Graphic Simulator for the Visual Control of Hyperredundant Robots

DORIAN COJOCARU*, RĂZVAN TUDOR TĂNASIE**
* Mechatronics Department, ** Software Engineering Department
University of Craiova
Bvd. Decebal, Nr. 107, 200440, Craiova, Dolj
ROMANIA
cojocaru@robotics.ucv.ro, http://robotics.ucv.ro/

Abstract: - Tentacle manipulators produce changes of configuration using a continuous backbone made of sections which bend. In this paper, a new project for controling a continuum robot is presented. The robotic structure uses cables to transmit forces to the elements of the arm. This new robot is actuated by stepper motors. Due to the difficulties of using distributed sensors along the manipulator structures, it was decided to use a vision system. The vision system is composed by two pan / tilt / zoom cameras and a frame grabber. A graphical simulator was designed and implemented in order to test the robot behavior under certain circumstances. The camera calibration using this graphic simulator is also presented. The term of "camera calibration" in the context of this paper refers to positioning and orienting the two cameras at imposed values.

Key-Words: - Hyperredundant robots, Visual Control, Graphic Simulator

1 Introduction

A tentacle manipulator is a hyper-redundant or hyper-degree-of-freedom manipulator and there has been a rapidly expanding interest in their study and construction lately. The control of these systems is very complex and a great number of researchers have tried to offer solutions for this difficult problem [8]. The inverse kinematics problem is reduced to determining the time varying backbone curve behavior. New methods for determining "optimal" hyper-redundant manipulator configurations based on a continuous formulation of kinematics are developed. The difficulty of the dynamic control is determined by integral-partial-differential models with high non-linearity that characterize the dynamic of these systems [4], [9].

In a paper presented at RAAD 2007, the problem of a class of tentacle arms of variable length was discussed [10]. First, the dynamic model of the system was inferred. The method of artificial potential was developed for these infinite dimensional systems. In order to avoid the difficulties associated with the dynamic model, the control law was based only on the gravitational potential and a new artificial potential. Servoing was based on binocular vision, a continuous measure of the arm parameters derived from the real-time computation of the binocular optical flow over the two images, and is compared with the desired position of the arm. The control error function was built in 3D Cartesian space by the visual information obtained by two cameras in two image planes. The two 2D errors obtained in the two image planes were determined by the two differences between the actual and desired continuous angle values that define the projections of the arm shape. The plane errors can be considered as the errors of the arm shape. These errors were used to calculate the spatial error and a control law was synthesized.

A tentacle arm prototype was designed and the practical realization is now running. It is a cable based mechanism having, in the first implementation, three segments. The tentacle arm is designed to be actuated by 3-phase stepper motors. 4-Axis Stepper Motion Controller boards are used. In order to implement the visual-servoing system, a benchmark was organized based on two color camera with 0.05lux low light sensitivity and the DT3162 frame grabber from Data Translation.

The cameras have motorized Pant/Tilt/Zoom (10x optical zoom) and are mounted in perpendicular planes offering the input for the frame grabber. The Pant/Tilt/Zoom precision is sufficient for this step of the application development (Pan: range +135, 10 50/sec; Tilt: range +90 45, 7 25 /sec; Zoom: 1x~10x optical zoom). Two white screens are placed in front of the cameras in order to increase the images contrast. The tentacle arm is placed between each camera and its screen.

The image processing tasks are performed using Global LAB Image2 from Data Translation. The robot control algorithms are implemented in a C++ program running on a Pentium IV PC. In order to facilitate the image feature extraction, a set of

markers are placed on joints along the backbone structure (see Fig. 1).

Fig. 1. a. Visual Control benchmark
b. "Old" experimental segment
c. CAD image of the tentacle design

2 Cameras Calibration

Two video cameras provide two images of the whole robot workspace. The two image planes are parallel with XOY and ZOY planes from robot coordinate frame, respectively.

The cameras provide the images of the scene that are stored in the frame grabber's video memory. Respective to the image planes are defined two dimensional coordinate frames, called screen coordinate frames or image coordinate systems. Denote X_{S_1}, Y_{S_1} and Z_{S_2}, Y_{S_2}, respectively, the axes of the two screen coordinate frames provided by the two cameras. The spatial centers for each camera are located at distances D_1 and D_2, with respect to the XOY and ZOY planes, respectively. The orientation of the cameras around the optical axes with respect to the robot coordinate frame, are noted by ψ and ϕ, respectively [5], [6], [7].

The control system is an image – based visual servo control where the error control signal is defined directly in terms of image feature parameters. The desired position of the arm in the robot space is defined by the curve C_d, or, in the two image coordinate frames $Z_{S_1}O_{S_1}Y_{S_1}$ and $Z_{S_2}O_{S_2}Y_{S_2}$, by the projection of the curve C.

The control problem of this system is a direct visual servo-control, but the classical concept of the position control, in which the error between the robot end-effector and target is minimized, is not used. In this application the control of the shape of the curve in each point of the mechanical structure is used [1], [2]. The method is based on the particular structure of the system defined as a "backbone with two continuous angles $\theta(s)$ and *q(s)*".

