METR4202 Tutorial 6 Solutions

1) -

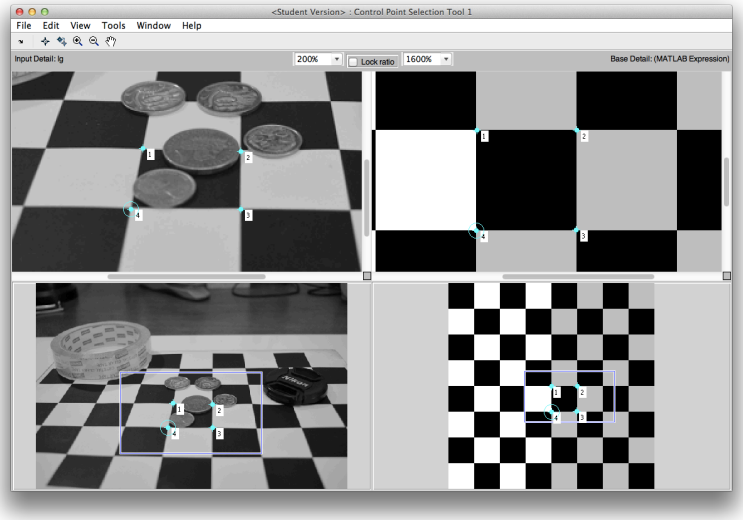
2) 'chess_circles.jpg' was taken with a Nikon D5100 Camera. The focal length was 26mm, the exposure time 1/60 of a second and F-Number 4.2.

3) The following code can be used for part 3. Smoothing before performing canny edge detection improves the performance of the subsequent Hough transform.

```
% Read in image
I = imread('chess_circles.jpg');

% Scale down 50%
I = imresize(I, 0.5);

% Convert to grayscale
Ig = rgb2gray(I);

% Get a gaussian kernel for blurring
K = fspecial('gaussian');

% Blur the image
% Note that multiple passes with a fixed kernel
% are the same as a single pass with a larger kernel
Igf = imfilter(Ig, K);
Igf = imfilter(Igf, K);
Igf = imfilter(Igf, K);

% Detect edges
E = edge(Ig, 'canny');

% Perform Hough Line transform
[H, T, R] = hough(E);

% Get top N line candidates from hough accumulator
N = 10;
P = houghpeaks(H, N);

% Get hough line parameters
lines = houghlines(Igf, T, R, P);

% Show lines overlaid on original image, and hough space
figure;

% Show original image
subplot(2, 1, 1);
imshow(I);
title('chess\_circles.jpg');
hold on;

% Overlay detected lines - this code copied from 'doc houghlines'
for k = 1:length(lines)
   xy = [lines(k).point1; lines(k).point2];
   plot(xy(:,1), xy(:,2), 'LineWidth', 2, 'Color', 'blue');

   % Plot beginnings and ends of lines
   plot(xy(1,1), xy(1,2), 'x', 'LineWidth', 2, 'Color', 'yellow');
   plot(xy(2,1), xy(2,2), 'x', 'LineWidth', 2, 'Color', 'red');
end

% Show Hough Space
subplot(2, 1, 2);
imshow(imadjust(mat2gray(H)), 'XData', T, 'YData', R,
```

```
'InitialMagnification', 'fit');
title('Hough Line Transform');
xlabel('\theta');
ylabel('\rho');
axis on;
axis normal;
grid on;
hold on;

% Display as colormap
colormap('jet');
```

4) The following code can be used for part 4. The four control points used are shown below.




```
% Select four control points as shown in the figure,
% then select File > Export Points to Workspace
%cpselect(Ig, checkerboard);

% Use the selected points to create a recover the projective transform
tform = cp2tform(input_points, base_points, 'projective');

% Transform the grayscale image
Igft = imtransform(Ig, tform, 'XYScale', 0.2);

% Crop the tranformed image to only include relevant parts
% Note that in matlab image matrices, the first dimension is the y
% (vertical) direction, and the second is the x, starting from the top
% left hand corner
Ic = Igft(round(size(Igft, 1)*0.7):end, :);
Ic = Ic(:, round(size(Ic, 2)*0.1):round(size(Ic, 2)*0.5));

% Use imtool to manually measure the size of the coins in the resultant
% image
%imtool(Ic);

min_radius = 15;
max_radius = 20;

% Detect and show circles
houghcircles(Ic, min_radius, max_radius);
```

5) The following code can be used for part 5. HSV is a useful colour model for computer vision as the hue of an object changes relatively little under differing lighting conditions. RGB on the other hand roughly matches the way the human

eye perceives colour, and is intuitive to think about in terms of colour mixing, however is not as useful for image processing as all three channels are sensitive to lighting changes.

```
% Save compressed jpg at very low quality (8 = 8%)
imwrite(I, 'chess_circles_compressed.jpg', 'Quality', 8);

% Read back in
I_compressed = imread('chess_circles_compressed.jpg');

% Convert to HSV
[h, s, v] = rgb2hsv(I_compressed);

% Show individual channels for comparison
figure;
subplot(2, 2, 1);
imshow(I_compressed);
title('Compressed Image');

subplot(2, 2, 2);
imshow(imsc(v));
title('Value (Brightness) Image');


subplot(2, 2, 3);
imshow(imsc(h));
title('Hue (Color) Channel');

subplot(2, 2, 4);
imshow(imsc(s));
title('Saturation (Color Intensity) Image');

% Show edge comparison between original and compressed images
figure;

subplot(1, 2, 1);
imshow(edge(rgb2gray(I), 'canny'));
title('Original Image Edges');

subplot(1, 2, 2);
imshow(edge(rgb2gray(I_compressed), 'canny'));
title('Compressed Image Edges');
```