Combining View-based Object Recognition with Template Matching for the Identification and Tracking of Fully Dynamic Targets

M. Oliveira, V. Santos, *Member, IEEE*

***Abstract* — This paper describes a system intended to identify and track dynamic targets that may change appearance while moving. The full system includes a pan and tilt unit to ease tracking and keep the interesting target in the center of the image. View-based Haar-like features are used for object recognition while template matching continues to track the object even when its view is not recognized by object recognition system. Some of the techniques used to improve the template matching performance are also presented. Preliminary results are given and system performs well up to 15 frames per second on a 320 x 240 image on an ordinary laptop computer.**

I. INTRODUCTION

Object recognition using computer vision is a complex problem. View-based strategies are receiving an increasing attention because it has been recognized that 3D reconstruction is difficult in practice and also because of some psychophysical evidence for such strategies [1]. Therefore, to have a detector that can recognize an object and track it from every possible view is still a very demanding challenge. Furthermore, the dimension of a database to contain all of the objects possible points of view should be immense just for each single object, unleashing some other problems concerned with real time processing.

This paper proposes a method for tracking fully dynamic objects (that may rotate over any axis and, to some extent, modify shape), based on Haar features [2] [3] that are used as a single view identifier and complemented by template matching to track a previously classified object. Templates are self-updated when Haar features fail and redefined when they succeed, allowing the object to freely move and rotate overcoming temporary failures of the identification module.

First, the paper describes some of the ways that were used to capture the systems attention so that a particular image region may be processed by the identifier. Secondly, the tracker's implementation in a pan and tilt unit is briefly described and finally some of the several identifying and tracking techniques studied so far are discussed.

This work's final objective is to identify and track objects moving and rotating trough dynamic environments in real time (15Hz). Dynamic environments are difficult to handle because of the difficulties in achieving an accurate background subtraction without depth measurements that could be taken from stereo vision or laser. Moreover, light conditions may change considerably. Although it is still an ongoing work, there are some results mainly with using Haar-like features for identification and tracking with template matching and the implementation is following a modular perspective that may allow particular sectors of the task to be independently developed.

For visual tracking, a servo controlled pan & tilt unit is used. It supports a velocity of up to 300º/sec in both axes. Two IEEE1394 cameras are installed on the unit, though, for now, only one camera is used.

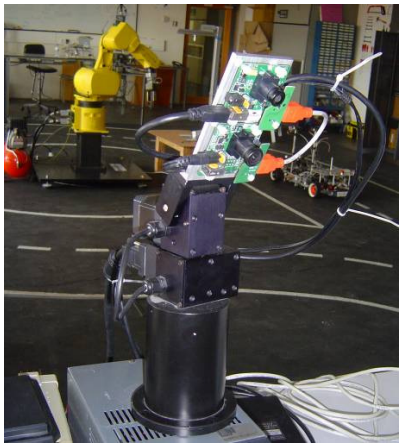

Figure 1 – Pan and tilt unit.

The camera's lens has a wide angle (89º) which facilitates tracking since the object is not easily lost from the camera view. The camera's vertical axis is coincident with the tilt plane, while pan movement shifts the image horizontally. The pan and tilt uses RS232 communication protocol and supports position, velocity and acceleration configuration.

II. ATTENTION MECHANISMS

The starting action in object tracking is, of course, to find the object that is to be followed. For this purpose, several methods are available, ranging from techniques that try to find the object in the whole image, to others that focus on some particular object characteristics. This chapter intends to describe some simple algorithms that play the role of attention mechanisms. Attention mechanisms are processes that center the system's attention into an area of visualization based on particular, usually simple, features. These are meant to be used when the system is searching for a known object to track.

Simple color recognition may work as a good attention

Reviewed manuscript received April 13, 2007. Authors are with the Department of Mechanical Engineering, University of Aveiro, Portugal. E-mails: {mriem, vsantos}@mec.ua.pt.

mechanism. Ude *et al.* use it as signal detectors that deploy attention on a particular image blob [1]. The object to be followed can be physically tagged with color markers and these will capture the system's attention. Alternatively, the object dominant color can be set as one worthy of attention. Color recognition is performed in the HSV color space which suits perfectly for color recognition since, unlike RGB, it separates color from light intensity and saturation. Therefore, the simplest approach might be to filter all the pixels with Hue values between some defined limits. Using a mask built with these conditions, the pixels with the desired color are filtered.

$$\begin{aligned} &if\,(\min_{Hue} > src_{Hue} > \max_{Hue}) \Rightarrow mask_{val} = 1 \\ &else \Rightarrow mask_{val} = 0 \end{aligned} \quad (1)$$

Where src_{Hue} is the Hue value for a given pixel and $mask_{val}$ is the value of the Boolean mask that is being built and indeed holds the object being segmented. The undeniable advantage of this method, its simplicity, is on the other hand contradicted by the fact that min_{Hue} and max_{Hue} values have to be precisely tuned. A very restricted interval may discard some of the object pixels, while a large, undemanding interval fails to filter background noise and does not entirely segment the object. A balance can be achieved if post processing is used to attempt to extract the remaining background pixels with some other criteria. Upon the application of an additional filter to remove isolated pixel, this operation commonly gives satisfactory results. Another possibility is to use pixel connectivity to separate the image into several spots and then select the spot with the largest amount of pixels. After some experiments, it was found that the final solution is to find adequate values for max_{Hue} and min_{Hue}, perform an isolated pixel filtering and, finally, calculate the pixels mass center. Mass center works well because the weight of the objects pixels (being far more than the rest) pulls it to the object center. The mass center is calculated using the following expression:

$$MC\left(\vec{x}\right) = \begin{bmatrix} \dfrac{\sum_{c=0}^{c=w-1}\sum_{l=0}^{l=h-1} mask_{val}(l,c)\times l}{n} \\ \dfrac{\sum_{c=0}^{c=w-1}\sum_{l=0}^{l=h-1} mask_{val}(l,c)\times c}{n} \end{bmatrix} \quad (2)$$

Where *MC* is the vector representing the mass center's coordinates, *l* is the line, *c* the column and *n* represents the total number of filtered pixels.

OpenCV's functions to access image pixels (`cvGet2D`) are quite slow [4] and only unrestricted pointer based access to pixels is very fast (about 10 times faster than `cvGet2D`). To avoid free pointer access and to keep software structure it is preferable instead to use OpenCV's built-in functions for mass center calculation, for which a method has been devised. Two matrices with the same size of the image are defined, where the first, called M_{line}, has for each "pixel" its corresponding line, and the second, M_{column}, has for each "pixel" values equal to its column, defining what is frequently called a mesh grid.

$$M_{line} = \begin{bmatrix} 0 & 0 & \dots & 0 \\ 1 & 1 & \dots & 1 \\ \dots & \dots & \dots & \dots \\ h-1 & h-1 & \dots & h-1 \end{bmatrix} M_{column} = \begin{bmatrix} 0 & 1 & \dots & w-1 \\ 0 & 1 & \dots & w-1 \\ \dots & \dots & \dots & \dots \\ 0 & 1 & \dots & w-1 \end{bmatrix} \quad (3)$$

Both these matrices are constant for every image size and are therefore created only once at the beginning of the program. No real time computational power is spent in this calculation. To get *MC*, it is simply necessary to sum the pixels for every mesh grid, using the color mask as a conditioner for the operation, and divide the final result by *n*, the total number of pixels of the object ($n = \sum\sum mask_{val}$):

$$MC\left(\vec{x}\right) = \frac{\begin{bmatrix} \sum\sum\left(mask_{val} \otimes M_{line}\right) \\ \sum\sum\left(mask_{val} \otimes M_{column}\right) \end{bmatrix}}{n} \quad (4)$$

Where $\otimes$ is the symbol for pixel multiplication. This procedure avoids user retrieval of the original image information (safe access), using only OpenCV's functions, and so, boosting the time of *MC* calculation when compared to the common OpenCV's image data retrieval functions (8~9 times faster than `cvGet2D`).