The control of the system is based on the control of the two angles $\theta(s)$ and *q(s)* (see Fig. 2).

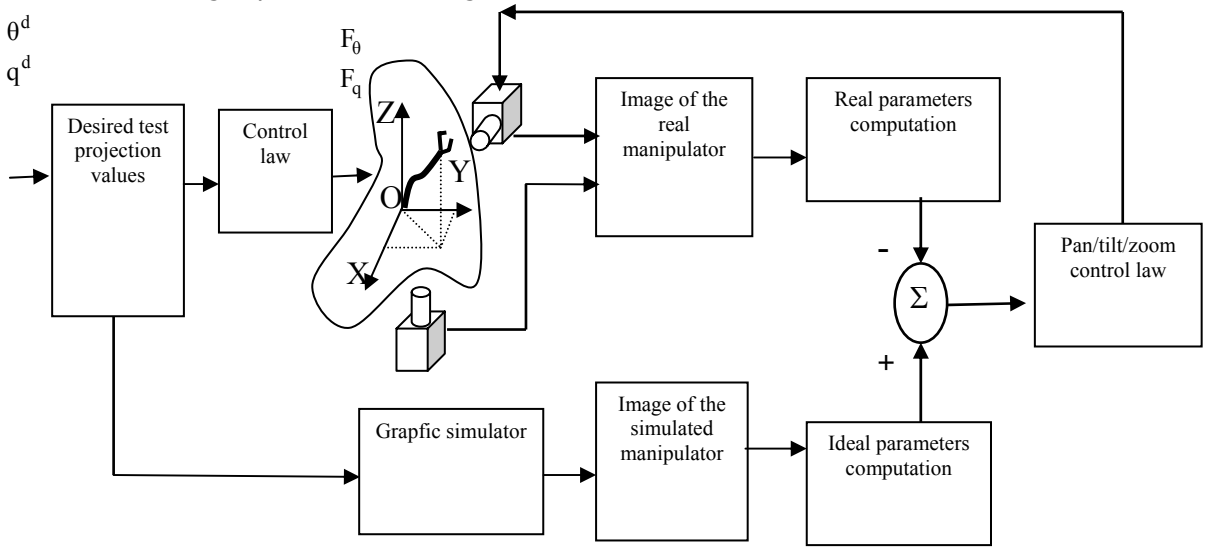

Fig. 2 Camera calibration system

These angles are measured directly or indirectly. The angle $\theta(s)$ is measured directly by the projection on the image plane $Z_{S_1}O_{S_1}Y_{S_1}$ and *q(s)* is computed from the projection on the image plane $Z_{S_2}O_{S_2}Y_{S_2}$.

A very important task in developing this application is to control the camera position and orientation [3]. From this point of view, the calibration operation assures that the two cameras' axes are orthogonal. In the beginning, the tentacle manipulator receives the needed commands in order to stand in a test pose (imposed position and orientation).

The same commands are sent to the Graphic Simulator. Two different sets of images are obtained: real images acquired by the real cameras and simulated images offered by the Graphic Simulator. From these two sets of images, two sets of parameters are computed: real parameters are computed from real images and, respectively, ideal parameters are computed from synthetic images.

Comparing the two sets of parameters and knowing the image/parameters behavior for the camera orientation, the cameras are orientated (pan/tilt/zoom) in order to minimize the error.

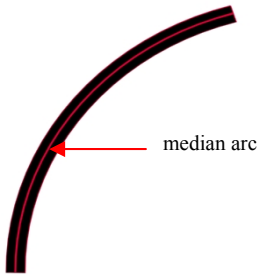

Fig. 3 An arched segment

3 Graphic Simulator

A graphical simulator was designed and implemented in order to test the robot behavior under certain circumstances.

3.1 Simulation

A graphical simulator was designed and implemented in order to test the robot behavior under certain circumstances. The simulator approximates the curved segments of the hyperredundant robot and considers constant the length of the median arc of each segment (Fig. 3).

To ease the presentation, the term segment will be used in all that follows referring to the median segment (arched or un-arched). For the arched segment, its median arc remains constant.

The inputs for the simulator are: robot configuration; robot initial position; control laws for each of the segments of the hyperredundant robot.

The robot configuration consists of the number of segments the hyperredundant robot has, the length of each segment and the angles that the cords make with the Ox axis. The arching angles are computed from these angles. An arching angle is defined as the angle made by the cord (determined by the ends of the arched segment) and the original un-arched segment.

For the direct kinematics problem, the control of the robot simulation is accomplished by giving the Ox angles for each of the segments in their final position and the output of the simulation is the hyperredundant robot's end-effector final position in the operation space.

In order to compute the final position of the end-effector and the hyperredundant robot's behavior during its motion, a few elements must be computed: the relation between the arching angle and the angle at centre determined by the arched segment (this angle determines the length of the arc); the cord length; the relation between an Ox angle and an arching angle; the final arching angles – recurrent set.

The computation of the relation between the arching angle and the angle at centre determined by the arched segment is determined by the following axiom:

Axiom 1: Consider a cylindrical segment of the hyperredundant robot described in the first part. The median line of the arched segment is tangent in its origin to the median line of the un-arched segment.

