
DORIAN COJOCARU*, RAZVAN TUDOR TANASIE**
*Mechatronics Department, **Software Engineering Department
Faculty of Automation, Computers and Electronics
University of Craiova
Blvd. Decebal, Nr. 107, 200440, Craiova
ROMANIA

Abstract: This paper presents a project dealing with visual servoing applications. A general structure for the software system was developed starting from the ViSP architecture model [1]. Software modules were already implemented in beta versions: computer graphics procedures for robotic applications visualization and simulation and image processing procedures for visual servoing applications. Fuzzy algorithms and methods were used in order to support the implementation of the image processing procedures. The project is financed by Romanian National University Research Council CNCSIS.


1 Introduction
The main task of visual servoing is to control a dynamic system, as a robot, using the information obtained from digital images.

Usually, every visual application is implemented in a particular way based on his own hardware and software structures.

Developing this project, two main requirements were to be fulfilled:
- to create an open software architecture which allows an easy expansion by adding new modules,
- to create an open software architecture which allows working with different hardware architectures.

Virtual visual servoing applications were also considered for this project and, as consequence, a strong graphic support is needed.

In order to support these tasks, a modular structure was chosen (Fig. 1):
- a data acquisition module, including image acquisition,
- an image processing module for visual servoing, including features extraction,
- a computer graphic module for robotic application visualization and simulation,
- a visual servoing module, including robot control cinematic and dynamic laws and robot controller.

Regarding robot control, the research team designed and implemented a wide variety of control laws, both for usual robotic structure and for hyper redundant ones [2][3][4]. Following, fuzzy based functions for the Image Processing module and simulation functions for the Computer Graphics module will be presented.

Fig. 1 Visual servoing architecture
2 Image Processing Module

Fuzzy image processing is not a unique theory. It is a collection of different fuzzy approaches to image processing.

Fuzzy image processing has three main stages:
- image fuzzification,
- modification of membership values, and,
- if necessary, image defuzzification.

Typically applications ask the enhancement process to be capable of removing noise, smoothing regions where gray levels do not change significantly, and sharpening abrupt gray-level changes. Edge detection is a critical part of many computer vision systems. Ideally, edges correspond to object boundaries, and therefore edge detection provides a means of segmenting the image into meaningful regions. As a general concept, the job of the cluster analyze is the data partition into a number of groups, or clusters. Applying this partitioning operation on images, the image segmentation - a very important task for image processing - is obtained.

2.1 Fuzzy Image Enhancement Algorithm

This algorithm is based on Choi’s and Krishnapuram’s algorithm [5][6] who view image enhancement as replacing the gray level value of every pixel in the image with a new value depending upon the local information. If the local value region is relatively smooth, then the new value of the pixel may be a type of average of the local values. On the other hand, if the local region contains an edge or noise points, a different type of filtering should be used. So we need to create a bank of filters from which to choose the needed one. The choosing, unlike in crisp algorithms, need not be of only one of the filters but according to fuzzy logic each filter is used. So we need to create a bank of filters from which to choose the needed one. The choosing, unlike in crisp algorithms, need not be of only one of the filters but according to fuzzy logic each filter is used. So we need to create a bank of filters from which to choose the needed one. The choosing, unlike in crisp algorithms, need not be of only one of the filters but according to fuzzy logic each filter is

The FIRE system of rules used is:

If $M$ is mic, then use filter $B$
If $M$ is mare, then use filter $C$
Else use filter $A$

