How to define the colour ranges for an automatic detection of coloured objects

The colour detection algorithms scan every frame for pixels of a particular quality. To recognize a pixel as part of a valid object, its Y, U and V components must fall within the ranges defined in the *Thresholds* section of the *colour definition file*. The latter is a regular text file with at least two sections, *Colors* and *Thresholds*. The *Colors* section has an entry for each object to be detected. It defines an RGB colour triplet, a merge parameter (0 … 1), a colour identifier (0 … 31) and a colour label (text). Every entry in *Colors* must have a corresponding entry in *Thresholds*. The latter defines ranges for a pixel's Y-component (brightness), its U-component (first colour attribute) as well as its V-component (second colour attribute). A typical colour definition file has the following structure:

```
[Colors]
(255, 255,   0)    0.95  0  yellow
(255,   0,   0)    0.95  1  red
(0  ,   0, 255)    0.95  2  blue

[Thresholds]
(228:234, 12:28, 130:134)       % too narrow yellow
(88:98, 110:118, 185:197)       % working red
(115:136, 191:203, 59:70)       % too narrow blue

[not used...]
(218:234, 8:32, 127:135)        % working yellow
(88:98, 108:120, 185:197)       % working red
(96:148, 175:204, 59:79)        % working blue
```

The section following the *[not used…]* tag is ignored. In fact, the algorithms only read as many lines from the *Thresholds* section as there are in the preceding *Colors* section. This makes it easy to experiment with differing colour schemes. To change from one scheme to another, simply exchange the contents of the two sections *Thresholds* and *not used*…. Comments such as the ones found in the *Thresholds* section are ignored during the parsing of a colour definition file.

A suitable choice of these ranges can easily be made using the camera training utilities *trainCamera* (working of live images from the camera or a pre-recorded AVI movie) and *trainStill* (working of a pre-recorded image file, image format: YUYV). A chosen colour scheme can then be tested using the MATLAB m-file script *test.m*. The following graphs present screen shots which have been made using these utilities.

trainStill

This utility function allows a user to determine the YUV ranges of an object which is chosen from a pre-recorded (still) image. The image is expected to be stored in a regular MAT-file, containing a variable 'yuyv' (YUYV format) or 'rgb' (RGB format). The default filename is 'testimages/testimageYUYVRGB.mat'; it can be modified in the source code of *trainStill* (see line 10 of the script file 'trainStill.m').

Once started, *trainStill* displays the loaded image file as well as three smaller graphs for the YUV statistics of the current object. As nothing has been selected yet, these graphs appear empty. An object can be defined for colour detection using the mouse. Left-hand click on the image, keeping the mouse button pressed. The appearing rubber band box can be moved around until the mouse button is released. Choose an area of the object which includes its representative colour – and as few other colour pixels as possible. The larger the variety of colours found inside the chosen area, the less accurate the colour detection algorithms will work. It should be pointed out, however, that too narrow training could result in the failure of detecting an object at all, especially under varying lighting conditions.

Following the selection of an object, *trainStill* displays histograms for the Y, U and V components of all associated pixels. Thin red lines are displayed to indicate the chosen ranges. The choice is made automatically, based on 'the essential weight' of the histogram of each component (YUV). At present, *trainStill* uses up to two standard deviations, centred at the mean value of the histogram. To make this selection more acute, change the variable *numberSTD* to value smaller than 2. Figure 1 shows the tool *trainStill* with the included test image. The yellow object has been selected; this relatively bright colour is characterised by a high Y value (brightness), a diffuse distribution of U values and a reasonably narrow spread of V values. Qualifying pixels have been shaded black and white.

