INSTITUT FÜR INFORMATIK

UND PRAKTISCHE MATHEMATIK

Learning Manipulator Behaviors using Visual Information

Josef Pauli (ed.)

Bericht Nr. 9901
Januar 1999

PAX OPTIMA RERUM

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

KIEL

Institut für Informatik und Praktische Mathematik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

Learning Manipulator Behaviors using Visual Information

Josef Pauli (ed.)

Bericht Nr. 9901
Januar 1999

e-mail: jpa@ks.informatik.uni-kiel.de

″Dieser Bericht ist als persönliche Mitteilung aufzufassen.″

Preface of the report

The research of the Cognitive Systems Group Kiel is focused on the design and implementation of behavior-based autonomous robot systems. This report summarizes selected work with the common aim of solving industry-relevant manipulation tasks based on visual information. Our manipulator is a STÄUBLI RX90 robot arm with six rotation joints and a parallel jaw gripper. One or two gray-level SONY cameras are mounted on an immovable tripod or on the manipulator end-effector. We describe four manipulation tasks which have been conducted by the editor of this report and implemented by graduated students in Computer Science. The approaches have already been published and presented at international conferences in 1997 and 1998. Extended versions are available from the archives of our group (theses, written in German).
In the first work (chapter 1) the manipulator has been equipped with the behavior of arranging technical objects according to a spatial relation, which must be visually demonstrated prior to application phase [1, 2].
In the second work (chapter 2) the manipulator has been equipped with the behavior of moving along the boundary of technical objects, which is relevant for processing or visually inspecting the boundary [3, 4].
In the third work (chapter 3) the manipulator has been equipped with the behavior of learning gripper trajectories for handling working tools, e.g. turning a screw spanner by a circle trajectory [5, 6].
In the fourth work (chapter 4) the manipulator has been equipped with the behavior of finding obstacle-avoiding trajectories towards a goal position, e.g. taking out food objects from a refrigerator [7, 8].
Currently, further industry-relevant developments are nearly finished, e.g. using the robot arm as a carrier of cameras for the purpose of detailled object inspection or object surface reconstruction.

Table of contents

1 A vision based robot system for arranging technical objects

1.1 Outline of the chapter

Robot programming by demonstration simplifies the task of robot programming. Our implemented vision based robot system makes use of this approach and is able to arrange objects in a 2D-scene (e.g. a conveyor belt). To perform this task, the video camera takes images from two relevant objects and the movement of the robot hand is determined in such a way that both objects are arranged in a desired manner. The taught relation between two objects, e.g. a screw-wrench at a screw-nut, can be restored automatically, independent of their initial position and orientation. Almost all necessary information is extracted from images of the scene (very little a priori knowledge). The procedure of object recognition is based on the contour of the objects and derived features. The recognition procedure is invariant w.r.t. scaling, rotation and position of the objects, and actually this implies the generalization ability.

1.2 Introduction

The subject of this project was to implement a vision based system which gives a robot the ability to rearrange objects in a desired manner. It puts into practice the method called *programming by demonstration* [9]. Systems based on this method differ by the ability of performing their tasks in a generalized way. Beside the ability of extracting the contour and derived features which are necessary to recognize, localize and identify objects automatically, our system stores the spatial relationship between objects. This relation can be restored by the robot in the sense that it is independent of the initial position and orientation.

There are some premises. Objects are assumed to be flat so there is no need of a three dimensional reconstruction of the scene. The pair of objects which shall be arranged must not overlap mutually or with other objects in the scene, so that the whole closed contours of the desired objects can be extracted. The object which shall be manipulated has to be grasped. This separate task is complicated and will not be part of this paper. However the object which is grasped is recognized automatically.

The system consists of four main parts:

1. *Modelling of shapes and objects*
   Images of the objects are taken which shall be arranged (appearance based object recognition [10]). The required models are extracted and stored.

2. *Calibration*
   The calibration takes place by multiple positioning of an object in the scene and relies on the localization procedure. This part will not be discussed in this paper.

3. *Demonstration of the spatial relation*
   By positioning a *grasped object*, it is related geometrically to another object, the

target object, on the working plane. The relation between these two objects is stored afterwards.

4. *Arrangement of objects*
   Known and unknown objects in the scene which can be arbitrarily positioned and orientated are automatically selected, recognized and localized. The grasped object is positioned by the robot so that the taught relation is restored.

1.3 Configuration of robot and camera

The STÄUBLI RX90 robot is characterized by six rotation joints so that the end effector can be arbitrarily positioned and orientated in the working space. The processing unit solves the problems of inverse kinematics and inverse dynamics. In essence only the position and orientation of the working tool must be sent to the processing unit. The images are taken by one grey scale camera which is mounted fix on a tripod. The size of an image is 512 × 512 pixels. The focal length of the lense was 12 millimeters. The optical axis and the object plane are perpendicular to each other. Image processing and object recognition was performed on a Sun SPARC workstation.

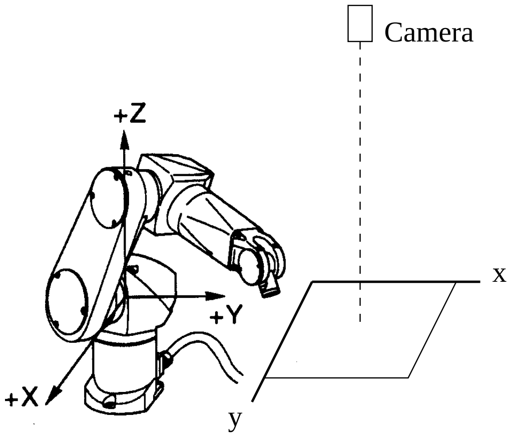

Figure 1: Configuration of robot and camera.

1.4 Object recognition

1.4.1 General requirements

The choice of the method which recognizes, localizes and selects objects is determined by the properties of the images of the scene. First, the vision system must be able to distinguish between parts of the robot and objects in the image. A direct identification of robot parts in the image is difficult because of their nonrigidity. Second, it must be able to separate the image background from the desired objects. Because of the nonhomogenous background and noise there can be contours to which no real object belongs to. Third, it must also be able to distinguish between the objects themselves. The scene can be enriched with objects which are not intended to be arranged, these objects must be rejected. Finally, the most important point is that the recognition procedure has to cope with the arbitrary position, orientation and scaling of the objects.

To fulfil these requirements, we use a model based approach. Therefore, the images of the desired objects are taken in the modelling phase and suitable models are computed through image processing. Thus the problem of assigning an initial meaning [11] is solved. During the recognition procedure only objects to which a model fits are regarded. In this pragmatic way the system is informed about relevant objects for further recognition.

1.4.2 Representation of the contour and matching procedure

The discussion in the previous section has shown the necessity of a sophisticated representation of the contours and a suitable matching method. Contour based approaches have been proved to be favourable, e.g. in [12], [13] and [14], which in particular differ in their complexity.
Arkin et al. [15] published an efficient method for comparing polygonal shapes. They establish the notion of the *turning function* which represents the shape of an object. In the case of piecewise constant turning functions they present an $O(mn\log(mn))$ matching algorithm, where m and n are the numbers of vertices of the two polygons. This algorithm has been applied and turned out to be very fast.

Let Q be a point on the contour of the object O with the length l_O. If one walks from Q to a point P on the contour so that the interior of the object is on the left side, one has covered a way of length $w \in [0, l_O]$. At first, one assigns to w the angle in radians between the tangent in P and the horizontal, in consideration of the circulations. Through a parametrization $[0,1] \to [0, l_O]; s \mapsto s \cdot l_O$ one obtains the turning function $\Theta_{A,Q} : \mathbb{R} \to \mathbb{R}$, where A is the shape of object O. The turning function is easily extended from the unit interval to $\mathbb{R}$ by including the circulations, i.e. by adding or subtracting multiplies of 2π. Figure 2 shows an example of a turning function.

The parametrization implies the invariance under scaling. There are two degrees of freedom, the choice of the starting point Q and the orientation of the object. This leads to the equivalence class

$$[\Theta_A] := \{\Theta_{A',Q'} \,|\, \exists\, t, \varphi \in \mathbb{R}\, \forall s \in [0,1] : \Theta_{A',Q'}(s+t) + \varphi = \Theta_{A,Q}(s)\} \qquad (1)$$

of turning functions which belong to the same shape A.

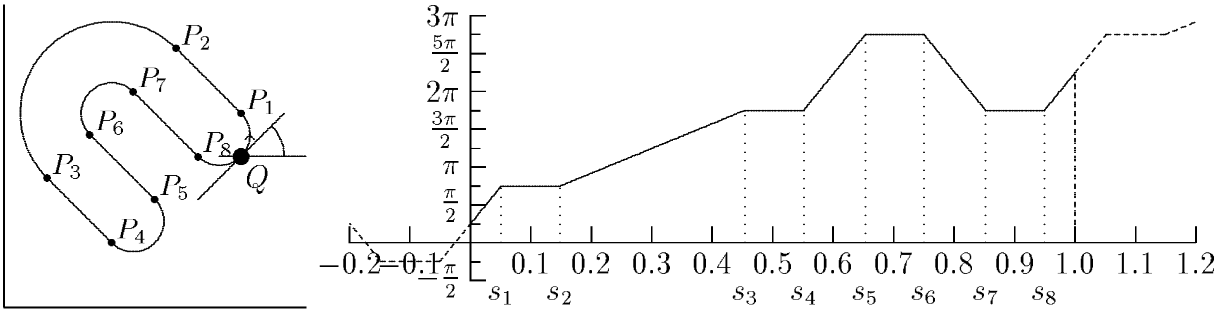

Figure 2: Object and its turning function.

The metric on these equivalence classes is defined as

$$d_2(A,B) = \left(\min_{t\in[0,1],\varphi\in\mathrm{I\!R}} D_2^{A,B}(t,\varphi)\right)^{\frac{1}{2}} \quad \text{where} \quad D_2^{A,B}(t,\varphi) = \int_0^1 |\Theta_A(s+t)+\varphi-\Theta_B(s)|^2\,ds \tag{2}$$

It is shown that

$$d_2(A,B) = \left(\min_{t\in[0,1]}[h(t) - (\alpha - 2\pi t)^2]\right)^2 \tag{3}$$

where

$$h(t) = \int_0^1 [\Theta_A(s+t) - \Theta_B(s)]^2\,ds \quad \text{and} \quad \alpha = \int_0^1 \Theta_B(s)\,ds - \int_0^1 \Theta_A(s)\,ds. \tag{4}$$

In the case of an approximation of the contour by straight lines the turning function becomes a piecewise constant function. This leads to a simple representation of the form

$$\tilde{\Theta}_{A,Q_A} = \{(\sigma_0^A, \alpha_0^A), \cdots, (\sigma_{m_A-1}^A, \alpha_{m_A-1}^A)\} \qquad \text{with} \quad m_A = n_A + 1, \tag{5}$$

where n_A is the number of vertices of the polygon which approximates the contour of a shape A, σ_i^A is the i-th supporting point and α_i^A the angle between the horizontal line and the tangent in the current point P in the environment on the right side of σ_i^A. The algorithm in [15] which computes the distance between two shapes A and B is of complexity $O(m_A m_B \log_2(m_A m_B))$. A side effect of the algorithm is that the best matching angle between the two objects is computed.

1.4.3 Modelling of objects

The shape is represented by the piecewise constant turning function and the normalized area $F_A := \frac{F_O}{l_O^2}$. F_O is the area and l_O the contour length of the image object. F_A is a measure for the circularity of the shape. The circularity of a narrow object is small and a circle shaped object has maximum circularity. F_A is used as a preselection criterion to decide whether two shapes are similar enough. So the algorithm which computes the distance between two turning functions must only be applied to shapes with similar circularity and this makes the whole matching procedure more efficient. Note that the circularity of two shapes can be equal for two different shapes, e.g. if the two shapes are mirrored. So the comparison of the circularities is only a necessary but not sufficient criterion.

The object is modelled by its shape and features that determine its size, i.e. its area F_O and its contour length l_O. F_O and l_O serve as selection criterion for objects.