$M$ is a coefficient that is computed for each pixel and its value depends directly on the values of the pixels in the sliding matrix and indirectly, and consequently in a smaller proportion, on every pixel in the region. The membership functions for the linguistic labels “mare” and “mic” are defined on the domain of $M$ ($0 \leq M \leq 1$). They are fuzzy membership functions that take an exponential form. These functions depend on two parameters: $M$ and the image enhancement coefficient. Its value must be selected carefully considering the type of the image that is enhanced. In order to solve the if-then-else rules system, multiplication is selected as composition operator, minimum for aggregation, weighted average for the defuzzification function and the filter output for the consequent, so the following output is obtained:

$$I(X_i)=\frac{c_1 \cdot f_b + c_2 \cdot f_c + c_3 \cdot f_a}{c_1 + c_2 + c_3}$$

where: $I(X_i)$ = the gray-level of pixel $X_i$; $c_1 = mic (M)$; $c_2 = mare (M)$; $c_3 =1 - \max(c_1, c_2)$; $f_a$ = the output from filter A; $f_b$ = the output from filter B; $f_c$ = the output from filter C;

After applying the defuzzification process to the entire image, the enhanced image is obtained.

2.2 Fuzzy Edge Detection Algorithm

The algorithm used in this application is based on Russo and Ramponi algorithm [7]. This edge detector is relatively immune to noise and is based on the if-then-else paradigm (FIRE). In order to detect the edges gray-level differences in a 3x3 neighbourhood are used as inputs to the fuzzy rules. Let $X$ denote the central pixel in the window:

$$x_i=I(X)-I(X_0), \text{ for } i=1...8.$$ (2)

In this equation $I(X)$ represents the gray level in the central pixel while $I(X_i)$ represents the gray level of the $i^{th}$ pixel in the matrix. The variables $x_i$ are used in the antecedent clauses of the rule base. In the general case, there is a various number of rules in the system. Let $c_i$ be the overall degree of satisfaction of the $i^{th}$ “then” rule of the system. If $N_i$ is the number of input variables in the $i^{th}$ rule, $A_j(\ )$ is the membership function of the linguistic label $A_j$ and the mean operator is used for aggregation (that is the way it is used in this algorithm):
Letting $c_T$ denote the overall degree of satisfaction of the first $M$ “if” rules (the system has $M$ “if” rules and one “else” rule) and $c_E = c_{M+1}$ denote the degree of satisfaction of the “else” rule we have:

$$
\begin{align*}
c_T &= \max(c_m) \quad m = 1\ldots M \\
c_E &= c_{M+1} = 1 - c_T = 1 - \max(c_M) \quad m = 1\ldots M
\end{align*}
$$

The output $y$ is obtained by adding the effects induced by the “then” and “else” actions and by performing a suitable defuzzification. For this application, a simple edge detector was selected. It uses only one if rule. The FIRE system used is the following:

If ($x_2$ is zero) and ($x_4$ is zero)  
Then (make $y$ white) Else (make $y$ black)

The “zero” membership function is defined on the domain $[-G+1, G-1]$, where $G$ is the number of color levels. The function is a fuzzy membership function that takes an exponential form. It depends on two parameters: $x_i$ and the edge detection coefficient. As a consequence of the fact that the system has only one “if” rule, the values for $c_1$ and $c_2$ are:

$$
\begin{align*}
c_1 &= \frac{a_1 + a_2}{2} \\
c_2 &= 1 - c_1
\end{align*}
$$

where: $a_1$=$\text{zero}(x_2)$; $a_2$=$\text{zero}(x_4)$

The value of the $i^{th}$ pixel of the edge image is obtained from the formula:

$$
y = \frac{c_1 \cdot 255 + c_2 \cdot 0}{c_1 + c_2} = c_1 \cdot 255
$$

2.3 Fuzzy Image Segmentation Algorithm

There are many techniques [8] of fuzzy image segmentation. Fuzzy C-means is an algorithm based on one of the oldest segmentation methods which allows data to have membership of multiple clusters, each to varying degrees [9][10]. The algorithm is based on minimization of the following function:

$$
J_m = \sum_{i=1}^{N} \sum_{j=1}^{C} u_{ij}^m ||x_i - c_j||^2 
$$

where: $m$ is any real number greater than 1; $u_{ij}$ is the degree of membership of $x_i$ in the cluster $j$; $x_i$ is the $i^{th}$ of d-dimensional measured data; $c_j$ is the d-dimension center of the cluster; $||*||$ is any norm expressing the similarity between any measured data and the center. This algorithm [11] realizes an iterative optimization of the $J_m$ function, updating membership $u_{ij}$ and the cluster centers $c_j$ using the following formulas:

