Object Trajectory Prediction Application to Visual Servoing

C. Pérez, N. García, O. Reinoso, J.M. Sabater and J.M. Azorín

Abstract— **Visual Servoing is an important issue in robotic vision. Considering tracking as a particular case of visual servoing, motion estimation algorithms are used to predict the location of target and generate a feasible control input to keep the target in the center of the image. Several well known algorithms can be used for trajectory prediction such as *Kalman* filter, $\alpha\beta/\alpha\beta\gamma$ filters, circular prediction algorithms and so on, but in this paper, we present a new filter based on existing filters that improves the prediction made by any one of them. This new filter is based on parameter optimization of a fuzzy system, therefore, we have named it: *Off-Line Optimized Fuzzy* FILTER (*OLOF* FILTER). The robustness and feasibility of the proposed algorithm is validated by a great number of experiments and is compared with other robust methods.**

I. INTRODUCTION

VISUAL servoing is a well known solution to control the position and motion of an industrial manipulator envolved in unstructured environment [1][2]. This is achieved by processing the visual feedback and minimizing an appropriate error function. Based on the visual information, visual servoing systems can be classified in different groups [3]: position based (3D), image-based (2D) or hybrid ($2\frac{1}{2}$D, ...) visual servoing. In image based visual servoing, 2D visual information is extracted from the image and used directly in the control law to generate the control signal. The visual servoing process depends on features extraction, matching, tracking and motion estimation algorithms. In this paper, we propose the use of a new object/feature prediction when it is out of the image during one or several iterations of the visual servoing algorithm.

The tracking issue depends on the object motion knowledge and it is crucial to achieve this purpose. With a good knowledge of the object motion, one can improve the performance of tracking and thus increase the accuracy of motion predictors. The problem arises when we don't know what the motion is. The proposed algorithm works in two different steps: first estimates the type of motion (constant velocity, constant acceleration increasing, ...) and second step, application of *Sugeno* [12] type fuzzy algorithm for position/trajectory estimation.

Some research work about the motion estimation is presented in [13] and [14]. Further, some motion understanding and trajectory planning based on the *Frenet-Serret* formula is described in [15], [16] and [17]. Using the knowledge of the motion and the structure, identification of the target dynamics may be accomplished.

The filter proposed is based on well known filters like (*Linear Interpolation (LI)*, *Kalman* filter (with different dynamic models), $\alpha\beta$ filter and $\alpha\beta\gamma$ filter), so, analyzing the cases in which each one of them works better, we can establish rules by adapted coefficients.

The new filter design, parameter optimization and analysis of the results are made using a well known benchmark: the decreasing bounce of a ball on the ground. We have chosen this system because it has changes of acceleration and speed.

This paper is focused in object position prediction and is structured as follows: in section II-A we present the different dynamics that can be considered for the object, because knowing the object's dynamics we can estimate the trajectory better. In section II-B we have presented four different filters. In section II-C, we can find the trust region optimization algorithm [18] used in this work. Section II-D presents the Sugeno type fuzzy algorithm [12] and the filter proposed in this paper is presented in section III. In section IV we can see the results with simulated data, in V the computational load analysis for all algorithms and finally in VI the conclusions.

II. THEORETICAL BACKGROUND

A. The Dynamics of a Moving Object

The objective of this paper is to follow a moving object, to do it it is necessary to know what type or types of movement it would have [5], in other words, what the motion model is like. The more we know about the object movement, the better the estimation/prediction will be, therefore, the results will be better. In several papers, we can find information about these dynamics. In this paper we show time expressions (see from (1) to (5)) and space state representation (see from (7) to (10)) of the jerk model ($jerk = \frac{d}{dt}(acceleration)$) [6].

$$\frac{a - a_i}{t - t_i} = \frac{\Delta a}{\Delta t} = J_0 \quad (1)$$

$$x(t) = x_i + v_i(t - t_i) + \frac{1}{2}a_i(t - t_i)^2 + \frac{1}{6}J_i(t - t_i)^3 \quad (2)$$

$$v(t) = v_i + a_i(t - t_i) + \frac{1}{2}J_0(t - t_i)^2 \quad (3)$$

$$a(t) = a_i + J_0(t - t_i) \quad (4)$$

$$J(t) = J_0 \quad (5)$$

This work is supported by the *Plan Nacional de I+D+I 2004-2007*, DPI2005-08203-C02-02 of the Spanish Government (*Técnicas Avanzadas de Teleoperación y Realimentación Sensorial Aplicadas a la Cirugía Asistida por Robots*)