We have extentend the metric on shapes to a metric on objects in terms of

$$d_O(O_1,O_2) = \sqrt{d_2(A_1,A_2)^2 + [\omega\cdot(|\frac{F_{O_1}}{F_{O_2}} - 1| + |\frac{F_{O_2}}{F_{O_1}} - 1|)]^2}. \tag{6}$$

The parameter $\omega \in \mathrm{I\!R}_{>0}$ weights the relative errors of the areas. If ω tends to zero, only shape is measured.

To decide whether two objects are similar enough the distance must be compared with a threshold. If the distance of two objects is lower than this threshold, the objects are accepted as similar, otherwise they do not match.
This method is applied on all objects in the image. The obtained set of objects includes the candidates of target objects and the object which is grasped. The grasped object is determined by the method described in the next section.

1.5 Recognition of the grasped object

The appearance of the gripper distuinguishs itself from the objects during the grasping process. The gripper moves while the other objects are fix in the image. This feature is used to determine which of the recognized object is the grasped one. Two images are taken while the gripper is closing. By applying the difference operator on these images, one get an image with high grey values in the environment of the end effector. Additionally, edges are extracted to get dominant points from which the convex hull is computed. This convex hull is the geometric description of the gripper range. The object which intersects with the convex hull or lies inside the hull is the grasped object. This method has the advantage that it is fast and in particular independent of the appearance of the effectors. Figure 3 illustrates this method.

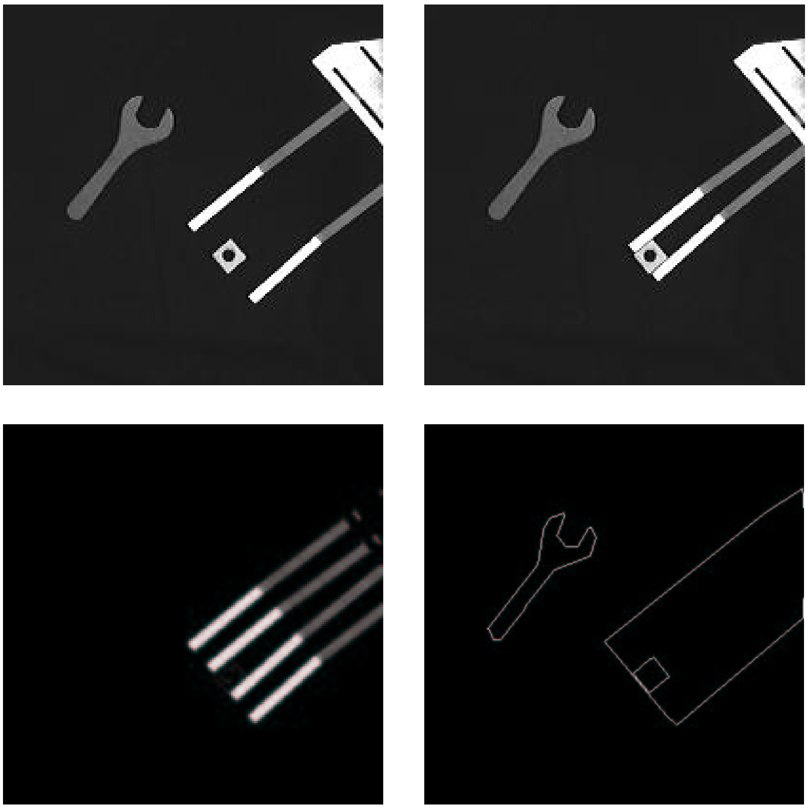

Figure 3: Image of the scene with opened gripper, image of the scene with closed gripper, difference image, contour of the grasped object and the target object with the convex hull which describes the gripper range.

According to this approach of object recognition, the system is able to recognize the candidates of target objects and the grasped object.

1.6 Learning and application of the knowledge

The operator puts two objects in the working plane and uses the control panel of the robot to arrrange the two objects manually and thus demonstrates the system the desired geometrical relation between both. The vision system computes the contours, recognizes the objects and computes a tripel which represents the relation beween the grasped and the target object. The tripel is defined by

$$(O_g, R, O_t), \tag{7}$$

where O_g und O_t are the models of the grasped resp. the target object. R is the tripel (ϕ, z, v) that stores the data to rearrange the two objects. ϕ is the angle in radians the grasped object was rotated. To allow that two objects are rearranged upon each other, z stores the vertical offset. $v = (x, y)$ describes the scale invariant position of the common center of area of the two objects. Figure 4 shows an example of two related objects.

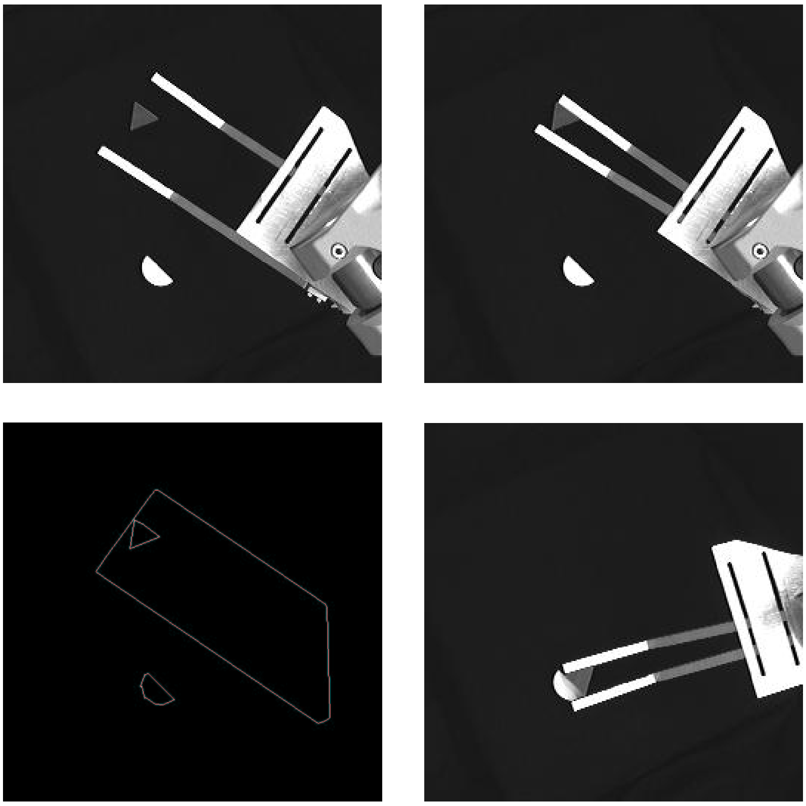

Figure 4: Scene with open gripper and scene with closed gripper before the positioning of the object by the operator, the contour of both objects with the convex hull which describes the range of the gripper, scene after relating the objects through the programmer.

The triangular shaped block was positioned side by side to the halfcircle shaped box in the way that two corners meet. An arbitrary number of relationships can be learned in this way for further application. The knowledge is represented by the union of tripels in (7).

The relation learned in the demonstration phase is the basis for rearranging the objects automatically. The objects in the relation set serve as the models to recognize objects in the image of the scene taken by the camera. Pairs (O_g, O_t) of candidates are extracted from the image. If there exists a matching tripel (O_g^r, R, O_t^r) in the stored relation tripels, the relative orientations of O_g and O_t are computed from O_g^r and O_t^r and thus the object O_g can be manipulated by the robot so that the relation code in R is restored. Therefore, the distance of a pair of objects is measured by

$$d_R((O_{g_1}, \cdot, O_{t_1}), (O_{g_2}, \cdot, O_{t_2})) = \max(d_O(O_{g_1}, O_{g_2}), d_O(O_{t_1}, O_{t_2})). \qquad (8)$$

The metric on triples of the relation is in terms of

$$d_R((O_g, O_t), (O_g^*, O_t^*)) = \min\{d_R((O_g, O_t), (O_g^r, O_t^r)) \mid (O_g^r, \cdot, O_t^r) \in R\}. \qquad (9)$$

A threshold operator is used again to decide whether two triples match or not.

Figure 5 demonstrates the precision the objects are rearranged. The underlying relation was the relationship between the triangle shaped block and the halfcircle shaped block. Consequently they are recognized as grasped object resp. target object and are manipulated by the robot in the desired manner.

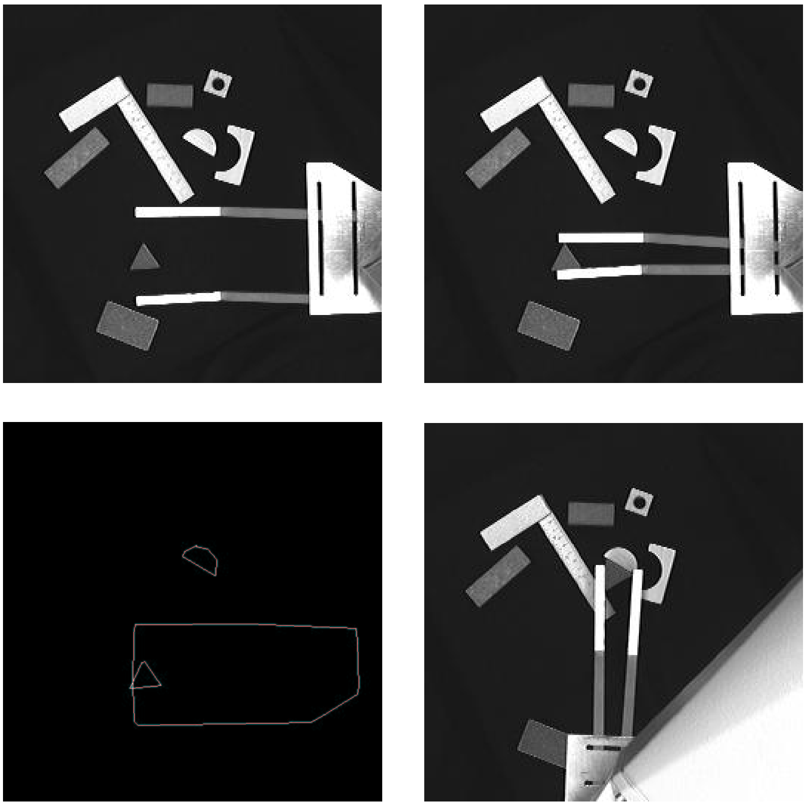

Figure 5: The restored relation of figure 4.

1.7 Conclusions of the chapter

We developed a system to arrange objects which works even if the objects are translated and rotated. The method of *programming by demonstration* is applied by using vision based object recognition. The theoretical basis for the shape matching procedure was given by [15]. A model based matching procedure for object recognition and localization was derived. The starting point of this procedure is a symbolic contour description which is generated by simple preprocessing which has to be adapted to the environment. Despite of unavoidable shadows and approximation of the contours by straight edges the programmed relations were restored with high precision. The recognition method is determined only in the two-dimensional plane, nevertheless many applications are conceivable.

2 Object boundary extraction by an active contour approach

2.1 Outline of the chapter

The manipulator has been equipped with the behavior of moving along the boundary of objects, which is relevant for processing or visually inspecting the boundary. In this work two calculation rules are presented for forcing a snake to approximate the contour of a selected object or the outer contour of an assembly of objects. Actually this second capability of extracting the contour of an object assembly characterises the novelty of the approach. The snake works on an image of flat scene objects. Therefore the border of the real 3D object can easily be reconstructed from the object contour in the image. It was intented to develop an appraoch which works with nearly arbitrarily chosen initial snake points, to get a snake which "finds" the object even if the initial points are put far away from the desired object. A further goal was to close gaps in the outer contour of an object assembly if the space between two objects has to be bridged respectively. The extracted contour is used a guiding line for moving the manipulator along.

2.2 Introduction

A usual way to detect the contour of an object is to extract the edges of the whole image using the magnitude of the grey level gradient, then to match an edge model of the object with the edge image and to locate it by looking for the strongest correlation. The gradient based edges come from grey level fluctuations within the object and from the object border. Often the edges within the object are undesirable due to shading caused by nonperfect lighting conditions. Instead of that the edges at the object border are stable and object detection should mainly take these into account.

Therefore a useful approach to detect border edges is to use a priori knowledge for distributing a set of points roughly around the object in a first step. In the second step the procedure iteratively moves the points towards the object to stop finally if all points are laying directly at the border. The polygon received by connecting every point of the sequence with its successor is an approximation of the object contour. This way of proceeding first was discussed by Kass et al. [16]. Such a sequence of points that change their location by time based on several constraints is called an *active contour* or figuratively, a *snake*.

The kind of constraints which let the snake move (or more exactly the points) is a characteristic property of every snake algorithm. There exist different snake algorithms, which minimize the so-called *snake energy*. The snake energy represents the grade of the approximation of the object border, it can depend on image grey levels, curvature, smoothness, snake length etc. (see [16], [17]).

In this paper two calculation rules are presented for forcing a snake to approximate the contour of a selected object or the outer contour of an assembly of objects. Actually this second capability of extracting the contour of an object assembly characterises the novelty of the approach. The snake works on an image of flat scene objects. Therefore the border of the real 3D object can easily be reconstructed from the object contour in the image. It was intented to develop a system which works with nearly arbitrarily chosen initial snake points, to get a snake which "finds" the object even if the initial points are put far away from the desired object. A further goal was to close gaps in the outer contour of an object assembly if the space between two objects has to be bridged respectively.

2.3 Calculation rules

With regard to objects and object assemblies it is not possible for the snake to decide if a nonconvex object should be mould or a gap between two objects should be bridged. For this reason there are two different algorithms which serve the two contradictory requirements and the user has to select the proper algorithm.
The two calculation rules are based on the analysis of the 8-neighbourhoods of each snake point (Figure 7a). The algorithms run in a loop of iterations and in each iteration the sequence of all points is considered. Each point is moved in a way that the energy of the snake can be reduced maximally. The loop stops when the energy has converged to a minimal value.

2.3.1 Minimization of the snake length

Using the snake length as the main constraint for appoximating the object, it is obviously possible to close gaps in the contour, in the same way as to span a bigger distance between two objects (Figure 6). The minimization takes place in the following way: Start by taking two successive points P_{i-1} and P_i into consideration. Look for the point $P_{i_{min}}$ in the 8-neighbourhood of P_i which minimizes the euclidean distance to P_{i-1}. If the replacing of P_i by $P_{i_{min}}$ is done for all points of the sequence iteration by iteration the snake polygon begins to shrink. Actually, the points rotate around the object and come closer and closer (Figure 7b). However, the replacement of point P_i by $P_{i_{min}}$ only makes sense if the grey level jump is smaller than a given threshold.

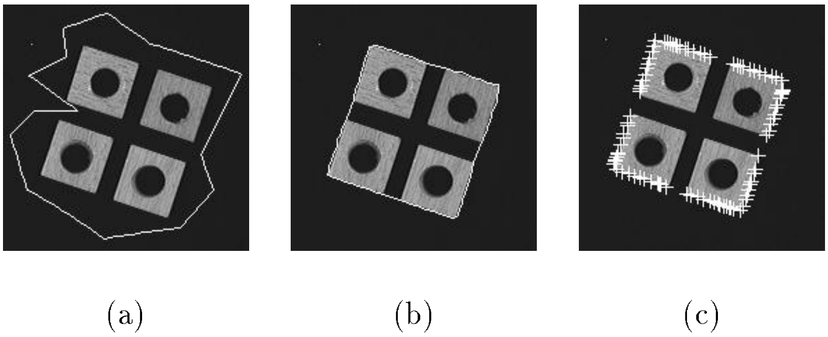
(a) (b) (c)

Figure 6: (a) start polygon, (b) end polygon, (c) distribution of the points at the end.

Applying this procedure the snake tends to approximate only convex objects or forms the convex hull of several included objects. The rotating behaviour of points can be used to make the algorithm more robust against salt and pepper disturbances in the image. Everytime when an edge is detected (because the grey level jump exceeds the threshold) a so-called hit is counted. Thus it can be introduced another parameter value for the maximum number of edge hits. A point does not have to stay constantly at an edge when

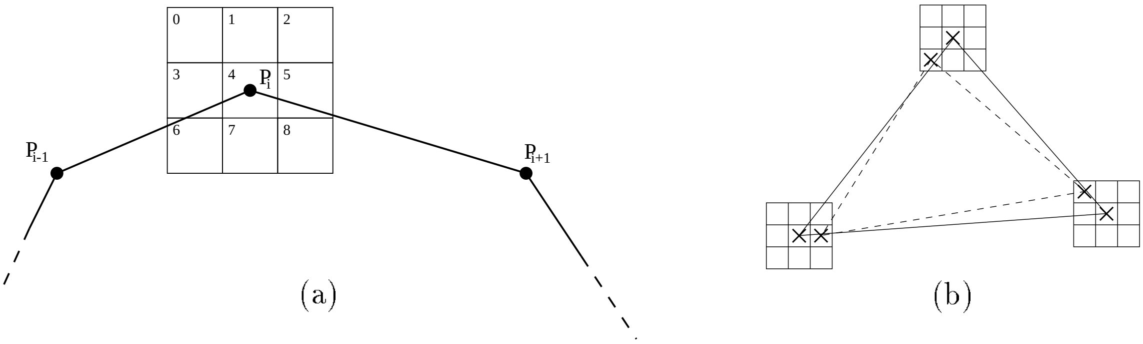

Figure 7: (a) 8-neighbourhood, (b) length minimization.

it is detected for the first time. Rather, in combination with the rotating behaviour a snake point can pass by isolated points in the image that do not belong to the desired contour. All together the snake points approach the object, run along the object contour and stay at remarkable places of the contour, e.g. edges (Figure 6c). To avoid too big gaps between points of the snake, a maximum value for the distance between the snake points can be used.

2.3.2 Minimization of the snake polygon area

As mentioned in the previous section, the minimization of the snake length is only a useful approach for an object assembly or a convex object. However, to mould nonconvex areas of an object a calculation rule is needed that works mainly by minimizing the area of the snake polygon. Instead of minimizing the whole area of the polygon many times in an iteration (equal to the number of snake points), it is more efficient to calculate the triangle areas (Figure 8a) spaned by the vectors

$$V_{1_i} := P_{i+1} - P_i \quad \text{and} \quad V_{2_i} := P_{i+2} - P_i, \quad \text{for} \quad P_i = \begin{pmatrix} x_i \\ y_i \\ 0 \end{pmatrix} \quad (\text{for} \quad i = 0, \cdots, n-2), \tag{10}$$

by using the following formula:

$$V_{triangle_i} := \frac{V_{1_i} \times V_{2_i}}{2}, \qquad A_{triangle_i} := ||V_{triangle_i}|| = |\pi_3(V_{triangle_i})|. \tag{11}$$

The × denotes the cross product and π_3 the projection on the third component of the vector. The $V_{triangle_i}$ is a 3-dimensional vector, and in our special case the first two components are obviously zero because V_{1_i} and V_{2_i} are laying in the x-y-plane.

The third component represents the area of the triangle and its sign is important for deciding if the triangle belongs to the polygon area or not. The sign depends on enumerating the points P_i clockwise or counterclockwise. For example the area of the triangle that is described by P_5, P_6, and P_7 in Figure 8b does not belong to the area of the polygon and its sign differs from the sign of all other triangle areas. So the area $A_{triangle_i}$ cannot be minimized because it is unknown whether $\pi_3(V_{triangle_i})$ is positiv or negativ. To manage

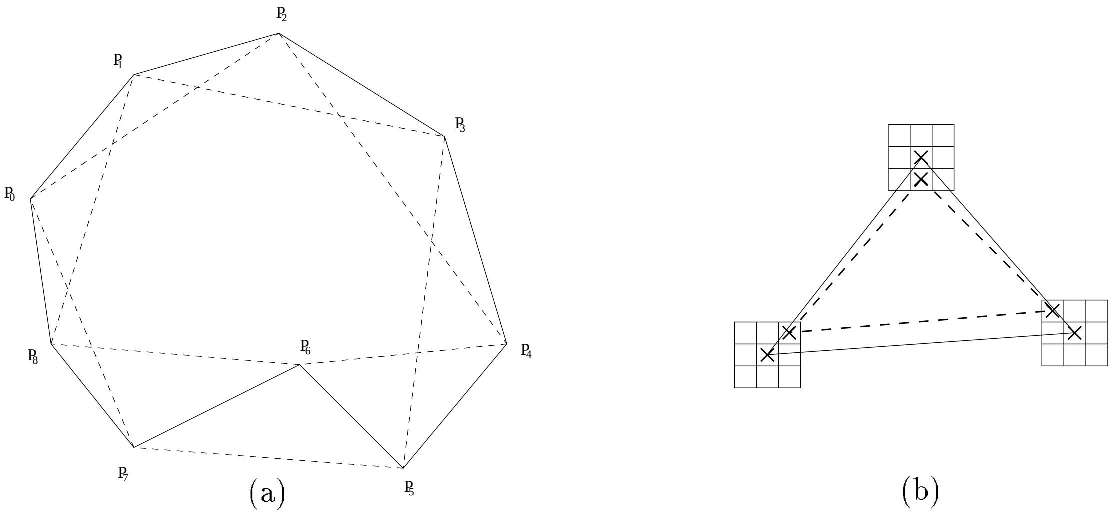

Figure 8: (a) single triangle minimization, (b) area minimization.

this problem a value σ is used,

$$\sigma := \operatorname{sgn}(\pi_3(\sum_{i=0}^{n-2} \begin{pmatrix} x_{i+1} - x_0 \\ y_{i+1} - y_0 \end{pmatrix} \times \begin{pmatrix} x_{i+2} - x_0 \\ y_{i+2} - y_0 \end{pmatrix})) \tag{12}$$

with which each single triangle area is normalized in some way. The sum of cross products calculates a vector with the (double) area of the snake polygon in its third component. The variable σ is positive if the points P_i of the polygon run counterclockwise. Consequently the expression

$$\sigma \cdot \pi_3(V_{triangle_i}), \tag{13}$$

is received, which describes one single triangle area that has to be minimized. That means, in every iteration of the algorithm it is tested which point in the 8-neighbourhood of P_i minimizes the area of the triangle area $\sigma\pi_3(V_{triangle_{i-1}})$ optimally. The grey level jump is tested in the same way as in the previous algorithm. The snake polygon shrinks in the way as shown in Figure 8b.

2.4 Experiments

We demonstrate exemplarily a combined use of the two calculation rules. The first calculation rule minimizes the contour length of an arch shaped object. Figure 9 shows the snake at every 10 iterations of the loop. The algorithm stops with a nearly rectangular contour, and all snake points are located at the convex part of the object border. If we now use these snake points as the initial distribution for the second calculation rule, also the nonconvex part of the object border can be approximated (before starting the algorithm the gap is automatically filled with equidistantly distributed points). Figure 10 shows the snake at every five iterations of this second loop.

The depicted objects are real objects and the image background has small grey level fluctuations. The initial distance of the snake points was five pixels and to detect edges a threshold of 20 grey levels was used. The maximum distance between a point and its successor was unlimited and the allowed number of edge hits was 25.

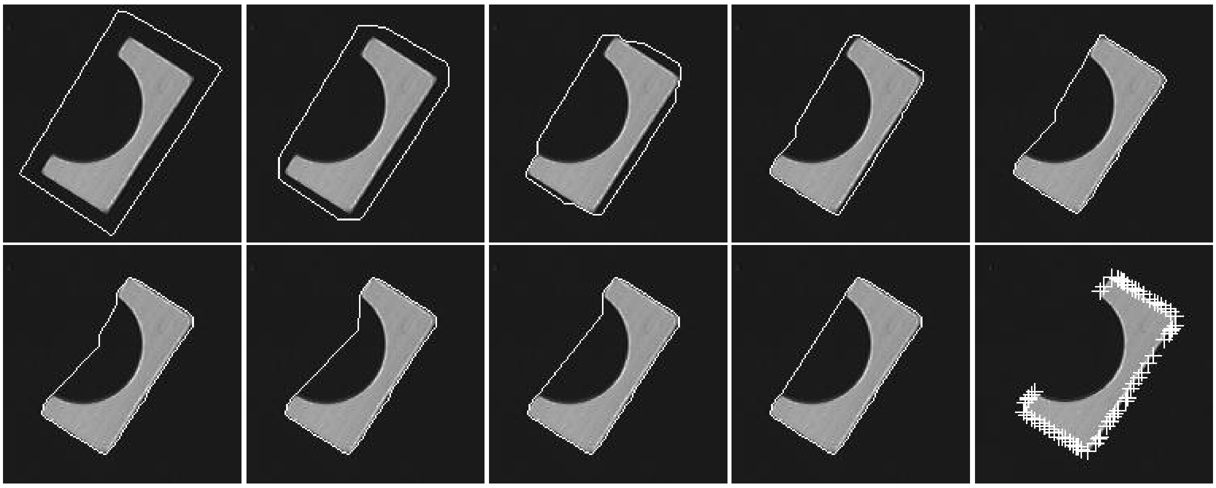

Figure 9: Approximation of the snake polygon to an object by length reduction, every image after 10 iterations. Upper left: initial polygon with equidistantly distributed points. Lower right: end positions of the snake points.

