Department of Electrical & Electronic Engineering
Imperial College of Science Technology & Medicine

3rd year Digital Systems Design Course Work

Introduction

This coursework is designed to give you practical digital design skills to complement the lectures. You will be designing a module (which may contain various sub-modules) to perform transformations or distortions on real-time video taken from a camera.

In order to reduce the coursework to a manageable size, you will be given a framework which performs the function of video capture and display. A separate document will provide a detailed description of that framework and how you may use it.

This specification document explains the idea behind the distortion algorithms and describes what are expected for different level of attainment (and marks).

You will be working in pairs (or singly if you prefer), and you are responsible to find your own partner.

Deadline

Demonstration (in Level 5 Lab): Friday, 27th March 2009 (a timetable will be set up)
Report: Friday, 3rd April 2009

Please submit electronically

- A no-nonsense report documenting your design (don't write long reports with words that do not add to its content), including clear diagrams showing the overall design and other diagrams/codes. (Use PDF)
- Some evidence that your design works.
- Justification on any design choices made.
- Signed copy of contribution of each member of the pair (if you work in a pair).
- A zipped design directory uploaded including the design and report.

Equipment

You will be using the Terasic DE2-70 boards (these are upgraded version of the DE2 board) in Level 5 3rd/4th year lab. The design environment will be Altera's Quartus II.

Basic Requirement of the project

In order to help you learn about the underlining problem, I have provided you with a MATLAB function that performs simple transformations. The goal is to perform this type of operations using the DE2-70 hardware in real-time.

You need to implement the following distortions as specified by SW0:

SW18=0: Rotation – rotate the video by θ degrees, where θ is specified by SW0-SW5 in increments of 5 degrees.
SW18=1 Spinning - As 1 above, but this time rotate the video by θ degrees every 50 ms.

If you use the table lookup method to generate the sine/cosine values, you will get a B-C grade. If you use a CORDIC module from elsewhere (such as free IP block desgined by others) to generate the sine/cosine values, you will get a B+ grade. If you design your own CORDIC module, you will get an A grade.

The following document provides guidelines about video distortion including rotation, edge detection and blurring. The algorithms are provided as MATLAB code for you to get familiar with. We only use a static image (Clown) for this purpose.

Getting Started

Loading the test image

To make things nice and easy for you we've given you an image and a Matlab function to display the image. The image shown in Figure 1 is a 200x320 grey scale image called "clown" for you to work with.

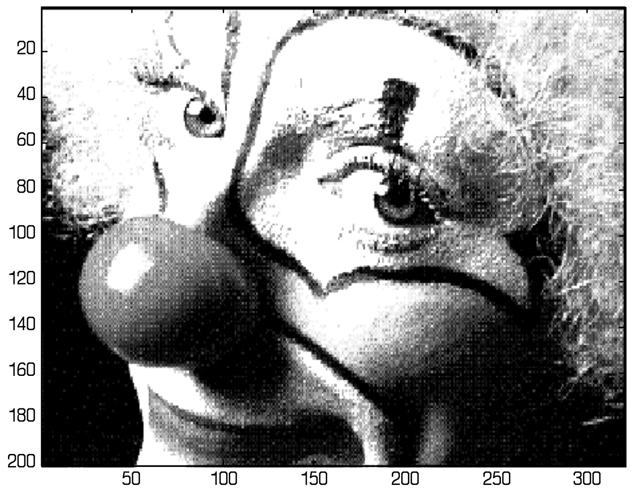


Figure 1 - Grey Scale 200x320 Image "clown"

To load the image into your workspace type (you can find the image file on the course webpage):

```
» load clown
```

The Image Format

The image is stored as a 2 dimensional array of grey scale values in the range 0 to 1. To return the grey scale value of the image at co-ordinate (**x,y**) type:

```
» clown (y,x)
```

So typing **clown (20,319)** Matlab responds with:

```
ans =

   0.1554
```

Which is the grey scale value of the image at (319,20).

Displaying Images

The function **show(*Image Name*)** has been given so you can display the images. To display the clown image you previously loaded type:

```
» show (clown)
```

Example - Image Rotation

The main task in this project is to perform rotation. As example is shown below where the original image is rotated by about 9 degrees, as shown in Figure 2.