C. Pérez, N. García, O. Reinoso, J.M. Sabater and J.M. Azorín are research staff of the Industrial Systems Department, Miguel Hernández University, Avda. de la Universidad S/N, 03202 Elche (Spain) carlos.perez@umh.es

where, x is the position, v is the velocity, a is the acceleration and J is the jerk. So the relation between them is:

$$x(t) = f(t);\ \dot{x}(t) = v(t);\ \ddot{x}(t) = a(t);\ \dot{a}(t) = J(t) \quad (6)$$

The matrix form of these expressions is:

$$x(k+1) = F \cdot x(k) + M \cdot m(k) \quad (7)$$

$$z(k) = H \cdot x(k) + N \cdot n(k) \quad (8)$$

$$\begin{pmatrix} x_{k+1} \\ v_{k+1} \\ a_{k+1} \\ J_{k+1} \end{pmatrix} = \begin{pmatrix} 1 & T & T^2/2 & T^3/6 \\ 0 & 1 & T & T^2/2 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_k \\ v_k \\ a_k \\ J_k \end{pmatrix} + + \begin{pmatrix} T^4/24 \\ T^3/6 \\ T^2/2 \\ T \end{pmatrix} \quad (9)$$

$$\begin{pmatrix} z_k \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_k \\ v_k \\ a_k \\ J_k \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \cdot n_k \quad (10)$$

Where the vector $x(k)$ is the space estate vector, and $z(k)$ is the output vector. F and H are the system matrices. $m(k)$ and $n(k)$ represents the system noise and the measurement noise respectively.

B. Trajectory prediction filters

1) Linear Interpolation (LI): The simplest analyzed filter is the *Linear Interpolation*. This filter is based on prediction calculus of the next position aligned with two immediately previous positions. For these data, the position prediction is calculated by the following expression:

$$x(t_{n+1}) = \frac{x(t_n) - x(t_{n-1})}{t_n - t_{n-1}} \cdot t_{n+1} + + \left(x(t_{n+1}) - \frac{x(t_n) - x(t_{n-1})}{t_n - t_{n-1}} \cdot t_{n-1} \right) \quad (11)$$

Thus, the next position will be on the line defined by two previous positions. All trajectories can be considered as a line for small values of ΔT.

2) The Kalman filter: This filter is recommended for systems affected by noise disturbance that cannot be modelled. It makes a *Bayesian* prediction of the state where the system model includes two random variables (*Gaussian* variables) with null average and a well known covariance (white noise), these variables corresponds to: the system error $v(k)$ and the measure error $w(k)$. The *Kalman* filter is based on a recursive expression of prediction and correction: it considers the current state from the prediction and adds a term of proportional correction to the prediction error, so this prediction error is minimized (optimal estimation). The algorithm can be divided in two differentiated phases: propagation and update, detailed in the following expressions:

The system model is:

$$x_{k+1} = F_k x_k + G_k u_k + v_k \quad (12)$$

$$z_k = H_k x_k + w_k \quad (13)$$

with Q and R covariance matrices of $v(k)$ and $w(k)$ respectively, we obtain:

Initial estimations:

$$\hat{x}_{k|k} \text{ and } P_{k|k}$$

1) Propagation step:

State prediction:

$$\hat{x}_{k|k} \text{ and } P_{k|k}$$

Measure prediction:

$$\hat{z}_{k+1|k} = H_{k+1} \hat{x}_{k+a|k} \quad (14)$$

Prediction covariance:

$$P_{k+1|k} = F_k P_{k|k} F_k^T + Q_k \quad (15)$$

2) Actualization step:

$$Inn_{k+1} = z_{k+1} - \hat{z}_{k+1|k} \quad (16)$$

$$S_{k+1} = H_{k+1} P_{k+1|k} H_{k+1}^T + R_{k+1} \quad (17)$$

$$W_{k+1} = P_{k+1|k} H_{k+1}^T S_{k+1}^{-1} \quad (18)$$

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + W_{k+1} Inn_{k+1} \quad (19)$$

$$\hat{P}_{k+1|k+1} = \big(I - W_{k+1} H_{k+1}\big) P_{k+1|k} \quad (20)$$

We can see that all the information that this filter needs, is stored in two variables, the actual estimation and the estimated covariance. The *Kalman* filter can be divided in three different types of filters depending on the system dynamics considered: *Kalman* filter for constant velocity objects (*Kv*), *Kalman* filter for constant acceleration objects (*Ka*), and *Kalman* filter for constant jerk objects (*Kj*). The difference between these three filters are: the dynamics considered for the object movement and the computational load needed (matrix dimensions are 2, 3 and 4 for velocity, acceleration and jerk model, see section II-A). This filter is widely used in visual servoing and tracking applications [4].

