Detecting pointing & calculating direction

1) Finding the skin-like pixels
2) Finding the blocks of skin
3) Getting the fingertip
4) Getting the pointing direction
5) Improvements

Colour Based

Skin is roughly the same colour, so we can find skin by looking for skin coloured pixels.

This is very fast to calculate.

However, errors occur due to lighting and shading.

Chromatic Colour Based

We need adjust for the lighting conditions.

We can do this by simply dividing by the total value at the pixel.

$$r = r / (r+b+g)$$

(or r+b+g+1 to avoid divide by 0 errors)

Skin Probability

Chromatic Colour Based

Original image

Chromatic Colour Based

Chromatic colour image

Chromatic Colour Based

High contrast chromatic colour image

Colour -> Skin Probability Value

Now we have a chromatic colour image, but we need skin probabilities.

We take an average skin colour and use it as the mean of a normal distribution.

Skin Probability

Colour -> Skin Probability Value

Shamelessly stolen from Face Detection by Henry Chang and Ulises Robles.

Skin Probability

Colour -> Skin Probability Value

Skin Probability

Colour -> Skin Probability Value

Finding the hand

But where's the hand?

We know which pixels are (supposedly) skin.

We need to find connected groups of pixels.

I wrote a 1-2 pass algorithm to do this.

The Algorithm (1)

Look at each pixel, and check its 8 connected neighbours.

If they are skin, then look to see which group they belong to.

Assign the lowest group number to the selected pixel.

If there are no neighbours, the pixel gets a new group number, groupNum. This is then incremented

The Algorithm (1)

X=min(A,B,C,D)

The Algorithm (2)

This isn't quite right though.

If C is lower than D, then D should be set to C.

So, we keep a replacement table.

For every group number, there is a replacement number.

e.g. replace[10]=4 means group 10 has been seen to be connected to group 4, so they should be joined.

The Algorithm (2)

replace[A]=min(replace[A],X)

replace[B]=min(replace[B],X)

replace[C]=min(replace[C],X)

replace[D]=min(replace[D],X)

The Algorithm (3), the last bit :)

Again, there is a slight problem.

We may have:

replace[10]=4

replace[4]=2

Therefore, 10 should be replaced by 2 in the image.

We loop over the replace array, from

n=length(replace) to 0:

replace[i] = min(replace[i] , replace[replace[i]])

Finding the hand

Now a picture

Edges

Now we find all the edges.

These are pixels in the largest group that are connected to pixels that aren't in the group.

Finding the finger

Now a picture

Finding the finger

Edges

- Now we find the edges on the outside of the hand
- This is done by first selecting the extreme left pixel.
- Then we look at its neighbours that are edges.
- We move to one of them, and mark it 'tried'.
- This repeated until either the start (meaning we've finished) or dead end is found.
- If a dead end is found, backtrack.

Getting to the point

We now have an outline.

We want to find the finger, though.

Since we are looking for a pointing finger, look for the pointiest bit of the outline

Finding the finger

curvature[i]=dotProduct(edge[i]-edge[i+k],edge[i]-edge[i-k])

Getting to the point

This should give us the fingertip.

Now we trace back from this, along the sides of the finger.

The average of the vector describing the sides of the finger gives us the direction.

Finding the direction

Getting to the point

Getting the direction

Now we have a vector in the 2D image.

Given calibrated stereo cameras, running the detector on both images will give enough information to calculate the 3D vector describing the direction of the finger.

Future work

Improvements

Adaptive threshold

Logical analysis

Motion sensitive

Future work