# Assisted living

L. Ovaere, C. Sysmans, S. Naert and C. Verstraeten

June 17, 2013

**Abstract**

Assisted living is a concept for seniors who need help with everyday tasks. They may need help with dressing, eating or brewing coffee, but they do not need full-time nursing care. Assisted living is more affordable than nursing home care, but it is still fairly expensive (staff costs). That is why it is interesting to reduce the amount of costs by implementing partial-automated home care. In this paper we introduce a robust vision framework that focuses on one specific task: brewing coffee. Two or more cameras are installed to recognize the different steps of brewing coffee. Afterwards the sequence of subtasks is processed and a warning or message is returned. The vision framework described in this paper can be used as a basis for automating other tasks in assisted living, e.g. taking a bath or doing the dishes.

**Keywords:** assisted living, robust vision framework, coffee brewing detection

## 1 Introduction

In this paper a robust framework for visual detection is introduced. It assists adults when brewing coffee and operates with two or three cameras. The framework can be expanded very easily and thus can be used as a basis for detecting other tasks. The engine of the framework is a *Deterministic Automata* (DA), that maintains the different states in the recognition of brewing coffee. The system is evaluated by four different scenarios and scores well on detection rate and handling false positives. To detect the different steps of brewing coffee markers, color segmentation and a calibration technique of the coffee machine are used.

### 1.1 Set-up

Figure 1: Set-up with three cameras

The vision framework works with two or three cameras, but it is also possible to work with only one camera (after some additional changes). To simplify we will focus on using three cameras in the remainder of the section. One camera needs to be placed (0,5m) above the coffeemaker and with the coffeemaker in the (approximate) center of its viewport. The top

camera also needs to be placed (0,3m) in front of the coffeemaker and with a small angle, so the coffeemaker is in the center of the viewport. The angle is required to detect the different LEDs of the coffeemaker. The other two cameras need to be placed (1m from the coffeemachine) in sideview, also with the coffeemaker in the approximate center of its viewport. In our set-up we used a 5mm lens for the top camera and two zoom lenses for the side cameras (Figure 1).

### 1.2 Overview

The remaining sections of this paper will discuss the different steps of brewing coffee (and the algorithms to detect them) and the implementation of the framework. Section 2 will discuss the different subtasks of brewing coffee in general. Section 3 will detail the structure of the framework and explain the important role of the deterministic automata. In section 4 we will describe the important use of markers and the algorithms we have used to detect the different subtasks introduced in section 2. At the end of this paper we will discuss the results of the different detection algorithms.

## 2 Different Steps Of Brewing Coffee

Before we can give a description of the detection algorithms in section 5, we should identify the different steps of brewing coffee. In general, there are four different steps:

1. Fill the coffeemaker with coffee.
2. Fill the water tank with fresh water.
3. Place the jug on the hotplate.
4. Turn on the appliance.

The order of the first three operations may vary, but they should be completed before the appliance is turned on. In addition, each of these steps consists of several subtasks. Therefore each step is divided below.

### 2.1 Division of the general steps

#### 2.1.1 Fill the coffeemaker with coffee

This first step consists of four subtasks. Their order is important and should be preserved.

1. Remove the filter holder from the machine.
2. Put a paper filter in the filter holder.
3. Put pre-ground coffee in the filter.
4. Place the filter holder back into the machine.

#### 2.1.2 Fill the water tank with fresh water

This second step also consists of four subtasks. Their order can be changed, however some orders are impossible. For example, it is impossible to close the lid before it was opened.

1. Open the lid of the water tank.
2. Remove the jug from the machine.

3. Fill the water tank with fresh water. During this subtask, the jug is located above the device.

4. Close the lid of the water tank.

### 2.1.3 Place the jug on the hotplate

In this step there are no subtasks. In addition, this task is not always a separate action and it is possible to combine this step with the previous one. However, this task should be completed before the final step is performed. Therefore it is seen as a third general step.

### 2.1.4 Turn on the appliance

The previous three general tasks can be performed in different orders. However, it is important that this last task is executed at the end. Also for this step, there are no subtasks. It is only required to press the on/off button to turn on the coffeemaker.

## 2.2 Dynamic approach

From the previous part, it is clear that the order of some tasks can be changed. Therefore a static approach where the program detects one action after the other is not possible. There is a dynamic approach needed where different tasks can be detected simultaneously. In order to make this possible, a specific design is used. This is described in Section 3. The detection techniques for these actions are explained is Section 5.

# 3 The Application

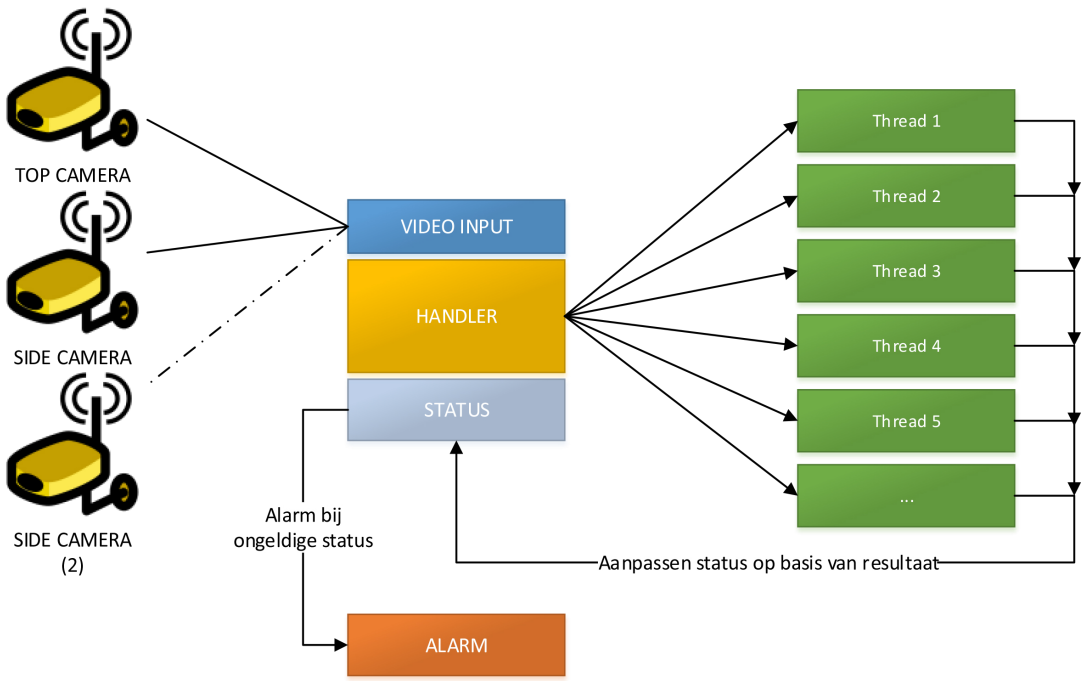


Figure 2: Back-end structure

The strength of the application is how the back-end is structured. By delegating each action to a separate thread, each action can be detected independently (Figure 2).

Dividing the application into smaller pieces also improved the efficiency during development. Each developer of our group was able to work completely independently and in the end all pieces fitted together perfectly.

In a single-threaded solution, the frame rate depends on the performance of the computer and the efficiency of the algorithms. In our multi-threaded solution there is no such limitation. Because all the actions are delegated to threads, the program can read the camera input with a fixed frame rate and therefore work on a live feed.

In general the back-end reads the **Video Input** and iterates over each frame. The back-end will start the threads at fixed intervals. This happens every $\frac{1}{3}$th of a second (333ms). The threads are only started when all previous threads have finished. Some threads are only started when they can change the state of the machine (e.g. when the jug is not outside of the machine, it is not necessary to start the thread containing the water detection algorithm, since the jug is required to put water inside the machine).

The reading of frames and starting the threads is done in the **Handler** class. At the end of each thread, it notifies the Handler class of its detected result. Each thread returns two items:

1. A boolean indicating the result of the algorithm.
2. A result image containing the current frame after the algorithms have been applied.

Based on the boolean, the Handler class changes the current state of the machine **Status**. When starting the threads, the Handler class will check the current state of the machine to decide which threads to start.

This can be compared with a deterministic automata where the machine is always in a specific state and can move to another state when it receives a certain input. When an action is detected, the Handler changes its current state, based on the previous state and the detected action.

When the **Status** object detects an invalid state, the alarm starts, writing the specific error to the output.

**False positives** Some threads use techniques such as color segmentation to detect actions. Since each video is different (light, shadows, ...) there can be variations in color. Another example is when a user moves his hand in front of the camera and the view is obstructed.

Since the above examples are probable, the threads can return a wrong result. When the Handler class receives a wrong result, the status of the machine should not be changed. To allow a few wrong results, a new class is introduced: *ThesholdBool*.

A ThesholdBool is a boolean - as indicated by its name - with a threshold value. Simply put, each time the Handler class receives a true/false value from a thread, it increases or decreases the inner Integer value of the ThresholdBool with a value of one. When this value is above a certain threshold (configurable for each ThresholdBool), the boolean is seen as TRUE, otherwise as FALSE. This method works well if there are less wrong results than correct ones.

You can compare this system with a bucket containing a hole in the bottom. When the ThresholdBool receives a TRUE value, a drop of water is added to the bucket. When receiving a FALSE value, a drop of water is removed. When the bucket is full, we say we have enough water (and thus the value is TRUE).

Using this technique actions (e.g. taking out the jug) are not seen instantly, but only after the threshold has been reached. This means there is a small delay before an action is detected. However, using this bucket system the program is more stable and temporary wrong results are ignored.

# 4 Techniques

In this section the techniques used to detect the different steps are described. Color segmentation in combination with different markers is the fundamental method of the system and is explained in Section 4.1. However, in order to detect each action successfully, the calibration mechanism of Section 4.2 is needed.

## 4.1 Markers

For the detection of the subtasks we decided to use markers to simplify and increase the recognition. We used different colored tape: yellow and green. These markers are posted on the coffeemaker, coffee holder and the jug. With color segmentation we are able to extract the positions of the different markers, and thus can be used to make specific calculations for each subtask.

The color segmentation is based on filtering in the HSV-channel. To complete this we have searched for the correct values of Hue, Saturation and Value for each color. Tests on multiple videos have confirmed that filtering in the HSV-channel delivers significantly better results than filtering in the RGB-channel. The practical use of the markers will be explained more detailed in Section 5. Table 1 illustrates the HSV-ranges for the different colors: yellow and green. These values where determined manually, based on three different situations (hue, saturation and value).

Table 1: HSV ranges

| Color | H | S | V |
|---|---|---|---|
| green | (60-73) | (132-256) | (16-256) |
| yellow | (0-98) | (208-256) | (166-256) |

## 4.2 Machine calibration

Some of the algorithms use only a specific section of the video input, e.g. the algorithm to detect whether or not the machine is running detects the color (more info in the next chapter) of the on/off switch. Since this color is not unique in most video inputs, it is required to crop the video input to the smallest possible region containing this indicator. Since not every video input is exactly the same (position of the machine, zoom, ...) calibration is required.

To make the algorithm above (and the others that will follow) work for every position of the machine, we developed an automated calibration system. When the application is started, the position of the machine is calculated, based on the calibration cross at the top of the machine.

In a first version, the openCV template matching technique was used. Since this technique does not give any info about the angle or size of the detected template, this technique could not be used.

Our own technique works like this (Figure 3):

1. Take the first frame of the video source (top camera) and filter out the cross as much as possible.
   (a) Use openCV's **medianBlur** function to filter out the noise.
   (b) Convert to grayscale.
   (c) Since the cross is clear white in grayscale, **threshold** the frame at a very white level so that only the cross remains (but unfortunately also other white objects).

2. Use openCV's **findContours** function to detect all the contours in the image.

3. Iterate the list of found contours and on each contour, use the **approxPolyDP** function to detect the points of the estimated polygon. Since a cross has eight contour points, all other contours are ignored.

4. It is possible but unlikely that another (white) shape with eight points is detected. To make sure the current contour is the one we are looking for, the smallest rectangle (containing the contour points of the entire shape) is calculated using openCV's **minAreaRect** function. This is shown in the following image:

Figure 3: Detection of cross, used in calibration

   The correct shape is chosen based on the width to height ratio of the expected cross.

5. Since we have information about the shape available such as contour points and the convexing rectangle, simple goniometry is used to calculate all the positioning information of the machine.

All the positioning information is stored in a positioning object that can be accessed by each thread. Using this object, a thread has positioning information available such as width and height of the cross, and (x, y) coordinates, representing its center. The zoom ratio is the ratio between the width found for the cross and the expected width and represents the zoom factor of the camera. The angle-attribute is the angle between the cross (and thus the machine) and the horizontal axis of the camera.