$$
u_{ij} = \frac{1}{\frac{1}{2} \sum_{k=1}^{C} \left( \frac{||x_i - c_k||}{||x_i - c_j||} \right)^{m-1}}
$$

$$
c_j = \frac{\sum_{i=1}^{N} u_{ij}^m x_i}{\sum_{i=1}^{N} u_{ij}^m}
$$

The minimization of $J_m$ is achieved only when the $u_{ij}$ function saturates:

$$
\max_{ij} \left\{ \left| u_{ij}^{(k+1)} - u_{ij}^{(k)} \right| \right\} < \varepsilon
$$

where $\varepsilon$ is a number between 0 and 1, and $k$ is is the iteration step.

The algorithm [12] has the following steps:

- Consider a set of $n$ data points to be clustered, $x_i$.
- Assume that the number of clusters, $c$, is known.
- Choose the level of cluster fuzziness, $m \in \mathbb{R} > 1$.
- Initialize the $(n \times c)$ sized membership matrix $U$ to random values such as $u_{ij} \in [0,1]$ and $\sum_{j=1}^{c} u_{ij} = 1$.
- Calculate cluster’s centers $c_j$ using (9) for $j=1..c$.
- Calculate the distance measures $d_{ij} = ||x_i - c_j||$, for all clusters $j=1..c$ and data points $i=1..n$.
- Update the fuzzy membership matrix $U$.

- If $d_{ij} > 0$ then $u_{ij} = \left[ \frac{c}{\sum_{k=1}^{c} \left( \frac{d_{ij}}{d_{ik}} \right)^m} \right]^{\frac{1}{m-1}}$
  
- If $d_{ij} = 0$ then the data point $x_i$ coincides with the cluster center, and full membership can be set $u_{ij} = 1$.
- Repeat from 5th step until the change in $U$ is less than a given tolerance, $\varepsilon$.

2.4 Module Implementation

The module was implemented in Microsoft Visual C++ from the package Microsoft Visual Studio .NET. In order to test the application, a user interface
was developed. It has a menu from which the user can select the desired function and a toolbar from which images can be opened, saved or the about dialog can be also opened.

3 Simulation Module

In this paper it will be presented, as an example, an algorithm for the simulation of the rotation joint for a walking robot. The robotic movement can be regarded as a sequence of positions. If these positions are considered states, the movement can be compared to a state machine. The first and most important step is to determine correctly these states. If this stage is not done properly, the entire simulation can be compromised, because it may lose the realistic factor.

3.1 Determine the States

The actual movement, the transition between the states, is accomplished by using time dependent matrices which represent the transformations applied to the model. The transformations needed for the actual movement are mostly rotations, some translations and scaling may be required in order to position the model in the initial scene.

The test model used for this application is a simple one – a part of a like a human robot. It consists of 5 cylinders: 1 for the body, 2 for the upper part of the legs and 2 for the lower part of the legs. They are not perfectly joined together in order to permit the complete observations of the movement of each part. Also the joints are invisible, the linked parts are bound together by applying the appropriate transformations in order to simulate the existence of joints. 14 states were identified. The movement can be split in three stages:

- **Stage 1.** Starting the movement. The robot is standing still and starts moving.
- **Stage 2.** Motion in progress. The character is moving and continues to move.
- **Stage 3.** Stopping the movement. The character is in motion and stops.

Considering this classification, the motion can be regarded as the following sequence:
- Start the movement. Apply stage 1 once.
- Moving. Apply stage 2 repeatedly.
- Stop moving. Apply stage 3 once, selecting the appropriate case.