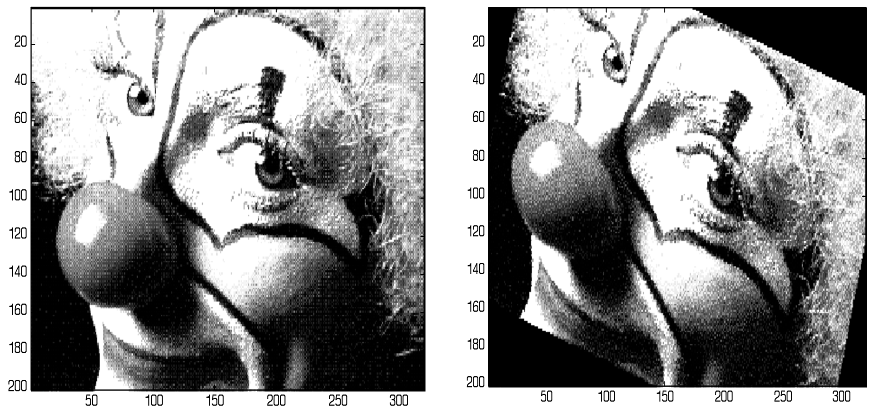
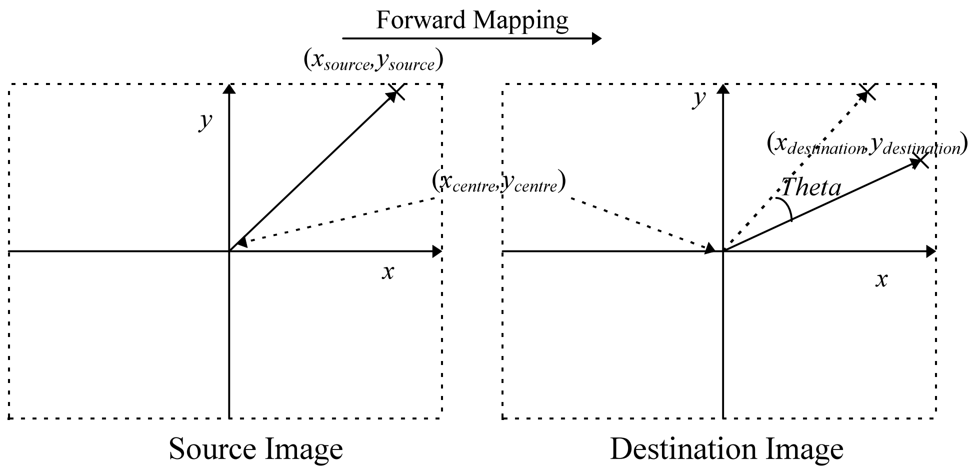


Figure 2 - Original "clown" Image & the image rotated by 0.3 radians

- The function has the following format in Matlab (angle specified as radians here, but you should use degrees in the hardware design):
  function [Out] = rotate(In, Theta)
- The resulting image has the same size as the original. (i.e. the matrix storing the image have the same dimensions, so some clipping of the image may occur)
- If a source pixel lies outside the image it is painted black.
- Use "nearest pixel" only: for example if the source pixel required is (34.43,46.667) the pixel at the location (34,47) in the source image is.
- The rotation is performed about the centre of the image.

Everything You Need To Know About Rotating Images

Just in case your maths is a bit rusty, here's the basics of image rotation:



Forward Mapping
(x_{source}, y_{source})
y
x
(x_{centre}, y_{centre})
y
$(x_{destination}, y_{destination})$
Theta
x
Source Image
Destination Image

Equation 1 - Equation for rotation of theta about image centre

$$\begin{pmatrix} x_{destination} \\ y_{destination} \end{pmatrix} = \begin{pmatrix} \cos(theta) & \sin(theta) \\ -\sin(theta) & \cos(theta) \end{pmatrix} \left(\begin{pmatrix} x_{source} \\ y_{source} \end{pmatrix} - \begin{pmatrix} x_{centre} \\ y_{centre} \end{pmatrix} \right) + \begin{pmatrix} x_{centre} \\ y_{centre} \end{pmatrix}$$

The way to use the forward mapping would be as follows:

```
For each pixel in the source image
{
    Work out the destination pixel location using the forward mapping
        equation.
    Paint that destination pixel with the source image value.
}
```

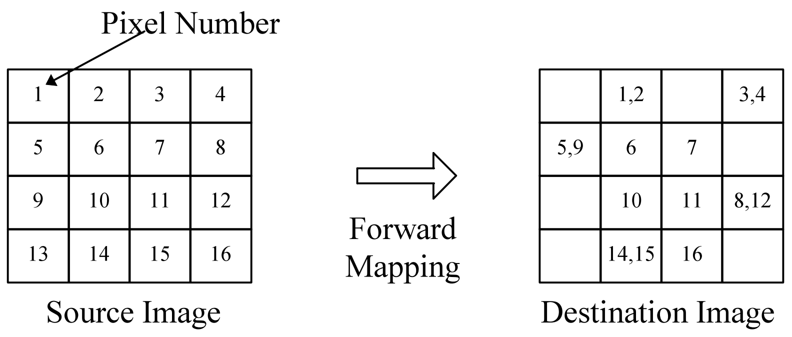


Figure 3 - Using Forward Mapping

But!…

The problem with using the forward mapping directly is demonstrated by Figure 3; Firstly there are pixels in the destination image with more than one source pixel. More of a problem is the fact that some pixels are never written to, leaving the destination image with holes!

The way around this is to use the *reverse mapping* equation in Equation 2. This works out where each destination pixel *came from* in the source image. This uses the inverse of the transformation matrix, which fortunately is easy to work out using the Matlab **inv()** function.

Equation 2 - Reversing mapping of equation 1

$$\begin{pmatrix} x_{source} \\ y_{source} \end{pmatrix} = \begin{pmatrix} \cos(theta) & \sin(theta) \\ -\sin(theta) & \cos(theta) \end{pmatrix}^{-1} \left(\begin{pmatrix} x_{destination} \\ y_{destination} \end{pmatrix} - \begin{pmatrix} x_{centre} \\ y_{centre} \end{pmatrix} \right) + \begin{pmatrix} x_{centre} \\ y_{centre} \end{pmatrix}$$

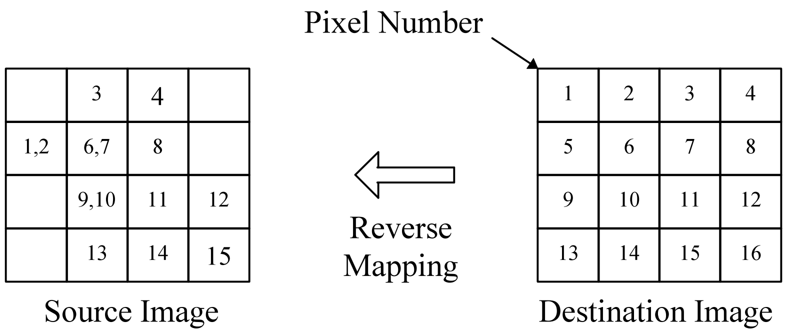


Figure 4 - Using Reverse Mapping

So the way to use the reverse mapping would be as follows:

```
Calculate the inverse transformation matrix
For each pixel in the destination image
{
        Work out where the pixel maps to in the source image, using the
            reverse mapping equation
        Paint the destination pixel with that source pixel value.
}
```

The MATLAB code that perform this rotate function is given in Appendix A.

Ripple Effect

If you are interested in getting A++ grade on this course work, read on.

The effect to be bonus marks (and a real challenge) is to add ripple effect to the moving image. The basic algorithm is as follows:

1) Translate destination pixel location according to centre of ripple. E.g. for ripple in centre:
   [x',y'] = [x-width/2,y-height/2]

2) Covert coordinates into polar [x',y'] -> [r, θ] (you can use CORDIC for this).

3) Modify r -> r' according to something like r' = r + A*sin(r*2*π/L + Offset)
   A = 'Amplitude'

L = 'Wavelength'
Offset = 'Angle offset' can be 0

(You can also use CORDIC for this step to give you the sin().)