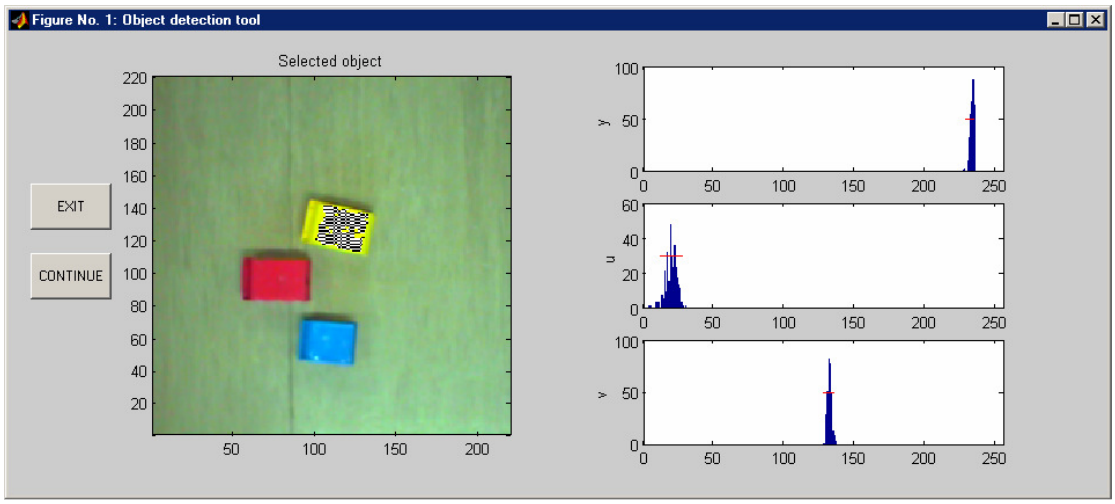

Figure 1: *trainStill* in action

The YUV thresholds are displayed in the MATLAB command window. In this case, *trainStill* suggests the following ranges:

```
-----------------------------------------------------------------
(Ymin:Ymax, Umin:Umax, Vmin:Vmax)
(228:234, 12:28, 127:135)
-----------------------------------------------------------------
```

To validate the chosen range, the m-file script 'test' can be run. This m-file calls upon the colour detection algorithms, which have been made accessible to the MATLAB

environment in form of a dynamically loaded library (DLL): *imgProc.dll*. The DLL expects the colours ranges to be found in the colour definition file *testcolors.txt*. Copy and past the above line to the first entry of the *Thresholds* section in *testcolors.txt* and run 'test.m'. Your colour definition file should resemble the following lines:

```
[Colors]
(255, 255,   0)    0.95  0  yellow

[Thresholds]
(228:234, 12:28, 127:135)
```

Run *test* by entering at the command line:

```
test('testimages/testimageYUYVrgb.mat',[1 2 3])
```

This tells *test* to load the image file *testimages/testimageYUYVrgb.mat* and to scan for the first colour of the colour definition file. Figure 2 illustrates the output of *test* using the above colour scheme. The yellow object has not been detected. A possible reason could be that the object selection was based on too few pixels.

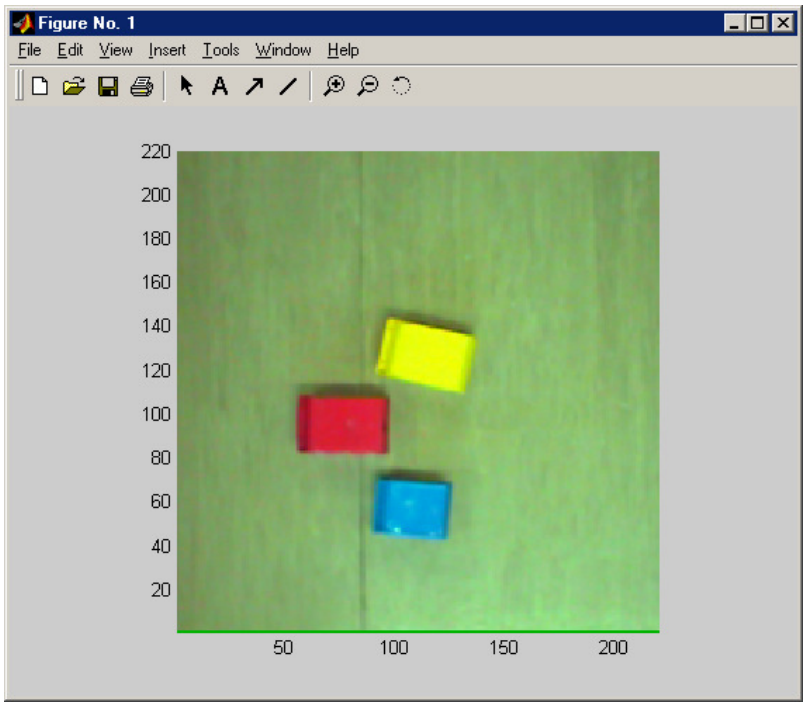

Figure 2: Validation of the chosen range: failure

Repeat the training, this time selecting a larger variety of shades of yellow (e. g. the bright central area as well as the less luminous fringes on the left and/or the right of the object). This could produce modified thresholds such as the following:

```
(217:234, 7:35, 127:135)            % modified selection
```

Notice that both U and V range have been expanded while the brightness (Y) remains unaltered. With the modified colour definition file (*testcolors.txt*), the yellow object is successfully detected (Figure 3).

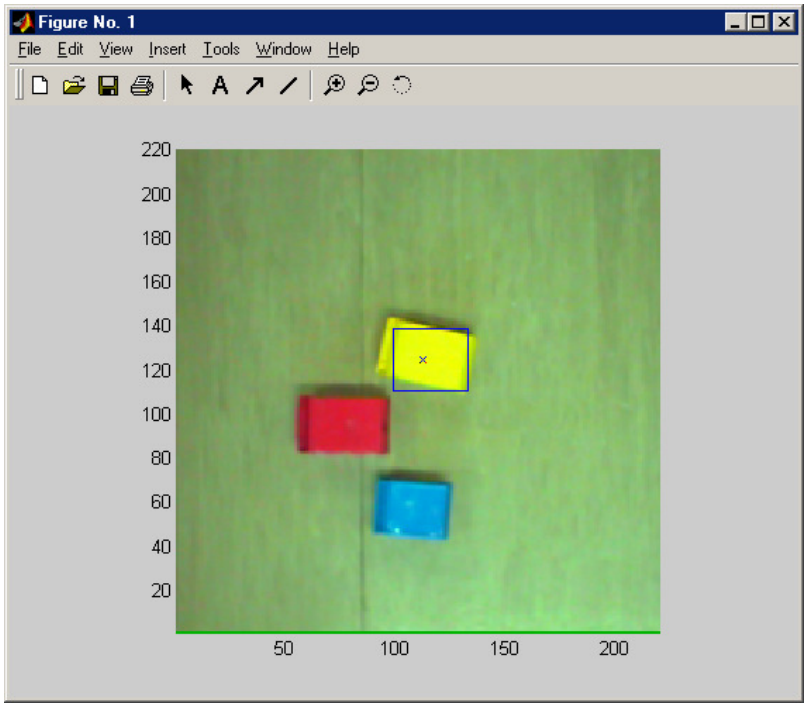

Figure 3: Validation of chosen range: success

Try and find appropriate colour ranges for all three objects (yellow, red, blue). To validate colour schemes with more than one colour definition enter the following command:

```
test('testimages/testimageYUYVrgb.mat', [1 2 3])
```

This instructs the processing DLL (*imgProc.dll*) to scan for the colour definitions with colour IDs 0, 1 and 2 (the DLL uses indices starting from '0' rather than '1'). Figure 4 shows the result.

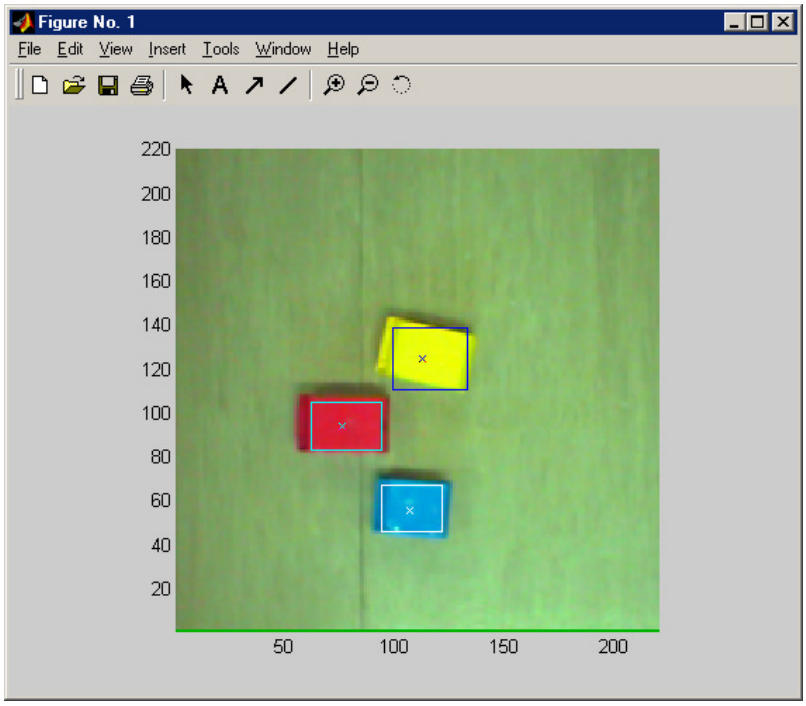

Figure 4: Successful detection of all three objects

trainCamera

The utility *trainCamera* allows a user to work from a life image stream rather than a previously recorded still image. The program first checks if the machine it is running from features a camera and the corresponding framegrabber card. Should this check return negative, a pre-recorded AVI movie clip is used instead. The default filename is *testmovies/test_short.avi* (see line 17 of the script file 'trainCamera.m'); this name can be modified to any suitable AVI movie file, e. g. the robot test movie *test_robot_LQ.avi* (low quality) or *test_robot_HQ.avi* (high quality). Both can be found in the folder *testmovies*.

The use of *trainCamera* is essentially identical to *trainStill*. The only difference is that the user first has to select an appropriate input frame from the stream of live images captured by the framegrabber card, or from the playing AVI movie. A frame can be frozen through clicking on the push button 'Freeze'. The following steps of object selection and range detection are the same as before. Figures 5 and 6 show the utility *trainCamera* in action.

