JavaCam Project 2005

Design and Organization of Autonomous Systems
University of Amsterdam

Jun Wu, 0310190, jwu@science.uva.nl
David Persons, 0100722, dpersons@science.uva.nl
Michael Metternich, 0012645, mjmetter@science.uva.nl
Daan van Schuppen, 0008524, dschuppe@science.uva.nl
Derk Crezee, 0138606, dcrezee@science.uva.nl

Supervisor: Peter van Lith peter@lithp.nl

Abstract

With the increasing demand for surveillance of public and private areas, the number of cameras has grown substantially and monitoring all of them has become a difficult task. A solution lies in the use of 'intelligent' cameras. These are cameras that can assist people in surveillance with use of certain inbuild intelligent functions. In this paper we will introduce and discuss some of these intelligent functions. We will focus on two tasks; people counting, object tracking and general movement extraction.

1 Introduction

Nowadays you could not think of a society without cameras. Cameras are used for several tasks of which security and surveillance are the most important. As cities grow larger, the task of conserving general security becomes a tedious task for humans. There is a great need for surveillance systems which could autonomously perform certain tasks or which could assist humans in conserving security. Examples are detecting people who are committing obscure actions like leaving an unknown packet at the airport, detecting pickpockets in public areas or detecting burglars or terrorists around industrial areas.
In this paper we describe the techniques used to tackle two tasks. In the first task moving objects or people have to be detected from a particular distance called the General Movement Extraction and Object Tracking. This task could be used for example in industrial areas where it is useful to know if a disaster happens in which way groups are moving. The second task consists of counting the number of people entering or leaving a room called the People Counting task. This again has to do with security. In this case it is useful to know how much people are in a room or building in order to an efficient evacuation in case of an emergency.
These two tasks are implemented in a Java environment in combination with a simple web camera. Java is not the most common programming language to develop image processing techniques because it is relatively slow. The reason that it is implemented in Java is that a great part of the software can be implemented in a hardware camera which works on Java byte code; this will speed up processing.

To begin we describe the Graphical User Interface of the used test program in paragraph 2.1. After that we discuss colour models and some standard image processing operations which must assist in solving the more intelligent tasks, in paragraph 3 and 4. Finally in the rest of the paper, we will introduce some solutions for the two tasks described above and discuss the outcome.

2.1 Graphical User Interface

For testing purposes a test program has been created in an earlier point in the project. Based on the original interface, a more natural feeling user interface was created to improve the usability of the java-based software. The original layout was rather unclear since too much information is given to the user at once. A lot of the functions would never be used, and the data-fields with port information or additional parameter information were obsolete as well. Therefore, these were simply removed. The play and stop functions were replaced from the functions at the right window since their more direct impact on the screen is better shown this way.
Furthermore, each function at the right window automatically calls the start and stop function. The user will not notice the stream of pictures was ever interrupted, but this makes it easier to switch between functions. The new interface is shown in figure 1.

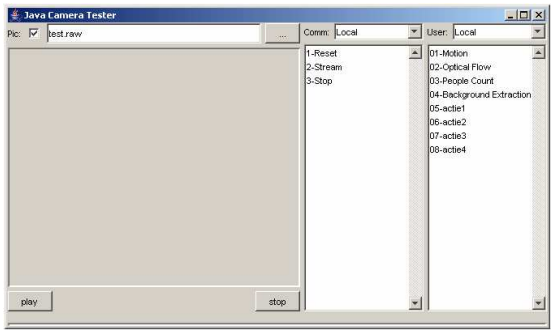


Figure 1: new interface

2.2 Dynamic Sensors

For some functions sensors need to be placed on the screen. How these sensors work exactly will be shown later in this paper, but we will now be talking about placing these sensors. Traditionally their position was hard-coded, but to create a usable, user-friendly program obviously this had to change. Since their position is specified by the user in the interface, but the location is needed in the hardware part, this problem turned out to be a lot harder then we initially anticipated. When a function that uses sensors is called, the sensors can be placed on the screen by dragging the mouse. When the play button is pushed these locations are stored in the user interface.