Image segmentation has some operations that are far more complex and effective. Pyramid segmentation [5] can be used to group pixels with color similarity and divide the image into a set of blobs. Each of these blobs represents a group of connected pixels whose color is similar. However, these methods are obviously more time costing and therefore are not used. In the authors' view, an attention mechanism must be a simple and fast process, since it does not try to detect complex features but only some particular attention capturing properties.

Another possible attention mechanism is the usage of optical flow. Optical flow techniques try to find pixels that correspond in two sequential frames, exiting also the possibility to match some particular features instead of all of the image's pixels, using a smaller amount of tracking data [6] [7].

These mechanism have already been implemented and tested, though the current object detection technique, Haar features (IV.B), do not make use of them since they already have an embedded attention cascade [2]. Nonetheless, attention mechanisms are relevant many other object detection techniques that require a previous background subtraction operation.

III. Tracking Controller

Object tracking implies following a predetermined object. However, in this approach and for the sake of program modularity, the tracker module does not need to know what it is following. This approach was chosen because it ensures total independence of the tracker module. Its inputs are the current image coordinates (C_{xy}) of the object to track and the

desired coordinates, i.e. where the object is and where it ought to be. Usually, the desired coordinates, named Target (T_{xy}) are the image's center. However, that may not always be the case.

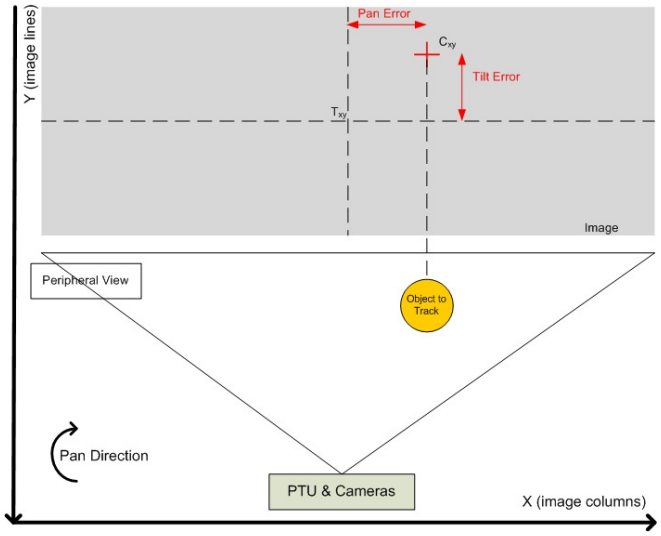

Figure 2 – Pan & Tilt controller schematics and controller implementation.

Figure 2 illustrates a schematic of the perception unit. The pan error (P_{error}) is given by:

$$P_{error} = C_x - T_x \tag{5}$$

While the tilt error (T_{error}) is defined as:

$$T_{error} = C_y - T_y \tag{6}$$

Two independent trackers, one for each image axis, i.e. one for pan control and one for tilt control, are implemented. Note that the philosophy of the whole implementation is not to use vision for exact metric measurements, but conversely, to use it for some kind of fuzzy control using only correlations between pixel distances. Therefore, there is no need to calculate the world coordinates that correspond to C_{xy} or to T_{xy}. Without possessing this information, it is hard to specify the desired angles to pan/tilt based only on the object image coordinates. These angles have to be specified to the hardware unit. To solve this problem, fixed angle values are set. That is, if the object is on the right side of the image, the pan position's value ($P_{position}$) is a predetermined one that pans the cameras entirely to the right, while the opposite occurs when the object is on the left.

$$\begin{aligned} &if\,(P_{error} > 0) \Rightarrow P_{position} = 200^{o} \\ &else \Rightarrow P_{position} = -200^{o} \end{aligned} \tag{7}$$

The controller is actually a speed controller (in practice, position only reports the signal of the speed value). Pan speed is controlled based on a PID controller that accounts for P_{error} magnitude, its past history and future trends whereas its signal is neglected.

$$P_{speed} = K_p \left\| P^n_{error} \right\| + K_i \sum_{n=0}^{N} \left\| P^n_{error} \right\| + K_d \frac{\left\| P^n_{error} - P^{n-1}_{error} \right\|}{\Delta t} \tag{8}$$

Where K_p, K_i and K_d are the proportional, integral and derivative constants respectively, n is the iteration index, N the max amount of iterations to account for, and Δt corresponds to the time that has elapsed between iterations n and $n-1$. Pan acceleration may also be controlled by a PID controller but for now it is set as a constant high value with acceptable results so far.

IV. View-based Object Recognition

A. Template Matching

Template matching is one of the simplest forms of identification. A template is a matrix that is tested on an image by finding some measure of similarity between the template and the image's pixel values. The template is tested in all possible image locations so, if the image has H × W pixels and the template h × w, then the matrix that results from the template matching operation [5] should have size:

$$(H - h + 1) \times (W - w + 1) \tag{9}$$

These so-called measures of similarity can be mathematically expressed in several ways.

Using the minimum square differences, the best possible match is obtained for the smallest R (smallest difference between image and template).

$$R(x, y) = \sum_{x'=0}^{w-1} \sum_{y'=0}^{h-1} \left[T(x', y') - I(x + x', y + y') \right]^2 \tag{10}$$

Where R is the resulting value, T the value of the template and I the value of the image, x' and y' the template's line and column coordinates respectively, while x and y are the images line and column coordinates.

Another method is the correlation technique:

$$R(x, y) = \sum_{x'=0}^{w-1} \sum_{y'=0}^{h-1} \left[T(x', y') \cdot I(x + x', y + y') \right] \tag{11}$$

In this case, the best match is achieved where R is higher (higher correlation). Template matching is a computationally demanding operation since the template is tested on every possible location on the image. However, using OpenCV's template matching function, and Intel Performance Primitives (IPP), one can do these operations in real time.

B. Haar Features

Haar features were first proposed by Viola *et al.* [2] as an alternative method for face detection. The general idea is to describe an object as a cascade of simple feature classifiers. Positive and negative set of example images are analyzed by a machine learning algorithm (Adaboost), that builds up a tree of features selecting, at each stage, the feature that best separates positive from negative examples. Lienhart *et al.* proposed to extend the features by utilizing also 45º rotated filters and have successfully optimized the classifier's performance [3]. Also, Haar features can be used not only for face detection [8].

For experimental purposes, we have decided to try to follow the rear of a small model of a car, whose color is very similar to the background in the laboratory and therefore the contrast between the object and the background is low. The

main idea would be to track the car rear area to, for example, later fetch its target plate while in free motion.

Figure 3 – Object to follow. Object is identified only when signaled area is viewed.

Approximately 1400 hand labeled images of the car's back were used as the positive set. Images were reduced to a size of 25x12. Identification's performance is very effective concerning scale variations (where variations of up to 400% of the training image size are successfully handled) However, when the object is shown at a slightly different angle (+/- 10º around any axis) than the one used for training, the detection are not sufficient.

V. Dynamic Object Tracking

This chapter describes how the previously mentioned modules, i.e. attention mechanisms, view-based object recognition and tracking, are used and integrated. Figure 4 shows the global architecture of the program, explaining the interaction between modules.

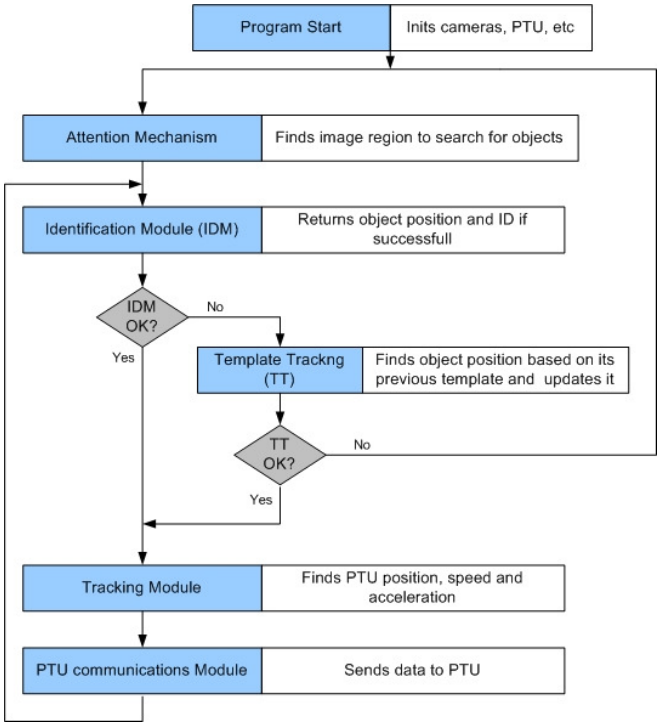

Figure 4 – A solution to integrate the proposed algorithms.

When an object is identified, a new tracking template is defined. Defining a template implies defining its size location and pixels intensity. Currently, due to real time demands, a grayscale template is used. When the identifier fails, the template tracking module is activated. This module executes template matching using the previously stored tracking template and the current frame. Afterwards, the new template's content is updated. The implementation of a car's rear identifier using Haar-like features has already been mentioned in chapter IV.B. Its performance is quite stable when the car is presented in the desired view. Whenever a match occurs in the identifier module, its output holds information about the target window, i.e. a rectangle whose position, width and height are defined by the identifier module.

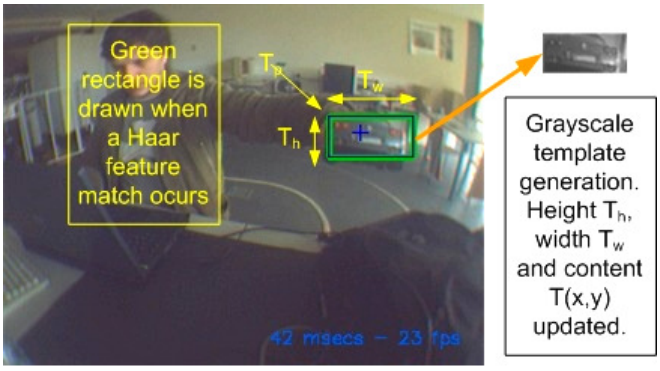

Figure 5 – Haar-like features based identifier. Template's position, size and content are updated.

The template's size, position and content are stored with the values provided by the identifier. If, in the next iteration, the identification module is unsuccessful, the new frame is scanned for the best possible match with the previously defined template using a template matching operation.

A. *Template Tracking Update*

The problem of how to update the template is also a difficult one to attend to. Kaneko *et al.* [9] have goaled the problem very well: "There is a trade-off relationship between accumulated errors and errors caused by image deformations. If templates are updated frequently, the accumulated errors become large. Conversely, if a template is not updated for a long time, a fatal large error occurs as a result of an image deformation." Kaneko *et al.* approach the problem in a much more complex way, defining inclusively an advanced template update criterion.

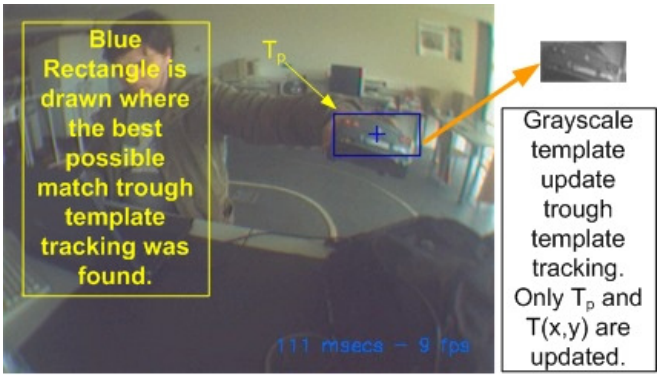

Figure 6 – Template matching when identification module fails.

A very simple way of updating templates is executed:

either the last defined template (N=1), or a simple average between the last N templates, work as the way to define tracking templates.

$$T_0 = \sum_{n=1}^{N} \frac{1}{N} \times T_n \qquad (12)$$

B. Template Matching Results Window

The results of template matching operation, which will be called TMR from now on, is a matrix like the one presented in Figure 7, taken shortly after the template definition of Figure 6.

Figure 7 – TMR. Best match for white regions.

As expected, the upper right corner of the frame (brightest part of TMR) is positioned on top of the car. In TMR, brightest points correspond to better matches (since the correlation technique was being used). In Figure 8, TMR is overlapped to its corresponding frame for a better comprehension of the process.

Figure 8 – Frame and template matching results.

As mentioned earlier, the size of TMR is equal to the size of the image subtracted by the size of the template.

C. Gaussian Conditioning

Figure 7 displays the results of the application of template matching to a frame. In that case, the highest probability point corresponds to the exact position of the car. However, since the template matching is done with grayscale images (for faster processing) and also because the car's color was selected (intentionally) very close to the background color, the template matching sometimes fails, warping to a completely different zone of the image. In the real world there is no warping and biological perception takes advantage of this fact. Of course that frame by frame analysis, as is the case, is always a discrete process and warping might occur due to low sampling frequency. However, assuming that the frame rate is high enough, it can be fairly trusted that an object early positioned at some coordinates will have, in the following iteration, a higher probability of being in the neighborhood of those coordinates than of having warped to some distant place.

In order to embed this "common sense" into our system, we have decided to use a 2D Gaussian probability function centered on the last templates position to build a matrix with the same size as TMR that will be referred to as GM. The matrix is calculated as follows:

$$G(u,v) = \frac{1}{2\pi\sigma^2} e^{\left(-\frac{u^2+v^2}{2\sigma^2}\right)} \qquad (13)$$

Where $G(u,v)$ is the function's value for every u (line) and v (column) and σ is the standard deviation expressing how "wide" the Gaussian is defined. In order to use the proper GM, the probability distribution needs to be centered on the object's previous iteration T_p.

Figure 9 – Using the Gaussian filter overlapped onto a frame.

Finally, to condition TMR with GM, it is necessary to multiply (pixel multiplication) TMR by GM. The output is that pixels that are far from the previous *Tp* reduce their probability (right on Figure 10).

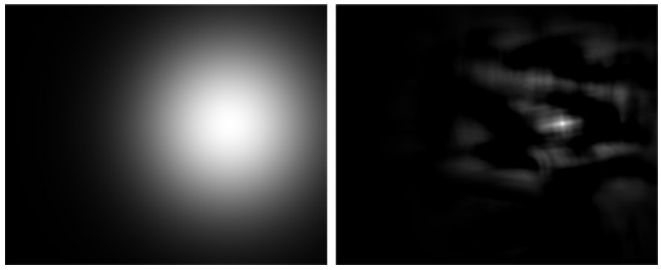

Figure 10 –Gaussian mask (left) and final probability map (right).

D. Fast Gaussian Computation

The Gaussian function calculation is not fast to compute. Doing this in all iterations (for every *Tp*) would considerably decrease the frame rate and an alternative method had to be developed. As a part of the program initialization processes, an extended Gaussian matrix (EGM) is calculated. To

Figure 11 –Haar detection (green window in A and I frames) fails at frame B, but tracking continues despite the objects movement and rotation.

calculate EMG minimum size, note that, by absurd, the smallest template one can use is of size 1×1.

Therefore, recovering chapter's IV.A template matching results matrix size equation, the biggest size that the template match results matrix can have is W×H, i.e., the image's size. The EGM is a Gaussian 2D function represented in an image of size (2W)×(2H), with the function centered in the center of the image. For every given T_p and TMR size, a specific sub-window of the EGM can be used without having to recalculate the probability values. Its upper left corner's coordinates are given by:

$$EGM_{upperleft} = \begin{bmatrix} EGM_{height} - \frac{TMR_{height}}{2} - T_{px} \\ EGM_{width} - \frac{TMR_{width}}{2} - T_{py} \end{bmatrix} \quad (14)$$

while its size is equal to the size of TMR. Using this technique, the Gaussian matrix's calculation is limited to the definition of a particular region of interest of the EGM, therefore saving precious computation time.