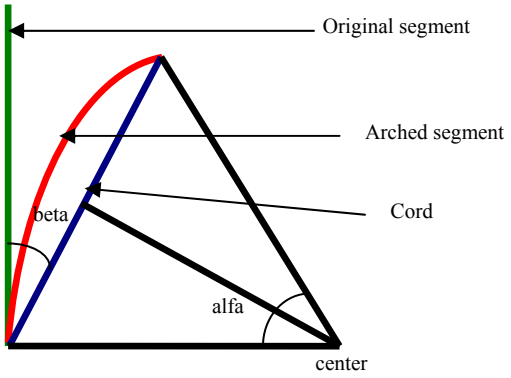

Fig. 4 Relation between arching angle and angle at center Legend*:* center – center of the circle, alfa – angle at the center, beta – arching angle.

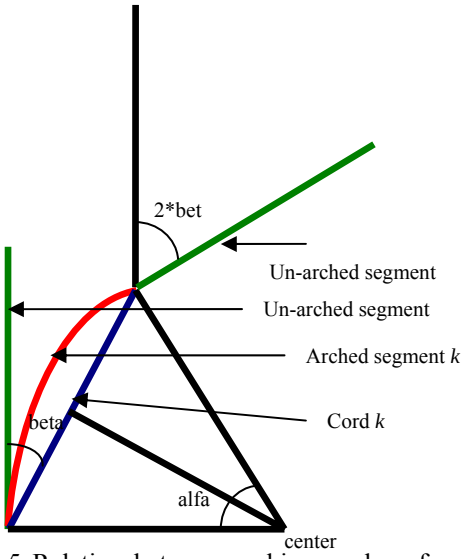

Fig. 5 Relation between arching angles of consecutive segments
Legend: center – center of the circle for segment k, alfa – angle at the center for segment k, beta – arching angle for segment k.

Based on Axiom 1, it results that the arching angle is given by:

$$\alpha_{arching} = \frac{\alpha_{center}}{2} \tag{1}$$

where: $\alpha_{arching}$ is the arching angle; α_{center} is the circle angle at centre (see Fig. 4).

Considering that the length of a circle arc is:

$$l_{arc} = \alpha_{center} * R \tag{2}$$

and the length of a circle cord is:

$$l_{cord} = 2 * R * \sin\left(\frac{\alpha_{center}}{2}\right) \tag{3}$$

where: l_{arc} is the length of the arc, l_{cord} is the length of the cord, R is the circle radius.

For a robot that has only rotation joints, the Ox angle increases (or decreases, depending on the selected positive direction) for each segment with the sum of rotation angles of each of the previous segments (including the current segment). This is true because the orthogonal system attached to the ith segment is obtained from its initial position and applying all the anterior transformations (the rotations of the segments 1 to i):

$$O^i = \prod_{k=i}^{0} Rot^k * O^0 \tag{4}$$

where: O^i is the column vector defining the position of the ith segment's origin; Rot^k is the rotation matrix of the kth segment.

For a hyperredundant robot the things are different (see Fig. 5). The arching angle is double the sum of each previous arching angle plus the current arching angle, because the un-arched segment is a prolongation of the previous segment. It yields that the Ox angle of a segment is given by:

$$\alpha_{Ox}^n = \alpha_{Ox}^0 - 2 * \left(\sum_{k=0}^{k=n-1} \alpha_{arching}^k \right) - \alpha_{arching}^n \tag{5}$$

where: α_{Ox}^i is the angle the ith segment makes with Ox axis; $\alpha_{arching}^i$ is the arching angle of the ith segment.

From Eq. (1), (2), (3) and considering that the arc length remains constant (and equal to the segment length), it yields:

$$l_{cord}^n = l_{arc}^n * \frac{\sin\left(\alpha_{arching}^n\right)}{\alpha_{arching}^n} \tag{6}$$

Using Eq. (6), the coordinates of the base points of each segment are computed:

$$x^n = x^{n-1} + l_{cord}^{n-1} * \cos\left(\alpha_{Ox}^{n-1}\right) \tag{7}$$

$$y^n = y^{n-1} + l_{cord}^{n-1} * \sin\left(\alpha_{Ox}^{n-1}\right) \tag{8}$$

where: x^n is the x coordinate of the nth segment's base point; y^n is the y coordinate of the nth segment's base point.

Eq. (5), (6), (7) and (8) yield through iterative

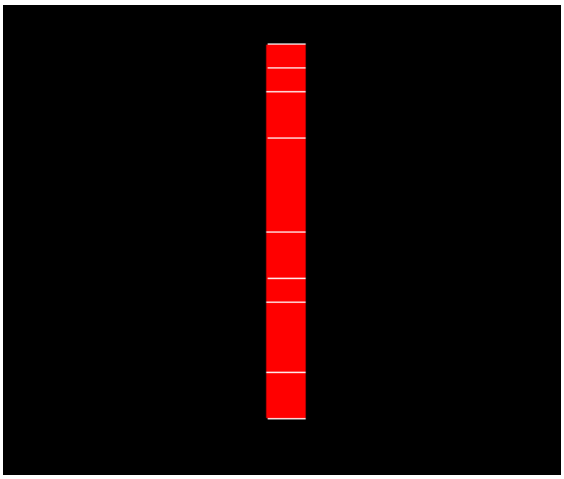

Fig. 6 The initial position of the hyperredundant robot.

calculus:

$$x^n = x^0 + \sum_{i=0}^{n-1} l_{arc}^i * \frac{sin\left(\alpha_{arching}^i\right)}{\alpha_{arching}^i} * \quad (9)$$
$$* cos\left(\alpha_{Ox}^0 - 2 * \left(\sum_{k=0}^{k=n-1} \alpha_{arching}^k\right) - \alpha_{arching}^n\right)$$

$$y^n = y^0 + \sum_{i=0}^{n-1} l_{arc}^i * \frac{\sin\left(\alpha_{arching}^i\right)}{\alpha_{arching}^i} * \quad (10)$$
$$* \sin\left(\alpha_{Ox}^0 - 2 * \left(\sum_{k=0}^{k=n-1} \alpha_{arching}^k\right) - \alpha_{arching}^n\right)$$

Once the values for the basis points are computed, an interpolation must be used in order to create the intermediate points that describe the arched segments.

3.2 Results

Many experiments were realized in order to test the simulator's behavior under different input conditions (different number of segments for the hyperredundant robot, different segments lengths, different final arching angles). One representative experiment is presented.

The hyperredundant robot has 8 segments, the base of the robot is situated in the origin (0,0), the segments have lengths of 2, 3, 1, 2, 4, 2, 1, 1 respectively and initially the robot is linear, aligned along the Oy axis (see Fig. 6).

In the final configuration, the cords make the following angles with Ox axis:

- segment 1: $0.55 * \pi$;
- segment 2: $0.65 * \pi$;
- segment 3: $0.8 * \pi$;
- segment 4: π ;
- segment 5: $1.15 * \pi$;
- segment 6: $0.9 * \pi$;
- segment 7: $0.8 * \pi$;
- segment 8: $0.65 * \pi$.

Fig. 7 presents and intermediate position acquired during the simulation, while Fig. 8 presents the final (obtained) position and the coordinates of the end-effector for this situation – E(-10.738, 5.201).

The simulator behaved correctly under all test conditions, creating a realistic motion simulation.

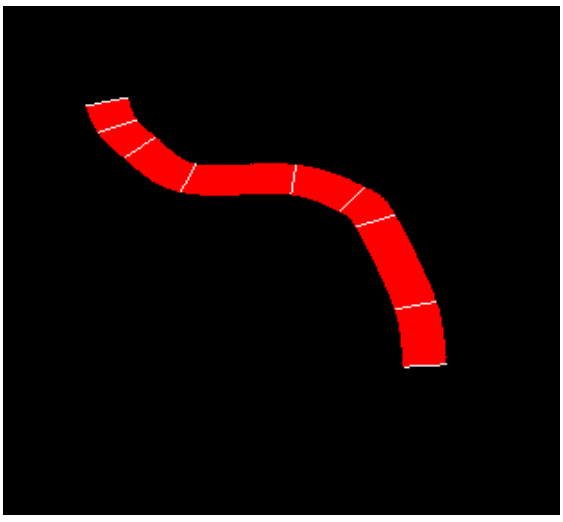

Fig. 7 An intermediate position of the hyperredundant robot.

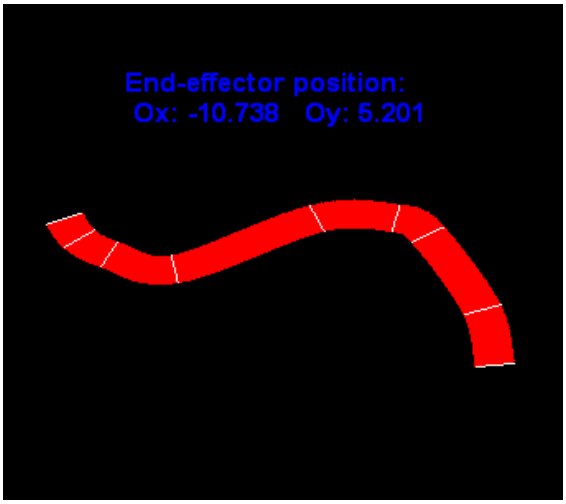

Fig. 8 The final position of the hyperredundant robot.

3.3 Implementation

The simulator was implemented in Microsoft Visual C++ .NET 2005 [13], [14]. It used the Microsoft DirectX SDK library for graphical purposes [12]. In order to simulate the circular arched segments a series of intermediate points (that are connected by lines) between the segment origins must be determined. The Catmull-Rom interpolation algorithm [11] was used for this simulator because it was need an interpolation algorithm that passes through the control points. Catmull-Rom splines are a family of cubic interpolating splines formulated such that the tangent at each point p_i is calculated using the previous and next point on the splines, $\tau(p_{i+1} - p_{i-1})$.

The geometry matrix is given by:

$$p(s)=\begin{pmatrix}1\\u\\u^2\\u^3\end{pmatrix}^T * \begin{pmatrix}0 & 1 & 0 & 0\\ -\tau & 0 & \tau & 0\\ 2*\tau & \tau-3 & 3-2*\tau & -\tau\\ -\tau & 2-\tau & \tau-2 & \tau\end{pmatrix} * \begin{pmatrix}p_{i-2}\\p_{i-1}\\p_i\\p_{i+1}\end{pmatrix} \quad (11)$$

where the parameter τ is known as "tension" and it affects how sharply the curve bends at the (interpolated) control points.

Designing and implementing algorithms to also solve inverse kinematics problems, designing and implementing algorithms that consider the dynamic components in solving direct and inverse kinematics problems are other features of the Graphic Simulator.

3.4 Calibration

Taking into account the presented structure of the tentacle - vision system, in order to apply the tested visual servoing algorithm, the two cameras must be positioned and oriented as:
- both focus on the robot,
- their axes are orthogonal,
- both have the same zoom factor.

Two different algorithms were implemented:
- one uses the graphical simulator.
- other uses a cylindrical etalon.

First, the algorithm working together with the graphic simulator will be presented.

It was proven that the two camera axes are orthogonal if, when both cameras are looking at the tentacle successively bended as circle's arcs in two orthogonal planes, are seeing also two circle's arcs.

The previous condition is fulfilled if each camera looks at the center of the circle containing the arc and the view line is orthogonal on the plane's circle (see Fig. 9).

Three calibration steps must be performed:

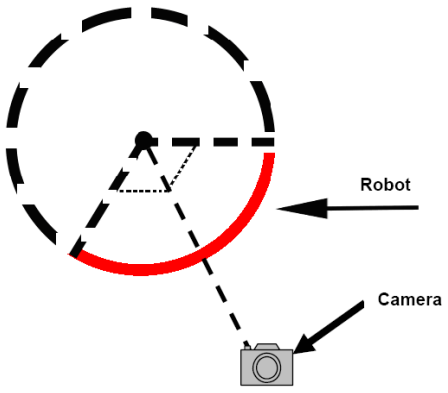

Fig. 9 Camera calibration

- Horizontal calibration - positioning and orienting the camera horizontally (pan),
- Vertical calibration - positioning and orienting the camera vertically (tilt),
- Zoom calibration - tuning the two cameras as both look at the robot from virtual equal distances.

The calibration algorithm has the following steps:

1. The graphic simulator computes the necessary bending and rotating arcs in order to transform the tentacle into an arc in the XOY plane.
2. The robot's control commands are generated in order to achieve the imposed positions and orientations.
3. The first camera acquires an image.
4. The image is segmented.
5. If the detected curve (the median axis of the tentacle) verifies the circle's equation then the algorithm continues from step 7.
6. Else the first camera is moved and algorithm returns to the step 3.
7. The graphic simulator computes the necessary bending and rotating arcs in order to transform the tentacle into an arc in the ZOY plane.
8. The robot's control commands are generated in order to achieve the imposed positions and orientations.
9. The second camera acquires an image.
10. The image is segmented.
11. If the detected curve (the median axis of the tentacle) verifies the circle's equation then the algorithm continues from step 13.
12. Else the second camera is moved and algorithm returns to the step 9.
13. The optical correlation for the equalization of "virtual" zoom factors is performed.

How to "move" the camera in the steps 6 & 12? The image behavior in accordance with camera's movements was studied. The effect of pan and tilt rotations on two points placed in a quadratic position on a circle was geometrically described. Matrix for coordinate transformations corresponding to rotations with pan and tilt angles, respectively for perspective transformation were used.

The following plot shows how is transformed a rectangle (inscribed in the circle and having the edges parallel with the axes OX and OY) when a tilt rotation is performed (Fig. 10).

Theoretically, by zooming, the distance between the two points varies in a linear way, as it is in Fig. 11.

The variation of the distance between the two points, placed in a quadratic position on the circle, and the centre of the circle, depending of the tilt angle X, are plotted in Fig. 12.

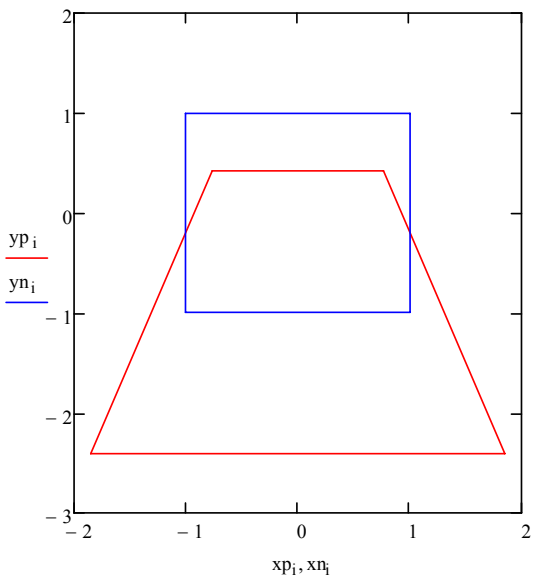

Fig. 10

Theoretically, by zooming, the distance between the two points varies in a linear way, as it is shown in Fig. 13.

Second, the algorithm using a cylindrical etalon will be presented.