3. Colour and colour-models

In realistic circumstances image conditions are not static. Different colour models could be used to create a view that is indifferent to shadows, viewing direction, surface orientation, and occlusion or cluttering of the object. In the Java Camera project, initially all functions used the RGB colour model. In order to improve the robustness of this system, we intended to use different colour models. Before we go further in image processing techniques we discuss different colour models.

3.1 The RGB colour-model

This system consisting of Red, Green and Blue components is a commonly used well-known model. This model comes from the principle of how the human beings perceive colours. The human retina contains three types of cones which are respectively sensitive to 620nm, 520nm and 450nm light spectrum (Red, Green and Blue). The wavelengths of perceived light are mapped to RGB values using the following equations:

$$R = \int_{\lambda} E(\lambda) U_R(\lambda) d\lambda$$

$$G = \int_{\lambda} E(\lambda) U_G(\lambda) d\lambda$$

$$B = \int_{\lambda} E(\lambda) U_B(\lambda) d\lambda$$

Where λ are the wavelengths, $E(\lambda)$ are spectral energy distribution and ($U_x \mid X \in \{R,G,B\}$ are colour component matching functions.

3.2 Intensity Invariant Colour-models

The HSI colour model represents all visible colours by their Hue, Saturation, and Intensity. Hue is referred to as the dominant wavelength of a colour. Saturation is defined as the amount of white mixed with the perceived colour. Intensity is the amount of light that is perceived. The HSI values of an image can be calculated from the image's RGB values as follows:

$$H(R,G,B) = \arctan\left(\frac{\sqrt{3}(G-B)}{(2R-G-B)}\right)$$

$$S(R,G,B) = 1 - \frac{\min(R,G,B)}{R+G+B}$$

$$I(R,G,B) = R+G+B$$

Another intensity invariant colour-space is the normalized RGB colour model (rgb). The rgb colour model is defined, with respect to the RGB space, as follows:

$$r(R,G,B) = \frac{R}{R+G+B}$$

$$g(R,G,B) = \frac{G}{R+G+B}$$

$$b(R,G,B) = \frac{B}{R+G+B}$$

The c1c2c3 colour model Gevers [3] defines the c1c2c3 colour model. This colour-model is defined as follows:

$$C_1 = \arctan\left(\frac{R}{\max(G,B)}\right)$$

$$C_2 = \arctan\left(\frac{G}{\max(R,B)}\right)$$

$$C_3 = \arctan\left(\frac{B}{\max(R,G)}\right)$$

Gevers also defines the l1l2l3 colour-model [3]. This colour-model is defined as follows:

$$l_1 = \frac{(R-G)^2}{(R-G)^2+(R-B)^2+(G-B)^2}$$

$$l_2 = \frac{(R-B)^2}{(R-G)^2+(R-B)^2+(G-B)^2}$$

$$l_3 = \frac{(G-B)^2}{(R-G)^2+(R-B)^2+(G-B)^2}$$

Some examples of the different colour models are shown in figure 2.

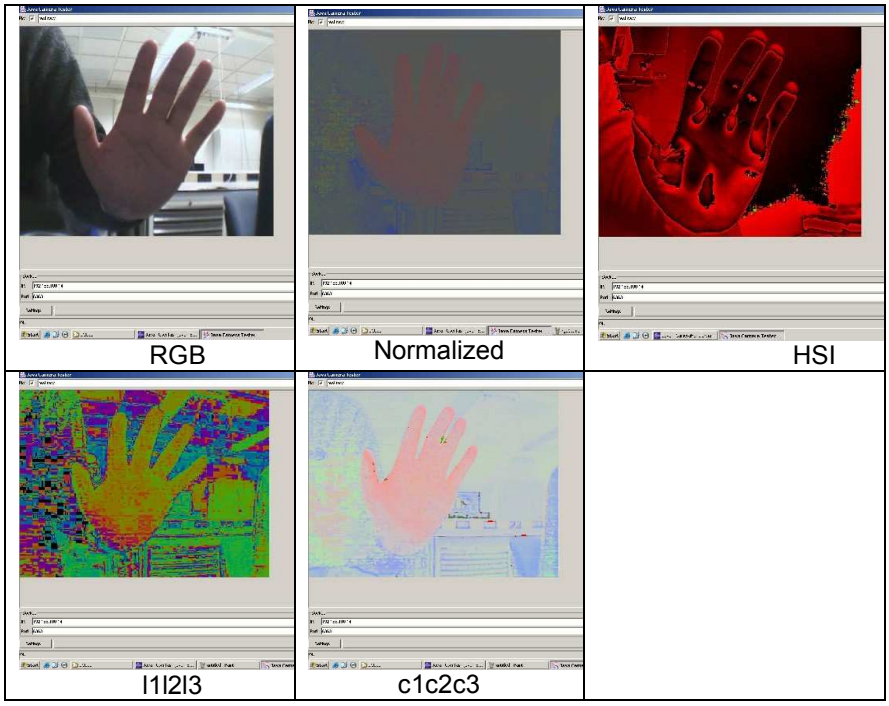


Figure 2: examples of different colour models

4 Background Subtraction

For the detection of moving objects we will make use of a method that is called Background Subtraction [9,10]. The first frame seen by the camera is saved, the background. This first frame is subtracted from all the next frames. When the value of a pixel is near to zero after the subtraction the pixel did not change, there is a match and the pixel is coloured black. When it does not match, it is coloured white.
There were some problems with our first implemented version of the Background Subtraction. When an object enters the screen for example, the focus of the camera goes to that object. Therefore the intensity of the background can change. Because of that it is better to do the Background Subtraction in an illuminant free colour-space. Therefore our standard Background Subtraction uses the normalized RGB colour-space. It is also possible that there are already some objects moving in the first frame. To neglect this problem, we take the average of the first few frames as the background (10 for example).

4.1 Morphological Operators

Figure 3 shows the Background Subtraction when a person is in the image. The Background Subtraction detects the object quite good but there is some noise at the left of the image and the object itself still contains groups of black pixels. Morphological operators [1] could help us improve the results.

Figure 3: background subtraction in normalized RGB

The different white groups of pixels can be seen as sets. Structural morphological operators are set operators that use structuring elements. Structuring elements are small sets that are used in the operator and correspond with the shapes we are looking for in the original set. The dilation and erosion are well known operators in mathematical morphology.

The dilation of two sets A and B can be interpreted as the vector addition of the two sets:

$$A \oplus B = \{c \mid c = a + b, a \in A, b \in B\}$$

where A represents the image being operated on and B is the structuring element.

$$A \oplus B = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

If we have a set operator that enlarges sets it might be handy to have an operator that shrinks sets as well. This is the erosion operator:

$$A \otimes B = \{c \mid B_c \subseteq A\}$$

with again A as the image being operated on and B as the structuring element.

$$A \otimes B = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

We implemented the erosion and dilation and combined it with the Background Subtraction. For our erosions and dilations we can use different sizes of structuring elements, depending on the scenery settings.

Figure 4.1: erosion of figure 2 with element size 5

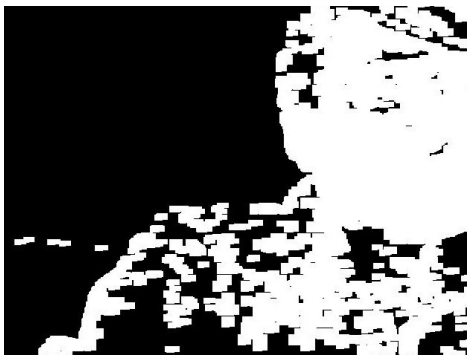


Figure 4.2: dilation of figure 2 with element size 15

From the images 4.1 and 4.2 can be seen that the erosion removes the noise and the dilation makes the object better visible. The result we get from first doing erosion and then doing dilation is called an opening and is a well known image operation. It's exactly what we need to improve our Background Subtraction.

Figure 5: background subtraction in normalized RGB with opening

We can see that morphological operators can improve our results of the background subtraction.

5 Sensors

To tackle the problem of counting the number of people leaving or entering the room and more generally to detect movements of objects or people in a sequence of images virtual sensors are used. The implemented sensors are an abstracted and simplified version of a human nerve cell called a neuron. In most cases a neuron reacts on the signals of the dendrites and sends out an output signal along the axon of the nerve cell. The general behaviour of theses sensors is to react upon a certain number of input signals in the form of an output signal. As in neurons the output signal will be activated if a certain threshold is exceeded. After this activation the neuron cell will be inactive for a certain period. This means that it is not sensitive to input signals.
To be more specific to the field of image processing a sensor operates on an area of pixels in an image. The pixels represent the input signals to a sensor. An input signal simply is a Boolean value representing the activeness of that pixel. There is a wide range of definitions of the activeness of a pixel, which could be defined by the sensing function. You could define the sensing function for example to become active on motions in the image or to become active if an object appears as the foreground of an image through background subtraction. If a particular percentage of these pixels become active the threshold of the sensor is exceeded and the sensor will output a signal. After exceeding the threshold the sensor is not able to sense for a certain defined period. In the Javacam application the timeline is represented as the different frames of the output stream of the camera.
A characteristic of both the neuron and sensor is that they could be connected to each other; in this way a hierarchy could be build which provide a certain functionality. To speed up the sensor is it possible to not

sense every pixel but to skip a number of pixels in an area. If a sensor is defined for a certain location with width w and height h it is not necessary to sense every pixel of the area w*h to still cover the whole area.

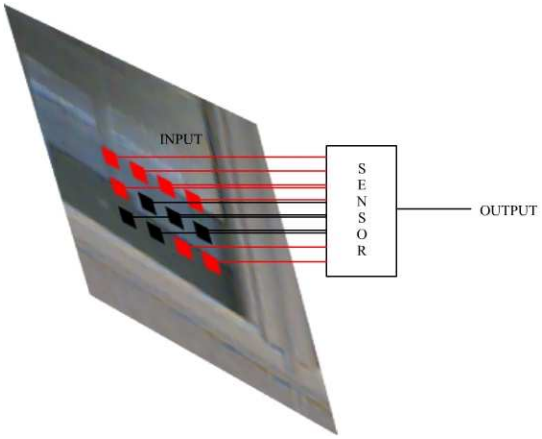


Figure 6: sensor with as input the pixels of the frame

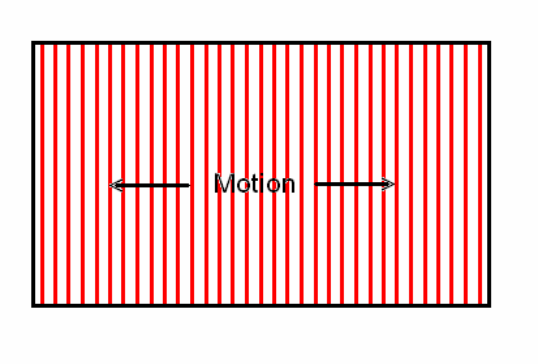
6 Object Tracking and General Motion Extraction

The first task we've had to solve was the task of Object Tracking and General Motion Extraction. The point was to extract information about global movement in a picture. This information could then be used for several higher level intelligent tasks, one of which is detection of panic (people running in one particular direction). For the extraction of global movement we've devised several solutions, which will be discussed in the next paragraphs.

6.1 Sensor Grid

The first solution tries to extract global motion with the use of sensors. Different sensors will be placed in a grid across the whole picture. By detecting motion in certain areas within successive frames we can try to derive motion in a particular direction. In the course of this project we've only tried to derive horizontal and vertical motion, but other directions could possibly also be extracted in this manner.

Broad shaped sensors were placed to detect vertical motion, and long shaped sensors to detect horizontal motion (see fig. 7.1 and 7.2).



7.1 Vertical sensors for horizontal movement detection

7.2 Horizontal sensors for vertical movement detection

Detection of movement was done by keeping track of the sensors 'triggered' at each frame. For all sensors we keep track of the frame number at which the sensor was 'triggered' last. From this information we can derive motion at a global scale. The way we extract motion from this information is by looking for ascending or descending traces within this triggered-sensor array. For instance if the frame numbers for sensors show a trace of {1,2,3,4} we could say that an object is moving in a certain direction, given the fact an object has moved through different sensors at different frames.