3) $\alpha\beta$ filter: The alpha-beta ($\alpha\beta$) filter is a particular case of the *Kalman* filter for a constant velocity system model. In this case, the filter gain is considered constant, so it is not calculated for each iteration. Also, it is not necessary to calculate the prediction of covariance estimation and innovation prediction simplifying the algorithm and decreasing the computational load.

The optimal values of α and β parameters depends of the relationship between the standard noise deviation and the standard noise average (relationship represented by λ):

$$\alpha = -\frac{\lambda^2 + 8\lambda - (\lambda + 4)\sqrt{\lambda^2 + 8\lambda}}{8} \quad (21)$$

$$\beta = \frac{\lambda^2 + 4\lambda - \lambda\sqrt{\lambda^2 + 8\lambda}}{4} \quad (22)$$

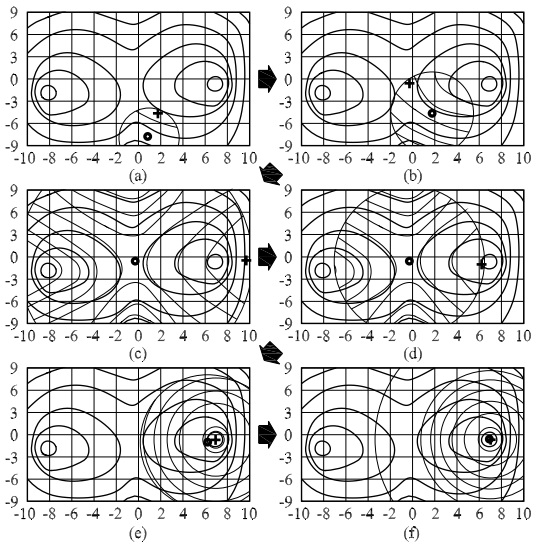

Fig. 1. Trust region algorithms evolution

Next we can see the expressions of this filter:
Initial estimation: $\hat{x}(k \mid k)$

$$\mathbf{W} = \begin{pmatrix} \alpha \\ \beta/T \end{pmatrix} \tag{23}$$

Filter gain:
Recursive loop:

$$Stateprediction:$$
$$\hat{x}_{k+1|k} = F_k \cdot \hat{x}_{k|k} + G_k \cdot u_k \tag{24}$$
$$Average\ prediction:$$
$$\hat{z}_{k+1|k} = H_{k+1} \cdot \hat{x}_{k+1|k} \tag{25}$$
$$Innovation:$$
$$Inn_{k+1} = z_{k+1} - \hat{z}_{k+1|k} \tag{26}$$
$$Optimal\ estimation:$$
$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + W \cdot Inn_{k+1} \tag{27}$$

4) $\alpha\beta\gamma$ filter: The alpha-beta-gamma ($\alpha\beta\gamma$) filter is again a particular case of the *Kalman* filter, but in this case, it is a filter based on a constant acceleration model system. As it has been commented before, the difference between them is the dimension of the vectors and matrices used to compute the predictions. Thus, the new gain matrix for the filter is (3x1 matrix):

$$\mathbf{W} = \begin{pmatrix} \alpha \\ \beta/T \\ \gamma/T^2 \end{pmatrix} \tag{28}$$

and the value of gamma is calculated using the expression: $\gamma = \beta^2/\alpha$

C. Trust region optimization algorithm

In this section we present the optimization method used explaining it by an example figure (figure number 1). For more information about trust region algorithm, see [18].

We have used an algorithm which makes the least possible number of function evaluations. Of course, to build a figure like 1, we evaluate the objective function a huge number of times. If you can construct figure 1, it means that your objective function is very cheap to evaluate and you might consider using an other type of algorithm like Rosenbrock [10]. We construct a local approximation $Q(x)$ of $f(x)$ around x_k, where $f(x)$ is the filter function for each segment. $Q(x)$ is represented in the figure 1 with bold lines. We will define a Trust Region around the current point x_k. The trust region is a disc of radius Δ_k centered at x_k. We will search for the minimum of $Q(x)$ inside the Trust Region. This minimum x_{k+1} is the black cross in figure 1.

We obtain $f(x_{k+1}) < f(x_k)$ (this is a success, see figure 1a), $Q(x)$ is a good local approximation of $f(x)$ and has given us good advice. We will move the current position to x_k and iterate $(k := k + 1)$. Since $Q(x)$ is so good we will also increase the trust region radius Δ_k. We will re-contruct a new quadratic interpolation $Q(x)$ around the new x_k. This re-construction can induce many evaluations of the objective function. In fact, in most optimization algorithms, this is where the greatest number of function evaluations are used. In the algorithm presented by [19], the author use a special heuristic method to reduce the re-construction cost. This heuristic method is based on *Multivariate Lagrange Interpolation Polynomials*. There are also other possibilities to construct $Q(x)$ like the *BFGS* method. See figure 1a for a graphical explanation of the second step of the algorithm. x_{k+1} (the black cross) is once again the minimum of Q inside the Trust Region.

Evolution of the algorithm presented in figure 1: We obtain, once again, a success (see 1b): $f(x_{k+1}) < f(x_k)$. We move to the new position. We Reconstruct $Q(x)$. We Increase Δ_k. This will be a failure (see 1c): $f(x_{k+1}) > f(x_k)$. $Q(x)$ is not anymore a correct approximation of $f(x)$. We thus reduce Δ_k. The current position x_k is not changed.

This is a partial success (see 1d): $f(x_{k+1}) < f(x_k)$ but the reduction is not as large as predicted by $Q(x)$, so we do not change Δ_k. The current position x_k is moved.

The next iterations do not present anything new (see figures 1e and 1f). Note that the trust region radius Δ_k is becoming very large at the end of the search. We stop when the step size $(=\| x_{k+1} - x_k \|)$ is becoming too small.

D. Sugeno-type fuzzy interface

The most common fuzzy inference process used is known as *Mamdani*'s fuzzy inference method. For this work, we have used the so-called *Sugeno*, or *Takagi-Sugeno-Kang*, method of fuzzy inference. Introduced in 1985 [12], it is similar to the *Mamdani* method in many respects. The first two parts of the fuzzy inference process, fuzzifying the inputs and applying the fuzzy operator, are exactly the same. The main difference between *Mamdani* and *Sugeno* is that the *Sugeno* output membership functions are either linear or constant.

A typical rule in a *Sugeno* fuzzy model has the form: If Input 1 = x and Input 2 = y and ..., then Output is $z =$

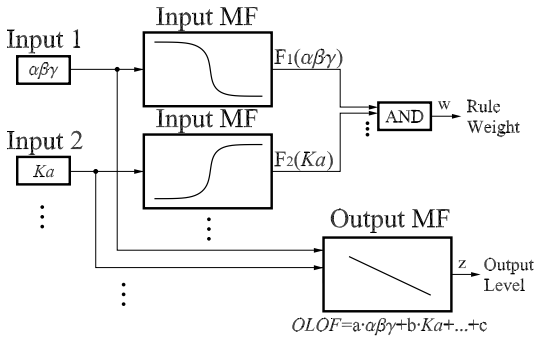

Fig. 2. Filter's operation diagram

$ax + by + c$.

For a zero-order *Sugeno* model, the output level z is a constant $(a = b = 0)$.

The output level z_i of each rule is weighted by the firing strength w_i of the rule. For example, for an AND rule with Input 1 = $\alpha\beta\gamma$,Input 2 = Ka, ... the firing strength is:

$w_i = AndMethod\big(F_1(\alpha\beta\gamma), F_2(Ka), ...\big)$

where $F_{1,2,...}(.)$ are the membership functions for Inputs 1, 2, ...

The final output of the system is the weighted average of all rule outputs, computed as:

$$\text{FinalOutput} = \frac{\sum_{i=1}^{N}(w_i z_i)}{\sum_{i=1}^{N}(w_i)}$$

A Sugeno rule operates as shown in figure 2.

III. *OLOF* FILTER

We have developed a new filter that has not a constant model of object movement, the model is estimated depending on the speed, acceleration and jerk using these simple expressions:

$$v_k = \frac{x_{k-1} - x_k}{T}; a_k = \frac{v_{k-1} - v_k}{T}; J_k = \frac{a_{k-1} - a_k}{T}$$

Depending on these values, we apply an specific combination of filters presented in section II-B.

We have used the decreasing bounce of a ball with the ground to design and test the proposed filter. Authors consider that the bounce of a ball behavior contains a wide representation of how an object trajectory can be (including changes of velocity and acceleration).

In figure 3, 4 and 5 we can see the position of the object (ball) that will allow us to take conclusions of its behavior (the data of these figures has been obtained with a set of expressions that models the system behavior, these expressions are not included in this paper).

In figure 3 we can see the prediction of filters presented in section II-B. We can see in figures 4 and 5 that for different

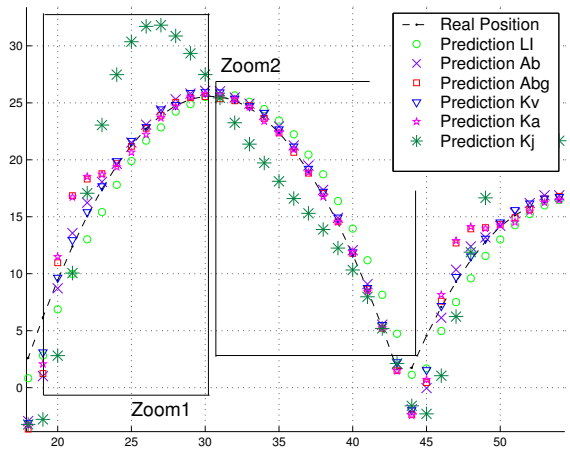

Fig. 3. Bounce of the ball on the ground

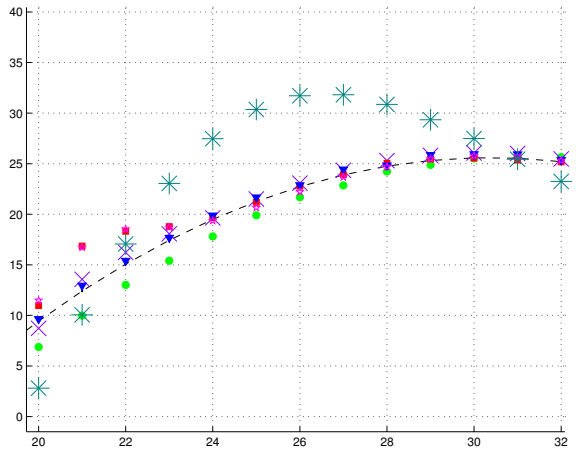

Fig. 4. Zoom 1

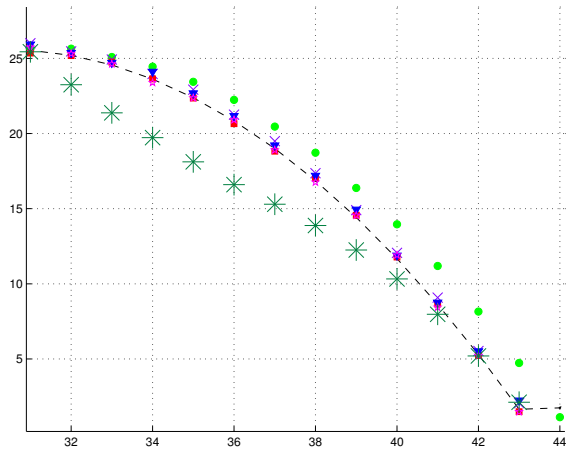

Fig. 5. Zoom 2

TABLE I

SCHEMATIC CLASSIFICATION OF THE SYSTEM BEHAVIOR WEIGH

Intermediate zone	
Places with high acceleration values	
Acceleration increasing	Acceleration decreasing
$case_1$	$case_2$
Places with low acceleration values	
Starting (first samples)	Ending (other samples)
$case_3$	$case_4$
Initial zone	**Final zone**
$case_5$	$case_6$

$$case_1 = 0.25 \cdot LI + 0.25 \cdot Kv + 0.25 \cdot Ka + 0.25 \cdot Kj$$
$$case_2 = 0.3 \cdot LI + 0.7 \cdot Kv$$
$$case_3 = 0.3 \cdot LI + 0.7 \cdot Kv$$
$$case_4 = 0.2 \cdot LI + 0.6 \cdot \alpha\beta\gamma + 0.2 \cdot Kv$$
$$case_5 = 0.2 \cdot LI + 0.6 \cdot \alpha\beta\gamma + 0.2 \cdot Kv$$
$$case_6 = 0.2 \cdot LI + 0.6 \cdot \alpha\beta\gamma + 0.2 \cdot Kv$$

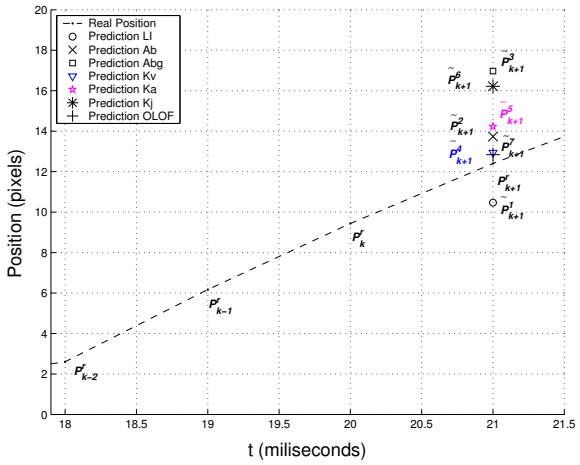

Fig. 6. Real position *vs* prediction

conditions (velocity and acceleration) of the object's trajectory one filter works better than the others, in other words, for specific values of velocity and acceleration we must apply an specific filter or filters and not others depending on dispersion values for each case. For example, for big values of acceleration, the best estimation is given by *Ka* filter but for small values $\alpha\beta\gamma$ filter works better.

Based experiments done, we can distinguish the system behavior in three different zones (places or conditions):

(I) Initial place: it is approximately the 5 first iterations, where all filters err a lot (not initialized), and where the best result corresponds with the filters that more quickly decreases the error. These requirements are acceptably obtained from filters *Kv*, $\alpha\beta$ and *LI*.

(II) Intermediate place: it is divided into two different places, acceleration picks and places between acceleration picks. For the first one, we distinguish two more cases, the increasing zone and the decreasing acceleration zone. For the increasing zone, the filters with a better behavior are *Kv*, $\alpha\beta$, *Kj* and *Ka*; for the decreasing zone, *Kv* and *LI* works better. For zones located between acceleration picks (small values of acceleration), we distinguish the initial values, that includes a maximum of 4 samples (time of filter update) or until the speed becomes negative (what has happened before), in this zone we consider the same than in decreasing zone (we considered the same behavior of the system). When this happens, error is stabilized in a value next to the average and the filters with better behavior in this case are $\alpha\beta\gamma$, *LI* and *Kv*.

(III) Final zone: the characteristics of this zone are a very small speed and a very small acceleration. It will be considered that the object is shaking if its speed is less than 1m/s and the filters with better behavior are *LI*, $\alpha\beta$ and *Kv*. Using this information, we can "mix" some filters to obtain the desired filter (for example, we can see in figure 5 that $\alpha\beta\gamma$ works very good, but in figure 4, this filters is bad near t=22. Using this empirical method and using figures 3, 4 and 5 we obtain coefficients shown in table I.

Constants presented in table I are empirically obtained, but we can use these values to initialize an optimization method to find the values that minimize the dispersion. So, we use a trust region algorithm (presented in section II-C) to find the local minimum closest to the values empirically obtained.

Once we have optimized all parameters [9], [10] and [11], we can establish the *Sugeno* type rules [7], [8] and [12] as we can see below (next we have the constants obtained after optimization process, which, are close to values shown in table I):

$case_1$: IF $i \geq 5$ AND acceleration IS high AND acceleration > 0 THEN $OLOF = 0.22 \cdot LI + 0.23 \cdot Kv + 0.26 \cdot Ka + 0.29 \cdot Kj$.

$case_2$: IF $i \geq 5$ AND acceleration IS high AND acceleration < 0 THEN $OLOF = 0.26 \cdot LI + 0.74 \cdot Kv$.

$case_3$: IF $i \geq 5$ AND $j < 4$ AND acceleration IS low THEN $OLOF = 0.33 \cdot LI + 0.67 \cdot Kv$.

$case_4$: IF $i \geq 5$ AND $j \geq 4$ AND acceleration IS low THEN $OLOF = 0.21 \cdot LI + 0.56 \cdot \alpha\beta\gamma + 0.23 \cdot Kv$.

$case_5$: IF $i < 5$ THEN $OLOF = 0.29 \cdot LI + 0.62 \cdot \alpha\beta\gamma + 0.09 \cdot Kv$.

$case_6$: IF $i \geq 5$ AND velocity IS low THEN $OLOF = 0.18 \cdot LI + 0.55 \cdot \alpha\beta\gamma + 0.27 \cdot Kv$.

where i is the iteration number of the visual control algorithm and j is the number of successive times that the acceleration is positive. This rules are based on the experience and knowledge of the filters behavior.

Once these rules are established, we have the new filter called *Off-Line Optimized Fuzzy* FILTER (*OLOF* FILTER), see figure 2.

