Vision Based Servoing and Grasping with Multiple Control Schemes

M. Eriksson* , **H.I. Christensen***, **and J.O. Eklundh***

Computational Vision and Active Perception
Numerical Analysis and Computing Science
Royal Institute of Technology
SE-100 44 Stockholm, Sweden

Abstract. In this paper we demonstrate a system for robust vision-based manipulation, using an eye in hand. By robust, we mean that the system should be insensitive to small perturbations in sensor readings. Our intention is to employ a trajectory generator that is "aware" of the limitations of the feature extraction of the vision system. This means that we are not looking for optimal solutions in terms of length of the trajectory; we are more concerned about finding *any* solution while allowing for some erroneous sensor readings. We are currently implementing a system that is to solve a jig-saw puzzle by identifying and picking up the pieces. In this paper we focus on the utilization of multiple control schemes in order to adapt the control to current conditions. We also describe an algorithm for recognizing the pieces of the puzzle that is to be solved.

1 Introduction

A large amount of work has been done in visual servo control for manipulation [1, 2, 4, 5, 9, 10]. A manipulation task typically consists of the following subtasks:

1. Recognize an object
2. Approach the object
3. Grasp the object
4. Manipulate the object

While many systems have successfully accomplished each of these individual subtasks, we feel that the integration of these subtasks in a robust manner still needs more work. The purpose of this paper is to demonstrate a system that is able to accomplish a small manipulation task integrating all subtasks listed above. The intended system is to be used on a service robot that should be able to operate 24 hours a day in a domestic environment. This fact stresses the importance of *robustness* and *integration*.

One important aspect of vision-based manipulation systems is the choice of image features to be used by the control scheme. There are many vision-based control schemes today that can perform specific tasks with high accuracy. However, when the environment in which the system operates is slightly modified, the performance might degenerate significantly, due to non-satisfactory vision-data. We believe that one main reason for this is that the focus of the research has been on the *control aspects* of the problem by somewhat avoiding the limitations of *computer vision*. Our system is designed with an architecture that is able to select between different control schemes depending on the vision data delivered. In other words, the complexity of the control scheme can be adapted to what the vision-system can facilitate. For instance, one control algorithm may require the vision system to extract exactly n non-collinear feature points in the image. If this is done correctly, the system operates in an optimal way; however, if one point cannot be correctly identified in one image, the system may not operate at all. At this point, we need another, probably less efficient, control scheme that can operate according to the constraints dictated by the vision system.

In order to perform the switching between different control schemes, we are making use of a *hybrid automaton*. This was implemented in MMCA(Mobile Manipulation Control Architecture) which is a software package, developed at our lab, for mobile manipulation in general. This architecture executes a control scheme

* Email: {eriksson,hic,joe}@nada.kth.se

until a certain condition is satisfied, that causes a switch to another scheme.We would also like to emphasize the distinction between two types of transitions between control schemes. A *task-specific* transition is due to a new stage of the overall task. For instance when going from the task of approaching an object to the task of grasping it. A *data-specific* transition occurs when the vision system begins to deliver new information (better or worse).

The robot we used is shown in Figure 1. It consists of a Puma560 mounted on top of a XR4000 platform from Nomadic Technology. The Puma560 is a 6 degree of freedom arm equipped with a CCD color camera on the end-effector in order to facilitate visual servoing. The end-effector is equipped with a J3 force-torque sensor.

Fig. 1.: The Nomad XR4000 platform with a Puma560 arm on top.

In order to demonstrate these ideas, we have decided to implement a puzzle solving routine for our robot. This would involve all the subtasks listed in the beginning. Specifically, the overall task can be broken down into the following subtasks:

1. Recognize a selected piece of the puzzle, lying on a table
2. Move the end-effector relatively close to the piece
3. Change the arm configuration so that it is possible to grasp the piece
4. Pick up the piece

The rest of this paper is organized as follows: In Section 2, we describe how the recognition module works. The discussion concentrates on how to extract and match 2D invariants. In Section 3, we describe how the servoing is performed. Two different servoing schemes are used for this task. One for approaching the piece and one for finding a configuration that makes it possible to grasp the piece. In Section 4 discuss the implementation. In section 5, we show some preliminary results, followed by brief summary in Section 6.

