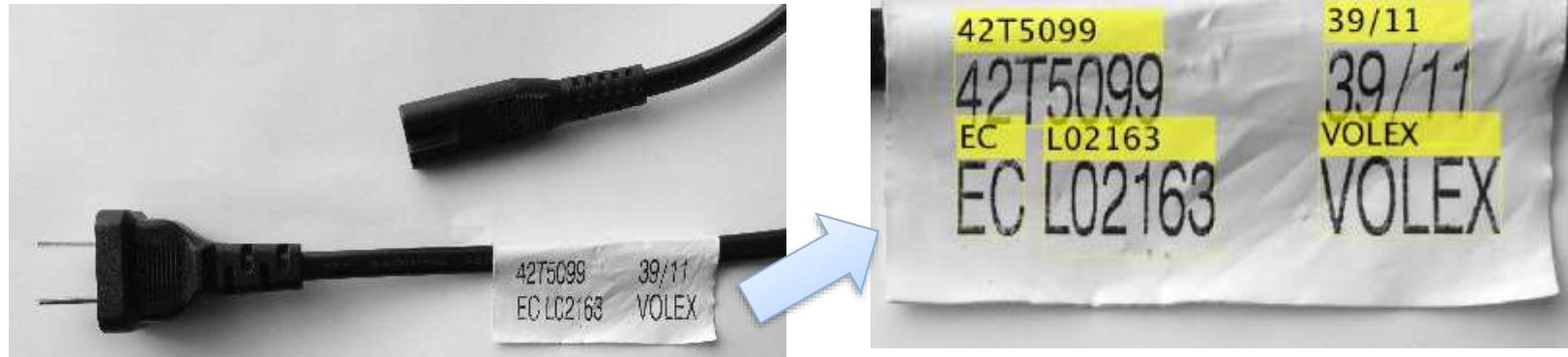
検出



## 5.3.1 文字認識

R2014a

- OCR (光学文字認識)



様々な画像処理機能を前処理として組み合わせ

```

G = imread('I4_3_ocrIMG_2517_cable.jpg');           %% 画像の読み込み・表示
G1 = imerode(G, ones(4));                         %% 白い部分を削り、字を太くつきりさせる
G2 = imbothat(G1, ones(19));                      %% ボトムハット処理で、字の部分のみを残す

results = ocr(G2)                                %% 文字認識

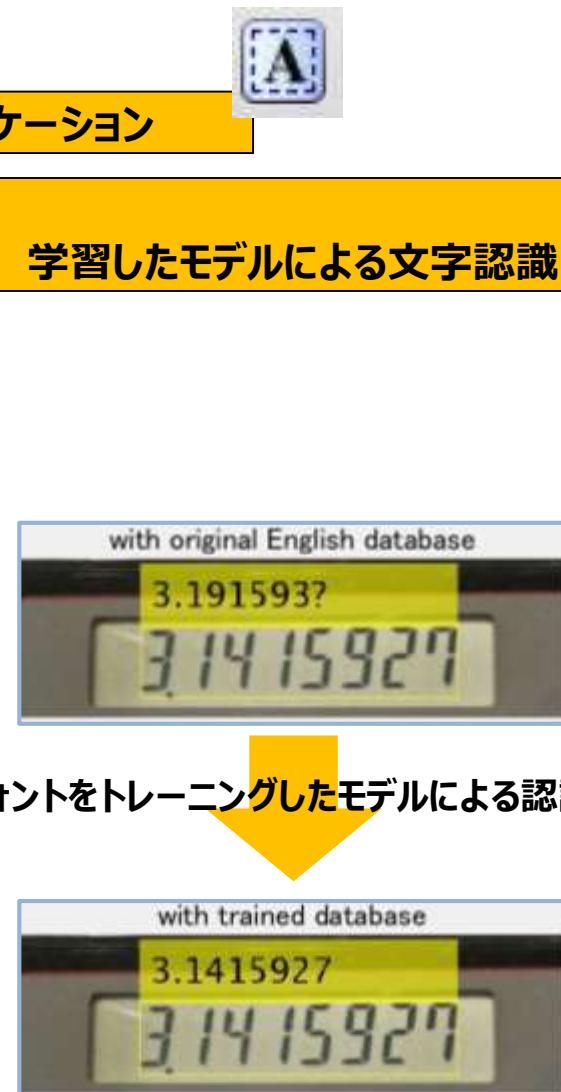
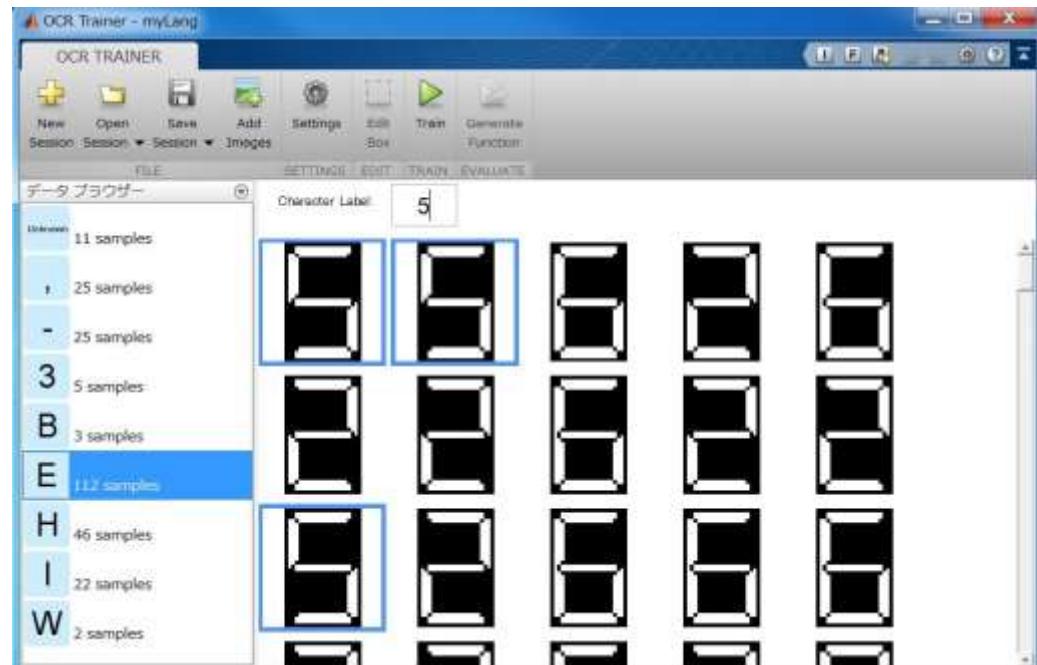
%% 確度の高い文字のみを表示
valid = (results.WordConfidences > 0.5) & ~strcmp(results.Words, '')
Ir = insertObjectAnnotation(G, 'rectangle', ...
    results.WordBoundingBoxes(valid,:), results.Words(valid));
imtool(Ir);

```

## 5.3.2 文字認識：カスタムフォント用学習アプリケーション

R2016a

```
ocrTrainer % 文字認識用モデルのトレーニング用アプリケーション  
  
results = ocr(BW1, 'Language', ...  
    'myLang\%tessdata\%myLang.traineddata') % 学習したモデルによる文字認識
```



## 5.4 機械学習ワークフロー：サポートベクターマシンの例

Statistics and Machine Learning Toolbox™



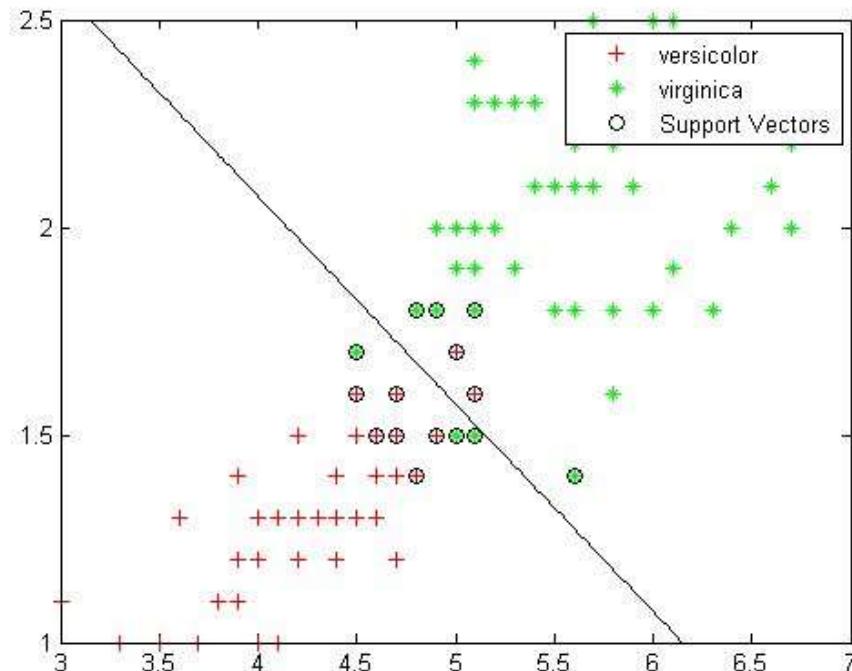
```

for i = 1:size(trainSet.Labels,1)
    img = readimage(trainSet, i); %トレーニング画像の読み込み
    img = imbinarize(rgb2gray(img)); % 二値化
    trainingFeatures(i,:) = ...
        extractHOGFeatures(img,'CellSize',cellSize); %特徴量の抽出
end
trainingLabels = (trainSet.Labels == '2'); % ラベルの生成
% サポート ベクトル マシンの分類器の学習
svmModel = fitcsvm(trainingFeatures, trainingLabels)

```

## 5.4.1 教師あり学習・分類：サポートベクターマシン (SVM)

1つのクラスの出来るだけ多くのデータ点を、他のクラスのデータ点から分離する超平面を作成（マージンを最大化） → **2クラス分類**



分類器の学習：

```
svmModel = fitcsvm(X, T);
```

学習に使う特徴量の集合

各特徴量の属するクラス（ラベル）

分類の実行：

```
predT = predict(svmModel, Y);
```

分類したい画像からの特徴量

予測されたクラス（ラベル）

多クラス分類器の場合：**fitcecoc()**

## 5.4.1 教師あり学習・分類：マルチクラス SVM

複数の2クラス分類器を組合わせて、多クラスの分類器を構成

完全2項 符号化設計	学習器1	学習器2	学習器3
Class1	符号化設計 : 1	1	1
Class2	1	-1	-1
Class3	-1	1	-1

1対1 符号化設計	学習器1	学習器2	学習器3
Class1	1	1	0
Class2	-1	0	1
Class3	0	-1	-1

1対他 符号化設計	学習器1	学習器2	学習器3
Class1	1	-1	-1
Class2	-1	1	-1
Class3	-1	-1	1

分類器の学習：

```
svmModel1 = fitcecoc(X, T);
```

学習に使う特徴量の集合

各特徴量の属するクラス（ラベル）

学習器数 :  $K(K-1)/2$

学習器数 :  $K$

学習器数 :  $2^{(K-1)} - 1$

## 5.4.2 教師あり学習・分類：k最近傍分類（kNN）

k個の最近傍点から、多数決で分類を決定 → 多クラス分類



ユーザーが指定

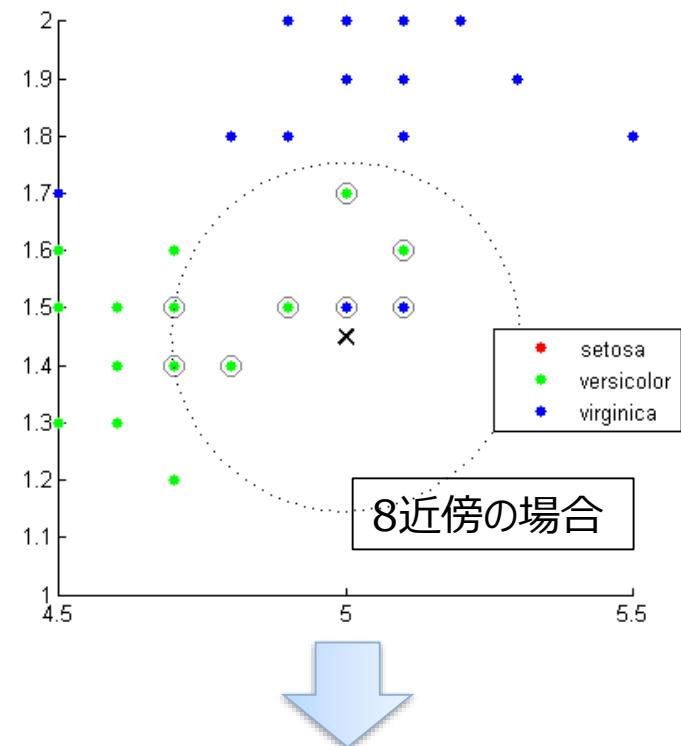
分類器の学習：

```
knnModel = fitcknn(X, T, 'NumNeighbors', 5);
```

学習に使う特徴量の集合

使用する近傍の数

各特徴量の属するクラス（ラベル）



分類の実行：

```
predT = predict(knnModel, Y);
```

分類したい画像からの特徴量

予測されたクラス（ラベル）

× 未知のデータのクラスは、8近傍中に6つある、

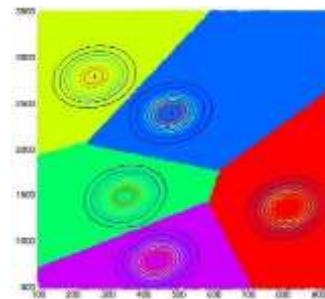
● versicolor

## 5.4.3 判別分類：線形判別分類・二次判別分類

各クラスのデータが、多変量正規分布を仮定して、クラスを分離する面を学習

線形判別分類：各クラスで共分散は同一。平均は各クラスで異なる

二次判別分類：各クラスで、共分散と平均共に異なる

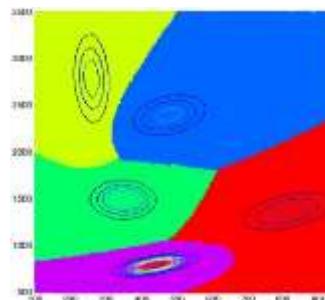


線形判別分類器の学習：

```
discrModel = fitcdiscr(X, T);
```

学習に使う特徴量の集合

各特徴量の属するクラス（ラベル）



二次判別分類器の学習：

```
discrModel = fitcdiscr(X, T, ...
    'DiscrimType', 'quadratic');
```