The significance of each state is (figure 2):
- **State 0** – not moving. This state is the initial state and the only final steady state;
- **State 1** – start moving by lifting the right foot;
- **State 2** – puts forward the right foot;
- **State 3** – steps forward on the right foot;
- **State 4** – lifts the left foot from behind;
- **State 5** – moves forward the left foot;
- **State 6** – straightens the left foot;
- **State 7** – steps forward on the right foot;
- **State 8** – lifts the right foot from behind;
- **State 9** – moves forward the right foot;
- **State 10** – straightens the right foot;
- **State 11** – steps forward on the right foot;
- **State 12** – prepares to stop by bringing the left foot near the right one not moving;
- **State 13** – prepares to stop by bringing the right foot near the left one.

Also some states may seem similar they are in fact different from one another due to the situation at that moment and to the past state. For instance, state 12 is different from state 4 because state 4 occurs in full motion, while state 12 appears only when the character prepares to halt.

3.2 State Transitions

As it can be observed from Figure 2, there are 17 possible transitions. Only one of them will be presented, the others are similar.

First of all is important to present some special vectors and matrices that will be used during the algorithm.
- **Vector pozA** – represent the coordinates of the center of the upper circle of the cylinder that represents the upper part of the left foot;
- **Vector pozB** – represent the coordinates of the center of the upper circle of the cylinder that represents the lower part of the left foot;
- **Vector pozC** – represent the coordinates of the center of the upper circle of the cylinder that represents the upper part of the right foot;
- **Vector pozD** – represent the coordinates of the center of the upper circle of the cylinder that represents the lower part of the right foot;
- **Matrices M0, M1, M2, M3, M4** – the overall transformation matrix for the body and the right's and the left's leg upper and lower part of the robot respectively;
Matrix Min – the overall transformation matrix for the initial positioning and scaling of the robot in the scene.

All the vector coordinates are expressed in model space. The body will also be called part 0, the upper part of the right leg – part 1, the lower part of the right leg – part 2, upper part of the left leg – part 3, the lower part of the left leg – part 4. In order to accomplish the movement, two more variables must be set: \( \alpha \) - the angle with which the leg is rotated and time - the period of time for a stage. Only one state transition will be presented in this paper, the other transitions being similar.

3.2.1 Transition state 0 \( \rightarrow \) state 1
This transition affects only 2 parts of the robot body:
- Part 1 – rotation around the modified pozC with the specified angle in the specified period of time;
- Part 2 – in order to keep the link with the upper part, this part’s transformation is composed of two transformations:
  - rotation around the modified pozC with the specified angle in the specified period of time;
  - rotation around the modified pozD with \( -2 \cdot \alpha \) in the specified period of time.

The transformation for part 1 is also a composed one. In order to rotate part one around the modified pozC, the next transformations must be performed:
- transform the coordinates of pozC in order to indicate the current position of the center of the upper circle of the cylinder that represents the upper part of the right foot;
- translate the part in the origin (translation with - poz) in order to rotate it around X axis (the movement is done along Z axis);

\[
\begin{bmatrix}
\mathbf{x}\text{.poz} \\
\mathbf{y}\text{.poz} \\
\mathbf{z}\text{.poz} \\
1
\end{bmatrix}^T =
\begin{bmatrix}
\mathbf{x}\text{.pozC} \\
\mathbf{y}\text{.pozC} \\
\mathbf{z}\text{.pozC} \\
1
\end{bmatrix}^T \cdot
\begin{bmatrix}
R_{l00} & R_{l01} & R_{l02} & 0 \\
R_{l10} & R_{l11} & R_{l12} & 0 \\
R_{l20} & R_{l21} & R_{l22} & 0 \\
T_{l00} & T_{l01} & T_{l02} & 1
\end{bmatrix}
\]

where \( Ml = \begin{bmatrix} R1 & 0 \\ T1 & 1 \end{bmatrix} \)

\[
Ml = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
- \mathbf{poz}\_x & - \mathbf{poz}\_y & - \mathbf{poz}\_z & 1
\end{bmatrix}
\]