2 Object Recognition

The general approach to perform object recognition can be divided into four different stages

1. Figure Ground Segmentation. This is one of the fundamental problems in computer vision.
2. Feature Extraction
3. Indexing
4. Verification

Our approach to each of these steps are described in the sub-subsections below. As the objects we are considering are quite different, we combined indexing and verification into one step. This step is explained in the section on curve matching. When the recognition algorithm starts, we assume that the arm is in a configuration so that the wrist-mounted camera will overview the table on which the puzzle is placed. The

puzzle consists of a board with empty slots and a set of pieces that are to be placed into these slots. The algorithm will now try to pair up one piece with one matching slot. A sample input image to the recognition module is shown in Figure 2(a).

(a) Original image (b) Image after thresholding (c) Extracted contours

Fig. 2.: The image of the puzzle to be solved taken from the camera on the end-effector.

The system starts by segmenting all potential slots and pieces from the background. One of the segmented slots is then randomly selected. The recognition system then selects the correct piece: i.e the piece that seems to fit in the slot. The recognition is based on matching geometric invariants on the contours [8]. This builds on the fact that two contours resulting from two different projections of a 2D shape differ only by a *homography*.

2.1 Figure Ground Segmentation

In order to do recognition, figure ground segmentation is generally needed in order to extract the potential objects from background. There is a rich collection of algorithms for figure ground segmentation designed for different tasks. In order to make our problem tractable, we decided to simplify the segmentation by using pieces and slots of very characteristic colors. This allows us to extract the pieces and empty slots with a color based segmentation approach. However, we have developed the system so that a more sophisticated figure ground segmentation algorithm can easily be incorporated into the system, and substitute the one we currently use. In our color based segmentation, we threshold the image in RGB-space by manually selecting a "box" of RGB-values. Any pixel with an RGB-value inside that box is considered to belong to a piece (or an empty slot, which are of similar colors as the pieces). All the other pixels are considered as background-pixels. A result of running this thresholding is shown in Figure 2(b). One obvious drawback with this approach is that the manually selected threshold is not robust under varying lighting conditions.

2.2 Feature Extraction

The features required by the recognition module is a set of contours of the possible pieces and slots. These contours are extracted directly from the blobs resulting from the figure ground segmentation step, by traveling one lap around each blob. The contours are represented as lists of image coordinates, which are passed along to the recognition module. Note that too small blobs are considered to occur due to noise, and are ignored. In order to reduce the effects of noise, the curves are smoothed with a Gaussian filter. The resulted contours found after the segmentation and smoothing are shown in Figure 2(c).

2.3 Shape Matching

Recognition of shapes in 2D is a relatively well understood problem [3, 7, 8, 12, 13, 14, 15]. As mentioned earlier, most approaches to comparing two planar objects are based on *projective invariants*. The fundamental fact behind these ideas is that the contour of the same object or two identical objects, taken from two different views differ only by a homography T. I.e let P_A and P_B be two corresponding points on the plane, expressed in homogeneous coordinates, lying on contours A and B respectively. Then

$$P_A = TP_B \tag{1}$$

where T is a 3x3 matrix.

Once the homoghraphy T is found, one of the shapes can be transformed into the frame of the other, and a match can be computed by, for instance, sum of squared distances between corresponding pixel positions. The projection matrix T can be determined (up to a scale factor) by identifying four matching points on the two contours. There are different ways to find such matching points based on the inflexion points (inflection points are invariant under projective transformations). Bitangents are another example of projective invariant points that we have tried to use in order to identify the homography. However, when using these points, we experienced cases when the detected reference points resulted in a very ill-conditioned transformation matrix T. We found that the algorithm worked best if the four points were evenly distributed over the contours. In our approach, we start by computing the curvatures of the two shapes, using the common formula for curvature given in equation 2. $x(t)$ and $y(t)$ are the coordinates for the contour-element with index t in the shape representation.

$$\kappa(t) = \frac{\dot{x}(t)\ddot{y}(t) - \ddot{x}(t)\dot{y}(t)}{(\dot{x}(t)^2 + \dot{y}(t)^2)^{\frac{3}{2}}} \tag{2}$$

The curvature plots of the two shapes that are compared are then normalized to the same length. The algorithm then tries to align the two curvature plots in a way so that concavities and convexities coincide as good as possible between the two curves. The curvature plots of two shapes, representing the same object, are shown in Figure 3. When this is done, we have a correspondence between the *indices* of the contour elements. Let contour A be of length l_a and and contour B of length l_b and the normalized curvature plot of B be displaced d units with respect to the normalized curvature plot of A. We then have that pixel number p in the pixel list of A corresponds to pixel number $(l_b/l_a)d+(l_b/L_a)p$ in the pixel list of B. Once this correspondence is established, we can chose four evenly distributed points on A, and also their corresponding points on B, which will give us the homography T. In fact, we can chose any number of points, but four is enough to compute T. The four points used to compute T in our example are shown in Figure 4