分類の実行 : `predT = predict(discrModel, Y);`

分類したい画像からの特徴量

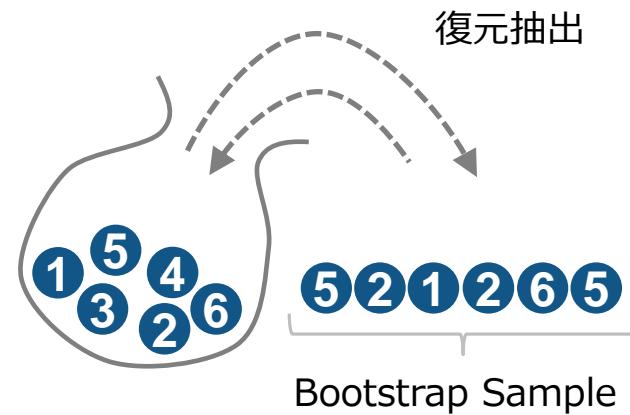
予測されたクラス（ラベル）

各種分類アルゴリズムの特性

## 5.4.4 バギングされた決定木 (Bagged Decision Tree)

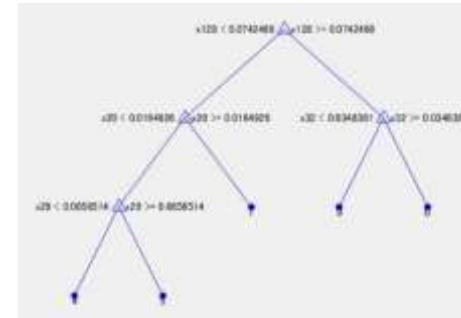
- バギング (Bootstrap Aggregating)

- Bootstrapping : 対象のデータから、復元抽出（重複可。未使用データあっても可）を行いデータのサブセットを生成
- Aggregation : 複数の識別器の認識結果に対し、多数決(分類の場合)や平均(回帰)を取る



重複サンプリングを許す  
使われていないサンプルあっても可

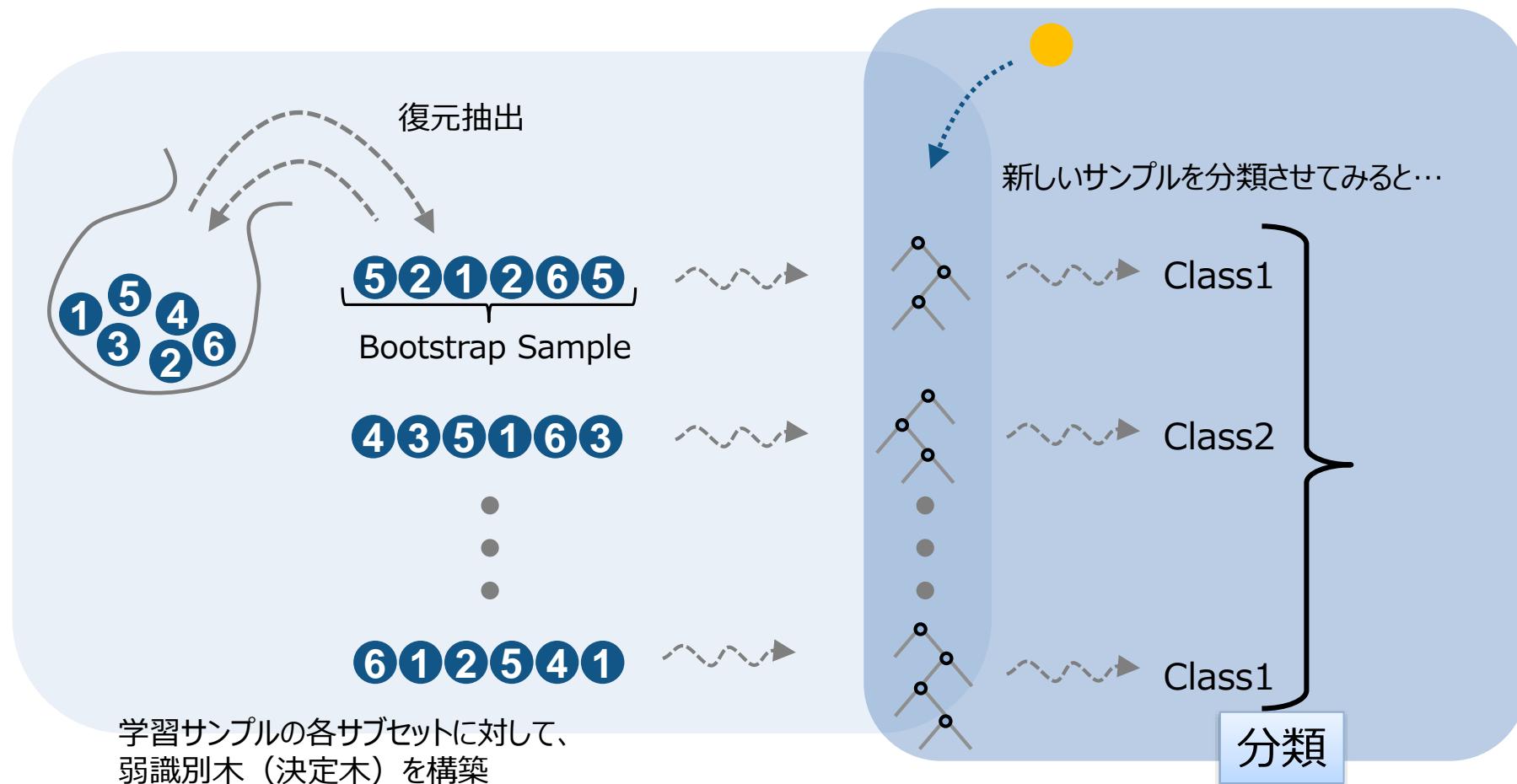
決定木 を学習



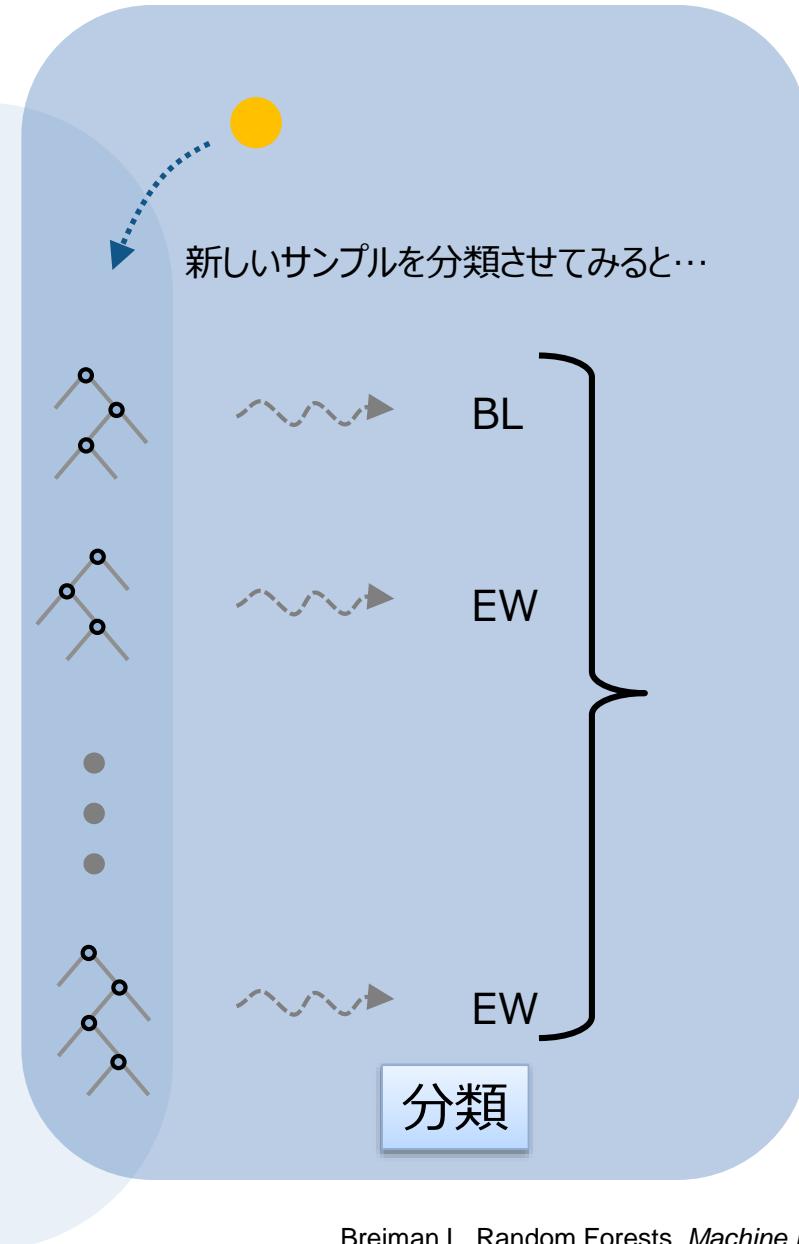
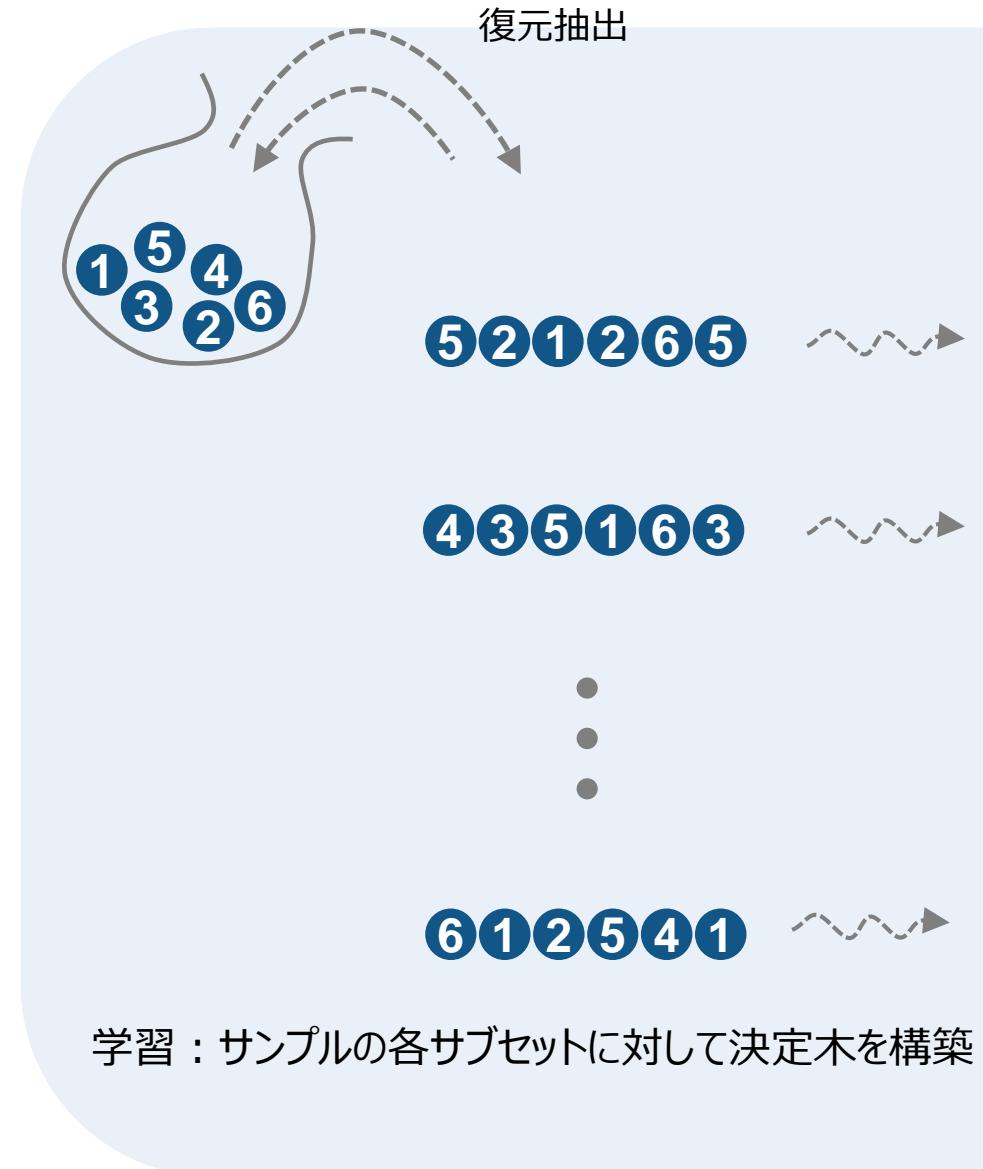
## 5.4.4 バギングされた決定木 (Bagged Decision Tree)

- バギング (Bootstrap Aggregating)

- Bootstrapping : 対象のデータから、復元抽出（重複可。未使用データあっても可）を行いデータのサブセットを生成
- Aggregation : 複数の識別器の認識結果に対し、多数決(分類の場合)や平均(回帰)を取る

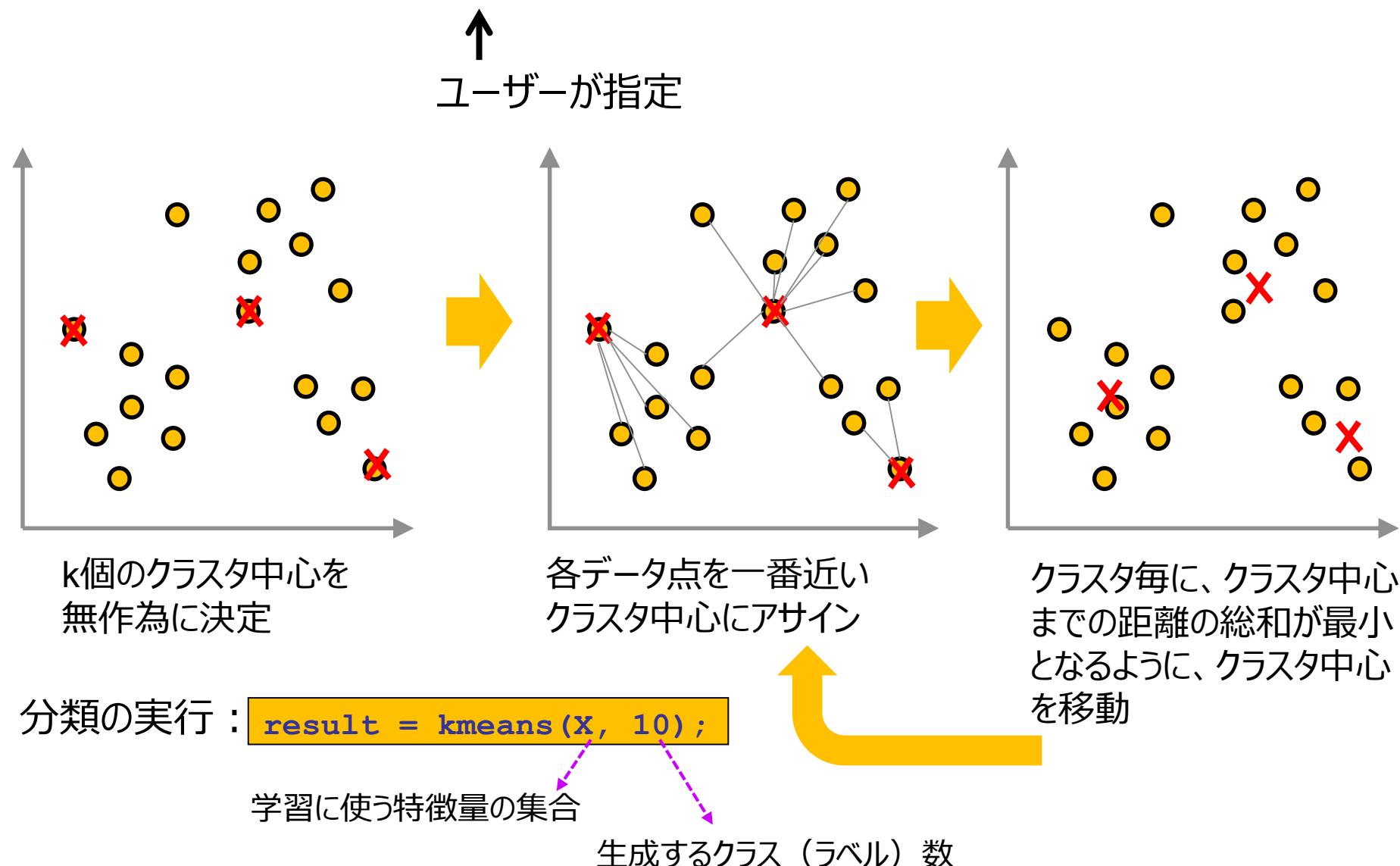


## 5.4.4 ランダムフォレスト



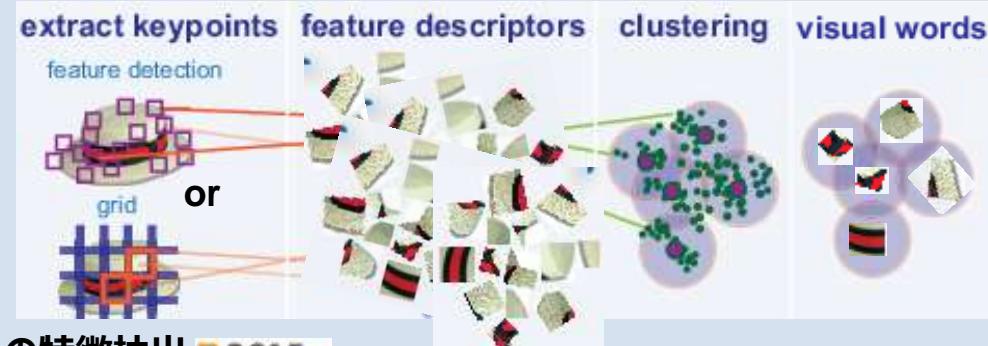
## 5.4.6 教師なし学習：k 平均 (k-means) クラスタリング

データの集合を、**k**個のグループに分割



## 5.5.1 Bag-of-Visual-Wordsを用いた画像カテゴリ分類

### 全画像からVisual Wordsの抽出



カスタムの特徴抽出 R2015a

### imageSet クラス

大量の画像取扱い用のクラス

### bagOfFeatures ()

全カテゴリーの画像から  
特徴量の抽出後、  
K-meansでクラスタリングし、  
K個のVisual Wordsを生成

### 各学習画像からVisual Wordsヒストグラムを生成し、分類器を生成



### trainImageCategory Classifier ()

各画像の特徴量をVisual  
Wordsに分類後、ヒストグラムを  
作成し、K次元の特徴ベクトル  
画像内の各特長の とする  
位置関係は考慮しない

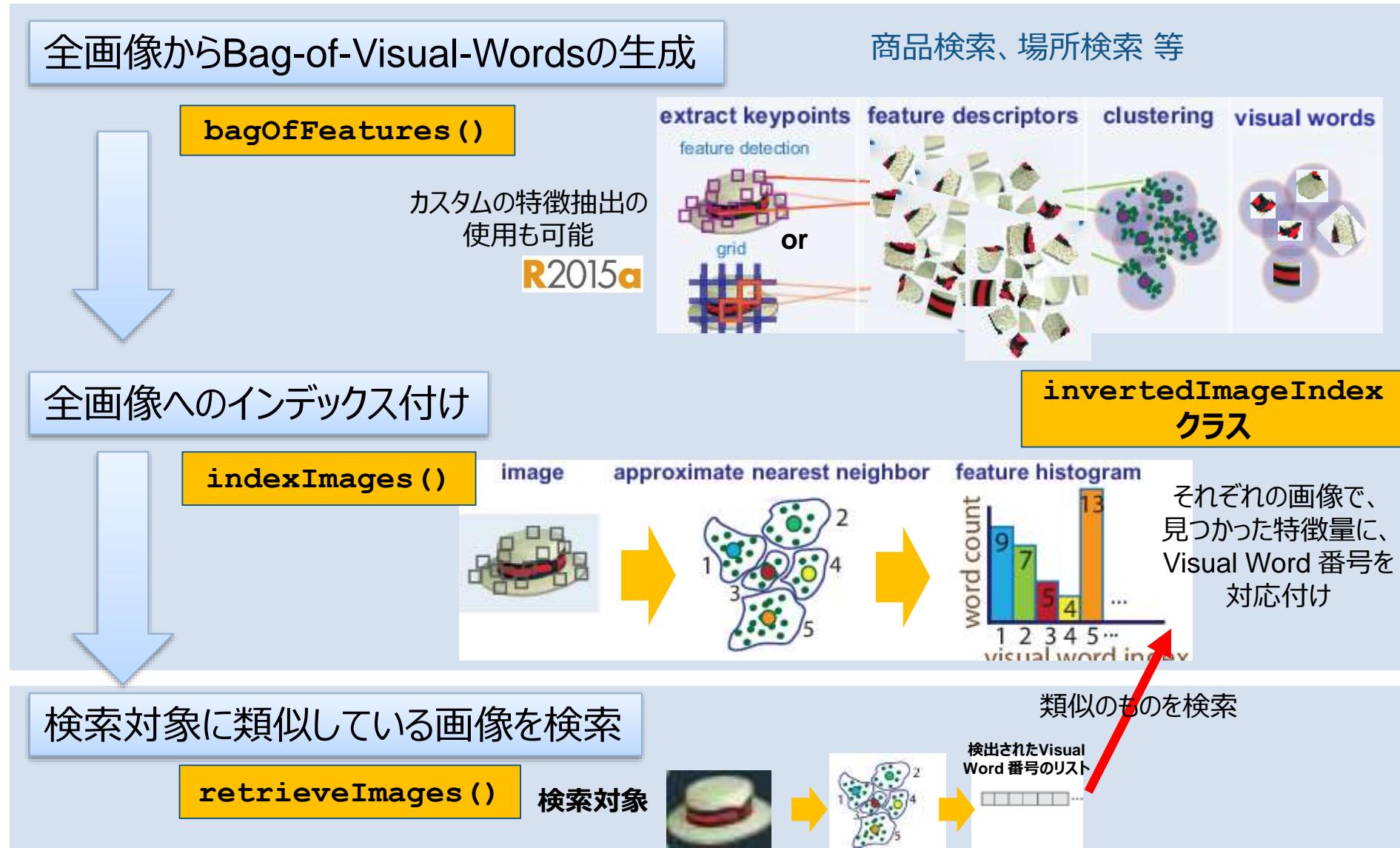
trainImageCategoryClassifier() は  
Statistics and Machine Learning Toolbox が必要



各学習用の画像を、Visual  
Wordsのヒストグラムで表し、機  
械学習により分類器を生成

## 5.5.2 Bag-of-Visual-Wordsを用いた類似画像検索

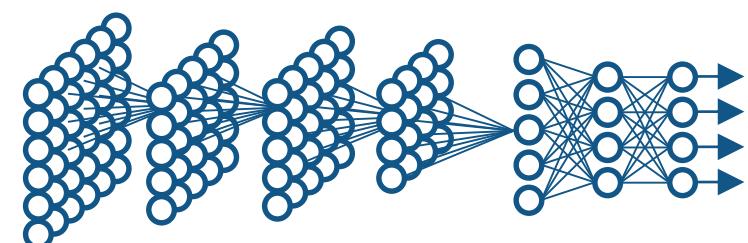
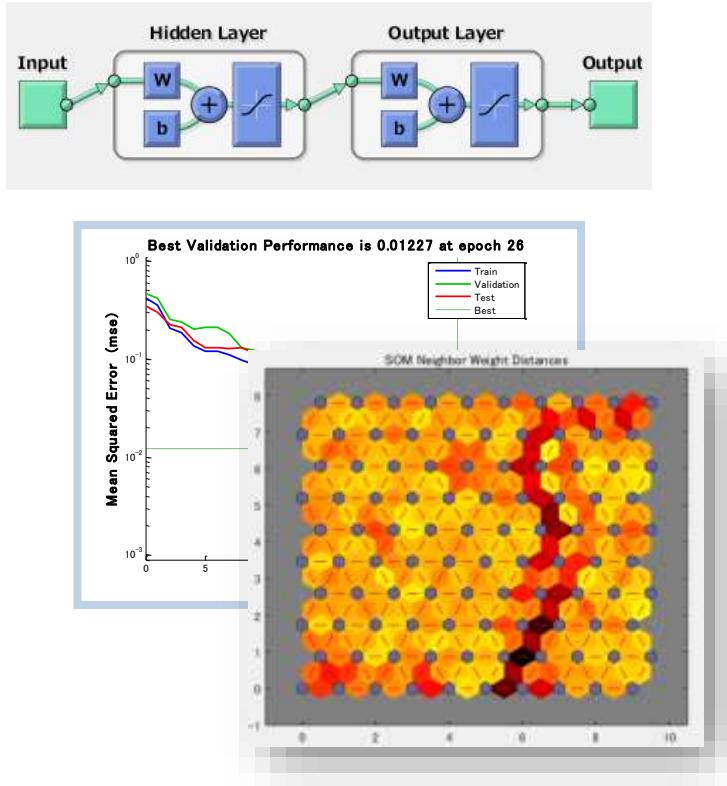
R2015a



# 5.6 ニューラルネットワークを用いた機械学習

Deep Learning Toolbox™

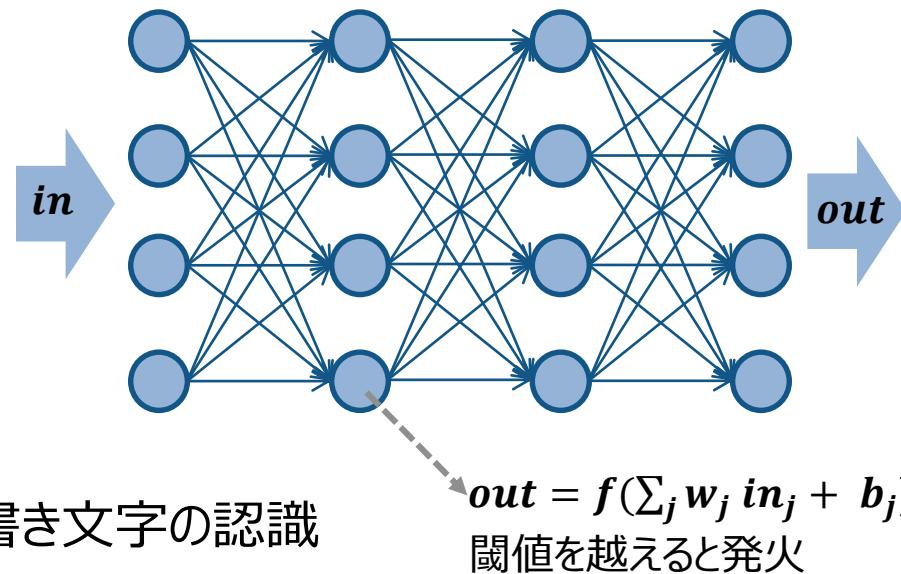
- 教師あり学習
  - フィードフォワードネットワーク
  - 放射基底ネットワーク
  - 動的ネットワーク (NARX, Layer-Recurrent)
  - 学習ベクトル量子化 (LVQ)
- 教師なし学習
  - 競合学習、自己組織化マップ (SOM)
- 汎化性能の改善
  - 正則化、早期終了
- 制御システムへの適応
  - モデル予測制御、線形化、モデル規範型適応制御
  - SimulinkブロックおよびSimulink Coderへの対応
- 大規模データ及び学習の高速化
  - 並列分散およびGPU上での演算
- 深層学習 (Deep Learning) への対応
  - Stacked Autoencoder
  - Convolutional Neural Networks(分類、回帰、カスタム)