Special starting conditions were imposed in order to support the image processing tasks (Fig. 14):

- white background,
- dark grey cylinder,
- red vertical equidistant (90 degrees) axes,
- friendly initial camera's positions and orientations,
- zoom x1.

Three succesive and dependent calibrations are performed:

- Horizontal (pan): position and orientation are obtained in two successive, but dependent steps.
- Vertical (tilt): position and orientation are obtained in two successive, but dependent steps.
- Zoom: tuning the two cameras as both look to the cylinder from virtual equal distances.

Horizontal orientation:

1. Image acquisition & segmentation using histogram threshold,

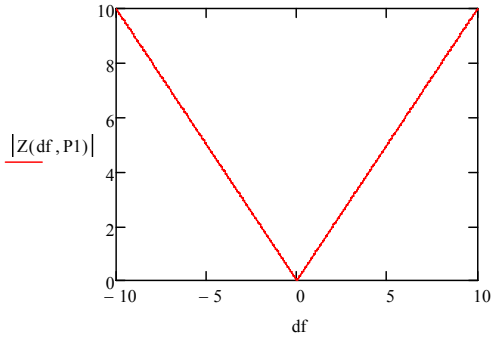

Fig. 11

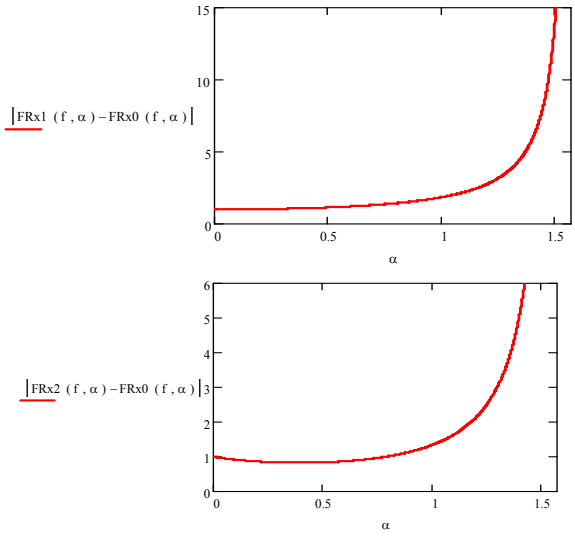

Fig. 12

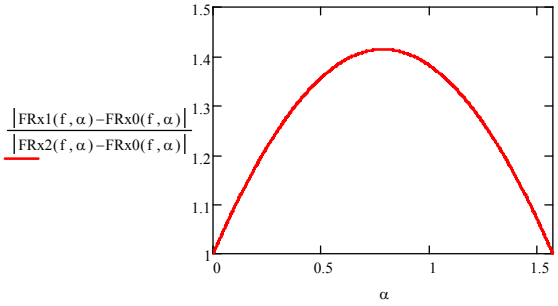

Fig. 13

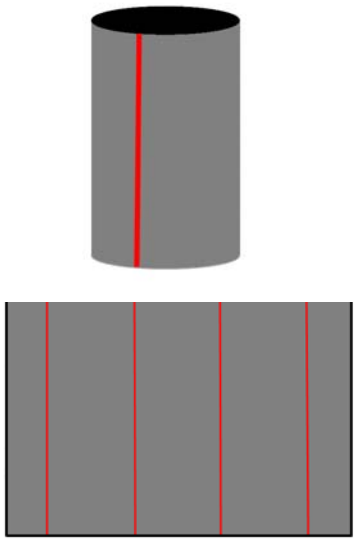

Fig. 14 Cylindrical etalon

2. Counting for each line in the image the number of black and respectively white pixels,
3. For the line with the greater number of black pixels, compute p_{st}^{oriz} and p_{dr}^{oriz}, the number of white pixels from the left and respectively from the right,

4. Compute the offset $\Delta^{oriz} = \left| p_{st}^{oriz} - p_{dr}^{oriz} \right|$,

5. IF $\Delta^{oriz} \le \varepsilon_{accept}^{oriz}$ THEN stop rotating the camera,

6. ELSE rotate the camera to the left or to the right using $\text{sgn}\left(p_{st}^{oriz} - p_{dr}^{oriz}\right)$,

7. Back to begin, 1.

Horizontal positioning:

1. Image acquisition & segmentation using histogram threshold,

2. Scanning the image horizontally, the axes are detected; if two axes are detected then the camera is rotated so the most lateral axes goes outside the image,

3. Counting for each line in the image the number of black and respectively white pixels,

4. For each line, the number of white pixels on the left of the axes is $p_{st}^{oriz-poz}$ and the number of white pixels on the right of the axes is $p_{dr}^{oriz-poz}$,

5. The horizontal offset is computed $\Delta^{oriz-poz} = \left| p_{st}^{oriz-poz} - p_{dr}^{oriz-poz} \right|$,

6. IF $\Delta^{oriz-poz} \le \varepsilon_{accept}^{oriz-poz}$ THEN stop translating the camera,

7. ELSE translate the camera to the left or to the right using $\text{sgn}\left(p_{st}^{oriz-poz} - p_{dr}^{oriz-poz}\right)$,

8. Back to begin, 1.

The full horizontal calibration is performed in the following steps:

1. Horizontal orientation,

2. Horizontal positioning,

3. The offset for horizontal orientation is computed $\Delta^{oriz} = \left| p_{st}^{oriz} - p_{dr}^{oriz} \right|$,

4. IF $\Delta^{oriz} \le \varepsilon_{accept}^{oriz}$ THAN the algorithm stops,

5. ELSE it goes to step 1.

Both offsets must be under the accepted thresholds. Else, the positioning destroyed the orientation and the procedure must be repeated.