# 5 Subtasks

In Section 2 we described the different steps of brewing coffee. For each of the four general steps, a specific technique has been developed. These are described below.

## 5.1 Fill the coffeemaker with coffee

We introduce a procedure that is able to detect the coffee holder and whether a coffee filter and/or coffee has been put inside. To succeed, a simple property, the circular shape of the existing coffeeholder is used. With the combination of a marker and the expanded version of the hough-transform for circles, we are able to detect the coffee holder. To detect the coffee holder, yellow tape is used and the HSV-ranges specified in Table 1 are applied.

When the coffee holder is temporarly out of the viewport of the top camera, information of the calibration will be used to check if the coffee holder was placed in the coffeemaker or not. This allows a coffee filter to be filled out of the viewport of the top camera. When the coffee holder would accidentally leave the viewport, the system will know the coffee holder has not been placed back in the coffeemaker, but is out of the range of the camera.

Figure 4: Fill coffeemaker with coffee

If the coffee holder has been detected, its position (x- and y-coordinate) will be returned. Afterwards, when the coffee holder was detected, a limited region (in function of the diameter, d) around this position is analysed. The grayscale intensity of each pixel in this region is measured. Therefore a frame will be converted to grayscale and the arithmetic mean of the intensities is calculated (Figure 5).

When the arithmetic mean is known, a three way decision is made (Table 2). Based on the sequence of decisions the deterministic automata is able to switch from one state to another and detect if the coffeemachine has been filled with coffee or not (Figure 4).

Table 2: Intensity ranges

| Type | Min | Max |
|---|---|---|
| holder | 0 | 79 |
| holder + filter | 80 | 139 |
| holder + filter + koffie | 140 | 256 |

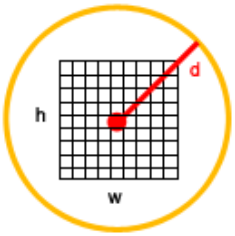


Figure 5: Mean calculation

## 5.2 Place the jug on the hotplate

Placing the jug on the hotplate is not treated as a real action. It is more like a condition that has to be satisfied. The jug must be on the hotplate when the machine is turned on.

There is a green marker on both sides of the machine and one on the jug as well. The marker is a small strip of tape that is wrapped around the jug. On the machine the marker is a broad vertical strip of green tape.

Green color filtering is used to detect these areas. Only the biggest horizontal and vertical area is saved. The distance between the middle of these two areas is calculated for the X-axis and Y-axis separately. If the jug is placed on top of the machine, this would not be detected as an mistake if only one distance is used. We can say that the machine is on the hotplate if these distances are both in a defined range.

## 5.3 Fill the water tank with fresh water

As mentioned in Section 2, this step consists of several subtasks. Therefore, different detection methods are needed. These are described below.

### 5.3.1 Open/close the lid of the water tank

To detect the lid of the water tank, we use the image of the top camera and a yellow marker on top of the lid. The yellow tape is placed over the entire width of the lid. As a result, the yellow area is large and easy to detect with color filtering. Since we use the image of the top camera, the yellow tape will only be visible when the lid is closed (Figure 6).

Figure 6: Water tank lid

### 5.3.2 Locate the position of the jug

For this action, the jug should be removed from the machine and located above the device. To track the position of the jug, we can use the same detection mechanism as in Section 5.2. When the horizontal distance between the two tape areas is large enough, we know that the jug was removed from the machine. The vertical distance indicates if the jug is located above the coffeemaker or not.

### 5.3.3 Check the indicator light

This third detection is not an action, but an additional check. Therefore, it was not mentioned in Section 2. Since it is difficult to detect the water itself and the previous two detections are not fully conclusive, this additional check was added.

The coffeemaker has a built-in indicator light. When the appliance is turned on and the water tank is not empty, the light is on. In this case, the orange color is detectable with color filtering. Since this color is rare in the top image, there is no need to crop it. If the color is not detected, there is no water in the water tank (Figure 4).

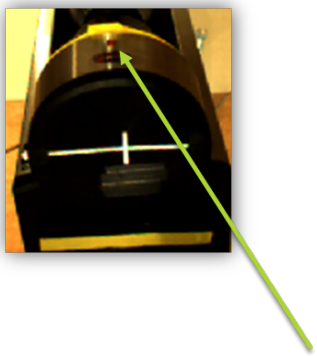

Figure 7: Indicator light

### 5.4 Turn on the appliance

In order to detect this action, we experimented with several techniques. To determine whether the coffeemaker is turned on or not, we should detect the position of the on/off switch.