## 5.6.1 積層自己符号化器 (Stacked Autoencoder)

R2015a R2015b  
Deep Learning Toolbox™

Parallel Computing Toolboxを使うことで、  
GPUや並列処理による高速化が可能



例) Stacked Autoencoder による、手書き文字の認識

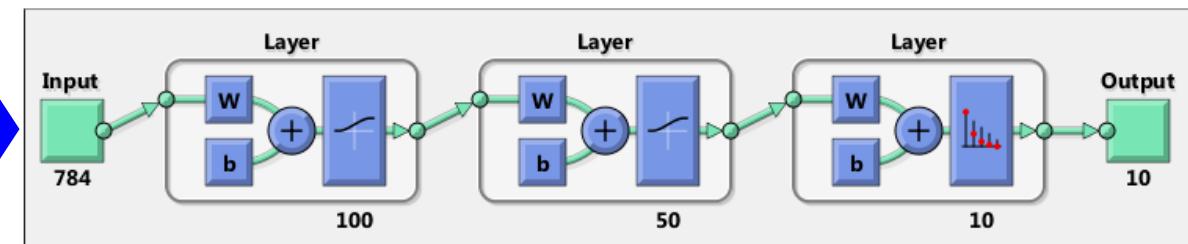
1

学習後の、Deep Neural Network

9

8

2



100個のノード

50個のノード

10クラスに分類

1

9

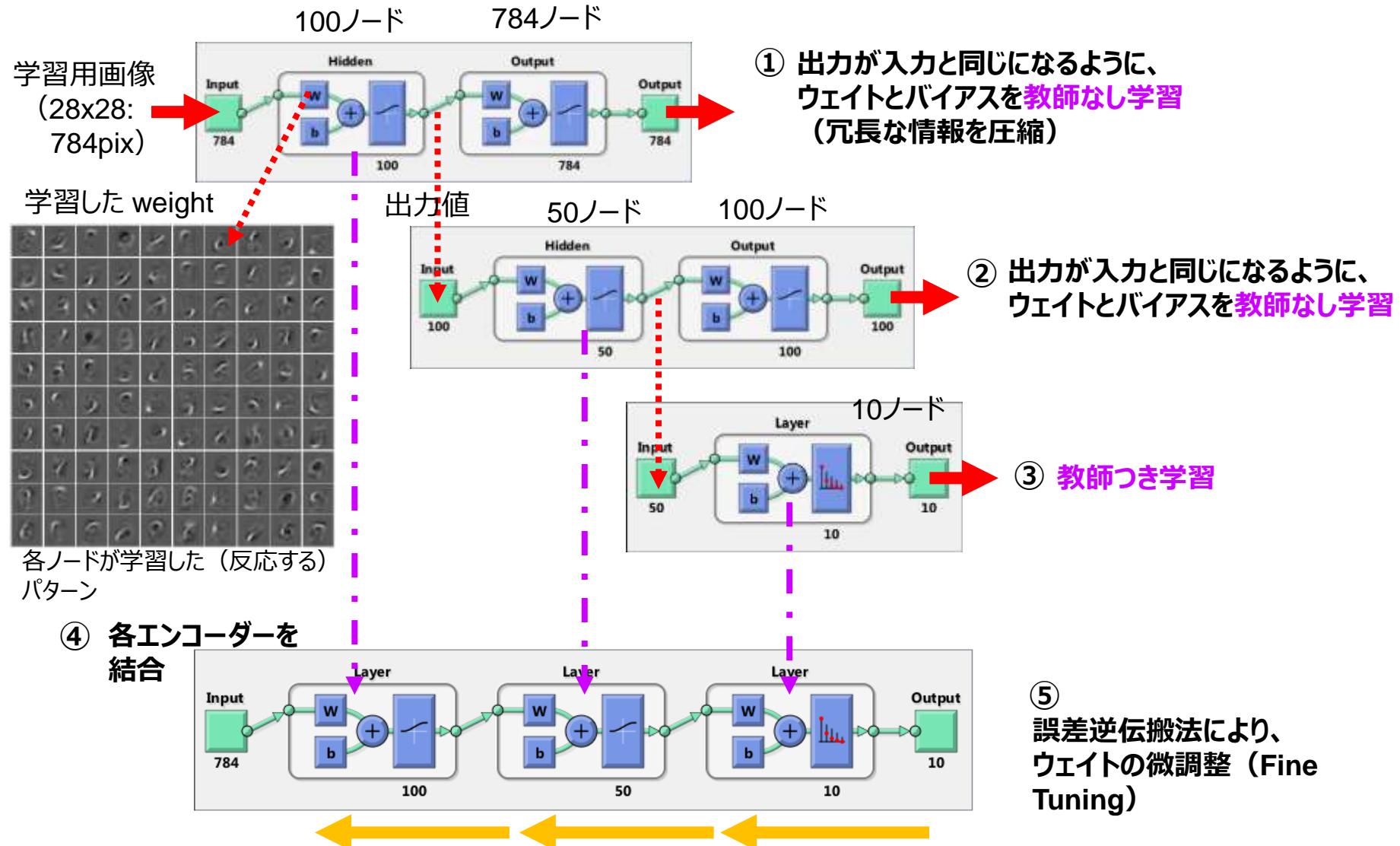
8

2

# 5.6.1 積層自己符号化器 (Stacked Autoencoder)

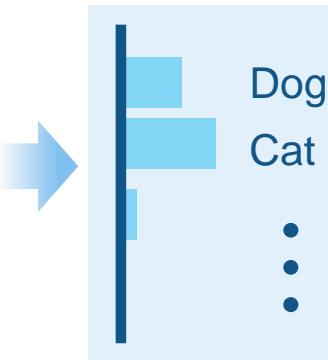
Deep Learning Toolbox™

Autoencoderによる Deep Neural Networkの学習：一層ずつ学習



## 5.6.2.1 ディープラーニングによる画像分類・検出・領域切り出し

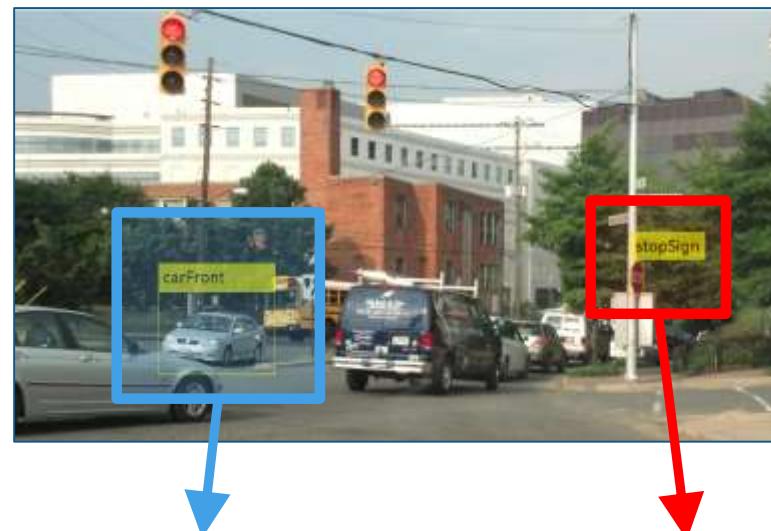
画像分類（画像全体）



画像

確率値

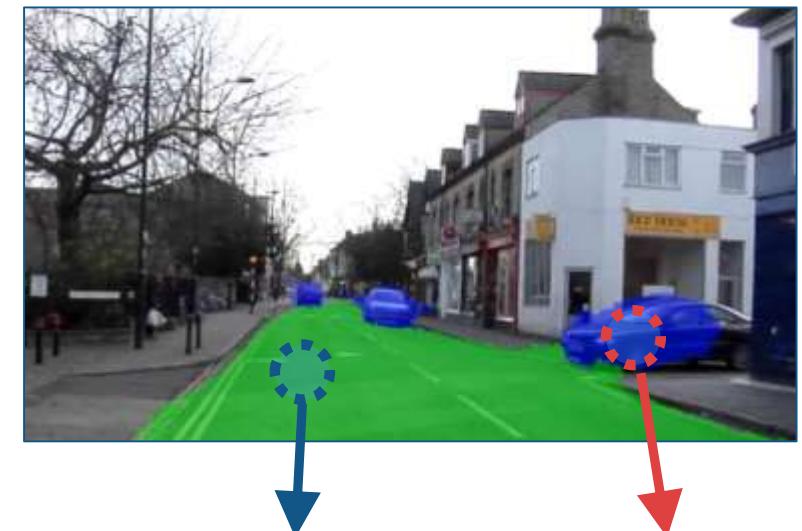
物体の検出



自動車の前面

停止標識

物体の検出と領域の切り出し



車道

自動車

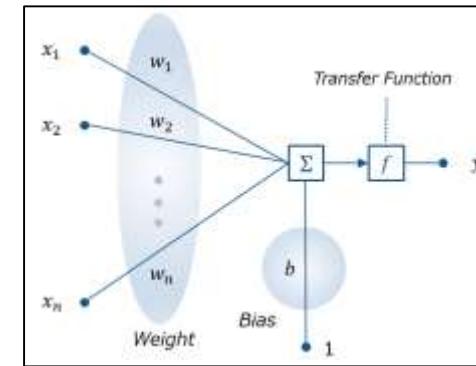
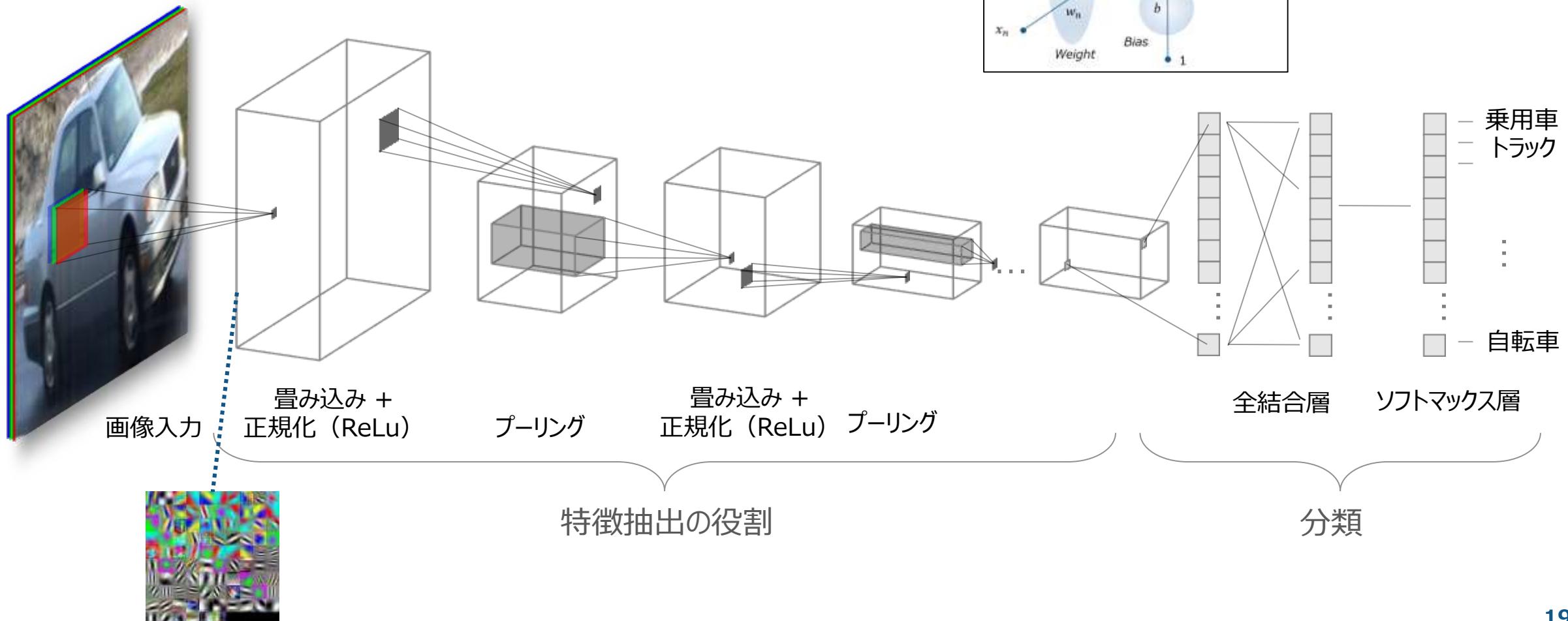
CNN (Convolutional Neural Network)

R-CNN / Fast R-CNN / Faster R-CNN

SegNet / FCN

## 5.6.2.1 ディープラーニング：畳み込みニューラルネットによる分類

畳み込み層・プーリング層・正規化層などを  
積み重ねて作られた多層のニューラルネットワーク  
畳み込みニューラルネットワーク (CNN)



## 5.6.2.1 ディープラーニング：畳み込みニューラルネットによる分類

R2016a

28×28 ピクセルの画像（数字）を認識させる例題でのネットワーク構築の例

```
digitDatasetPath = fullfile(matlabroot,'toolbox','nnet',...
    'nndemos','nndatasets','DigitDataset');
imds = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');
```

```
layers = [ ...
    imageInputLayer([28 28 1], 'Normalization', 'none');
    convolution2dLayer(5, 20);
    reluLayer();
    maxPooling2dLayer(2, 'Stride', 2);
    fullyConnectedLayer(10);
    softmaxLayer();
    classificationLayer()];
```

```
opts = trainingOptions('adam', 'MaxEpochs', 10);
net = trainNetwork(imds, layers, opts);
```



..... 畳み込み層・プーリング層・正規化層  
などの層を積み上げて定義

..... 学習率や最大反復数などを定義して  
学習の実行

GPU有無を自動で判定。あればGPU、なければCPUで学習。

<http://www.mathworks.com/help/releases/R2017b/nnet/ref/trainnetwork.html>

Deep Learning Toolbox, Parallel Computing Toolbox  
compute capability 3.0以上のCUDA GPUが必要