Vertical orientation:

1. Image acquisition & segmentation using histogram threshold,

2. Counting for each column in the image the number of black and respectively white pixels,

3. For the column with the greater number of black pixels, compute p_{sus}^{vert} and p_{jos}^{vert} the number of white pixels above and respectively under the cylinder,

4. Compute the offset $\Delta^{vert} = \left| p_{sus}^{vert} - p_{jos}^{vert} \right|$,

5. IF $\Delta^{vert} \le \varepsilon_{accept}^{vert}$ THEN stop rotating the camera,

6. ELSE rotate the camera up and down using $\text{sgn}\left(p_{sus}^{vert} - p_{jos}^{vert}\right)$,

7. Back to begin, 1.

Vertical positioning:

1. Image acquisition & segmentation using histogram threshold,

2. Scanning the image vertically up-down until the object is detected,

3. Scanning the image vertically down-up until the object is detected,

4. For each of this two lines the number of black pixels is computed, $p_{\text{inf}}^{vert-poz}$ and $p_{\text{sup}}^{vert-poz}$,

5. The vertical offset is computed

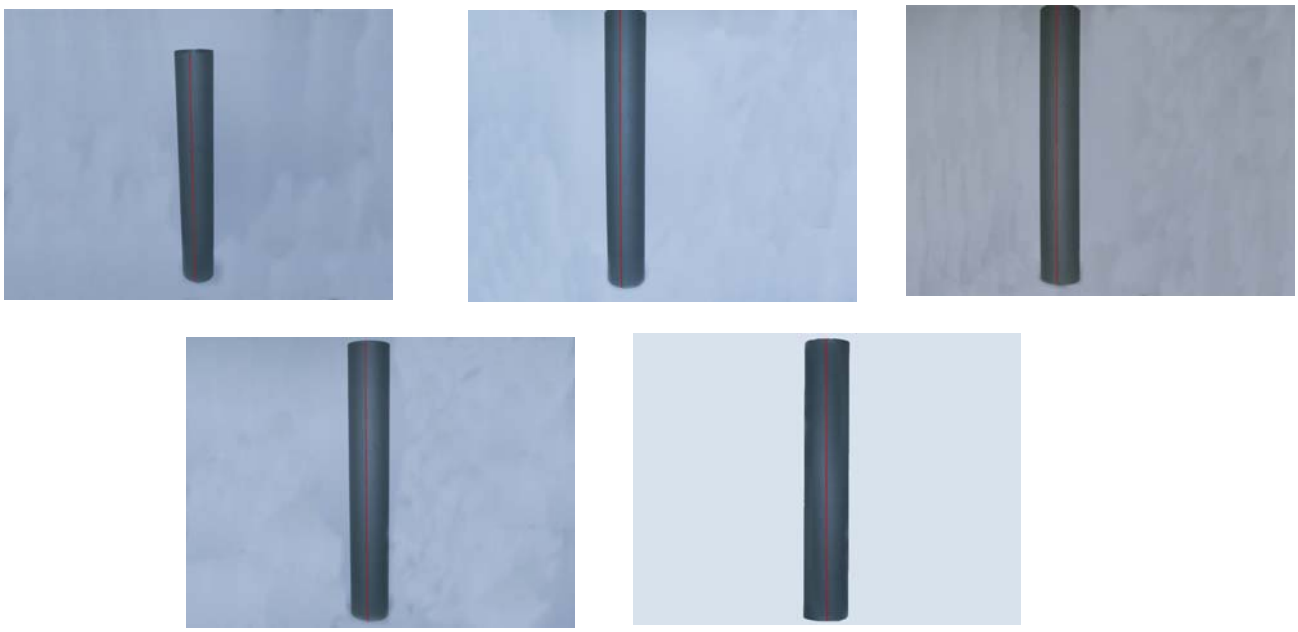

Fig. 15 Images from calibration procedure

$\Delta^{vert-poz} = \left| p_{\sup}^{vert-poz} - p_{\inf}^{vert-poz} \right|$,

6. IF $\Delta^{vert-poz} \le \varepsilon_{accept}^{vert-poz}$ THEN stop translating the camera,

7. ELSE translate the camera to up or down using $\operatorname{sgn}\left(p_{\sup}^{vert-poz} - p_{\inf}^{vert-poz} \right)$,

8. Back to begin, 1.

The full vertical calibration is performed in the following steps:

1. Vertical orientation,
2. Vertical positioning,
3. The offset for vertical orientation is computed $\Delta^{vert} = \left| p_{sus}^{vert} - p_{jos}^{vert} \right|$,
4. IF $\Delta^{vert-poz} \le \varepsilon_{accept}^{vert-poz}$ THAN the algorithm stops,
5. ELSE it goes to step 1.