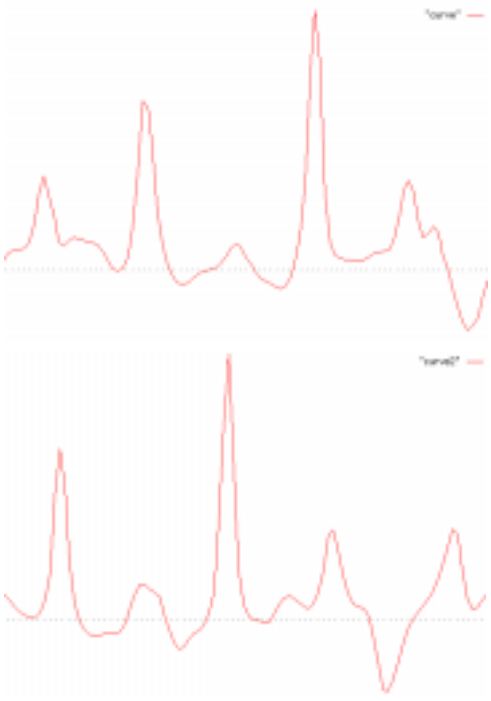

Fig. 3.: The curvature plots of two objects with the same shapes.

For the selected slot, this algorithm is ran on every possible piece, and returns the piece giving the best match.

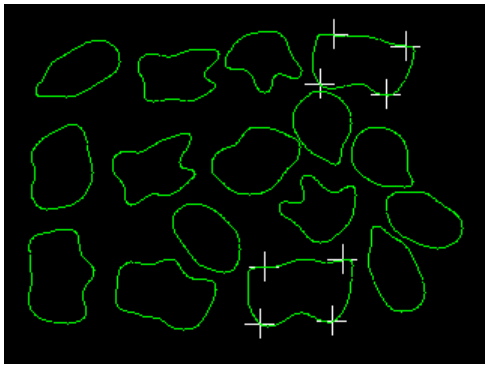

Fig. 4.: The board, the contours of the pieces and the computed point-sets of two matching pieces.

3 Servoing

Visual servoing is generally divided into two categories[6]; image-based servoing and position-based servoing. In this project we use strictly the image-based version. The general problem to be solved when doing image-based servoing is how to map a feature vector extracted from the image into a joint control vector to be applied to the set of joints on the manipulator. This mapping is generally referred to as the *image Jacobian*. I.e, let F^e be the feature vector extracted from the image, and F^d be the desired feature vector (the vector extracted from a picture taken from the desired position). Then the joint control vector, $\dot{\theta}$, is given by

$$\dot{\Theta} = J^J(\Theta)^{-1}(F^e - F^d) \tag{3}$$

Sometimes it is more convenient to work in Cartesian space, rather than in joint space. We then get the Cartesian control vector as

$$\dot{X} = J^C(X)^{-1}(F^e - F^d) \tag{4}$$

Here, $X = [V, W]^T$ where $V = [x, y, z]^T$ is the position and $W = [\alpha, \beta, \gamma]^T$ is the rotation of the end-effector in Cartesian space. By using the robot Jacobian, the joint angle velocities can be determined from the desired velocities in Cartesian space, as long as the pose is within the workspace of the manipulator. This relation is given by

$$\dot{X} = J^R(\Theta)\dot{\Theta} \tag{5}$$

and is referred to as inverse kinematics. Note that we use $J^J(\Theta)$ for the Jacobian in joint space, $J^C(X)$ for the Jacobian in Cartesian space and $J^R(\Theta)$ for the robot Jacobian. A subscript means that we reduce the dimensionality of the Jacobian.For instance $J^C_{xy}(X)$ means a $2x2$ Jacobian operating on the xy-plane. We can thus chose to express our control-law in joint space or in Cartesian space.