## 5.6.2.1 ディープラーニング：ディープネットワークデザイナーアプリ

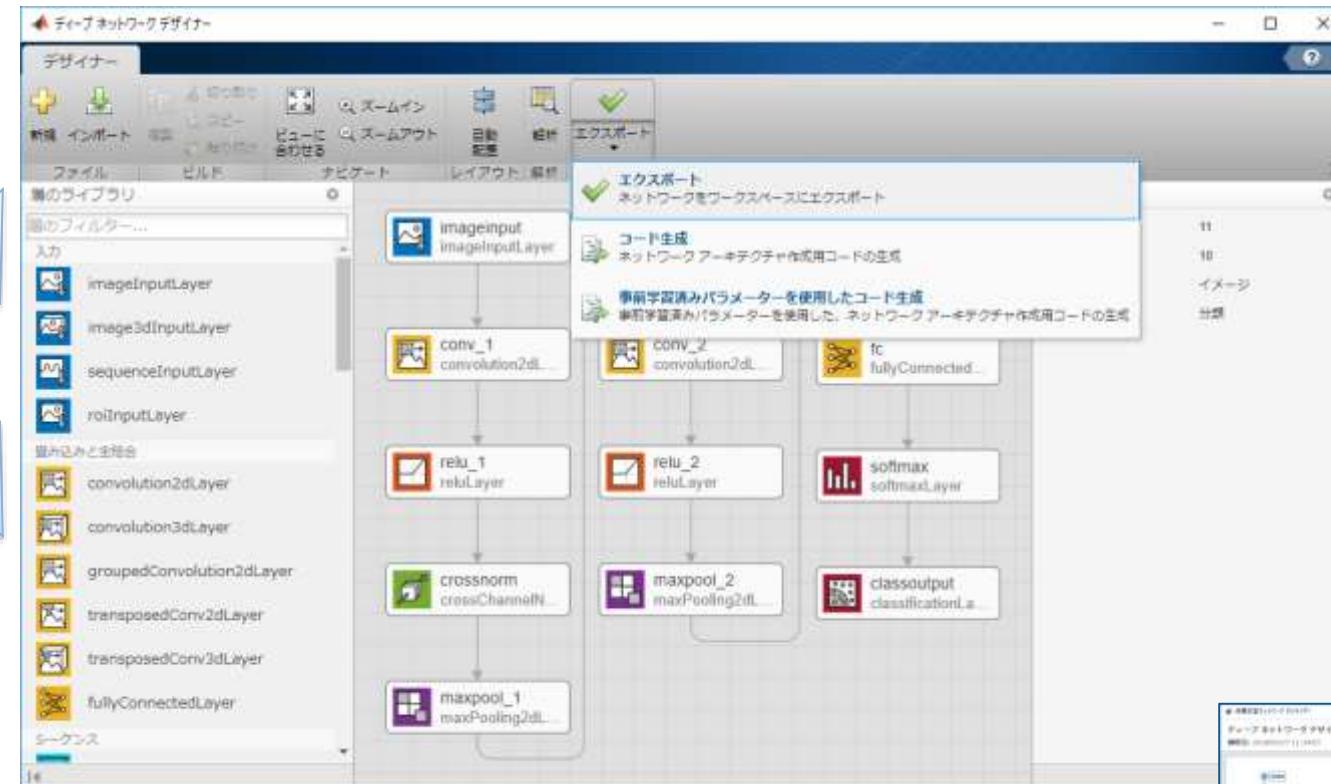
R2018b

Deep Learning Toolbox™

ゼロから設計

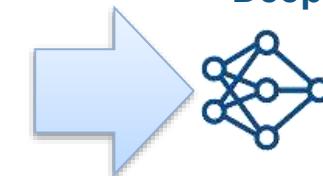


既存ネットワークの  
インポート

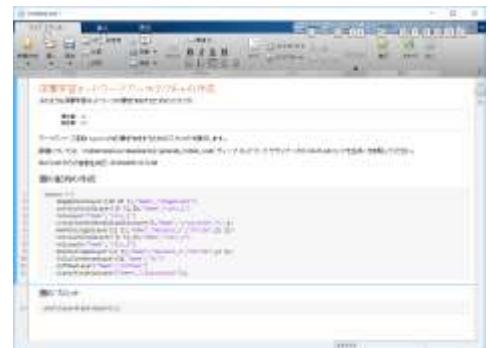


マウス操作でネットワークを設計  
Computer Vision関連のレイヤーに対応 R2019a

deepNetworkDesigner



ネットワークのエクスポート  
・レイヤークラス  
・MATLABコード R2019a

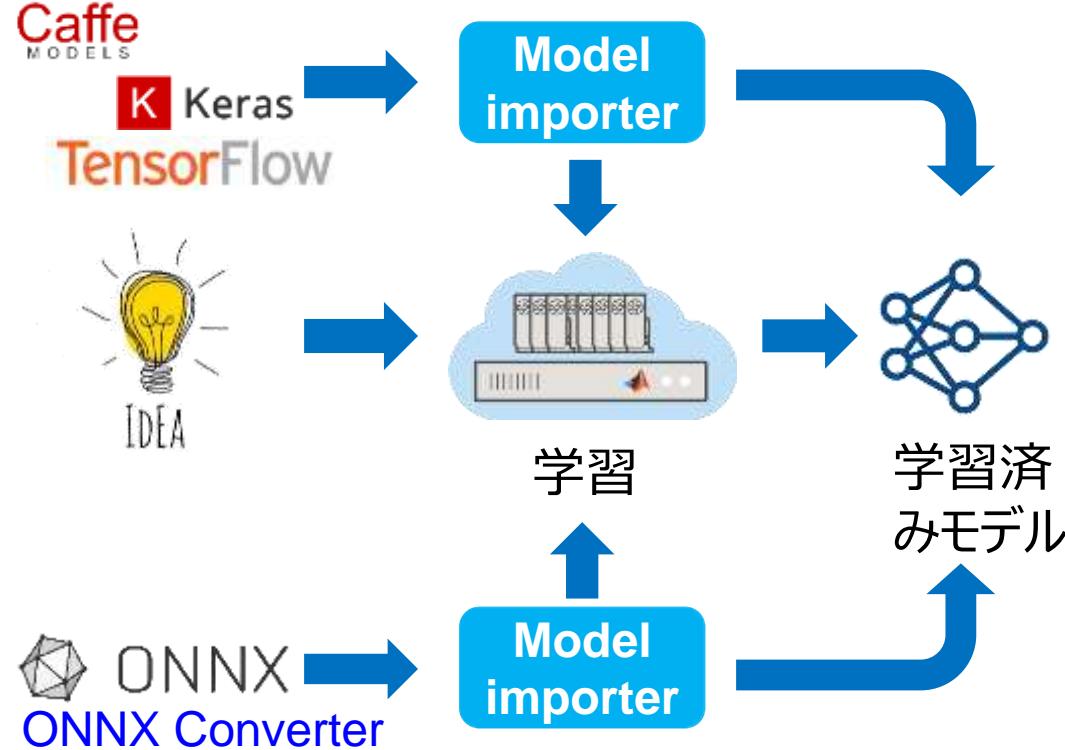


ネットワークの  
整合性を自動チェック

## 5.6.2.2 ディープラーニング：学習済みモデル

Deep Learning Toolbox™

提供されている学習済みニューラルネットワーク一覧 R2019b



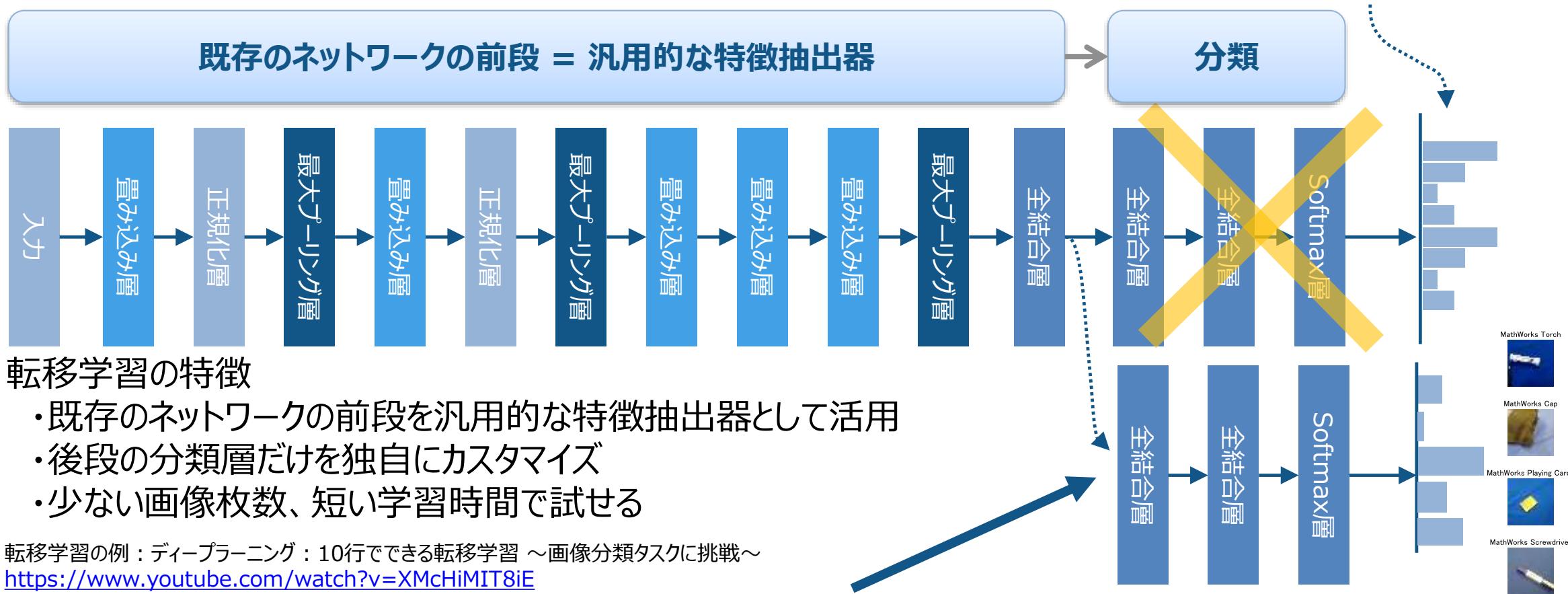
ネットワーク	深さ	容量	パラメーター数 (数弱万)	画像の入力サイズ
<a href="#">alexnet</a>	8	227 MB	61.0	227-by-227
<a href="#">vgg16</a>	16	515 MB	138	224-by-224
<a href="#">vgg19</a>	19	535 MB	144	224-by-224
<a href="#">squeezenet</a>	18	4.6 MB	1.24	227-by-227
<a href="#">googlenet</a>	22	27 MB	7.0	224-by-224
<a href="#">inceptionv3</a>	48	89 MB	23.9	299-by-299
<a href="#">densenet201</a>	201	77 MB	20.0	224-by-224
<a href="#">mobilenetv2</a>	53	13 MB	3.5	224-by-224
<a href="#">resnet18</a>	18	44 MB	11.7	224-by-224
<a href="#">resnet50</a>	50	96 MB	25.6	224-by-224
<a href="#">resnet101</a>	101	167 MB	44.6	224-by-224
<a href="#">xception</a>	71	85 MB	22.9	299-by-299
<a href="#">inceptionresnetv2</a>	164	209 MB	55.9	299-by-299
<a href="#">shufflenet</a>	50	6.3 MB	1.4	224-by-224
<a href="#">nasnetmobile</a>	*	20 MB	5.3	224-by-224
<a href="#">nasnetlarge</a>	*	360 MB	88.9	331-by-331

```

net = alexnet % 学習済みモデルの読み込み
I = imread('peppers.png'); % 画像読み込み
sz = net.Layers(1).InputSize;
I = I(1:sz(1),1:sz(2),1:sz(3)); % ネットワークの入力画像サイズに切り出し
label = classify(net, I) % 分類
  
```

## 5.6.2.2 ディープラーニング：学習済みモデルによる転移学習

ImageNet  
1000個のカテゴリ



```
alex = alexnet; layers = alex.Layers; % ベースのネットワークの読み込み
layers(23) = fullyConnectedLayer(10); % 独自のラベル数に変更(この例は10種類)
layers(25) = classificationLayer; % 分類レイヤーを新たに作成し付け替え
myNet = trainNetwork(imds, layers, opts); % 学習
```

## 5.6.2.3 ディープラーニング：領域ベース(R-CNN)による検出

R2016b

CNNにコンピュータビジョンの手法を組み合わせた物体(位置)検出・識別の手法

R-CNNに自動車の前面と停止標識を学習させた場合の例

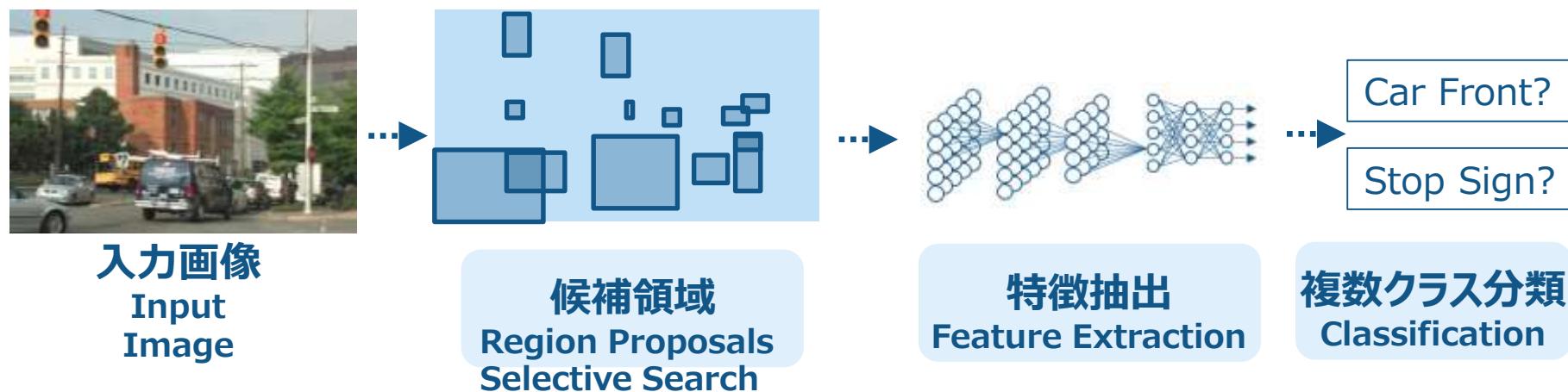


停止標識 (Stop Sign)