First, we have marked the top of the switch with a white color. In this way, the white section is visible only when the machine is turned on. Since there are too many other white parts, this technique can not be used on the full image.

Secondly, we have used several other colors to mark the switch. Because of the limited area of the button, the detection on the full image remains a challenge. Therefore we decided to crop the image of the top camera.

The final technique uses a cropped image and a white color. As a result, the detection is very accurate. In order to make this possible, the calibration mechanism of Section 4.2 is used. Without the calibration, the crop parameters should be adjusted manually when the coffeemaker is moved (Figure 8).

Figure 8: On/off switch

## 6 Results

### 6.1 Error ratios

Table 3 shows the results of the ROI measurements for every detection function. This is measured for a video about brewing coffee the correct way. Only one side camera is used. Most of

Table 3: ROI results for every funtion

| Function | Errors | Tests | ratio |
|---|---|---|---|
| Jug | 1 | 386 | 0.26% |
| Machine on | 2 | 386 | 0.52% |
| Machine running | 4 | 211 | 1.90% |
| Lid opened | 4 | 386 | 1.04% |
| Coffee holder | 6 | 386 | 1.55% |
| Coffee filter | 4 | 12 | 33.33% |
| Coffee | 13 | 44 | 29.55% |

the rates are really low and the faults can be explained easily. The machine generates steam when it is running. This interferes with the detections using the top camera. People can walk in front of the cameras at the side, which causes errors as well. Detecting the jug fails when a person is in front of the side camera. Using a camera on both sides solves this problem.

Detecting the coffee and the filter is more difficult than the other detection processes. Shadows are also a reason for faults in the detections.

Section 3 explains how these false detections are filtered out and this is a crucial step for a correctly functioning system.

### 6.2 Scenarios

The platform was tested using the following scenarios:

- Making coffee without any mistakes.
- Putting in a new coffee filter, but no coffee.
- Turn on machine without a jug.
- Never take out coffee holder (so no filter and no coffee).

The first scenario is valid so there is no alarm. In the other scenarios, the machine state is invalid, so an alarm is shown (as expected). Figure 9 shows the output from scenario 2.

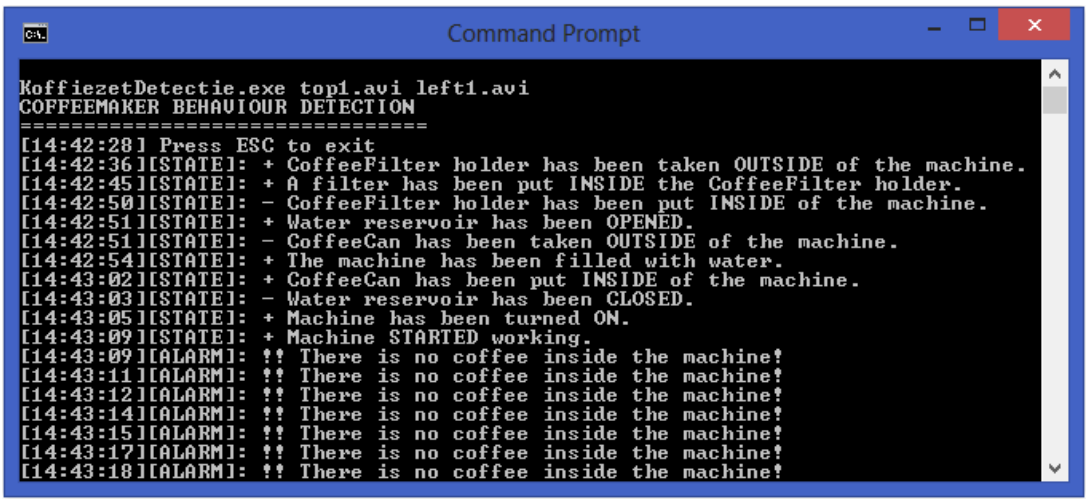


Figure 9: Application output with alarm

# 7 References


- http://opencv-srf.blogspot.be/2010/09/object-detection-using-color-seperation.html
- http://www.shervinemami.info/colorConversion.html
- http://docs.opencv.org/doc/tutorials/imgproc/histograms/template_matching/template_matching.html
- http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html