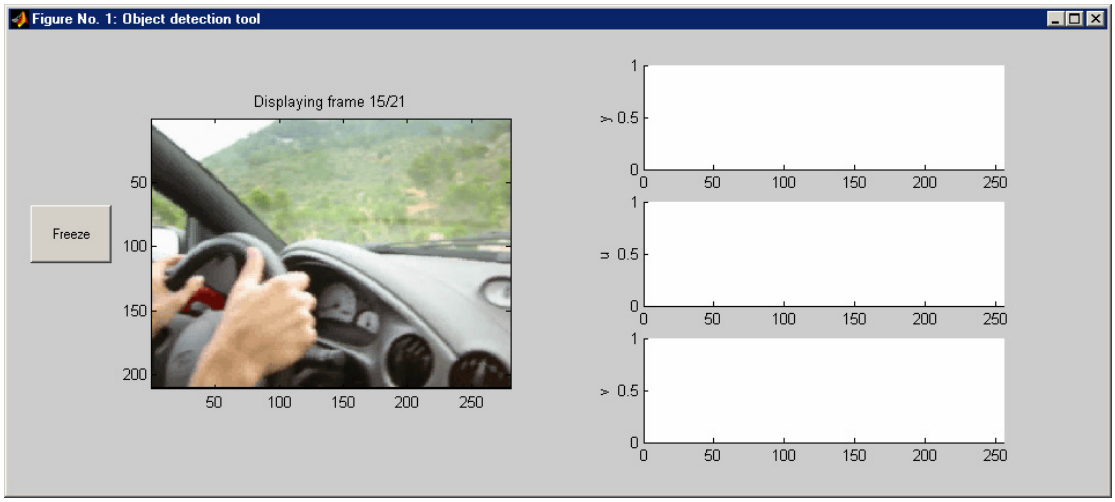

Figure 5: Utility *trainCamera* in action: click 'Freeze' to select a frame

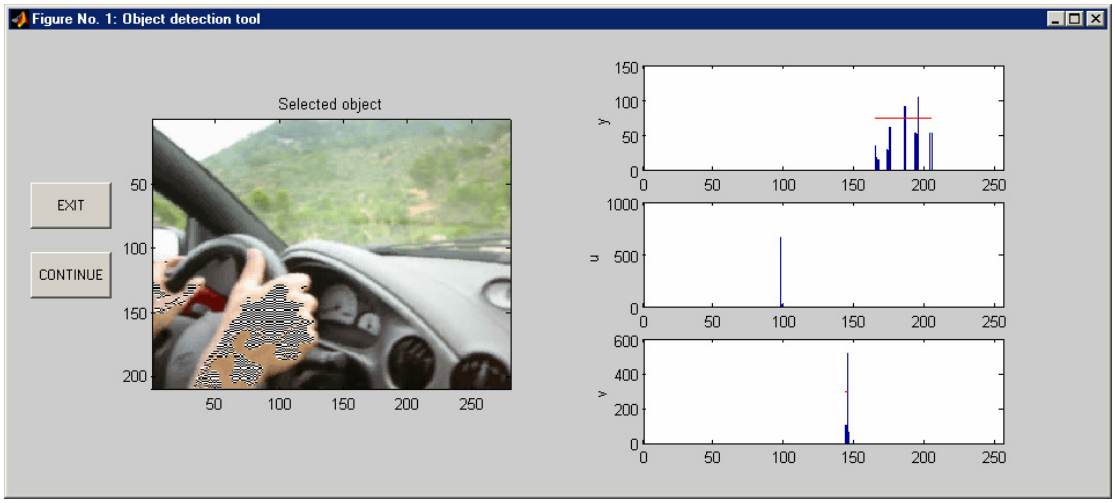

Figure 6: Utility *trainCamera* in action: selected area shaded in black and white

The corresponding colour range is displayed in the MATLAB command window.

```
-----------------------------------------------------------------
(Ymin:Ymax, Umin:Umax, Vmin:Vmax)
(178:210, 97:97, 142:144)
-----------------------------------------------------------------
```

To validate this colour scheme the captured frame has to be stored in a regular MAT file. The following few commands accomplish this:

```
rgb = uint8(CAPmov(15).cdata);
save testimages/testAVIframe15 rgb
test('testimages/testAVIframe15.mat', 1)
axis ij
```

First, the RGB data values of frame 15 of the currently loaded AVI movie (variable: CAPmov) are copied to a variable 'rgb' which is then stored in the MAT-file

'testimages/testAVIframe15.mat'. Running the *test* script confirms the successful detection of the selected colour scheme. For cosmetic reasons, the displayed image is rotated by 180° (axis ij).

Note that 'test.m' calls the DLL *imgProc.dll* which currently only works using the colour definition file with the hard-coded file name *testcolors.txt*. Prior to running *test* the user thus has to ensure that the above colour range is copied to the *Thresholds* section of *testcolor.txt*.

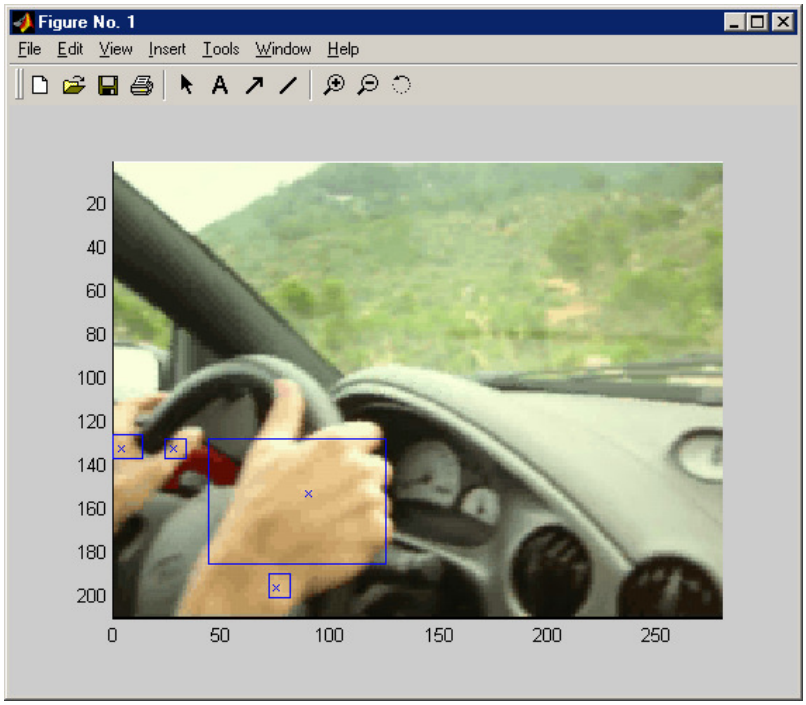

Figure 7: Validation of the selected range

Remark:

The U range of the above colour range is '97:97'. This effectively instructs the vision recognition algorithms to limit their search the pixels with a U-value of 97. To make the object detection more robust under varying lighting conditions, this range should be widened. This can be done manually, or by using a larger area in the object selection of *trainCamera*.

Frank Wörnle, April 2003.