Both offsets must be under the accepted thresholds. Else, the positioning destroyed the orientation and the procedure must be repeated.

Zoom calibration algorithm is performed in following steps:

1. Image segmentation using histogram threshold,
2. The white pixels are counted in both images, p_1^{zoom} and p_2^{zoom},
3. The zoom offset is computed $\Delta^{zoom} = p_2^{zoom} / p_1^{zoom}$,
4. IF $\varepsilon_{accept}^{zoom-\inf} \le \Delta^{zoom} \le \varepsilon_{accept}^{zoom-\sup}$ THEN the algorithm stops,
5. ELSE zoom-in one of the cameras. If $p_2^{zoom} / p_1^{zoom} < 1$, the second camera is chosen, else the first camera is chosen.
6. Back to begin, 1.

The image's segmentation is basically a threshold procedure applied to the image's histogram.

All the procedures included in the calibration algorithms were mathematically proven.

If the calibration algorithm was successfully applied then the system is ready to perform the visualservoing tasks.

4 Conclusions

This paper deals with a project for building and controlling a hyperredundant robot which is currently developing.

Different control methods and algorithms were proposed by the team members. A new tentacle manipulator using cables and stepper motors was designed and is under construction and a visual control using two video cameras is benchmarked.

A graphic simulator was created in order to support the system implementation. A virtual graphical simulator was designed and implemented in order to test hyperredundant robots behavior under certain circumstances for kinematics problems and to aid in the camera calibration process. The application was developed in Microsoft Visual C++ .NET 2005. It used the Microsoft DirectX SDK library for graphical purposes.

Algorithms for computing the positions of each segment base point were designed. The circular form of the segments was approximated using a cubic interpolating splines algorithm that passes through the control points: Catmull-Rom. A linear interpolation algorithm was developed for the motion simulation.

Several test scenarios were simulated. The cost reduction is one of the main advantages of using this application. In order to test the behavior of different hyperredundant robots – with different configurations - in real life, these robots must be built, the costs (high enough just for one robot) increasing with the number of test configurations.

Also, certain control laws could be proven to be harmful for the robot integrity (ex. too big arching angle), thus damaging the real robot. If the control laws are first tested in the simulator, such cases can be avoided.

Another advantage is the educational one, the simulator proving to be a useful tool in the process of teaching robot control and robot behavior.

In this paper, camera calibration using this graphic simulator was presented.

References:

[1] Walkera, I.D., Darren M. Dawsona, a.o. 2005. Continuum Robot Arms Inspired by Cephalopods, *DARPA Contr. N66001-C-8043, http://www.ces.clemson.edu/~ianw/spie05.pdf*

[2] Davies, J.B.C., Lane, D.M., a.o. 1998. Subsea applications of continuum robots, *Underwater Technology, Proceedings of the 1998 International Symposium on Volume*, Issue 15-17 Apr 1998 Page(s):363 – 369

[3] Walker, I.D. and Carlos Carreras. 2006. Extension versus Bending for Continuum Robots, *International Journal of Advanced Robotic Systems*, Vol. 3, No.2 (2006), ISSN 1729-8806, pp. 171-178.

[4] Hannan, M.W. and I. D. Walker. 2005. Real-time shape estimation for continuum robots using vision, *Robotica,* volume 23, pp. 645–651. © 2005 Cambridge University Press.

[5] Kelly, R. 1996. Robust Asymptotically State Visual Servoing, *Proc. IEEE Int. Conf. on Rob. and Autom.*, vol. 22, no. 15, pp. 759-765.
[6] Hutchinson, S., Hager, G. D. and Corke, P. F. 1996. A Tutorial on Visual Servor Control, *IEEE Trans. on Robotics and Automation*, vol. 12, no. 15, pp. 651-670.
[7] Grosso, E., Metta, G., a.o. 1996. Robust Visual Servoing in 3D Reaching Tasks, *IEEE Trans. on Rob. and Autom.*, vol. 12, no. 15, pp. 732-742.
[8] Ivanescu, M. 1984. Dynamic control for a tentacle manipulator, *Proc. Int. Conf. on Rob. and Fact. of the Future*, Charlotte, pp. 315-327.
[9] Ivanescu, M. 2002. Position Dynamic Control for a Tentacle Manip, *Proc. IEEE Int. Conf. on Rob. and Autom,* Washington, A1-15, 1531-1539.
[10] Ivanescu, M., Cojocaru, D., a.o. 2007, Visual Servoing By Artificial Potential Methods Of A Hyperredundant Robot, *Proceedings of 16th International Workshop on Robotics in Alpe-Adria-Danube Region - RAAD 2007*, June 7-9, Ljubljana, Slovenia.
[11] Watt, A. 2000. 3D Computer Graphics, *Addison Wesley.*
[12] Luna, F.D. 2003. Introduction to 3D Game Programming with DirectX 9.0, *Wordware Publishing*, Inc., Plano – Texas.
[13] Möller, T. and E. Haines, 2002. Real-TimeRendering. *2nd ed. Natick, Mass.: A K Peters*, Ltd.
[14] Jones, W. 2004. An Introduction to 3D Computer Graphics, *Course Technology PTR.*