IV. RESULTS WITH SIMULATED DATA

We show the effectiveness of the proposed method by the following simulation study. In this section we can find the most important results of these simulations. In figure 6 we can see positions P_k^r (actual object position), P_{k-1}^r (object position in $k-1$) and P_{k-2}^r (object position in $k-2$). Next real position of the object will be P_{k+1}^r and points from $\widetilde{P}_{k-2}^1$

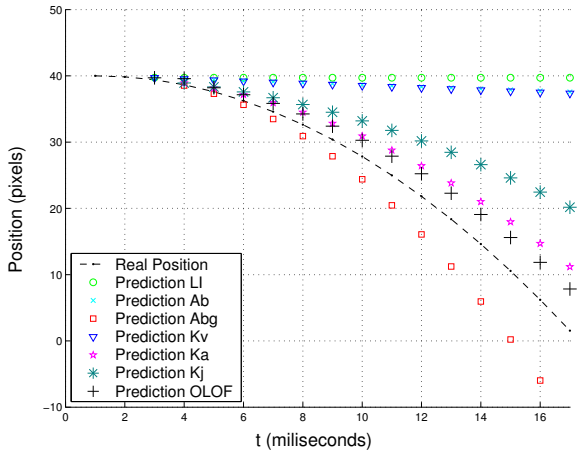

Fig. 7. Trajectory prediction

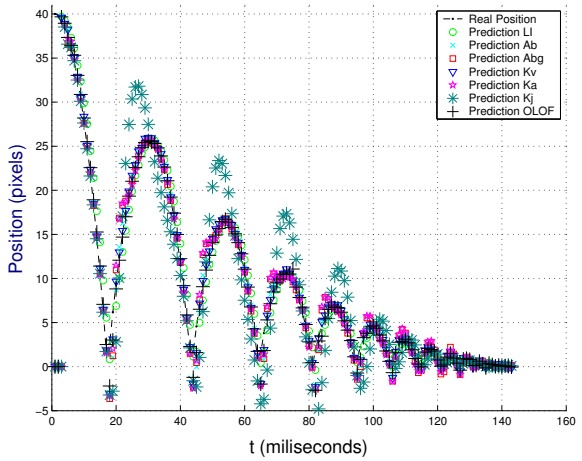

Fig. 8. Bounce of the ball with the ground

to $\widetilde{P}^7_{k-2}$ represents the prediction obtained for all filters. The best prediction is given by *OLOF* filter.

But really we require the trajectory prediction, not the prediction of the next position. In figure 7 we can see the real trajectory and the trajectory predicted by considered filters. For this experiment the best result is obtained by *OLOF* filter too.

We can consider the full trajectory of the ball (see figure 8) to compare the dispersion between filters. In table II, we can find the average dispersion for each filter. Again the best is the *OLOF* filter.

V. Computational load analysis

Obviously, the computational cost of any algorithm depends of the CPU power that is executing it and the used language (compiler effectiveness). For this reason, we have not made comparatives between them in seconds or milliseconds. We have made comparatives using one of them as a reference (the most used filter is the *Kalman* filter, so it is our reference filter. Time execution of *Kalman* filter with velocity model is the 100% of the reference time).

TABLE II

Numerical comparative for dispersion value of all filters implemented

Init. pos.	*LI*	$\alpha\beta$	$\alpha\beta\gamma$	*Kv*	*Ka*	*Kj*	*OLOF*
40	0.417	0.619	0.559	0.410	0.721	0.877	0.353
40(bis)	0.422	0.547	0.633	0.426	0.774	0.822	0.340
50	0.438	0.588	0.663	0.439	0.809	0.914	0.381
70	0.453	0.619	0.650	0.428	0.700	0.821	0.365
90	0.466	0.630	0.661	0.458	0.818	0.857	0.343
150	0.462	0.646	0.682	0.477	0.848	0.879	0.347

TABLE III

Comparison of computational load using as reference the *Kalman* filter with constant velocity model

LI	$\alpha\beta$	$\alpha\beta\gamma$	*Kv*	*Ka*	*Kj*	*OLOF*
22%	51%	65%	100%	133%	156%	188%

Filters $\alpha\beta$, $\alpha\beta\gamma$ and *LI* are faster than *Kalman* filter with constant velocity model, so their time execution are less than 100%. On the other hand, *Ka*, *Kj* and *OLOF* are slower algorithms, so their time execution is more than 100%.
We have explained in section III that the proposed filter (*OLOF* filter) is a linear combination of the others, so is the computational load of *OLOF* the sumatoria of *LI*(22%) + $\alpha\beta$(51%) + $\alpha\beta\gamma$(65%) + *Kv*(100%) + *Ka*(133%) + *Kj*(156%)? (see table III). Using efficient programming techniques we have obtained ***188%*** of computational cost for the proposed filter. This computational cost is smaller than the sum of the others because the filters have a great amount of common code and it was possible to take advantage of many calculations.

The slower filter is the proposed in this paper (*OLOF* filter) but only *OLOF* and *Kj* can predict trajectories with constant $jerk$ and in figure 3 we can see that only *OLOF* works good.