自動車の前面 (Car Front)

R2018a

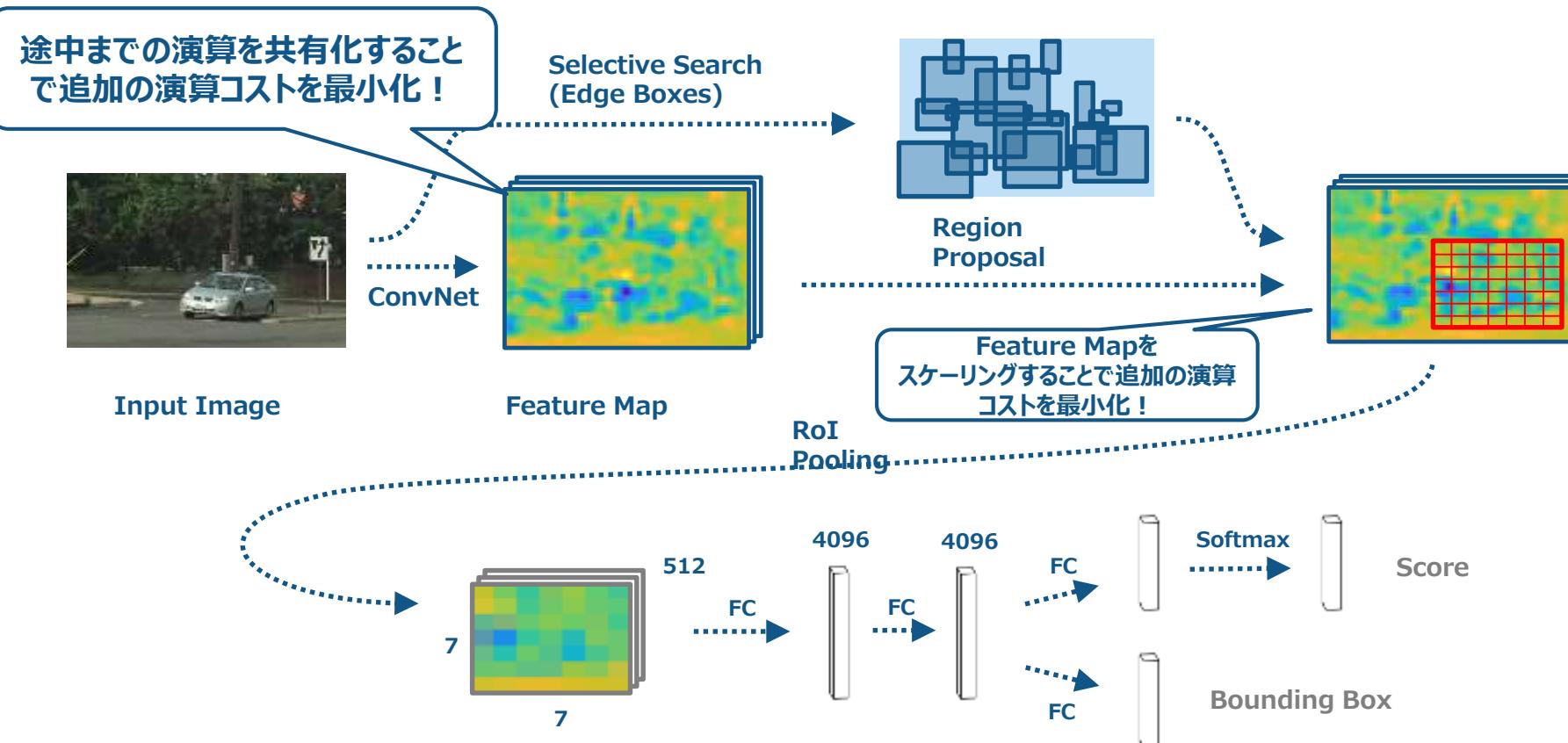
`selectStrongestBboxMulticlass`  
nonmaximal suppressionアルゴリズムを  
用いた最適な境界ボックスの選択  
(マルチクラス対応)



```
detector = trainRCNNObjectDetector(trainingData, network, options)
```

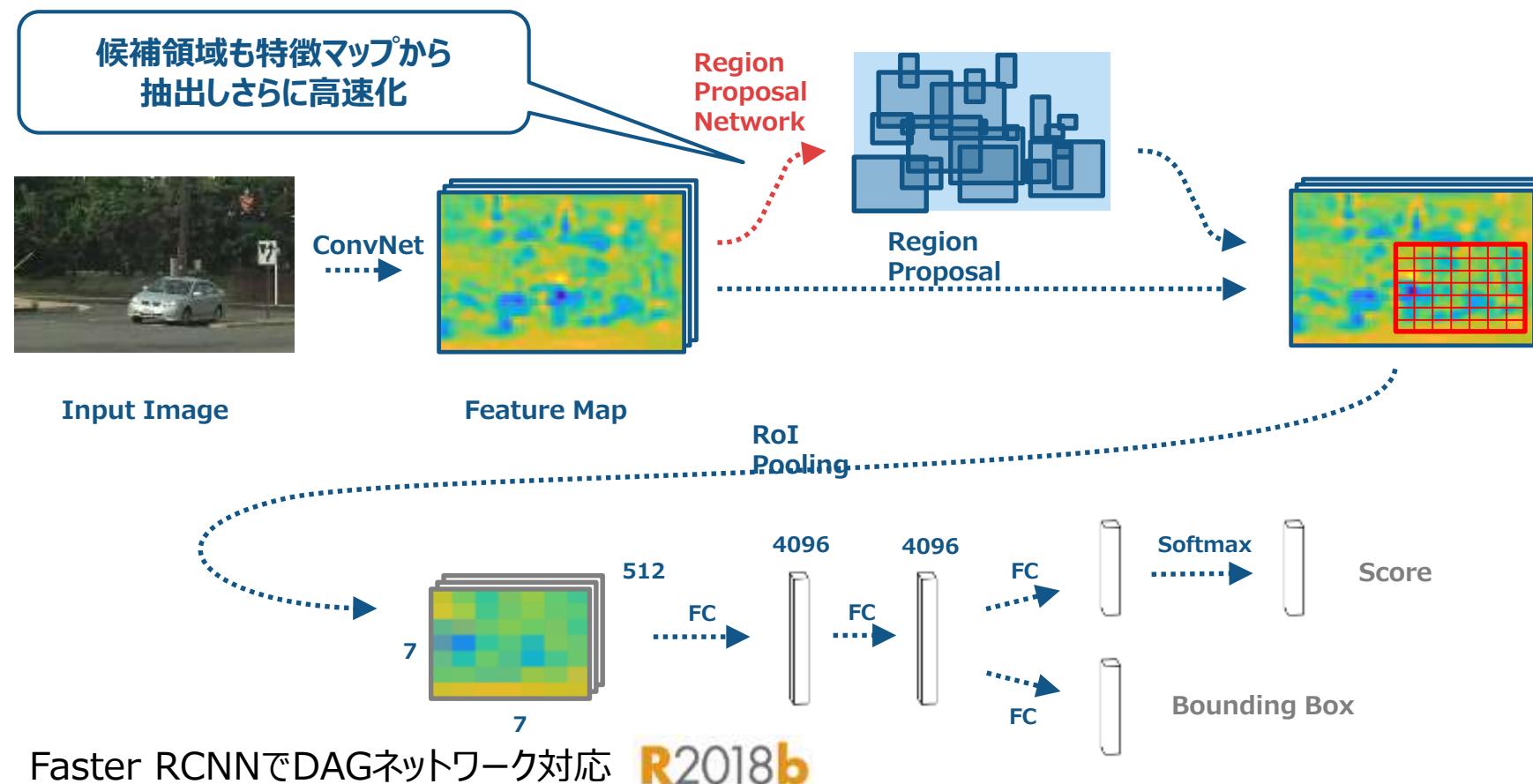
Computer Vision Toolbox, Deep Learning Toolbox, Statistics and Machine Learning Toolbox, Parallel Computing Toolbox  
compute capability 3.0以上の、CUDA GPUが必要。

## 5.6.2.4 ディープラーニング：Fast R-CNNによる検出



```
frcnn = trainFastRCNNObjectDetector(data, layers, options, ...
    'NegativeOverlapRange', [0 0.1], ...
    'PositiveOverlapRange', [0.7 1], ...
    'SmallestImageDimension', 400);
```

## 5.6.2.5 ディープラーニング：Faster R-CNNによる検出



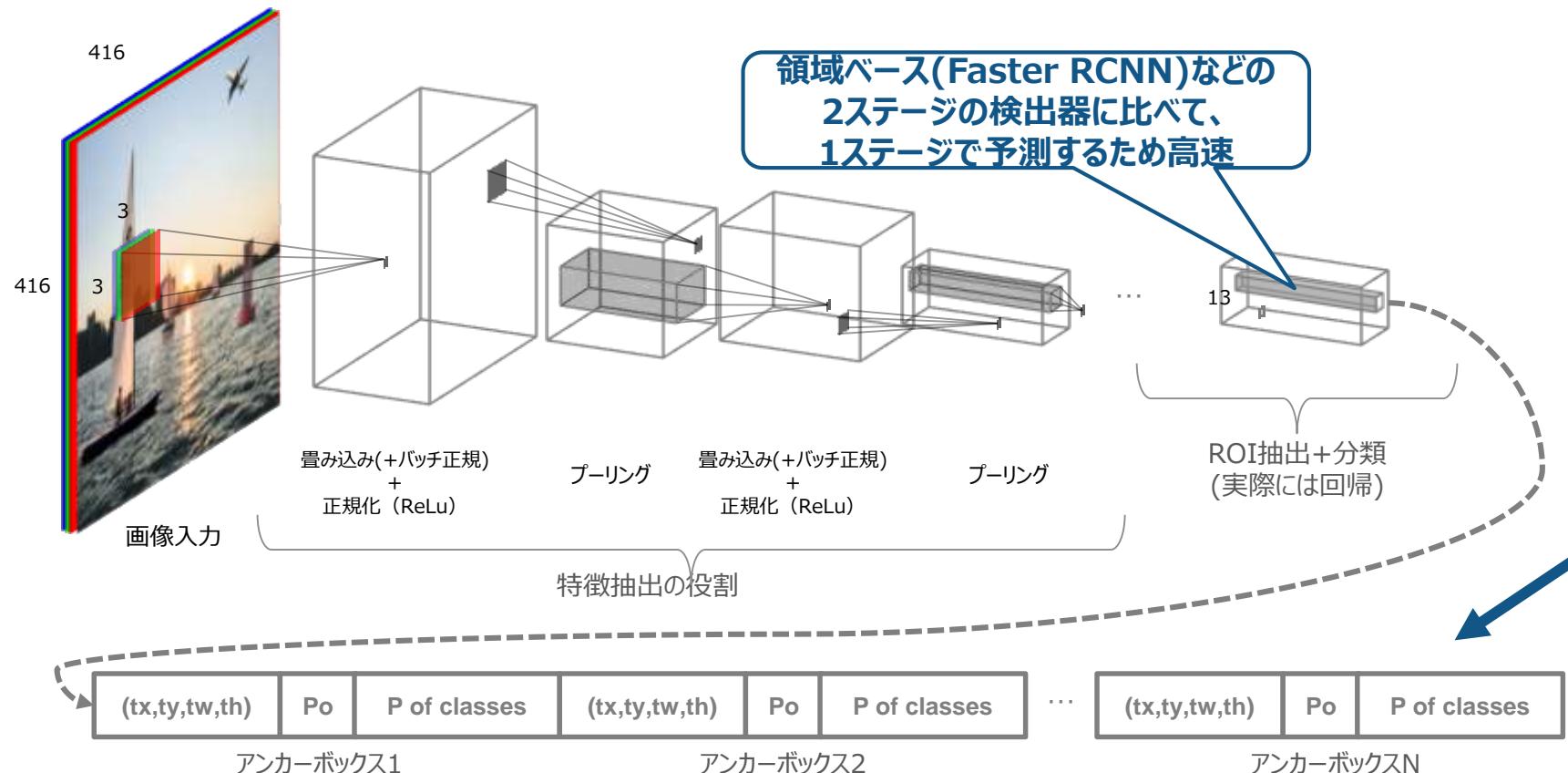
Faster RCNNでDAGネットワーク対応 R2018b

```
detector = trainFasterRCNNObjectDetector(data, layers, options, ...
    'MinibatchSize', [90 180; 128 128; 180 90], ...
    'BoxPyramidScale', 1.2,
    'NumBoxPyramidLevel', 3);
```

## 5.6.2.6 ディープラーニング：YOLO v2による物体検出

R2019a

Computer Vision Toolbox  
Deep Learning Toolbox  
Parallel Computing Toolbox  
compute capability 3.0以上の、  
CUDA GPUが必要。



R2019b

k-meansクラスタリングによる  
アンカーボックスサイズの推定  
**estimateAnchorBoxes**

```
[detector,info] = trainYOLOv2ObjectDetector(trainingData,lgraph,options);
```

GPU Coderを使って学習済み検出器からCUDAコード生成可能

```
net = loadDeepLearningNetwork('yolov2.mat')
[bboxes, scores, labels] = net.detect()
```

## 5.6.2.7 ディープラーニング：境界ボックスデータの拡張

R2019b

R-CNNやYOLOの学習時  
データ水増しに活用



境界ボックスのリサイズ

```
scale = 1.5;  
bboxB = bboxresize(bboxA,scale);
```

境界ボックスの幾何学的変換

```
tform = affine2d([-1 0 0; 0 1 0; 50 50 1]);  
rout = affineOutputView(size(I),tform);  
[bboxB,indices] = bboxwarp(bboxA,tform,rout)
```