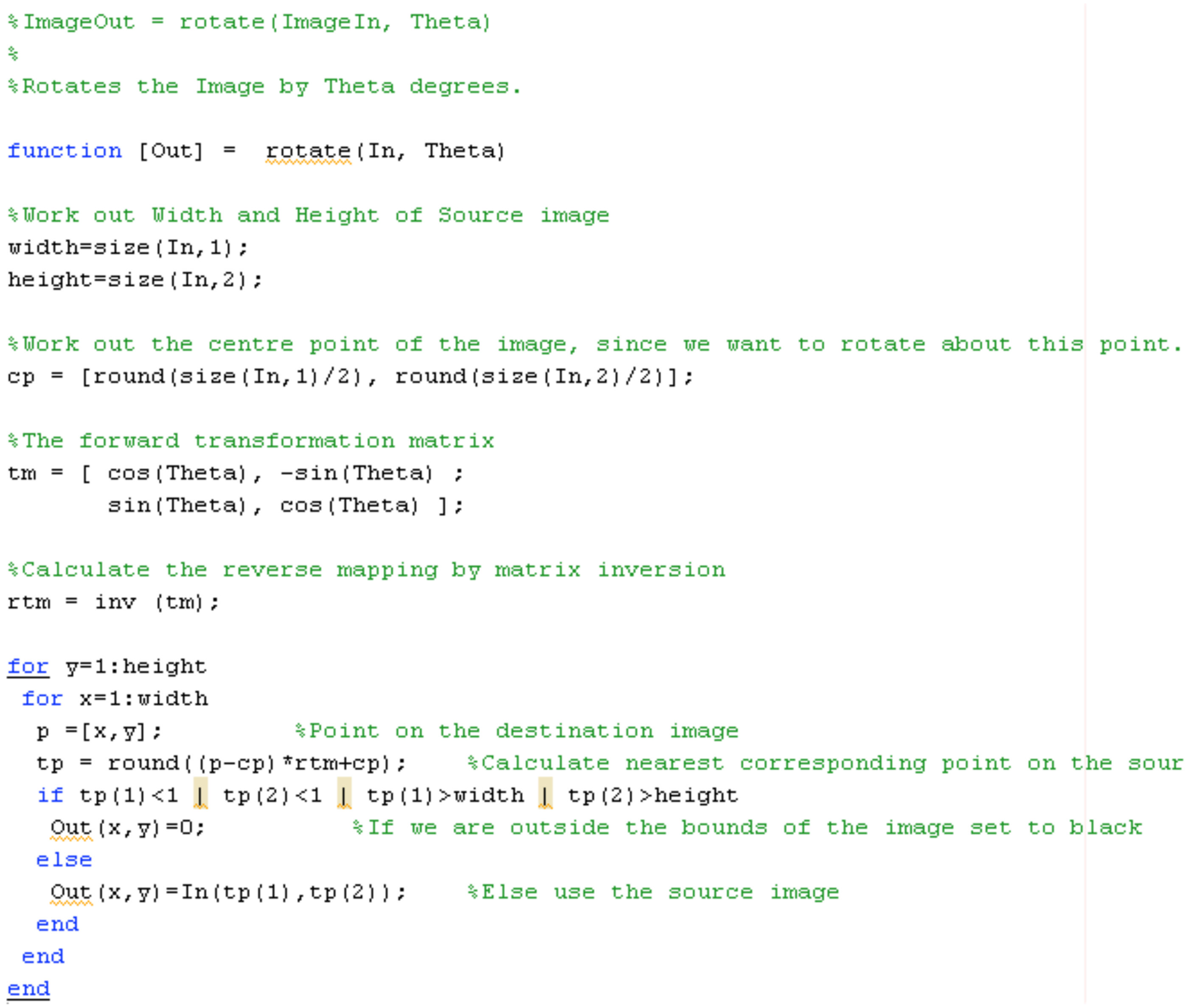
4) Convert back to Cartesian coordinates [r', θ] -> [x",y"] - CORDIC again!!!

5) Add back in original translation [x'",y'"] = [x"+Width/2,y"+Height/2]

6) Set destination pixel with source pixel Pdest[x,y] = Psource[x'",y'"]

The CORDIC can be made to give you the polar conversions directly, and can be used to calculate the additional sin() as well. Don't forget always to use inverse mapping (i.e. start from output frame and work out where in the input frame you should get the data).

APPENDIX A – Matlab Code to perform Rotation



```
%ImageOut = rotate(ImageIn, Theta)
%
%Rotates the Image by Theta degrees.

function [Out] =  rotate(In, Theta)

%Work out Width and Height of Source image
width=size(In,1);
height=size(In,2);

%Work out the centre point of the image, since we want to rotate about this point.
cp = [round(size(In,1)/2), round(size(In,2)/2)];

%The forward transformation matrix
tm = [ cos(Theta), -sin(Theta) ;
       sin(Theta), cos(Theta) ];

%Calculate the reverse mapping by matrix inversion
rtm = inv (tm);

for y=1:height
 for x=1:width
  p =[x,y];          %Point on the destination image
  tp = round((p-cp)*rtm+cp);    %Calculate nearest corresponding point on the sour
  if tp(1)<1 | tp(2)<1 | tp(1)>width | tp(2)>height
   Out(x,y)=0;          %If we are outside the bounds of the image set to black
  else
   Out(x,y)=In(tp(1),tp(2));    %Else use the source image
  end
 end
end
```