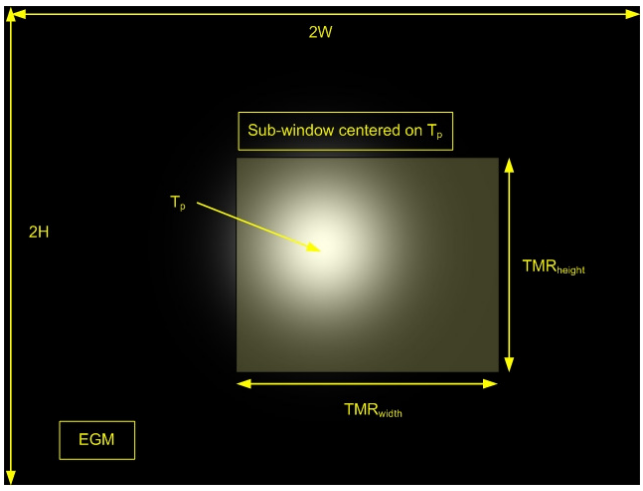

Figure 12 – Selecting a sub-window of the EGM.

VI. Results and Conclusions

This paper proposes a method for combining view-based identification algorithms with template matching trackers. Up to now, we are able to follow the test object even when it rotates and moves along the laboratory. Figure 11 shows a sequence of frames and the tracking results. In frame A, Haar detection succeeds and a new template is generated. In the following frames, the car's rear view disappears due to depth rotation. Object identification is no longer possible. At this point, tracking with template matching begins and the object continues to be followed even though, by frame D, it has rotated approximately 90 degrees. In frame I a new template was generated due to a Haar detection event. Therefore the template at frame I has a different size of the one in frame H.

All of Figure 4's modules are processed at a rate of 15Hz. Further work will scatter trough all of the modules. New object recognition methods will be implemented, namely Gabor filters [1]. Template tracking can be improved by doing sparse feature tracking. Conditioning matrices similar to the Gaussian ones may be utilized to further improve the tracker's performance. We are also planning to use more objects and to attempt to train a generalized cascade for the detection of cars.

VII. References

[1] A. Ude, C. Gaskett, G. Cheng, 2004, Support Vector Machines and Gabor Kernels for Object Recognition on a Humanoid with Active Foveated Vision, *Proceedings of 2004 IEEEIRSI International Conference on Intelligent Robots and Systems, Sendai Japan.*

[2] P. Viola, M. Jones 2001. Rapid Object Detection using a Boosted Cascade of Simple Features, *Conference on Computer Vision and Pattern Recognition 2001.*

[3] R. Lienhart and J. Maydt. An Extended Set of Haar-like Features for Rapid Object Detection. *IEEE ICIP 2002, Vol. 1, pp. 900-903, Sep. 2002.*

[4] OpenCV version 0.9.7 FAQ, *Included into OpenCV distribution.*

[5] OpenCV version 0.9.7 cxcore and cvaux documentation, *Included into OpenCV distribution.*

[6] J. Bouget. Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the Algorithm. *Included into OpenCV distribution.*

[7] D. Stavens, Introduction to OpenCV, Stanford Artificial Intelligence Lab. *Found at http://robots.stanford.edu/cs223b05/schedule.html.*

[8] G. Monteiro, P. Peixoto, U. Nunes, 2006. Vision-based Pedestrian Detection using Haar-like Features. *Encontro Científico, Festival Nacional de Robótica 2006.*

[9] T. Kaneko, O. Hori. Template Update Criterion for Template Matching of Image Sequences, *16th International Conference on Pattern Recognition (ICPR'02) - Volume 2, 2002.*