3.1 Approaching the object

A large problem with image-based servoing in general is how to estimate the image Jacobian J , and how to avoid J to be ill conditioned. Also, small errors in the feature vector extracted from the image can generate large perturbations in J. Even though working with a full rank image Jacobian does provide the mathematically fastest way towards the desired position, it is rather cumbersome to handle. Since our goal is to construct a robust system, we found it more natural to reduce the number of degrees of freedom that the robot can use at each step of the computation. More specifically, our first control loop iterates over the following steps:

1. Get the position of the object in image coordinates (from vision)
2. Find pitch and yaw angles so that the object becomes centered in the image
3. Move towards the object (along the z-direction in the reference frame of the camera)

The iteration terminates when the area (in image pixels) of the object exceeds a threshold. In order to center the object in the image in step 2, the control scheme works with an image Jacobian that only involves two degrees of freedom in Cartesian space, namely pitch and yaw. Thus, the control vector in Cartesian space is given by

$$\dot{X}_{pitch,yaw} = J^C_{pitch,yaw}(X)^{-1}(I_{xy} - I_{origin}) \tag{6}$$

where I_{xy} are the image coordinates of the center of mass of the object and I_{origin} is the image center. The movement in Step 3 involves translation in one direction of Cartesian space, namely z. The magnitude of this movement in each step of the iteration is inverse proportional to the size of object. This control vector is given by

$$\dot{X}_z = J^C_z(X)^{-1} I_A \tag{7}$$

where I_A is the area of the object in number of pixels. In this case we applied a strictly proportional control-laws by using $J^C_{pitch,yaw} = diag(K_\beta, K_\gamma)$ and $J_z = K_z$, where K_z,K_β and K_γ are constants. At the end of the iteration, we have brought the end effector relatively close to the object. However, this approach does not affect the *pose* of the end effector; it simply brings it into proximity of the object. The adjustment of the pose is taken care of in the pre-grasping step.

3.2 Pre-grasping

When the end effector is relatively close to the object, the system changes state to the pre-grasp state. Pre-grasping involves finding a configuration of the arm so that the z-direction of the end-effector is perpendicular to the table-surface. Also the roll of the end-effector should be in a position that allows for a grasp when the end-effector is brought to contact with the table. The first of these two steps is performed in a similar way as when the end-effector was brought close to the object. For simplicity, we assume that the table on which the pieces are placed is parallel to the ground-plane (and therefore oriented as the xy-plane in the reference-frame of first joint of the manipulator). The control algorithm now iterates over the following steps:

1. Update roll, pitch and yaw so that the end-effector comes closer to being perpendicular to the table, but *only as long as the object is inside the field of view.*
2. Center the object in the image *without changing the roll, pitch or yaw.*

Specifically, in the first step, the object is re-centered in the image using

$$\dot{X}_{xy} = J^C_{xy}(X)^{-1}(I_{xy} - I_{origin}) \tag{8}$$

while the adjustment to being perpendicular is achieved by

$$\dot{\theta}_{45} = J^J_{45}(\Theta)^{-1}[\pi - \theta_2 - \theta_3, \theta_4]^T \tag{9}$$

since when the end-effector is perpendicular to the ground-plane, $\Pi - \theta_2 - \theta_3 = \theta_4 = 0$. This is given by the definition of the Puma560 kinematics. After the completion of this loop, the end-effector is placed with the camera right above the object, oriented perpendicular to the table.

In order to grasp the object, a translation is required in order to move the gripper to the current position of the camera (since the servoing is done in the cameras reference frame). After this translation is performed, the actual grasping is done by moving the end-effector down towards the table until the force-torque sensor indicates that contact has been achieved. At this point, the gripper closes its fingers. Note that we have not yet implemented any grasp-planning, which generally becomes a more interesting problem when multi-finger grippers are used. However, for this project, we will implement a control scheme that, after the described pre-grasp, finds a roll of the end-effector that aligns the two fingers of the gripper with the major axis of the object. However, we have not yet integrated this feature in the system. In order to grasp along the major axis, the second-order statistics is given by

$$M_{kl} = \sum_{p \in Object} x_p^k y_p^l \qquad\qquad k + l = 2 \tag{10}$$

where the eigenvectors of

$$\begin{bmatrix} M_{20} & M_{11} \\ M_{11} & M_{02} \end{bmatrix} \tag{11}$$

give the major and minor axes.

4 Implementation

This system was implemented on a Nomadic XR 4000 platform with a Puma 560 on top. The image processing was carried out under Linux, while arm-control is carried out under the Real-Time Operating system QNX. The two computers are interconnected using TCP/IP implemented on top of a shared memory region. The control of the manipulator was defined as a Discrete Event System, implemented using the MMCA software systems[11].

5 Experimental Results

We decided to evaluate the recognition part and the servoing/grasping part of this system separately.

5.1 Recognition

The recognition naturally works best if the object to be recognized has may distinct concavities. Totally convex objects are virtually impossible to identify by using geometric invariants. In the case of the scene in Figure 2, the system could find the correct piece for seven of the eight slots. We have tested the algorithm on two different puzzles, and it correctly identified pieces for 14 out of 17 slots. We would like to emphasize the fact that the success of this approach relies on the number of inflection-points on the contours. This means that the system is not able to distinguish between squares and circles, since both these shapes are totally convex. There are of course other methods that can perform this recognition.

During the servoing phase, the vision algorithm iteratively sends the image coordinates of the position of the object that is selected to be grasped. The area, in number of pixels, of the object is also delivered. This data is sent by TCP/IP from the linux computer to the real-time computer that is controlling the arm. This data was delivered at about 4 Hz.

5.2 Servoing and Grasping

In order to successfully grasp a flat object, like a piece of a puzzle, it is important that the end-effector is accurately positioned above the object, in a correct pose (perpendicular to the ground-plane). In order to evaluate this, we ran the algorithm on a piece from the puzzle 10 times. The average errors in position was 1.5 cm, while the error in both pitch-angel and yaw-angle were slightly less than 1 degree.

If the error in yaw-angle is to large, the force-torque will signal contact with table when only one of the two fingers is in contact. This means that when the fingers close, the piece will not be caged properly. However, if the object to be grasped is thicker, the tolerance in Yaw-angle error becomes larger. In our experiments, the piece was successfully grasped in 7 out of 10 trials.

6 Summary

We have developed a framework that allows the control of a manipulator to adapt according to changes in the environment or changes in the sensor processing. However, for the project presented here, we have not switched between control schemes due to these changes. The switches we have demonstrated here are all task-driven, which means that they occur due to new stages of the overall problem. It is obvious that the system, in its present shape, is not very robust, due to unrealistic assumptions by the vision-module. Future work will involve developing control schemes that can relax the vision requirements. By building the system in a bottom-up fashion, we will have a robot that should always be able to operate, not necessarily in an optimal way, even though the conditions are challenging (i.e poor lightning, cluttered backgrounds, etc. By using a hybrid automaton, the system becomes very flexible, and it is easy to implement new control schemes. In this manner we are able to adapt the complexity of the control to the quality of the visual processing.

Acknowledgment

This work was sponsored in part by the Swedish Foundation of Strategic Research through its Center for Autonomous Systems at KTH.

References

1. P.I. Corke. *Visual Control of Robots*. Research Studies Press, 1996.
2. B. Espiau, F. Chaumette, and P. Rives. A new approach to visual servoing in robotics. *IEEE Transactions on Robotics and Automation*, 8:313–326, 1992.
3. F.Asada and M. Brady. The curvature primal sketch. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1):2–14, 1986.
4. J.T: Feddema and O.R. Michell. Vision guided servoing with feature based trajectory generation. *IEEE Transactions on Robotics and Automation*, 5:691–700, 1989.
5. K. Hashimoto, T. Kimoto, T. Ebine, and H. Kimura. Manipulator control with image based visual servo. In *IEEE International Conference on Robotics and Automation*, pages 2267–2272, 1991.
6. Seth Hutchinson, Greg Hager, and Peter Corke. A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12:651–670, 1996.
7. F. Mokhtarian and A. Macworth. Scale-based description and recognition of planar curves and two-dimensional shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1):34–43, 1986.
8. J.L Mundy and A. Zisserman. *Geometric Invariance in Computer Vision*. MIT Press, 1992.
9. N.P. Papanikolopoulos, P.K. Khosla, and T. Kanade. Vision and control techniques for robotic visual tracking. In *IEEE International Conference on Robotics and Automation*, pages 857–864, 1991.
10. N.P. Papanikolopoulos, P.K. Khosla, and T. Kanade. Visual tracking of a moving target by a camera mounted on a robot. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(1):14–35, 1993.
11. L. Petersson, M. Egerstedt, and H.I. Christensen. A hybrid control architecture for mobile manipulation. In *IEEE Proceedings of International Conference on Intellligent Robots and Systems*, 1999. Accepted for publication.
12. L.G. Shapiro and R.M. Haralick. Decomposition of two-dimensional shapes by graph-theoretic clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1:10–20, January 1979.
13. I. Weiss. Geometric invariants and object recognition. *International Journal of Computer Vision*, 10(3):207–231, 1993.
14. J. Wolfson. On curve matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(5):483–489, 1990.
15. F. Xia. On contour invariants: Relationships and application. In *IEEE International Conference on Pattern Recognition*, pages 136–140, 1996.

This article was processed using the TEX macro package with SIRS99 style