```
%============================ gradTempMatchQA ============================
%
%  dPos = gradTempMatchQA(template, image, ipos)
%
%
%  Computes the gradient of the template matching score for image matching
%  via gradient descent.  The current estimate of the template position
%  is required.  Uses interpolation to do sub-pixel matching on the image
%  data, but does not do so for the image gradient information.
%
%  Input:
%    template           - the grayscale template patch.
%    image                      - the image to find the template in.
%    ipos                       - the position of the template (centered).
%                       the position is in (x,y) coordinates.
%
%  Output:
%    dPos                       - the differential with respect to position.
%    pCost                      - the current matching score for ipos (optional).
%
%============================ gradTempMatchQA ============================

%
%  Name:        tmatchT
%
%  Author:      Patricio A. Vela, pvela@ece.gatech.edu
%
%  Created:             2012/02/18
%  Modified:    2012/02/18
%
%============================ gradTempMatchQA ============================
function [dPos, pCost] = gradTempMatchQA(template, image, ipos)

%==[1] Get the image and template dimensions.
ti = size(template,1);                          % Number of rows (size in y dir)
tj = size(template,2);                          % Number of cols (size in x dir)

[iM, iN] = size(image);                         % Image size rows x cols (y size, x size)


%==[2] Get image patch at the specified location.  Make the patch bigger
%               by one pixel all around in order to compute the gradients.
%
if ( rem(tj,2) == 0 )                           % Even size template in x direction.
  xinds = ipos(1) + [(-0.5-tj/2):(tj/2+0.5)];
else
  xinds = ipos(1) + [-(1+(tj-1)/2):(1+(tj-1)/2)];
end

if ( rem(ti,2) == 0 )                           % Even size template in y direction.
  yinds = ipos(2) + [(0.5-ti/2):(ti/2-0.5)];
else
  yinds = ipos(2) + [-(1+(ti-1)/2):(1+(ti-1)/2)];
end

%==[3] Extract the image data from the specified location, plus compute
%               the image gradients.
imdat = interp2(1:iN, (1:iM)', image, xinds, yinds');
[gradIx, gradIy] = gradient(imdat);
```

```
%==[4] Compute the gradient (compensating for the extra border).
tdiff = (template - imdat(2:end-1,2:end-1));

%-- Form the G matrix
%               a 2x2 matrix from the image gradients.
gradItempX = gradIx(2:end-1,2:end-1);
gradItempY = gradIy(2:end-1,2:end-1);
gradI = [gradItempX(:), gradItempY(:)];
G = gradI' * gradI;
%-- Form the E vector
%               a 2x1 vector from the image gradient and matching error.
E = (gradI' * tdiff(:));

%-- Solve for the position update.
%               computes the solution to the local quadratic approximation at ipos.
dPos = (G \ E);

%==[5] Compute the cost.  We have it almost computed, so this is an almost
%               free computation.
pCost = sum(tdiff(:).*tdiff(:));

end
```

```
Not enough input arguments.

Error in gradTempMatchQA (line 35)
ti = size(template,1);                          % Number of rows (size in y dir)
```

Published with MATLAB® R2016b