6.2 Image Segmentation and Matching

A second solution for the task is by making use of image segmentation. After the background subtraction discussed earlier we are left with a bitmap which tells us what is background and what is foreground. These bitmaps could be filtered of noise by the use of opening and closing. At this point we have a picture like the one in figure 5. The picture consists of a black background, and white (moving) regions that represent the foreground. Image segmentation could be used to extract the white regions in the image, where after these regions could be matched in successive frames, to extract motion. The motion of all regions could in the end be combined to give an indication of global motion.

Many image segmentation algorithms have been proposed [4,5,6,7,8]. In this task we will use a simplification of the seeded region growing algorithm introduced by Adams and Bischof [4]. The algorithm uses similarity and dissimilarity between neighbouring pixels to grow regions with respect to a number of starting points, known as seeds. We start with number of seeds grouped together in n groups: $A_1,...,A_n$. Groups can consist of any number of pixels as long as the group is not empty. At each step a pixel is added to one of the groups. Let T be the set of unallocated pixels which border at least one of the regions. This means that T is the set of pixels that form the total boundary of all regions formed up to now.

$$T = \left\{ x \notin \bigcup_{i=1}^{n} A_i \mid N(x) \bigcap \bigcup_{i=1}^{n} A_i \neq 0 \right\}$$

N(x) is the set of neighbouring pixels of x. In our case we will usually look at the 8-connectivity of a pixel, but other connectivity masks can also be used to speed up computation.

At each step the algorithm takes a pixel x from T, and adds it to one of its neighbouring regions. Then we look at N(x), the neighbours of x, and put them in T in ascending order, according to some distance measure, with respect to their neighbouring regions. The distance is simply the measure of dissimilarity with the neighbouring regions mean (intensity).

$$\delta(x) = \left| g(x) - \underset{y \in A_i(x)}{mean}[g(y)] \right|$$

This keeps T sorted with respect to the distance, so that less distant pixels are considered first for adding them to a region. If a pixel is on the boundary of more than one region, we have to decide to what region we should add it. We can simply calculate the distances of this pixel to each neighbouring region, and add the pixel to that region, for which the distance is the smallest.

A pseudo code of the algorithm is given below. In the code a simple sorted list (SSL) is used to store elements of T.

Initialization:
Label seed points according to their initial grouping.
Put neighbours of seed points (the initial T) in the SSL.

Region Growing:
While SSL is not empty **do**
  Remove first pixel y from the SSL.
  Test the neighbours of this pixel:
    if all neighbours of y which are already labelled (other than boundary label) have the same label
    then
      Set y to this label.
      Update running mean of corresponding region.
      Add neighbours of y which are neither already set nor already in the SSL to the SSL according to their value of delta
    else
      Flag y with the boundary label.

The algorithm we used is a simplification of the algorithm described above. Because we are only working with bitmaps, we don't have to keep track of any mean, or have to make checks for distance. We can just check whether pixels on the boundary are white and, if they are, add them to the region. Furthermore we should discuss the way we select seeds. What we do is loop through all pixels in the bitmap image and take the first white pixel we come along as a seed. Then we start growing the seed until it stops and a region is formed. Now we subtract this region from the bitmap, and continue the loop through the pixels looking for new seeds.

6.3 Movement detection using regions

Now that we can distinguish the different regions in a frame we should be able to detect the movement of these regions (representing moving objects or people) in time and to quantify the speed of the motions.
Difficulties arise in the detection of motions in the successive frames. In each frame the regions are known, nevertheless the mapping of the regions of the previous frame to the regions in the current frame is not known. It is never guaranteed that the region in one frame representing an object is the same region in the other frame representing the same object.
Another problem is that one object is not always represented as one region, because the background subtraction is not an object segmentation algorithm. An object could be in one frame consisting of a large region and in another frame consisting of multiple small regions.
As objects are moving through the screen of the camera it is also possible that certain regions will appear or disappear, so there will always exist regions that could not directly mapped to a region in the previous frame.
An approach to motion detection of objects is the Nearest Neighbour method [11]. This method assigns a region in the current frame to a region in the previous frame, which is closest to the region in the current frame with respect to certain properties of a region. The use of the Nearest Neighbour algorithm is based on the assumption that when an object moves through the camera the location of this object in one frame will only differ relatively small from the location in the next frame. Also size constancy is assumed; the size of an object will only smoothly change from one frame to another frame. The Nearest Neighbour method compares the regions of two successive frames with respect to the distance from region to region and the difference in size of the regions.
If the previous frame does not contain any regions (there are no objects in front of the camera) and there are regions in the current frame then they will be considered as new objects appearing in the screen. Because these new regions could not be mapped to the previous frame they will not have any begin speed. If regions can be mapped to previous regions they build up an average speed, which is the movement in pixels from region to region. If the current frame does not contain any regions and the previous does contain regions, it means that the objects have moved out of the screen, so no mapping is needed.
If both frames contain regions a mapping is needed. This mapping is done by calculating for each region in the current frame the difference in distance and size compared to the regions in the previous frame. For both the difference in location and the difference in size a ranking is built. The region in the current frame will be mapped with the region in the previous frame which has a lowest total ranking of the difference in location and the difference in size. The region in the current frame will take over the average speed of the mapped region of the previous frame; also the average speed will be updated.
The mapping of regions with respect to the least distance between the regions is not always a good measure. If for example there is only one region in two successive frames,

these regions will always be mapped together even if the distance between them is large. A solution to this problem is to restrict the distance between the regions to a maximum distance. This maximum distance is difficult to define because there it is not one predefined distance; it depends on the speed and size of an object. Objects moving at high speed should be tolerated to a higher maximum than objects moving at low speed. To get rid off the possible noise of the background subtraction algorithm only regions of a particular size will be considered.

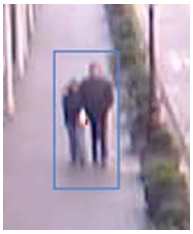


Figure 8 the people in the frame are detected as a moving region at low speed

A consequence of the assumption that locations of an object in one frame and the next frame will differ relatively small is that one object or region could melt together with another object. For example if there is one object that is not moving in the frame and another object is moving to this object, the moving object will melt together with the non moving object.

7 People Counting

One aspect of counting the number of people entering or leaving a room is the detection of motion in a certain area of the frame. A second important aspect is the detection of the direction of the motion. Both aspects could be solved by using sensors and sensor grids.

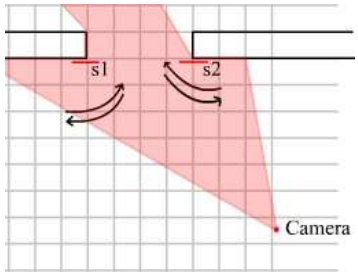


Figure 9a

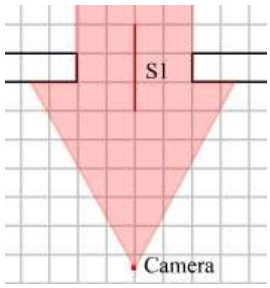


Figure 9b

Two horizontal grid sensor placed at the left en the right of the doorway (figure 9a). One vertical grid sensor placed at the floor of the doorway while the camera is at the ceiling (figure 9b). We implemented two methods for the task of counting people.
The first one places two horizontal grid sensors at the doorway (see figure 9a s1 and s2), which are able to detect horizontal motion. The camera is placed at the right or the left from the doorway.
The second one places one vertical sensor which is able to detect vertical motion. (see figure 9b s1) The camera is placed in front of the doorway at the ceiling looking down to the ground.

8 Discussion and future work

In the task of object tracking and general motion extraction by using sensor grids we ran into a number of problems. First of all the speed of a moving object was a problem. Object moving at high speed could pass sensors in successive frames, without triggering them. The reason for this is that for a sensor to register motion, the object should be within the sensors region for at least one frame. If this isn't the case, then there won't be any detection of motion. An object moving at high speed could thus lead to an erroneous triggered-sensors array, what would seriously affect the extraction of global motion. A second problem is the scale used to detect motion. With the scale I mean the size of the objects moving in the picture. The problem with scale is that it is directly related to the sensitivity of the sensors. For large object this should be small, but for small object, or objects seen from a distance, we should use very sensitive sensors. The problem with sensitive sensors is that they are prone to noise. The more sensitive a sensor is, the more interference there will be from noise. Noisy sensors will lead to an erroneous triggered-sensors array, thus affecting global motion extraction. Results of the use of sensor grids showed that sensor grids are too global and sensitive to noise to detect motion from a long distance. It is hard to locate the real movement in the frame.

The task of object tracking and general motion extraction by using regions heavily relied on the performance of the background subtraction. If the background subtraction does not perform well, the noise of it will be propagated trough the object tracking and general motion extraction. This was noticeable when multiple regions showed up in the image when there were no moving objects in the image. The noise was clearly visible in the background subtraction. Under static lighting conditions the RGB colour model performed well for the background subtraction and thus for the object tracking and general motion extraction. When people or objects showed up in the screen they were clearly visible by a bounding box around them. Also the speed arrow performed well.
Under variable lighting conditions the RGB model showed much noise. For the variable lighting conditions we tried to test the normalized RGB. In general it showed out that there was less noise. One disadvantage of the normalized RGB model is that it becomes less accurate for low intensity values (especially when using a simple webcam). We have tried to combine other colour models with background subtraction without explicit results.
As mentioned in the section 6.3 the threshold at which a region is matched with another region or should be considered as a new object, should not be a fixed value but should be a dynamical threshold depending on the speed and size of a region. To discriminate between moving and non moving objects the Nearest Neighbour method should also consider the difference in speed of the regions. To be more general regions could be compared by building histograms of them with respect to certain properties of the region (size, speed, RGB values). Multiple small regions which are close together and have more or less the same speed should be considered as one big region.

When testing the people counting task we concluded that the performance of the two counting methods (horizontal or vertical sensor grids) is strongly depending on the size of the sensors, the threshold of the sensors (depend of the size), the time that the sensors are inactive and the number of sensors placed.
It showed out that the best setting was the use of a grid sensor consisting of about 6 to 9 small but wide sensors. If too less sensors are used fast walking people were not counted, because all the sensor were active in the same frame. Therefore to still detect the direction of movement more sensors must be used. Also a small threshold and an inactive time of about 10 frames did work well.

Improvements could be made by defining a better sensing function for the sensors. In the implemented methods a motion detection function was used as sensing function. Most of the time this works well, but in some lightning conditions it could not detect motion.

Overall we can conclude that the software developed still needs some improvements to make it generally applicable. However we do think intelligent cameras can be made and would be an invaluable asset to certain applications.

References

1. Reader of the course Machine Vision, University of Amsterdam. Rein van den Boomgaard and Leo Dorst
2. Tijmen Majoor "Face Detection using Color Based Region of Interest Selection", Master's Thesis, 2000, University of Amsterdam, Faculty of Science.
3. Theo Gevers and Arnold W.M. Smeulders, "Color Based Object Recognition", 1999, University of Amsterdam, Faculty of Science.
4. R. Adams, L. Bischof: Seeded region growing. IEEE Transactions on PAMI 16, No. 6 (1994) 641 -647
5. J. Freixenet, X. Munoz, D. Raba, J. Marti, X. Cufi: Yet Another Survey on Image Segmentation: Region and Boundary Information Integration. University of Girona
6. R. Haralick, L. Shapiro: Image Segmentation Techniques. Computervision, Graphics and Image Processing 29 (1985) 100-132
7. K. Fu, J. Mu: A survey on image segmentation. Pattern Recognition 13 (1981) 3-16
8. N. Pal, S. Pal: A review on image segmentation techniques. Pattern Recognition 26 (1993) 1277-1294
9. S. Cheung, C. Kamath: Robust Techniques for background subtraction in urban traffic video. Center for Applied Scientific Computing Lawrence Livermore National Laboratory
10. S. Jabri, Z. Duric, H. Wechsler, A. Rosenfeld: Detection and Location of People in Video Images Using Adaptive Fusion of Color and Edge Information. Department of Computer Science, George Mason University and Center for Automation Research, University of Maryland
11. Tom M. Mitchell: Machine Learning. McGraw-Hill 1997