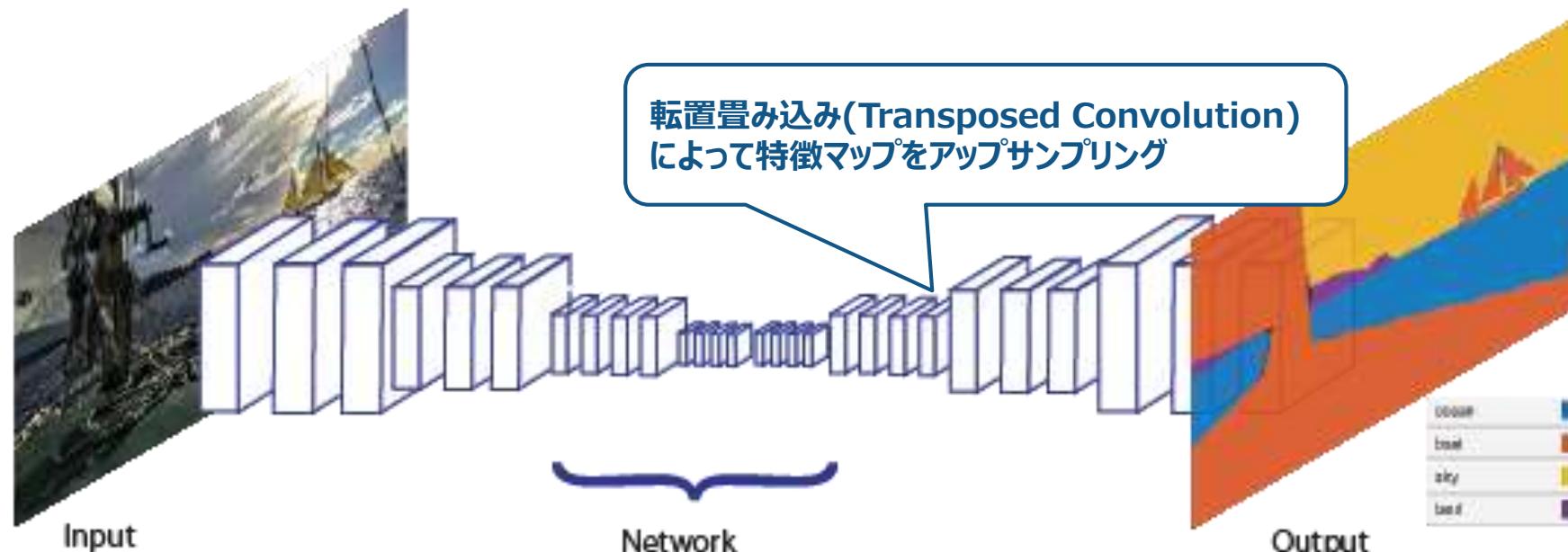
境界ボックスの切り抜き

```
targetSize = [256 256];  
win = centerCropWindow2d(size(I),targetSize);  
[bboxB,indices] = bboxcrop(bboxA,win);
```

## 5.6.3.1 ディープラーニング：FCNによるセグメンテーション

R2017b

FCN(Fully Convolutional Network; 全層畳み込みネットワーク)によるセマンティックセグメンテーション  
全結合層を転置畳み込み層に置き換え→位置情報が保存されたままアップサンプリング→画素ごとのクラス分類



R2018b

任意の位置の画像  
をペアで切り出す

RandomPatchExtractionDatastore  

- 3次元データに対応
- 並列実行に対応
- (Parallel Computing Toolboxが必要)

R2019b

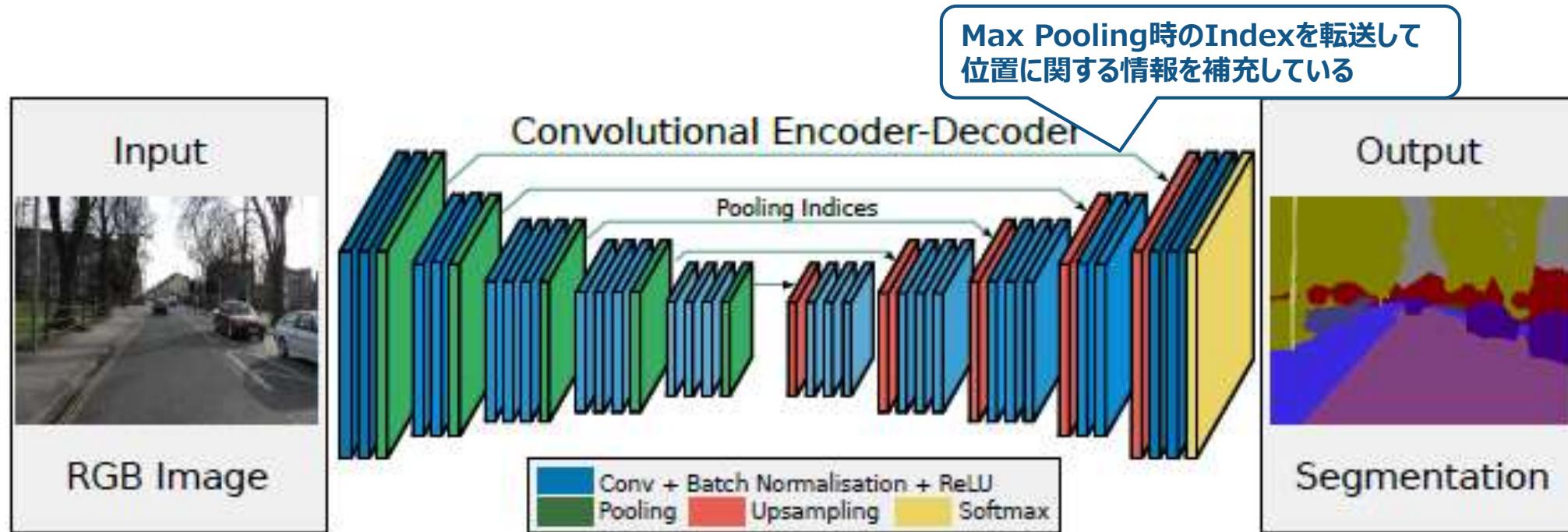
FCN-AlexNetの例：CamVidデータセットを用いたセグメンテーション  
<https://jp.mathworks.com/matlabcentral/fileexchange/65851-semantic-segmentation-using-fcn-alexnet>

```
imageSize = [480 640]; numClasses = 5;
lgraph = fcnLayers(imageSize, numClasses)
plot(lgraph)
```

Computer Vision Toolbox, Deep Learning Toolbox, Parallel Computing Toolbox, compute capability 3.0以上の、CUDA GPUが必要。

## 5.6.3.2 ディープラーニング：SegNetによるセグメンテーション

R2017b



SegNetの例：液塗抹標本画像の3クラスセグメンテーション

<https://jp.mathworks.com/matlabcentral/fileexchange/66448-medical-image-segmentation-using-segnet>

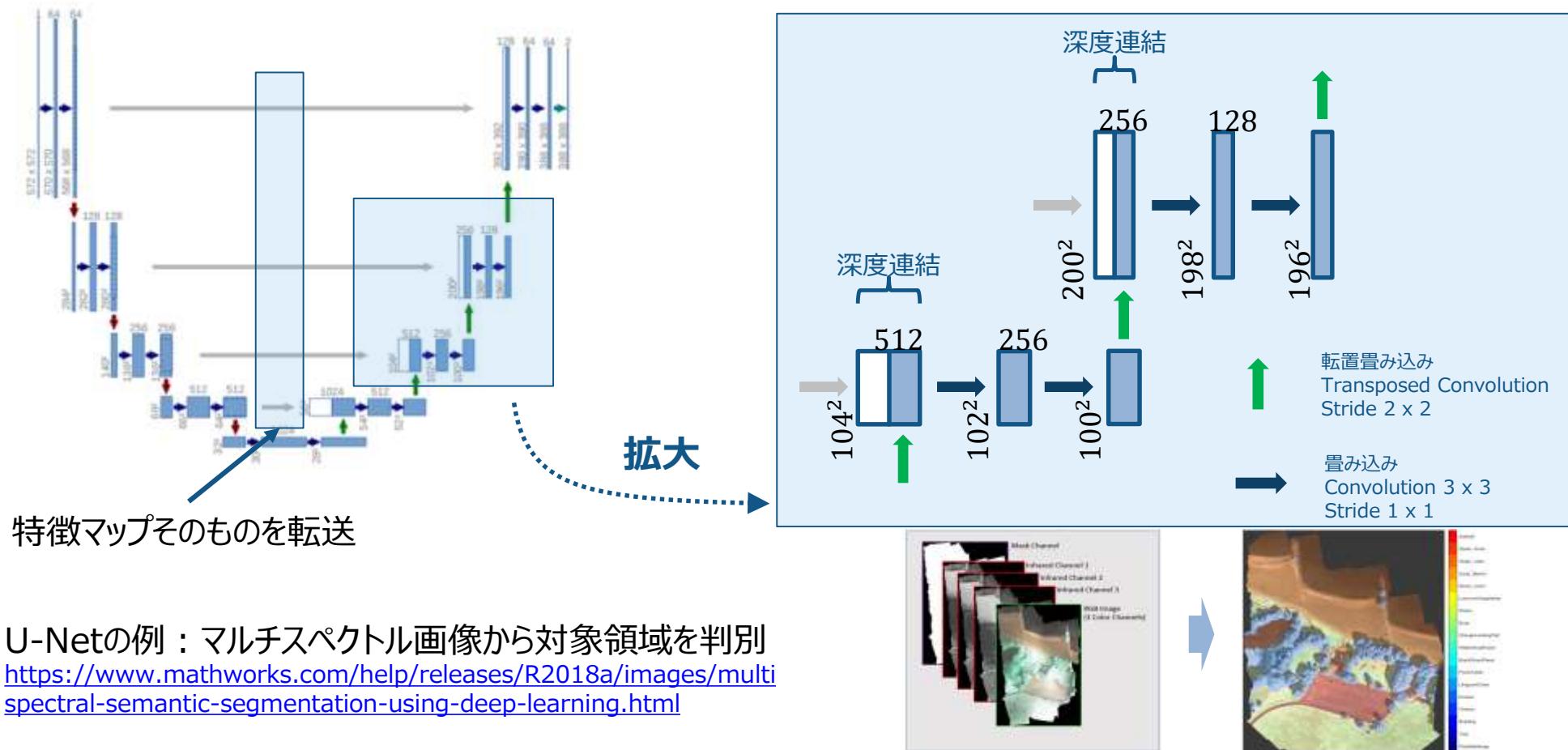
DeepLab v3+に例題アップデート R2019a

```
lgraph = helperDeeplabv3PlusResnet18(imageSize, numClasses);
```

```
imageSize = [32 32]; numClasses = 2;
lgraph = segnetLayers(imageSize,numClasses,2)
plot(lgraph)
```

Computer Vision Toolbox, Deep Learning Toolbox, Parallel Computing Toolbox, compute capability 3.0以上の、CUDA GPUが必要。

### 5.6.3.3 ディープラーニング：U-Netによるセグメンテーション

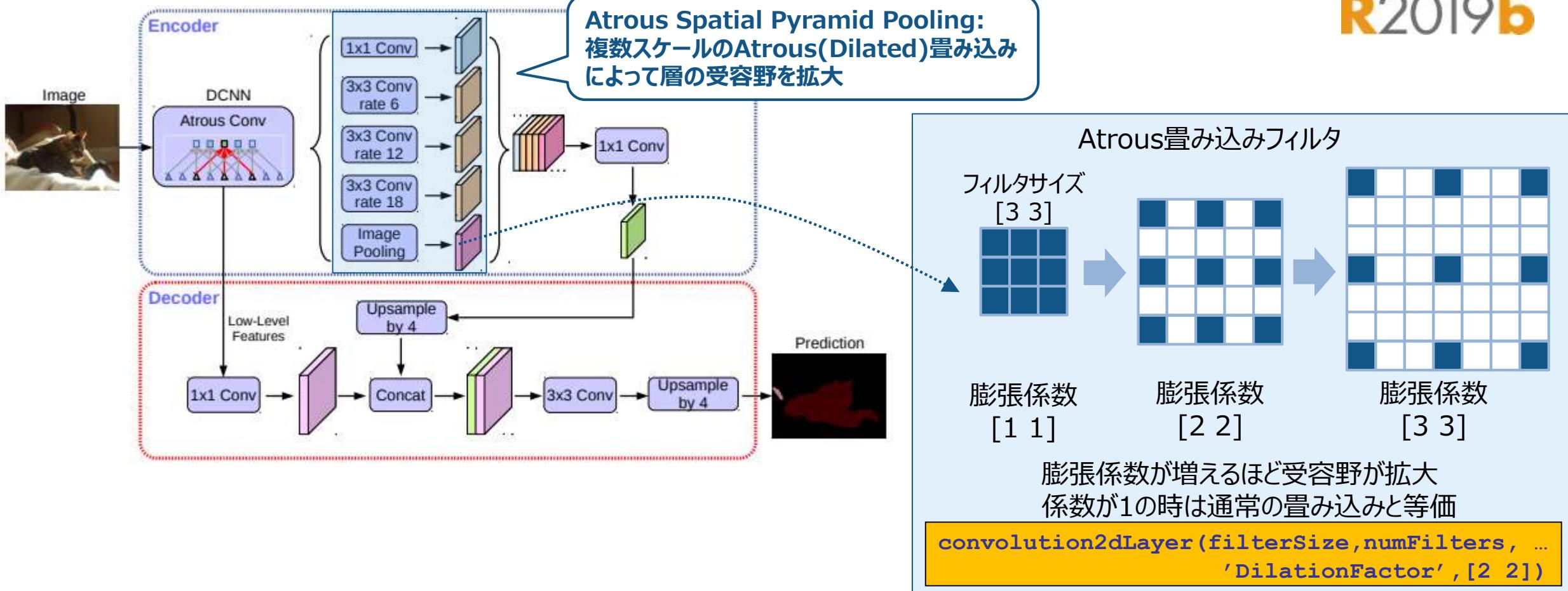


```
imageSize = [480 640 3]; numClasses = 5; encoderDepth = 3;
lgraph = unetLayers(imageSize, numClasses, 'EncoderDepth', encoderDepth)
plot(lgraph)
```

Computer Vision Toolbox, Deep Learning Toolbox, Parallel Computing Toolbox, compute capability 3.0以上の、CUDA GPUが必要。

## 5.6.3.4 ディープラーニング：DeepLab v3+によるセグメンテーション

R2019b

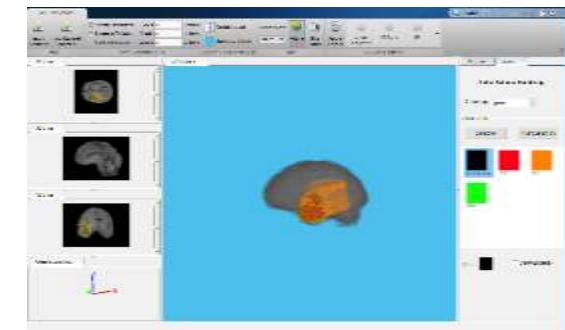


```
imageSize = [480 640 3]; numClasses = 5; network = 'resnet18';
lgraph = deeplabv3plusLayers(imageSize,numClasses,network);
plot(lgraph)
```

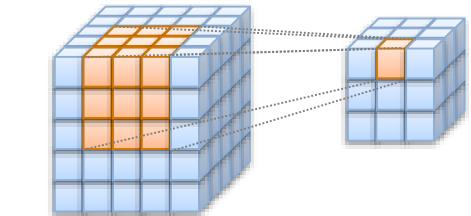
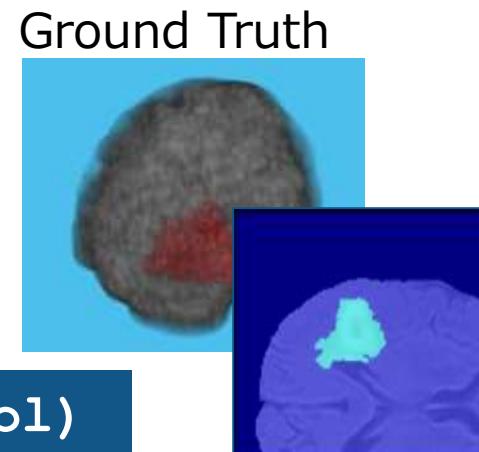
R2019a  
R2019b

## 5.6.4.1 ディープラーニング：3D U-Netセグメンテーション

- セマンティックセグメンテーション用関数の3次元サポート
   
`unet3dLayers`: 3次元U-Netネットワークの生成関数 **R2019b**  
`semanticseg`: セマンティックイメージセグメンテーションの実行  
`evaluateSemanticSegmentation`: セグメンテーション予測精度の評価  
`pixelClassificationLayer`: ピクセル分類層の作成  
`dicePixelClassificationLayer`: Dice損失を使用するピクセル分類層の作成 **R2019b**  
`pixelLabelDatastore`: ピクセルラベルデータ用のデータストア  
`labelvolshow`: 3Dのラベルの可視化



`volshow(vol)`



3次元フィルタの畳み込み  
(3x3x3のフィルタの例)

`semanticseg(vol, net)`



セグメンテーション結果

```
imageSize = [128 128 128 3]; numClasses = 5; encoderDepth = 2;
lgraph = unet3dLayers(imageSize, numClasses, ...
    'EncoderDepth', encoderDepth, 'NumFirstEncoderFilters', 16)
```

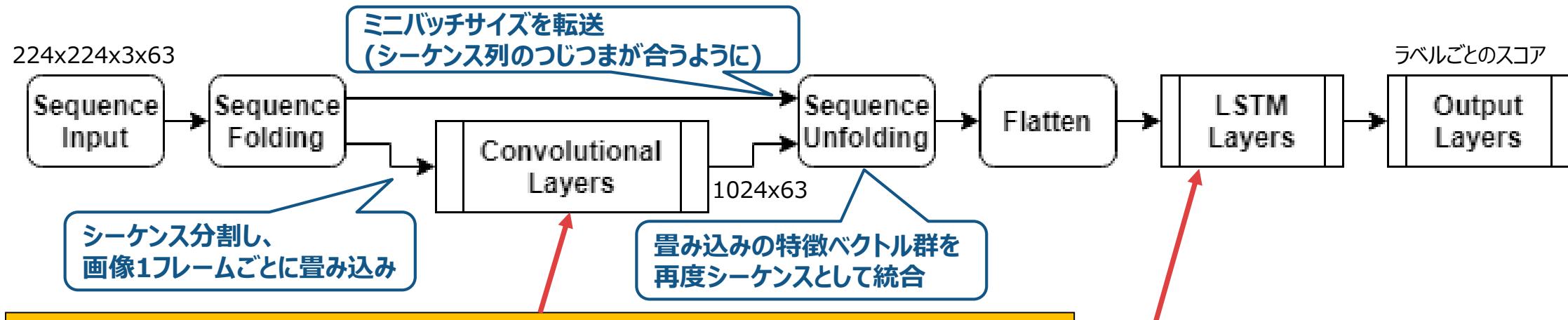
Computer Vision Toolbox, Deep Learning Toolbox, Parallel Computing Toolbox, compute capability 3.0以上の、CUDA GPUが必要。

他の例：肺の腫瘍のセグメンテーション: <https://www.mathworks.com/matlabcentral/fileexchange/71521-3-d-deep-learning-lung-tumor-segmentation>

## 5.6.5.1 ディープラーニング：動画の分類

R2019a

- 学習済みの畳み込みニューラルネットワークを使用してビデオを特徴ベクトルのシーケンスに変換
- ビデオラベルを予測するためのLSTMネットワークを学習
- 両方のネットワークのレイヤーを組み合わせてビデオを直接分類するネットワークを構築



```
netCNN = googlenet;
inputSize = netCNN.Layers(1).InputSize(1:2);
layerName = "pool5-7x7_s1";
activations(netCNN,video,layerName,'OutputAs','columns');
```

```
layers = [ sequenceInputLayer(numFeatures,'Name','sequence')
            bilSTMLayer(2000,'OutputMode','last','Name','bilSTM')
            dropoutLayer(0.5,'Name','drop')
            fullyConnectedLayer(numClasses,'Name','fc')
            softmaxLayer('Name','softmax')
            classificationLayer('Name','classification')];
```



## 5.7.7 検出漏れ/誤検出の補償(トラッキング)

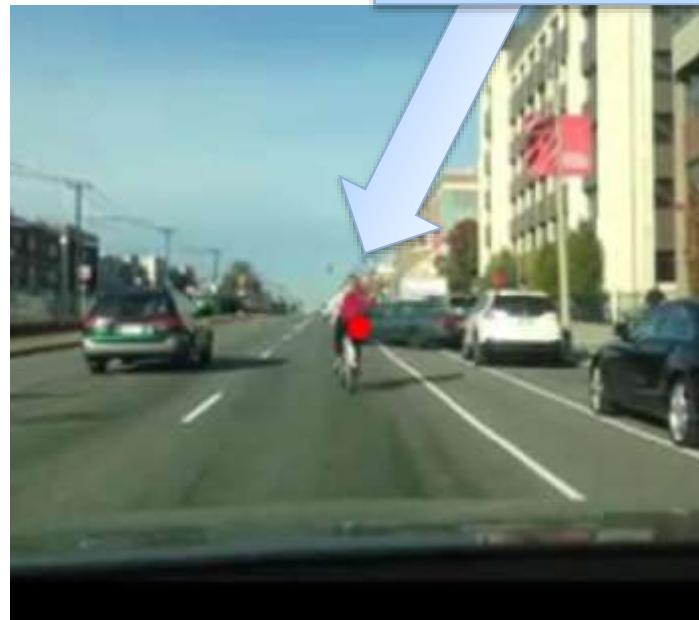
Computer Vision Toolbox™  
Automated Driving Toolbox™

人検出結果



補償

カルマンフィルタによる  
トラッキングで、位置を予測



`vision.KalmanFilter`  
`trackingKF`  
`trackingEKF`  
`trackingUKF`

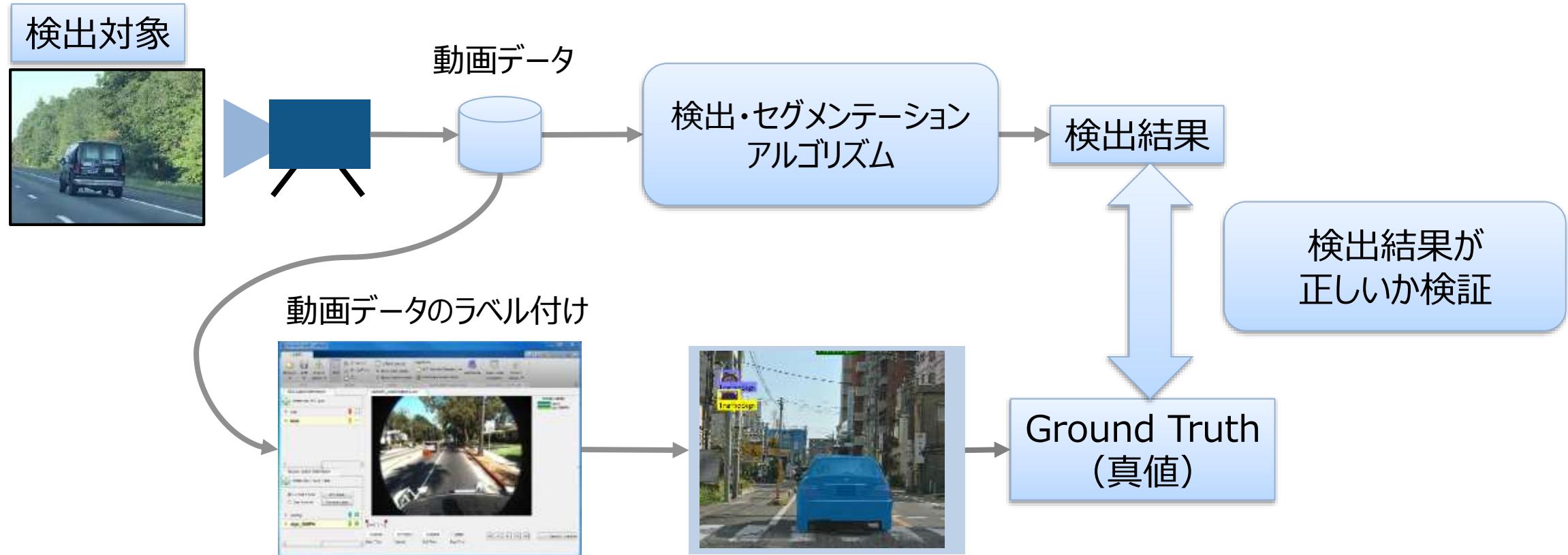
汎用線形カルマンフィルター(複数物体は非対応)  
線形カルマンフィルター **R2017a**  
拡張カルマンフィルター **R2017a**  
Unscentedカルマンフィルター **R2017a**

} 複数物体のトラッキングに対応  
(**multiObjectTracker**)

Automated Driving Toolbox™

## 5.7.8 検出器の性能評価

Computer Vision Toolbox™  
Automated Driving Toolbox™



`evaluateDetectionPrecision`

検出精度の算出

R2017a

`evaluateDetectionMissRate`

誤検出率の算出

R2017a

`evaluateSemanticSegmentation`

セマンティックセグメンテーションの評価指標 (混合行列 / 精度・IoUなどの各種メトリック)

R2017b

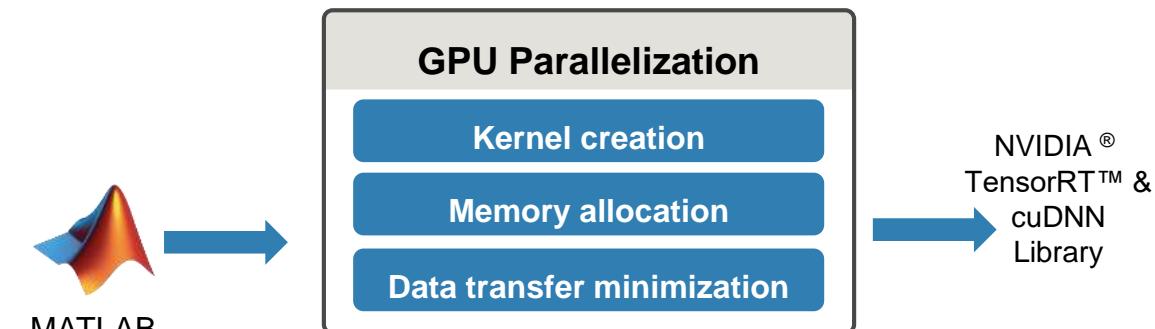
## 5.7.9 ディープラーニング実装

MATLABコードからCUDA®コードの生成

- GPU Coder専用GUIを使ったコード生成
- 初めてでも使いやすいGUI
- SeriesNetwork（カスタムレイヤ以外）の推定部分、もしくは並列処理を含むMコードを変換
- YOLOv2に対応
- プラグマによる関数解析とカーネル生成
- CUDAの文法を知らなくても利用できる
- MATLAB コードを、CPUで動かすCとGPUで動かすCUDAコードへ自動的に分割
- MEX生成してシミュレーション高速化
- コード生成後DLL化しSimulinkへ統合

GPU Coder™ R2017b

MATLAB Coder, Parallel Computing Toolboxが必要  
深層学習の実装には、Deep Learning Toolboxが必要  
compute capability 3.2以上のCUDA GPUが必要。  
Mac環境でのコード生成は未対応



# アジェンダ

1. MATLAB/Simulinkの概要
2. 各種画像処理例
3. 連携機能
4. コンピュータービジョン処理例
5. 画像の機械学習・ディープラーニング
6. まとめ

# まとめ

- MATLAB・Simulinkはアイディアを即座に試行できる統合開発環境
- 画像処理・コンピュータービジョンのアルゴリズムを対話的に検討・検証可能
  - 多くの高速/高度な関数・ブロックを用いた様々な方式検討
  - 様々な方式・パラメーターのトライ&エラーを迅速に実施可能
  - 統合された開発環境上で、並列処理/GPGPU高速化、C・HDL生成による実装を含む、多くのツール・外部言語との連携機能
  - テクニカルサポートや、多くのユーザ・研究者の方々による  
MATLAB Centralなどの共有リソースの有効活用

画像処理・コンピュータービジョンの開發生産性の向上



© 2019 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.