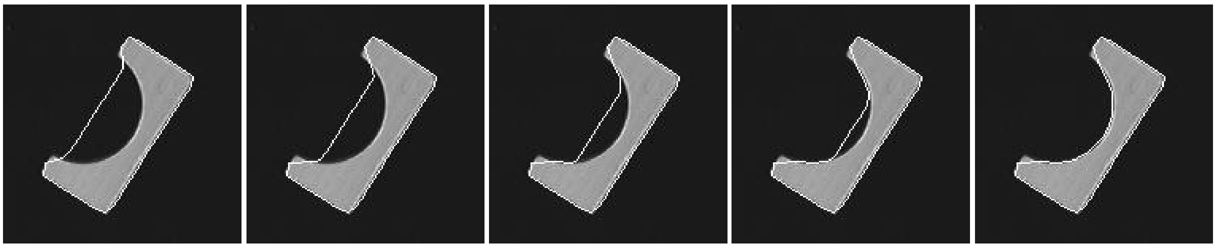

Figure 10: Minimization of the included area, starting with a polygon similar to the end polygon in figure 9 but with equidistantly distributed points.

2.5 Applications

Using a simple camera arrangement with the optical axis directed perpendicular to the flat objects, it is easy to reconstruct the image snake points into real world coordinates. Thus a manipulator end-effector can move along the boundaries of arranged objects, e.g. to weld the objects together.

2.6 Conclusions of the chapter

So far our snake algorithm does not make use of gradient based information. However, a gradient based algorithm like the one presented in [18] could optimize the grade of approximation. Therefore such an algorithm could use our final points as initial points. Rather, our snake uses a threshold for the grey level jump to detect edges if they are not too smooth. This strategy is useful if the initial contour is far away from the object border. Until now the initial snake points are set via mouse clicks. For an automatical setting several approaches are conceivable, e.g. learning a point distribution model (see [19]).

3 Vision based learning of gripper trajectories for a robot arm

3.1 Outline of the chapter

As opposed to the usual approach of programming robot arms by giving a complete description of the trajectory (e.g. *direct programming*) this work is based on the idea of *programming by demonstration.* Only few intermediate positions, which approximate the trajectory, are given and recorded by a stereo-camera-system. The idea is to extract the path of the manipulator gripper from the images by geometrical connecting this positions to form a smooth trajectory. This is done by tracking the appearance of the gripper in the sequence of stereo images. The gripper position in previous images and the gripper appearance are used for locating it in the next images. Based on this sequence of positions a *trajectory-structure* is formed, which allows to execute a vision based movement. The trajectory is general in the sense that the starting position and the orientation can be specified variable. The approach can be used in the assembly industry for approaching a working tool to a certain object and handling it (e.g. a screw spanner).

3.2 Introduction

This work is based on the idea of robot *programming by demonstration*, which means to indirect program a system by giving examples. Basic ideas and related work on this topic can be found in [20] and [9]. The operator uses the control panel to move the gripper in discrete steps to intermediate positions of the desired trajectory. A stereo-camera-system records this sequence and a computer-vision-system reconstructs a smooth 3D-trajectory. This trajectory only serves as an example for a whole class of trajectories having variable starting position and orientation. In the application phase this two unknowns have to be determined (e.g. manually by the operator or, perhaps, automatically with the help of a vision system). Furthermore, the information of the gripper trajectory can be used to anticipate collisions with unexpected objects in order to early interrupt the course. The gripper is tracked in the images by using its appearance in the previous image as decribed in section 3.3. Section 3.4 briefly explains the calibration used. In section 3.5 a *trajectory structure* is defined, which allows to execute a vision based movement. Figure 11 shows the whole system and the way the components depend on each other.

3.3 Tracking the gripper

To extract a series of gripper positions from a sequence of images, the gripper has to be located in every single image. To be more precise, in every single image a certain *reference point* of the gripper must be detected. Based on this precondition it is possible to consider the extracted positions as intermediate points of the movement.

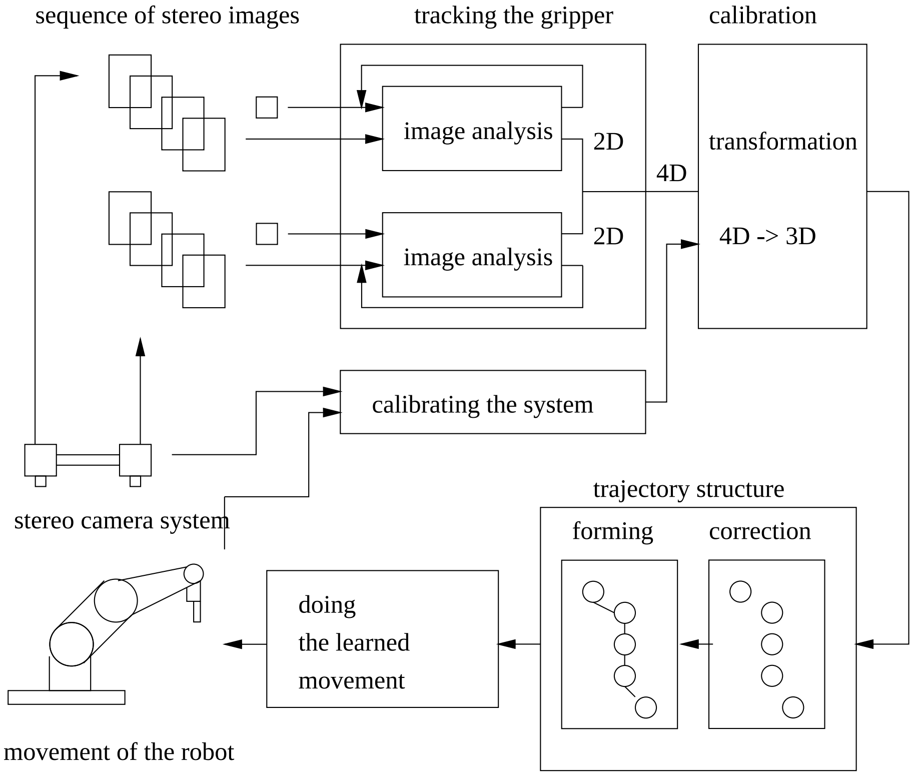

Figure 11: The components and their interaction.

A two step procedure is applied to extract the reference point from the images:

(i) roughly locate final segment of the gripper by using its appearence (greylevels) from previous image

(ii) exactly determine the reference point by geometrical analysis in the selected *gripper image region* (in the following named *patch*).

In figure 12 the procedure for tracking the gripper is illustrated graphically.

The input of the procedure is a sequence of stereo images representing the movement of the gripper in discrete steps. Furthermore, two image patches are supplied - one for each image sequence - which depict the appearance pattern of the gripper in the starting position of the movement. Both image sequences of the stereo cameras are analysed the same way but independently.

3.3.1 Locating the gripper region

The gripper image region is located by correlation matching using the expected gripper appearance (instead of using a model of the gripper). An $(m \times m)$-image B depicts the whole scene in which the robot arm is working and an $(n \times n)$-patch P contains the final segment of the gripper. Now a correlation image C is computed, by defining $C(k, l)$ as

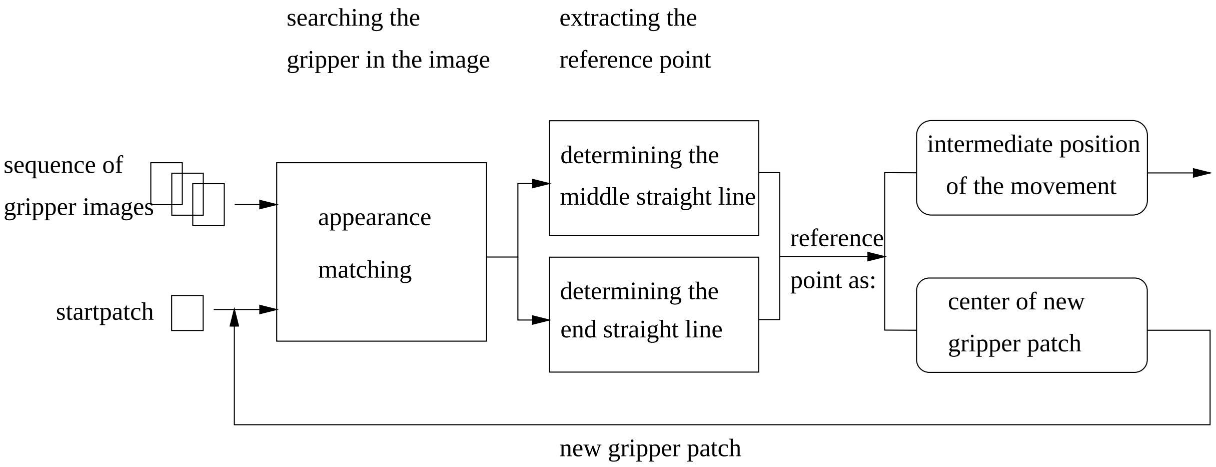

Figure 12: Tracking the gripper - a two step procedure.

sum of squared distances

$$C(k,l) = \sum_{\substack{i\in\left\{k-\frac{n}{2},\dots,k+\frac{n}{2}\right\}\\ j\in\left\{l-\frac{n}{2},\dots,l+\frac{n}{2}\right\}}} \Big(B(i,j) - P(i-(k-\tfrac{n}{2}), j-(l-\tfrac{n}{2})\Big)^2$$

for each image position (k,l).
Figure 13 shows a scene, the relevant gripper image region and the resulting correlation image (brighter greylevel indicates better correlation).

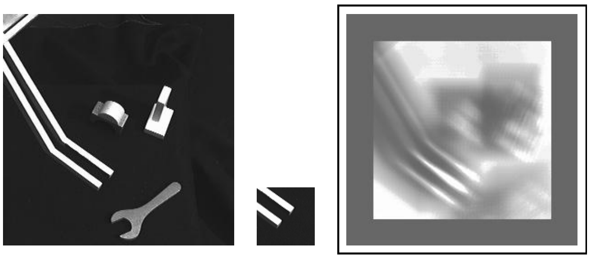

Figure 13: Scene, gripper image region and resulting correlation image.

The position of maximal correlation is looked for by starting the search in the position of the patch located in the previous image and expanding the catchment area (for reasons of efficiency). The position with the least sum of squared distances is expected to be the center of the relevant $(n \times n)$-region containing the tip of the gripper fingers.

3.3.2 Locating the reference point

Next, a certain reference point on the depiction of the gripper must be defined and located in the image as exact as possible. It will be used both as an intermediate position of the whole movement and as the center of the gripper image region for locating the gripper in the following image of the sequence. Figure 14 shows graphically the reference point of the gripper defined for this purpose. It is the virtual point of intersection between the *middle straight line* and the *end straight line* of the robot gripper. To extract these straight lines it is necessary to first recognize the top faces of the gripper.

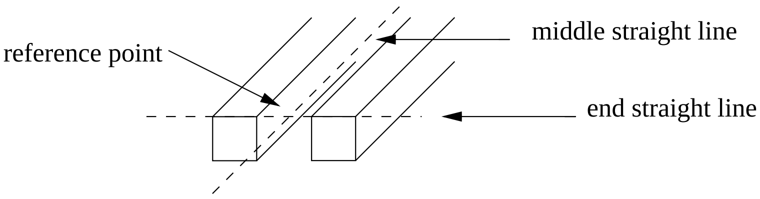

Figure 14: Definition of a reference point of the gripper.

Extracting the top faces of the gripper

This is done in two different ways using simple heuristics, depending on whether an object is grasped or not. In the case of a free gripper it is assumed that the image patch can be segmented into regions as follows. The background area of the gripper is approximately homogenous and therefore can be segmented in one region, which is expected to have the largest area of all regions. Furthermore, the gripper fingers are spacely disconnected, the top faces of the two gripper fingers are homogenous and can be segmented into one region for each, and they are the second and third largest areas. Figure 15 shows exemplarily the validity of these heuristics in the case of a free gripper. However, if an object is grasped these preconditions are no longer valid and that is why another heuristic is needed. It has to be mentioned, that the scene is lighted quite well and the gripper is made of reflecting material. This guarantees the top faces to be those parts of the image with the highest greylevel. Figure 16 illustrates the validity of these heuristic in the case of a grasped object.

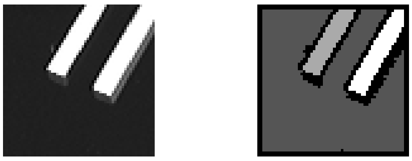

Figure 15: Gripper patch without object and its segmentation image.

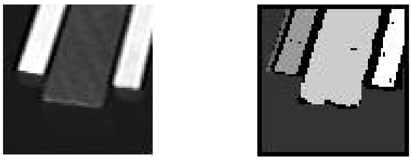

Figure 16: Gripper patch with grasped object and its segmentation image.

From the segmentation result a reference point of the gripper must be extracted as intersection between the middle straight and the end straight line.[1]

Detecting the middle straight line

For this a middle straight line is determined exactly between the extracted regions of the two gripper fingers which come from the top faces of the gripper. Each point on this middle straight line is characterized such that the Euclidean distance to both regions is equal. Alternatively, a city block metric, which computes distances only in x- respectively y-direction, works as well (see figure 17). If the distance in positive and negative x- (y-) direction is equal this point is added to a set M of points near or on the middle straight

[1] For other robotic equipment special attributes must be explored which are suitable to detect the gripper in the image (e.g. specific gripper color or an identity tag) and extract a certain gripper reference point.

line. The middle straight line is obtained by fitting a straight line through the points of M.

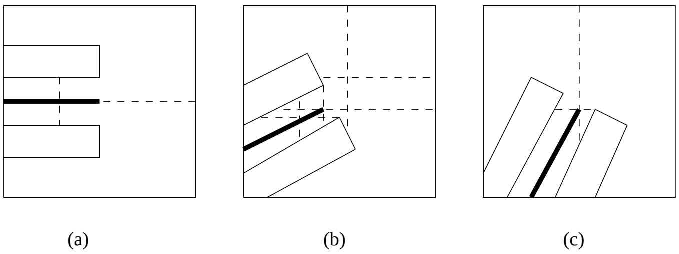

Figure 17: Determining points between top faces of fingers to extract a middle straight line.

Detecting the end straight line

First a set of relevant grey level edges is extracted from the gripper image region by computing the gradient magnitudes and setting a threshold. Care must be taken with this threshold to guarantee that at least two corner points of the final segment of the two gripper fingers are in the set of edges. In a second step the relevant corners have to be detected in the binary image of edges. A binary correlation mask (see figure 18) is used having edge points at the middle vertical axis. The mask is applied at every edge of the binary gripper image and rotated in discrete steps of angles in the interval of $[0, 360]$ degrees.

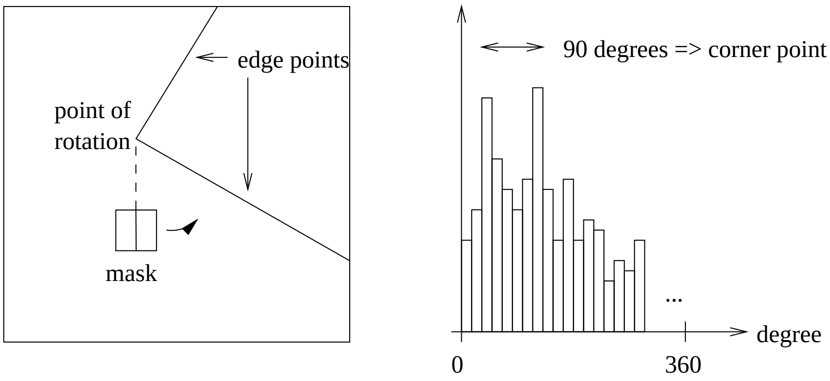

Figure 18: Rotating a squared mask to detect corners of the gripper fingers.

For each rotation step a value is retained which describes how many edges of the middle axis of the mask correspond to edges of the gripper. If two maximum peaks are found, being about 90 degrees apart from each other, then the rotation point belonging to that situation is taken as corner point. Only those points are considered, which are close to the top faces. The end straight line is obtained by fitting a straight line through these corner points.

3.4 Calibrating the camera system

Calibrating a stereo camera system means to find the relation between *image coordinates* (2×2D) and *world coordinates* respectively *robot coordinates*. Figure 19 shows the dependencies.

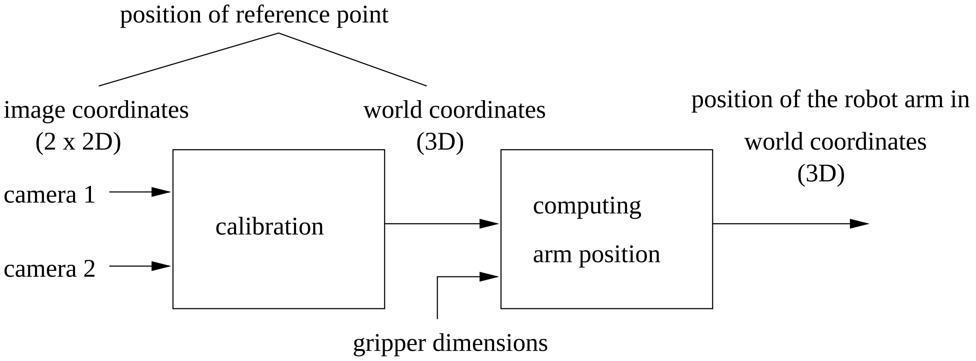

Figure 19: Transformation from 4D-image coordinates to 3D robot coordinates.

To obtain the function $calib : 4D \rightarrow 3D$ we use the technique proposed in [21]. For each camera a matrix

$$P_i = \begin{pmatrix} p^i_{11} & p^i_{12} & p^i_{13} & p^i_{14} \\ p^i_{21} & p^i_{22} & p^i_{23} & p^i_{24} \\ p^i_{31} & p^i_{32} & p^i_{33} & p^i_{34} \end{pmatrix}$$

$(i = 1, 2)$ is computed by using pairwise combinations of 3D world points and the 4D stereo points. The usage of the matrix is specified within the following context. Given a point in world coordinates (x_w, y_w, z_w) the position in image coordinates can be obtained by solving

$$\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \cdot \delta_i = P_i \cdot \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix},$$

where δ_i is an arbitrary scale factor. However, starting with a stereo image point (x_1, y_1, x_2, y_2) a transformation is needed for computing the world coordinates (x_w, y_w, z_w).
By combining P_1 and P_2 an overdetermined linear equation system ($\vec{b} = P\vec{a}$) is obtained. It can be solved for the vector $\vec{a}$ which contains the unknown 3D world coordinates and the two unknown scale factors by computing the pseudo inverse matrix of P. The matrix P and the vector $\vec{b}$ contain given calibration attributes and the given stereo image point for which the 3D world coordinates are requested.

$$\vec{a} = (P^T P)^{-1} \cdot P^T \cdot \vec{b}$$

In order to compute the calibration attributes a set of associations between 3D world points and 4D stereo image points is needed. For the purpose of this work it is possible to compute the relation between image and robot coordinates directly without an intermediate world coordinate system. Due to the dexterity of the robot manipulator only one

3D world point is needed namely the reference point of the gripper described in section 3.3. The manipulator is moved in discrete steps through the working space and along this course the reference point is recorded in 3D (known from the control unit of the robot) and is additionally detected in the stereo images to acquire the series of 4D stereo points.

3.5 Trajectory structure

Using the calibration result the reference point of the gripper can be reconstructed into 3D robot coordinates for arbitrary positions of the gripper in the working space. Furthermore, taking the same coordinate system into account, the position of obstacles could be determined by reconstruction from stereo data. Based on both, Euclidean relations between the robot gripper and obstacles could be represented. In order to learn a trajectory (possibly through a collection of obstacles) the system user is asked to demonstrate an example trajectory. For this, the user steers the gripper through the working space and stops at certain intermediate places, and the system computes a sequence of 3D positions $(x_1, y_1, z_1), ..., (x_n, y_n, z_n)$ extracted from the series of stereo images. This section describes how to acquire and use a smooth trajectory.

3.5.1 Definition of the trajectory structure

A trajectory structure will be defined by having in mind, that the operator should demonstrate just an example of a class of congruent trajectories. That is, during the application phase the example trajectory will be adapted to any desired starting position and orientation. Therefore, it must be possible to easy include the concrete starting position and orientation as soon as they become known. Accordingly, during the demonstration phase a trajectory structure will be constructed, which represents only the geometric relationship between intermediate points and not the absolute positions and orientations.
The trajectory structure is a sorted sequence $(k_1, ..., k_m)$ of nodes. Each node refers to an intermediate point of the trajectory and is defined by three components:

- vector $(v_x, v_y, v_z) \in I\!R^3$, written as $k_i.(v_x, v_y, v_z)$
- increment to the following node $(dx, dy, dz) \in [-1, 1]^3$, written as $k_i.(dx, dy, dz)$
- orientation $\varphi \in [0, 2\pi]$, written as $k_i.\varphi$

Vector $k_i.(v_x, v_y, v_z)$ decribes the relation between point (x_i, y_i, z_i) and (x_1, y_1, z_1), defined as

$$k_i.(v_x, v_y, v_z) = (x_i, y_i, z_i) - (x_1, y_1, z_1) \quad , \qquad \text{for all } \ j \in \{1, ..., n\}$$

The increments are given by

$$k_i.(dx, dy, dz) = k_{i+1}.(v_x, v_y, v_z) - k_i.(v_x, v_y, v_z)$$

for a node k_i $(i \in \{1, ..., n-1\})$.
The orientation φ of the gripper at node k_i can be chosen free. In the implementation reported here it should keep an orthogonal orientation to the tangent at every point of the trajectory.

3.5.2 Using the trajectory structure

To perform a movement the trajectory has to be defined absolutely by incorporating the starting point (x_s, y_s, z_s) and three rotation angles $\varphi_{xy}, \varphi_{xz}, \varphi_{yz}$ (describing the rotation in the xy-, xz- and yz-coordinate-plane). The structure is moved into the starting position, which means every node is translated with (x_s, y_s, z_s), and then it is rotated as defined by $\varphi_{xy}, \varphi_{xz}, \varphi_{yz}$. Figure 20 shows a simple example (for simplicity in 2D).

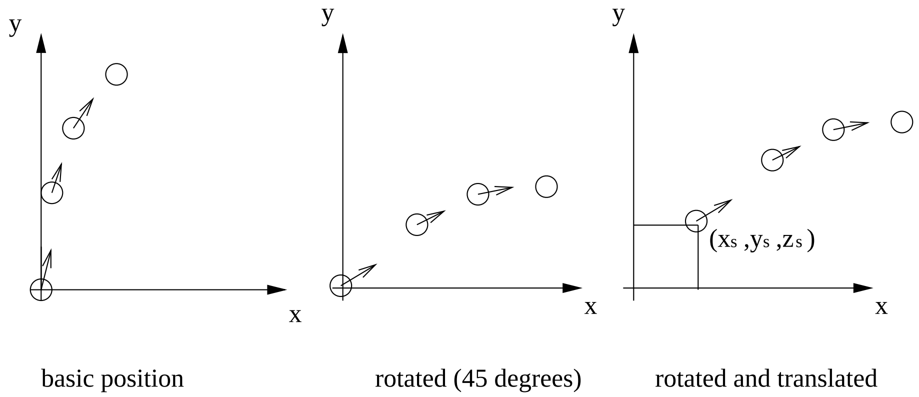

Figure 20: Moving the trajectory structure in the starting position and orientation.

When the robot starts to move the fingers beginning from (x_s, y_s, z_s) the second node works as an *attractor* until it is reached. Now the next node comes into play and the procedure is repeated for all nodes of the trajectory structure. A criterion is needed to test, whether a node is passed or not. For this a plane defined by the position of the node $k_i.(v_x, v_y, v_z)$ and the stored increment $k_{i-1}.(dx, dy, dz)$ is used. When the robot continues to move the fingers it is tested node by node whether the respective plane is passed. For a position (x_p, y_p, z_p) of the fingers the difference vector $(x_d, y_d, z_d) = (x_p, y_p, z_p) - k_i.(v_x, v_y, v_z)$ and the scalar product s between (x_d, y_d, z_d) and $k_{i-1}.(dx, dy, dz)$ is calculated. There are three cases to deal with:

- $s > 0 \Longrightarrow -90 < |\angle((x_d, y_d, z_d), k_{i-1}.(dx, dy, dz))| < 90$
- $s = 0 \Longrightarrow |\angle((x_d, y_d, z_d), k_{i-1}.(dx, dy, dz))| = 90$
- $s < 0 \Longrightarrow 90 < |\angle((x_d, y_d, z_d), k_{i-1}.(dx, dy, dz))| < 270$

If $s < 0$ then the fingers have not yet reached the plane belonging to a certain point of the trajectory, if $s = 0$ then the plane is reached, and $s > 0$ means that it is passed.
Figure 21 illustrates this criterion. Finally the orientation of the gripper at a certain point on the trajectory has to be found. This is done by first calculating the distances to attracting node k_i and preceding node k_{i-1} as

$$dist_v = \sqrt{(x_p - k_{i-1}.v_x)^2 + (y_p - k_{i-1}.v_y)^2 + (z_p - k_{i-1}.v_z)^2}$$

and

$$dist_a = \sqrt{(x_p - k_i.v_x)^2 + (y_p - k_i.v_y)^2 + (z_p - k_i.v_z)^2}$$

and then computing the orientation of the gripper as weighted mean of $k_{i-1}.\varphi$ and $k_i.\varphi$:

$$\varphi' = \frac{dist_a \cdot k_{i-1}.\varphi + dist_v \cdot k_i.\varphi}{(dist_a + dist_v)}$$

Figure 22(a) shows a simple example of the path from node to node. Furthermore, figure 22(b) shows an interesting behavior if a certain intermediate point can not be reached exactly, maybe due to an obstacle at that place. In this case the subsequent node attracts the gripper and thus it comes back to the original trajectory. Therefore, the course of the robot gripper can partially deviate from the learned trajectory at local areas due to certain requirements and adaptively return back to the target trajectory afterwards.

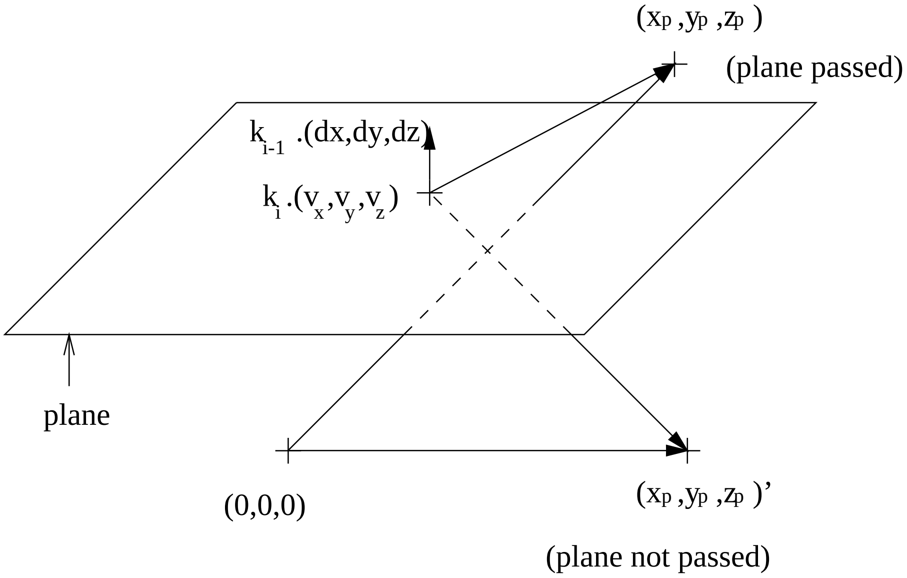

Figure 21: Testing whether a position of the demonstrated trajectory is passed.

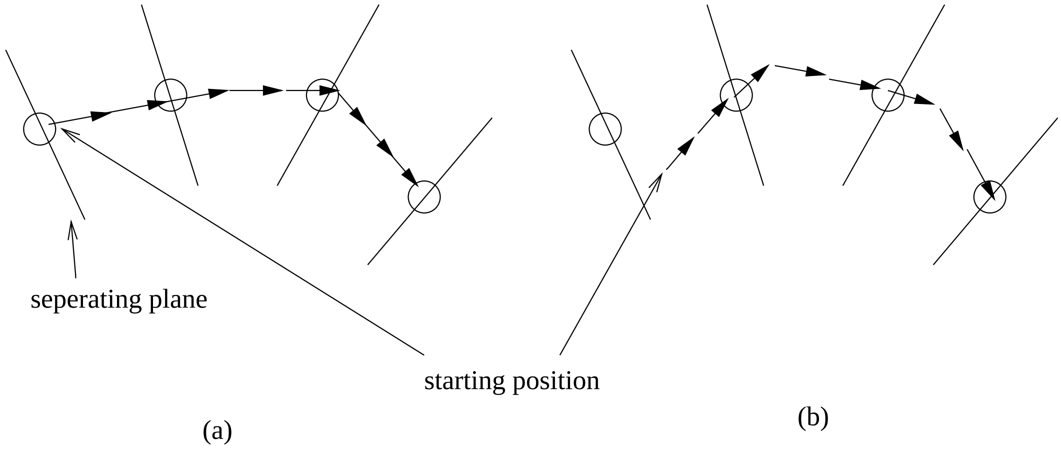

Figure 22: Example showing the resulting movements beginning in the starting point (a) or somewhere beside of it (b).

3.5.3 Smoothing

Usually the positions extracted from the images are unprecise, because of resolution limit of the image or errors in calibration. To get a smooth trajectory a simple method can be used by considering the position of the neighbouring nodes. For a node k_i the position is adjusted by first calculating the distance between k_{i-1} and k_{i+1} as

$$d_1 = \sqrt{(k_{i+1}.v_x - k_{i-1}.v_x)^2 + (k_{i+1}.v_y - k_{i-1}.v_y)^2 + (k_{i+1}.v_z - k_{i-1}.v_z)^2}$$

Then $$(x_m, y_m, z_m) = k_{i-1}.(v_x, v_y, v_z) + \frac{d_1}{2} \cdot k_{i-1}.(dx, dy, dz)$$

is the point in the middle position between k_{i-1} and k_{i+1}.
The distance between $k_i.(v_x, v_y, v_z)$ and (x_m, y_m, z_m) is

$$d_2 = \sqrt{(k_i.v_x - x_m)^2 + (k_i.v_y - y_m)^2 + (k_i.v_z - z_m)^2}$$

To do the adjustment two parameters $\varepsilon_1, \varepsilon_2 \in [0, 1]$ are defined. ε_1 defines whether a correction has to be done (e.g. in the case of large distances between points) and ε_1 says *how strong* it should be done. With $(x_d, y_d, z_d) = (k_i.v_x - x_m, k_i.v_y - y_m, k_i.v_z - z_m)$ the new position is

$$k_i.(v_x, v_y, v_z)' = \begin{cases} k_i.(v_x, v_y, v_z) + \varepsilon_2 \cdot (x_d, y_d, z_d) & , \text{ if } \quad d_2 \geq \varepsilon_1 \cdot d_1 \\ k_i.(v_x, v_y, v_z) & , \text{ otherwise} \end{cases}$$

This adjustment is done for the position of every single node starting with k_2.

3.6 Experiments

The experiments have been carried out with an industrial articulation robot having six rotational degrees of freedom and a two-fingered gripper. Image processing is done in the KHOROS environment on a SUN workstation 10/40. Exemplarily, the system was engaged to learn to handle a screw spanner. This means that a course of the manipulator must be learned such that the screw head will be turned around in a circle. In order to automatically learn the required trajectory of the robot gripper the system user steered the robot in 30 discrete steps of 3 angle degrees and according to that a quarter of a circle is approximated. Figure 23 shows for three intermediate steps on this course some processing results involved in visually evaluating the trajectory of the gripper fingers. The three images in the first row depict these intermediate steps. The second row shows pairwise the located regions of the finger tips of the gripper and corner points of the fingers. Finally, the images in the third show the result of computing the middle straight line and the end straight line of the fingers and the intersection point between both which is used as reference point. This point is extracted in all 90 images of the quarter cycle demonstration and a smooth gripper trajectory is approximated and reconstructed thereof. To get an impression for the accuracy of the learned movement the deviation from the radius has been measured. This deviation was at most 3 mm for a screw with head radius 15 mm.

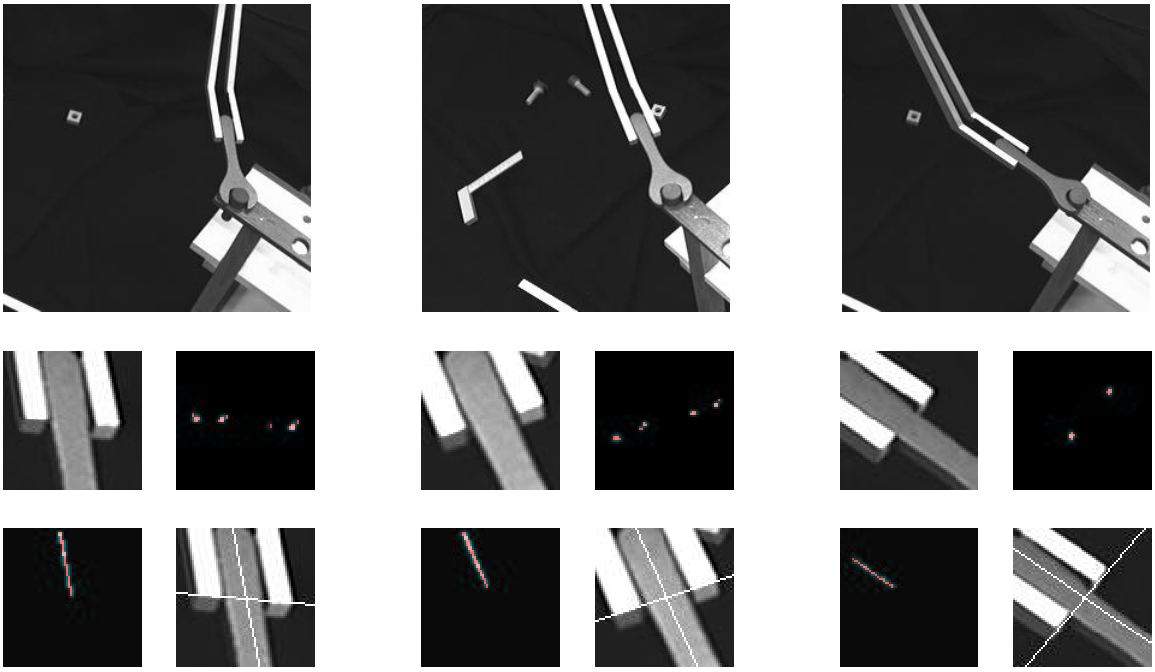

Figure 23: Intermediate steps of handling a screw spanner, and visual processing.

3.7 Conclusions of the chapter

The proposed framework enables an operator to program a robot by just giving intermediate positions recorded by a stereo camera system. For the operator there is no need for exact knowledge of how to program the robot. Instead, it is possible to use the example trajectory in various starting position and orientation by simple translation and rotation of it. Furthermore, it is possible to involve a computer vision system, which analyses the sequence of stereo positions for an early avoidance of collisions with new objects in the scene. The ability to react on unexpected appearance of obstacles and return to the original trajectory is one of the most important features of robot systems to be used in praxis.

4 Vision based manipulator navigation using RBF networks

4.1 Outline of the chapter

This work reports on close range manipulator navigation, i.e. searching collision-free trajectories of the robot hand to approach and handle goal objects. Neural network learning with *radial basis functions (RBFs)* is involved twofold. First, a function is learned for reconstructing from the optical flow of detected obstacle points their three-dimensional positions. Second, a function of the inverse manipulator kinematics is learned which is used for describing the non-rigid space occupied by the agile manipulator. Furthermore, based on the goal position and the continually detected obstacle positions a vector field is created dynamically by using the gradient of RBFs as basis fields. The vector field simulates attracting and repelling forces for navigating the manipulator hand. To overcome the curse of dimensionality and reach acceptable efficiency in function learning we applied mixtures of RBF neural networks and strongly emphasized divide-and-conquer strategies. The parallel approaches for neural learning (and image processing) are implemented on a four-processor workstation.

4.2 Introduction

In our behaviour-based robot system the manipulator has been equipped with a monochromatic video camera, fastened onto the robot hand. During the goal-oriented movement of the camera through the working space the system must detect obstacles continually. The SUSAN edge detector [22] is used for extracting greylevel corners probably arising from obstacles. Based on corresponding features between two successive images the obstacle position must be reconstructed into 3D space.

4.3 Learning to reconstruct from optical flow

The reconstruction function is learned offline using a *hierarchical mixture of expert (HME) networks* [23] in which RBF networks are arranged in two layers. Figure 24 shows the application of such a mixture of networks for reconstruction from optical flow vectors. Each RBF network of the first layer is trained for a small image area and is used for reconstructing from the optical flow therein the depth coordinate Z. Each RBF network in the second layer is trained for a small range of depth, i.e. ranges of Z, and is used for computing the space coordinates X and Y. The merit of this architecture is twofold. First, the non-linearity of the RBFs takes care for the nonlinear type of reconstruction which is due to significant image distortions. These distortions are a consequence of using a wide-angle objective (lens with small focal length, e.g. $4.2mm$) needed in close range navigation for depicting a wide view volume. Second, the modular architecture makes it possible to train each network efficiently by taking only a small subset of the whole training set into account. The output on each of the two layers is calculated by linear combining the respective outputs of a small set of relevant RBF networks. The combination factors are supplied by one gating network for each layer (not shown in the figure), which are pre-specified in our application.

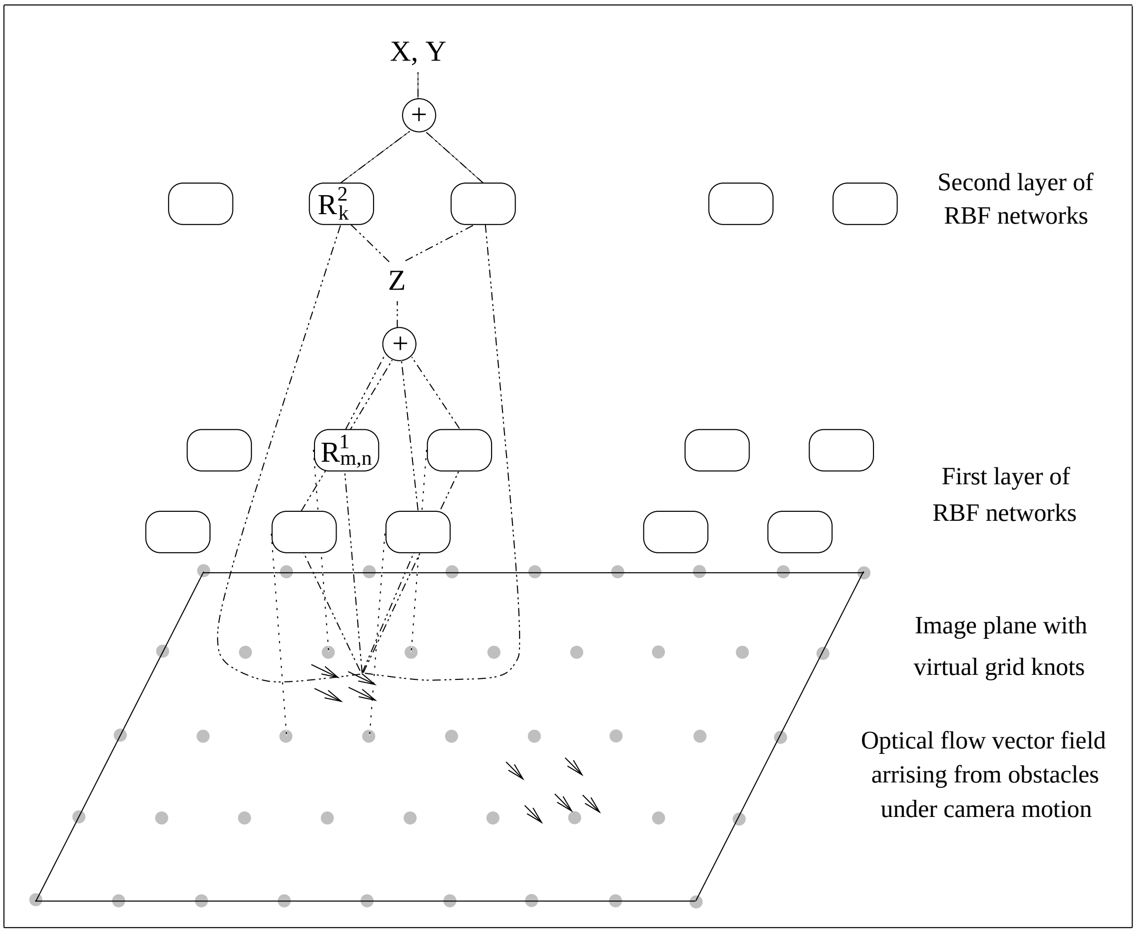
X, Y
+
R_k^2
Second layer of RBF networks
Z
+
$R_{m,n}^1$
First layer of RBF networks
Image plane with virtual grid knots
Optical flow vector field arrising from obstacles under camera motion

Figure 24: Hierarchical mixture of RBF networks, e.g. two layers.

The reconstruction function f_{rec} is learned as follows.
1. A sheet of paper depicting a regular distributed set of calibration dots is put at a fixed place of a ground plane. Beginning in a near position the camera is moving off the sheet in discrete steps with the optical axis approximately normal to it (e.g. 10 steps of $50mm$ each). At each step an image is taken and the calibration points are extracted. Furthermore these points are determined in the coordinate system of the robot hand which is translating step by step.
2. For every two successive camera positions the corresponding image positions of the calibration points are associated with the Z coordinates of their 3D positions (i.e. relative to the second of the two camera positions). A regular grid is defined for the image plane and one RBF network created for each grid knot respectively (first layer in the HME network). Each RBF network is trained efficiently by using a small set of calibration points (more conretely using the flow vectors) located in the neighborhood of the respective grid knot. The ISODATA clustering algorithm is used for defining the hidden nodes and a singular value decomposition (SVD) applied for determining the weights.

3. For each discrete camera position the image positions of the calibration points are associated with the (X, Y) coordinates of their 3D positions. One RBF network is defined with respect to each discrete camera position (second layer of the HME network). These RBF networks are trained (using ISODATA and SVD) by taking the respective associations into account.

Let (x_1, y_1) and (x_2, y_2) be corresponding positions of an obstacle in the image before and after camera motion. The second position is used to determine four neighboring grid knots $g_{m,n}$, $g_{m+1,n}$, $g_{m,n+1}$, $g_{m+1,n+1}$. The respective RBF networks $R^1_{m,n}$, $R^1_{m+1,n}$, $R^1_{m,n+1}$, $R^1_{m+1,n+1}$ of the first HME network layer is applied to the optical flow $(x_2 - x_1, y_2 - y_1)$. The linear combined output gives the depth coordinate Z. Just this coordinate is used for selecting two RBF networks R^2_k, R^2_{k+1} from the second layer, which are most sensitive to Z. They are applied to (x_2, y_2) and the combined output gives the coordinates X and Y of the obstacle.

Figure 25 shows two drinks cans (left and middle) and a beer bottle (right) stored in a refrigerator. The manipulator has to approach the goal object (assuming the can in the middle) by bypassing the obstacle objects (left can and bottle). The SUSAN edge detector has extracted a set of interesting points (see white dots) arising from the imprints of the three objects. Figure 26 shows for these detected image points the 3D reconstruction using the mentioned HME network. The X and Z coordinates are shown for points on the goal object (G) and obstacle objects $(H_1$ and $H_2)$.

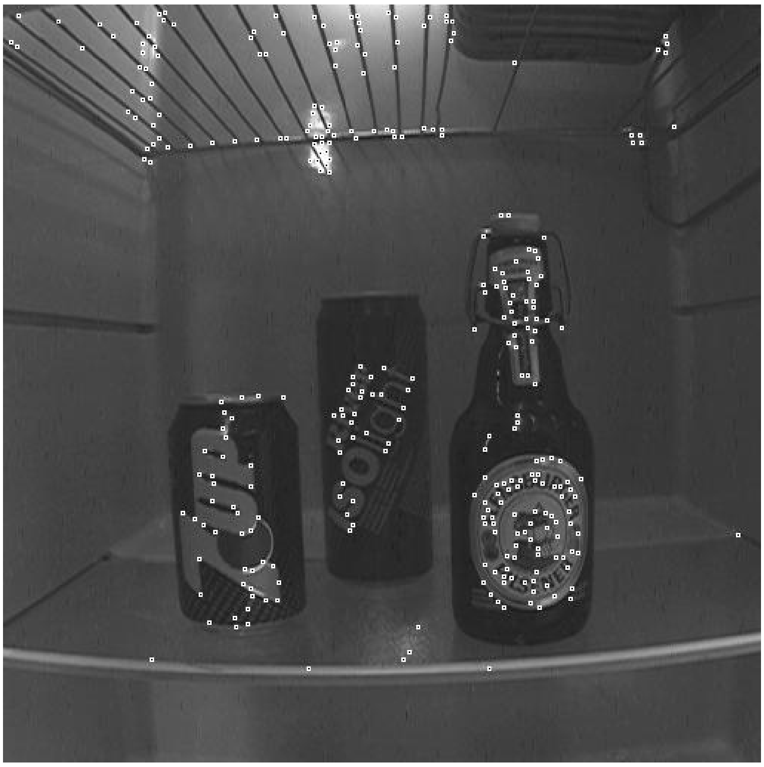

Figure 25: Detected grey level corners at two obstacle objects (left and right) and one goal object (middle).

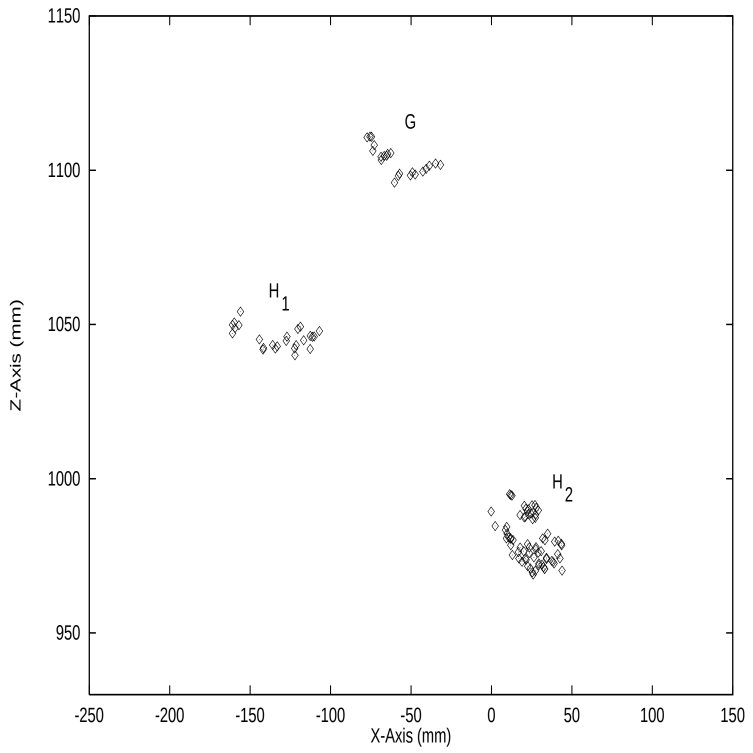

Figure 26: Reconstructed 3D coordinates X and Z of detected points from obstacles H_1, H_2 and goal G.

4.4 Learning the inverse manipulator kinematics

Suppose the manipulator must approach a goal position G, but in close neighborhood an obstacle H has been detected. Before approaching the goal it must be determined whether the arm segments will probably collide with H (see figure 27). This is done by just simulating a movement to G and there describing the occupied space of the manipulator. Figure 28 shows the rotation angle ω_i and position p_i of the joints, and the length l_i and diameter d_i of the links. Assuming that l_i, d_i are known a priori and p_i are computable from l_i and ω_i we easily compute an approximation of the occupied space V_m of the manipulator. Finally it must be checked whether obstacle H is contained in the virtual manipulator space V_m, and if this is not the case, the manipulator actually can appraoch goal G.
The only problem is to solve the inverse manipulator kinematics [24], i.e. determine the mapping of the 3D goal position $G = (X, Y, Z)$ into the relevant vector of rotation angles $\Omega = (\omega_1, \cdots, \omega_n)$. We build one layer of RBF networks in which each network is responsible for a certain range of Z and in consequence of that the dimension of the input space is reduced into 2D by dropping the Z component. Each RBF network is trained with associations between vectors G and Ω, taking only vectors G with relevant Z values into account. The efficiency of training arises by taking for each RBF network only a subset of the whole training samples into account. In the application phase we determine for an input vector G those RBF networks whose responsible ranges contain the Z value (e.g. two or more networks), apply these networks to the (X, Y) tuple, linear

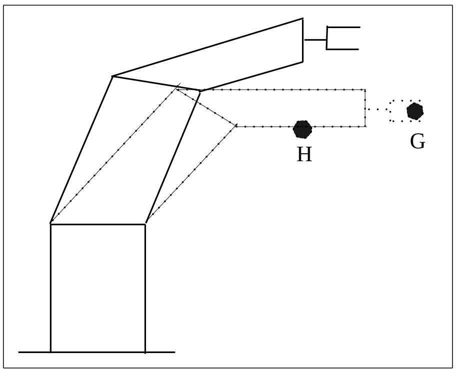

Figure 27: Manipulator and simulated obstacle collision.

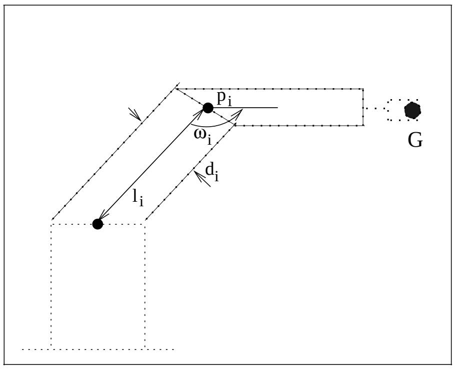

Figure 28: Characterizing the manipulator kinematics.

combine the respective outputs, and this gives the vector of rotation angles Ω. According to our experiments the approximation errors in the rotation angles ω_i can be reduced to 1° degrees in the mean. Applying forward kinematics this results in a mean positioning error of about $3mm$ which is good enough for checking criteria of obstacle avoidance.

4.5 Dynamic construction of a force vector field

The manipulator must navigate towards a goal position while avoiding obstacles. This is done by dynamically constructing a vector field of simulated forces [25]. The goal position is the center of an attractor field, i.e. in a working space all discretized points specify the origin of a vector which is directed towards the goal (Figure 29, left).

As the manipulator begins to move from an arbitrary position it will be attracted from the forces in the goal. Whenever the vision system detects an obstacle a repellor field is created at that position (Figure 29, middle). The summation of attractor and repellor field results in appropriate forces, i.e. the manipulator will be pushed off and thus bypasses the obstacle for approaching the goal (Figure 29, right).
The attractor field is simply defined by vectors of unique length α.

$$AF_G(P) := \alpha \frac{(G - P)}{\parallel G - P \parallel} \tag{14}$$

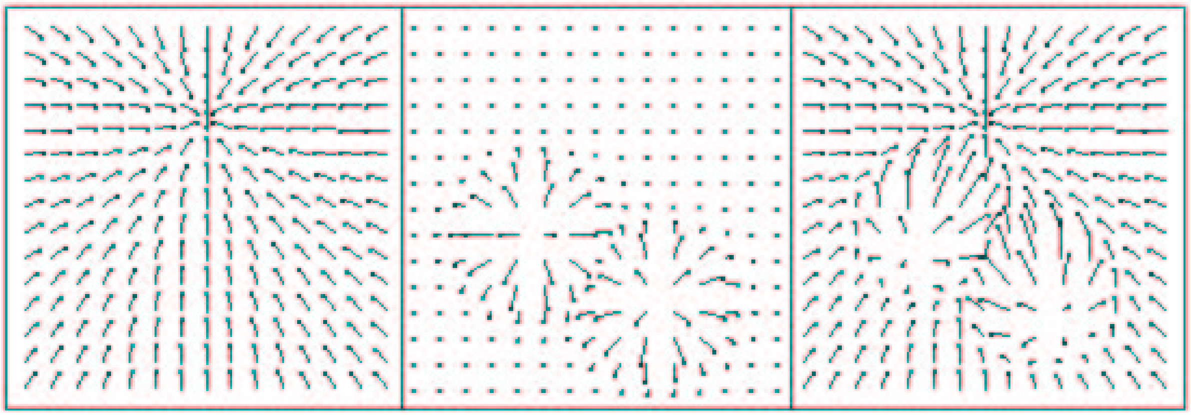

Figure 29: Attractor field for the goal object (left), two repellor fields for two obstacle objects (middle), summation of both fields (right).

The repellor field is defined by computing the gradient of a negated radial basis function centered at an obstacle position.

$$\Phi_\sigma(P, H) := -exp(-\frac{\| P - H \|^2}{\sigma^2}) \tag{15}$$

$$RF_H(P) := \frac{\partial \Phi_\sigma(P, H)}{\partial P} = 2(P - H)\Phi_\sigma(P, H) \tag{16}$$

The unknown σ value of the gaussian is computed by considering a desired minimal distance from the obstacles and taking the (small) inaccuracy of reconstruction into account. In order to exploratory navigate towards the goal position the manipulator is moving step by step, and the vision component is detecting obstacles. In these cases repellor basis fields are constructed, and the force vector field is changed dynamically. During the process locally a set of null vectors can arise which is similar to a local minimum in a potential field. These places are simply treated as obstacles, i.e. putting repellor fields there in order to generate repellent forces. The emerging vector field implicit represents a trajectory towards the goal position. For globally exploring the scene the navigation is repeated for different starting positions and thus an overall force vector field is constructed which implicit represents trajectories towards the goal position starting arbitrarily.

4.6 Conclusions of the chapter

Mixtures of RBF neural networks have been used for vision-based manipulator navigation. A two-layer mixture of RBF networks is appropriate for reconstructing 3D positions of obstacles especially for the case of significant image distortions which result from wide-angle objectives. A one-layer mixture of RBF networks is involved for efficiently solving the inverse manipulator kinematics, which is important for computing the occupied space of the manipulator. A force vector field is constructed dynamically by detecting obstacles and placing repellor fields, which are specified by the gradient of negated RBFs.

Acknowledgment: Many thanks to Prof. Sommer, the head of the Cognitive Systems Group Kiel, for continuous discussions and support.

References

[1] S. Kunze. Visuell basiertes Anordnen von Werkstücken mit dem Greifer eines Roboterarmes. Studienarbeit, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, Lehrstuhl Kognitive Systeme, 1996.

[2] S. Kunze and J. Pauli. A vision based robot system for arranging technical objects. In *International Symposium on Automotive Technology and Automation (ISATA)*, pages 119–124. Florence, Italy, Automotive Automation Limited, Croydon, 1997.

[3] F. Lempelius. Visuell basiertes Abfahren der Aussenkonturlinie von Szeneobjekten mit dem Endeffektor eines Roboterarmes. Studienarbeit, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, Lehrstuhl Kognitive Systeme, 1997.

[4] F. Lempelius and J. Pauli. Active contour based object detection. In V. Krasnoproshin, J. Soldek, S. Ablameyko, and V. Shmerko, editors, *International Conference on Pattern Recognition and Information Processing (PRIP)*, pages 171–175. Minsk, Belarus, Politechniki Publishing Szczecin, Poland, 1997.

[5] M. Päschke. Visuell basiertes Erlernen von Trajektorien eines Roboterarmes. Studienarbeit, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, Lehrstuhl Kognitive Systeme, 1996.

[6] J. Pauli and M. Päschke. Learning manipulator behaviors based on visual demonstration. In *Workshop zur Selbstorganisation von Adaptivem Verhalten (SOAVE)*, volume 663 of *Mess-, Steuerungs- und Regelungstechnik*, pages 212–221. Ilmenau, Germany, VDI-Verlag, Düsseldorf, 1997.

[7] W. Blase. RBF basierte Neuronale Netze zum automatischen Erwerb hindernisfreier Manipulator-Trajektorien, basierend auf monokularen Bildfolgen. Diplomarbeit, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, Lehrstuhl Kognitive Systeme, 1998.

[8] W. Blase and J. Pauli. Vision-based manipulator navigation using mixtures of RBF neural networks. In *International Conference on Neural Networks and Brain*, pages 531–534. Peking, China, 1998.

[9] H. Friedrich, S. Münch, and R. Dillmann. Robot programming by demonstration - Supporting the induction by human interaction. *Machine Learning*, 23:163–189, 1996.

[10] H. Murase and S. Nayar. Visual learning and recognition of 3D objects from appearance. *International Journal of Computer Vision*, 14:5–24, 1995.

[11] G. Cottrell, B. Bartell, and C. Haupt. Grounding meaning in perception. In H. Marburger, editor, *14th German Workshop on Artificial Intelligence*, volume 251 of *Informatik Fachberichte*, pages 307–321, 1990.

[12] G. Flammia. Dynamic programming solves approximate planar matching. In P. Johansen and S. Olsen, editors, *Theory and Applications of Image Analysis*, pages 99–109. World Scientific, Singapore, 1992.

[13] S. Dubois and F. Glanz. An autoregressive model approach to two-dimensional shape classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:55–66, 1986.

[14] A. Blumenkraans. Two-dimensional object recognition using a two-dimensional polar transform. *Pattern Recognition*, 24:879–890, 1991.

[15] E. Arkin, L. Chew, D. Huttenlocher, K. Kedem, and J. Mitchel. An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:209–215, 1991.

[16] M. Kass, A. Witkin, and D. Terzopoulos. Active contour models. *International Journal of Computer Vision*, 1:321–331, 1987.

[17] D. Williams and M. Shah. A fast algorithm for active contours and curvature estimation. *Computer Vision, Graphics, and Image Processing – Image Understanding*, 55:14–26, 1992.

[18] D. Young. Active contour models (snakes). http://www.cogs.susx.ac.uk /users/davidy/teachvision/vision7.html, 1995. Lecture at the University of Sussex.

[19] T. Cootes, C. Taylor, D. Cooper, and J. Graham. Active shape models - Their training and application. *Computer Vision and Image Understanding*, 61:38–59, 1995.

[20] R. Heise. Programming robots by example. Technical Report 92/476/14, University of Calgary, Department of Computer Science, 1992.

[21] O. Faugeras. *Three-Dimensional Computer Vision.* The MIT Press, Cambridge, Massachusetts, 1993.

[22] S. Smith and J. Brady. SUSAN: A new approach to low level image processing. *International Journal of Computer Vision*, 23:45–78, 1997.

[23] M. Jordan and R. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.

[24] J. Craig. *Introduction to Robotics.* Addison-Wesley Publishing Company, Massachusetts, 1989.

[25] F. Mussa-Ivaldi and S. Giszter. Vector field approximation – a computational paradigm for motor control and learning. *Biological Cybernetics*, 67:491–500, 1992.