VI. Conclusions

In previous sections we can see the accuracy of the estimation of the new filter and the computational cost to obtain this estimation. Obviously the improvement supposes a greater computational load than the others, but compared with *Kj* it is not more than 17%, the reason being for relatively complex movements of the object, it would not suppose too much computational cost. Obviously, if the behavior of the object to follow is constant speed, the use of the proposed filter would not be recommended (the *Kv* and $\alpha\beta$ filters would work similar and its computational cost would be lower). The authors belive that the use of the proposed filter is to be recommended if the behavior of the object is unknown a priori and probably the object would have speed, acceleration and jerk changes. In this case, *LI*, $\alpha\beta$, $\alpha\beta\gamma$, *Kv* and *Ka* can't work properly and only *Kj* and *OLOF* filters can be used. If the target has a combination of several dynamics (general case) as we can see in the example used in this work (beam of a ball) the *OLOF* filter works much better than *Kj*.

VII. ACKNOWLEDGMENTS

The authors gratefully acknowledge the contribution of National Research Program of the Spanish Government and reviewers' comments.

REFERENCES

[1] Hutchinson, S., Hager, G. y Corke,P. "A tutorial on visual servo control". IEEE Trans. on Robotics and Automation, Vol. 12, No. 5, 1996, pp. 651-668

[2] Peter I. Corke "Visual Control of Robots: High Performance Visual Servoing", Research Studies Press, 1996. ISBN: 0 86380 207 9 - 353 pages

[3] E. Malis, F. Chaumette and S. Boudet. "$2\frac{1}{2}$D visual servoing". IEEE Trans. on Robotics and Automation, Vol. 15, 1999, pp. 234-246

[4] Mikhel E. Hawkins, "High Speed Target Tracking Using Kalman Filter and Partial Window Imaging", Thesis Presented to The Academic Faculty. George Woodruff School of Mechanical Engineering. Georgia Institute of Technology. April 2002

[5] X. Li and V. Jilkov, "A survey of maneuvering target tracking: Dynamic models," in SPIE: Signal and Data Processing of Small Targets 2000.

[6] Mehrotra, Kishore and Mahapatra, Pravas R (1997) "A Jerk Model for Tracking Highly Maneuvering Targets". IEEE Transactions on Aerospace and Electronic Systems 33(4):pp. 1094-1105.

[7] Wang, Li-Xin, "Course in Fuzzy Systems and Control Theory", Imprint: Pearson US Imports & PHIPEs. Publisher: Pearson Higher Education. Date Published: 4/06/1997

[8] Wang, Li-Xin, "Course In Fuzzy Systems and Control, A", ISBN: 0-13-540882-2. Publisher: Prentice Hall Copyright: 1997

[9] Gill, P. E.; W. Murray, W y Wright, M. H.: "Practical optimizacion" Academic Press. 1981.

[10] Rosenbrock, H. H., Comp. J.,"An automatic methodfor finding the greatestor least value of a function" (1960).

[11] A. R. Conn, K. Scheinberg, and Ph. L. Toint. "A derivative free optimization algorithm in practice". In Proceedings of the AIAA St Louis Conference, 1998.

[12] Sugeno, M., "Industrial applications of fuzzy control", Elsevier Science Publications Company, 1985.

[13] S. Soatto, R. Frezza and P. Perona,"Motion Estimation Via Dynamic Vision", IEEE Trans. Automatic Control, vol. 41, no.3, pp.393-413, Mar 1997.

[14] Z. Duric, J. A. Fayman and E. Rivlin, "Function From Motion", IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 18, no.6, pp.579-591, June 1996.

[15] J. Angeles, A. Rojas and C. S. Lopez-Cajun, "Trajectory Planning in Robotics Continuous-Path Applications", IEEE J. Robotics and Automation, vol. 4, no.4, pp.380-385, Aug 1988

[16] Z. Duric, E. Rivlin and A. Rosenfeld, "Understanding the Motions", Image and Computing, vol. 16, no.6, pp.785-797, 1998.

[17] Z. Duric, E. Rivlin and L. Davis, "Egomotion Analysis Based on the Frenet-Serret Motion Model", Proc. IEEE 4th Int. Conf. Computer Vision, pp.703-712, April 1993.

[18] Frank Vanden Berghen, "CONDOR: a constrained, non-linear, derivative-free parallel optimizer for continuous, high computing load, noisy objective functions", Ph. D. in the University of Brussels (ULB - Universit Libre de Bruxelles), Belgium. 2004.

[19] M.J.D. Powell, "UOBYQA: Unconsrained Optimization BY Quadratic Optimization", DAMTP 2000/NA14. Math. Program. 92B, 555582.