- rotate it around X axis with \( \alpha \) over the period of time. In order to accomplish this, the rotation angle for a frame is \( \alpha \cdot \Delta / \text{time} \), where \( \Delta \) is the period of time since the last frame was rendered;

\[
\Delta = \sum \, \Delta \cdot \text{time}
\]

\[
\begin{bmatrix}
1 & 0 & \cos(\alpha \cdot \Delta / \text{time}) & -\sin(\alpha \cdot \Delta / \text{time}) & 0 \\
0 & 1 & \cos(\alpha \cdot \Delta / \text{time}) & \sin(\alpha \cdot \Delta / \text{time}) & 0 \\
0 & 0 & 1 & 0 & 0
\end{bmatrix}
\]

\[
M = Ml \cdot Ml1 \cdot Ml
\]

The transformation for part 2 is also composed of a number of similar transformations.

3.2.2 Time problem
One more problem needs to be taken into consideration [13]. As stated previously, the transition between states, implicitly the entire movement is realized based on time. For each transition, a certain period of time is allocated. Generally this period is the same for all transitions, except for the transitions that do not affect any transformation matrix, where the allocated period of time is zero. This period of time was previously called time.

In order to keep track of the period of time “spent” by each state transition at a certain moment of time another variable is used which will be called spent time. When each frame is rendered the following operation takes place:

\[
\text{spent\_time} = \text{spent\_time} + \Delta
\]

where \( \Delta \) represents the time passed since the previous frame was rendered.

Here a problem may occur. The time spent by each frame is in fact the sum of all \( \Delta \)s of that frame. Knowing that a certain transformation is divided between frames by splitting its effect to be equal to \( \text{transformation} \cdot \Delta / \text{time} \), it means that the total effect of the transformation over all the state transition frames is equal to \( \text{transformation} \cdot \sum \Delta / \text{time} \). But this equation needs to be equal to transformation, which implies the fraction must be one.

That means \( \sum \Delta = \text{time} \). Before each application of equation 19, the following test should be applied:
If spent_time + \Delta > time \\
Then \Delta = time - spent_time \\
After each frame rendering it is tested if the time allocated for that stage passed, that is: \\
If spent_time = time \\
Then Move to next stage \\
Else Continue stage

3.3 Module Implementation

Time is an essential aspect in graphics simulation applications (animations), because most of the transformations are time dependent. There are some problems regarding time that must be taken into consideration when constructing such an application:

- time spent on a transition;
- time influence in the transformations;
- time influence in each frame;
- exceeding the time allocated for a transition.

The algorithms were tested and implemented in a Microsoft Visual C++ .NET application. This program uses Microsoft DirectX 9.0c libraries' structures [13][14] (such as D3DXMATRIX) and functions (such as rotations, translations) in order to accomplish the desired implementation.

4 Conclusions

A general structure for a software system dedicated to visual servoing applications was developed starting from the ViSP architecture model. The project is financed by Romanian National University Research Council CNCSIS.

Software modules were already implemented in beta versions: computer graphics procedures for robotic application visualization and simulation and image processing procedures for visual servoing applications.

Fuzzy algorithms and methods were used in order to support the implementation of the image processing procedures. Further development of the project would be to add new tools and improving the already implemented ones by adding new rules to the FIRE systems or modifying the weights computing functions.

This paper presents also the methods to realize robot movement simulation and then focused on the technique based entirely on programming. In order to complete the algorithm, the problem was considered as a state transition one. The first step was to identify the states, select the begin and end states, followed by the determination of the links between them: state transitions, conditions for transitions, particular cases. As further development, a more complex simulation could be considered. Feet, arms, hands, neck and head could be added to the present model. For an even more realistic simulation, fingers, toes, in fact an entire human like robot, could be simulated.

From the general application point of view, the other modules of the system must be designed and implemented. Regarding robot control, the research team designed and implemented a wide variety of control low, both for usual robotic structure and for hyper redundant ones.

References: