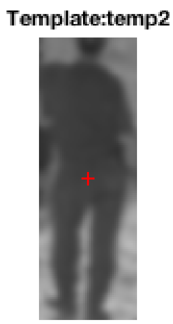
```
fprintf('Final Point:[%f,%f]\n',pos(1),pos(2));
```

findTemplate_I1_temp2.m

```
Image: I1
Template: temp2
Parameters:
(c=0.000024,nsteps=200,pos=[640,840])
Final Point:[586.983502,672.585061]
```

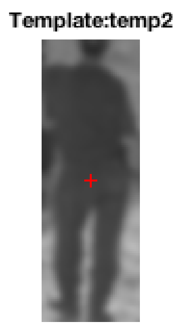


Published with MATLAB® R2016b

```
fprintf('Final Point:[%f,%f]\n',pos(1),pos(2));
```

loseTemplate_I1_temp2.m

```
Image: I1
Template: temp2
Parameters:
(c=0.000024,nsteps=200,pos=[520,600])
Final Point:[519.446550,543.022444]
```

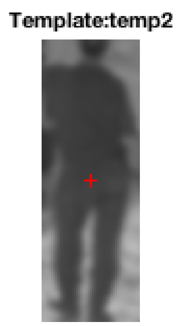


Published with MATLAB® R2016b

```
fprintf('Final Point:[%f,%f]\n',pos(1),pos(2));
```

findTemplate_I2_temp2.m

```
Image: I2
Template: temp2
Parameters:
(c=0.000024,nsteps=200,pos=[780,690])
Final Point:[707.141200,628.116144]
```

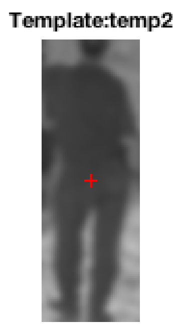


Published with MATLAB® R2016b

```
fprintf('Final Point:[%f,%f]\n',pos(1),pos(2));
```

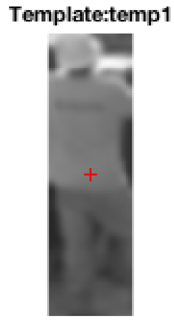
loseTemplate_I2_temp2.m

```
Image: I2
Template: temp2
Parameters:
(c=0.000024,nsteps=200,pos=[580,790])
Final Point:[564.441330,701.957442]
```



```
Image: I3
Template: temp1
Parameters:
(c=0.000025,nsteps=200,pos=[520,790])
Final Point:[574.472192,715.056294]
```

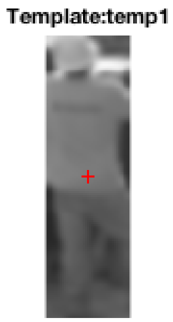
findTemplate_I3_temp1.m



Published with MATLAB® R2016b

```
Image: I3
Template: temp1
Parameters:
(c=0.000025,nsteps=200,pos=[620,790])
Final Point:[676.103055,809.589928]
```

loseTemplate_I3_temp1.m



Published with MATLAB® R2016b

```matlab
%============================ 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)
```

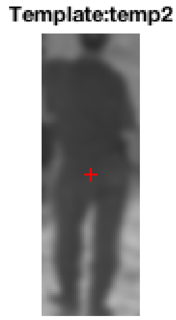
Published with MATLAB® R2016b

The convergence for the quadratic approach happens much quicker than the linear approach, since there is no need to use the c factor to take small steps. In other words, the approach allows the descent to jump to the local minimum of the quadratic approximation in one step.

The basin of attraction was similar, since similar blurring standard deviations were used for both approaches.

```
Image: I1
Template: temp2
Parameters:
(nsteps=50,pos=[640,860])
Final Point:[592.967259,673.053170]
```

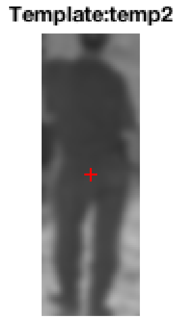
findTemplateQA_I1_temp2.m



Published with MATLAB® R2016b

```
Image: I1
Template: temp2
Parameters:
(nsteps=50,pos=[540,770])
Final Point:[507.319135,709.341531]
```

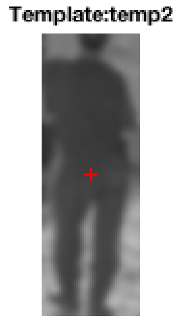
loseTemplateQA_I1_temp2.m



Published with MATLAB® R2016b

```
Image: I2
Template: temp2
Parameters:
(nsteps=50,pos=[640,860])
Final Point:[685.652961,670.868350]
```

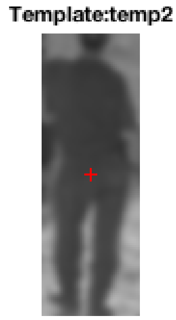
findTemplateQA_I2_temp2.m



Published with MATLAB® R2016b

```
Image: I2
Template: temp2
Parameters:
(nsteps=50,pos=[570,730])
Final Point:[564.495365,701.928916]
```

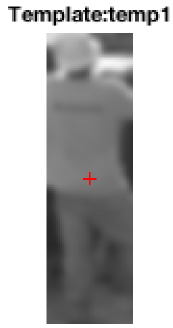
loseTemplateQA_I2_temp2.m



Published with MATLAB® R2016b

```
Image: I3
Template: temp1
Parameters:
(nsteps=50,pos=[480,820])
Final Point:[574.484935,715.115290]
```

findTemplateQA_I3_temp1.m



Published with MATLAB® R2016b

```
Image: I3
Template: temp1
Parameters:
(nsteps=50,pos=[620,790])
Final Point:[676.103055,809.589928]
```

loseTemplateQA_I2_temp2.m