Examensarbete

Visual Servoing Based on Learned Inverse Kinematics

Examensarbete utfört i Bildbehandling
vid Tekniska högskolan i Linköping
av

Fredrik Larsson

LITH-ISY-EX--07/3929--SE
Linköping 2007
Visual Servoing Based on Learned Inverse Kinematics

Examensarbete utfört i Bildbehandling
vid Tekniska högskolan i Linköping
av

Fredrik Larsson

LITH-ISY-EX--07/3929--SE

Handledare: Erik Jonsson
ISY, Linköpings universitet

Examinator: Michael Felsberg
ISY, Linköpings universitet

Linköping, 12 March, 2007
Initially an analytical closed-form inverse kinematics solution for a 5 DOF robotic arm was developed and implemented. This analytical solution proved not to meet the accuracy required for the shape sorting puzzle setup used in the COSPAL (COgnitiveSystems using Perception-Action Learning) project [2]. The correctness of the analytic model could be confirmed through a simulated ideal robot and the source of the problem was deemed to be nonlinearities introduced by weak servos unable to compensate for the effect of gravity. Instead of developing a new analytical model that took the effect of gravity into account, which would be erroneous when the characteristics of the robotic arm changed, e.g. when picking up a heavy object, a learning approach was selected. As learning method Locally Weighted Projection Regression (LWPR) [27] is used. It is an incremental supervised learning method and it is considered a state-of-the-art method for function approximation in high dimensional spaces. LWPR is further combined with visual servoing. This allows for an improvement in accuracy by the use of visual feedback and the problems introduced by the weak servos can be solved. By combining the trained LWPR model with visual servoing, a high level of accuracy is reached, which is sufficient for the shape sorting puzzle setup used in COSPAL.
Abstract

Initially an analytical closed-form inverse kinematics solution for a 5 DOF robotic arm was developed and implemented. This analytical solution proved not to meet the accuracy required for the shape sorting puzzle setup used in the COSPAL (COgnitiveSystems using Perception-Action Learning) project [2]. The correctness of the analytic model could be confirmed through a simulated ideal robot and the source of the problem was deemed to be nonlinearities introduced by weak servos unable to compensate for the effect of gravity. Instead of developing a new analytical model that took the effect of gravity into account, which would be erroneous when the characteristics of the robotic arm changed, e.g. when picking up a heavy object, a learning approach was selected.

As learning method Locally Weighted Projection Regression (LWPR) [27] is used. It is an incremental supervised learning method and it is considered a state-of-the-art method for function approximation in high dimensional spaces. LWPR is further combined with visual servoing. This allows for an improvement in accuracy by the use of visual feedback and the problems introduced by the weak servos can be solved. By combining the trained LWPR model with visual servoing, a high level of accuracy is reached, which is sufficient for the shape sorting puzzle setup used in COSPAL.
Acknowledgments

This thesis represents the final part of my Master of Science in Applied Physics and Electrical Engineering degree at Linköpings Universitet. This work has been carried out at the Computer Vision Laboratory at the Department of Electrical Engineering.

I would like to thank all people working at the Computer Vision Laboratory. You have all in one way or another contributed to this thesis, with technical advice or by contributing to the friendly and inspiring atmosphere.

I would especially like to thank my supervisor Erik Jonsson and my examiner Michael Felsberg. Erik for constantly answering my more or less thought through questions, even though I suspect I will never get to know ‘how long a rope is’. Michael for invaluable advice and for introducing me to the subject of machine learning and cognitive systems.

This work has been supported by EC Grant IST-2003-004176 COSPAL. This thesis does not represent the opinion of the European Community, and the European Community is not responsible for any use which may be made of its contents.

Fredrik Larsson 2007-02-28
6 Experiments

6.1 Test Scenario .............................................. 35
6.2 Simulator Setup ........................................... 35
   6.2.1 Results ............................................. 36
   6.2.2 Discussion ......................................... 37
6.3 Real World Setup ......................................... 37
   6.3.1 Results ............................................. 38
   6.3.2 Discussion ......................................... 39

7 Conclusions ................................................. 41

7.1 Conclusions .............................................. 41
7.2 Future Work ............................................. 41

A Leave-One-Out Cross-Validation Without Retraining ........ 43
   A.1 The Sherman-Morrison-Woodbury Theorem ............ 43
   A.2 Leave-One-Out Cross-Validation Without Retraining . 43

B Experimental Setup ........................................ 45
   B.1 Experimental Setup ................................... 45
   B.2 Data Acquisition ...................................... 45
      B.2.1 Detection of 2D Positions ......................... 45
      B.2.2 Stereo Triangulation ............................... 46
      B.2.3 Mapping Positions into Robot Frame ............ 47
      B.2.4 Estimating the Configuration of the End-Effector 47

Bibliography .................................................. 49
Chapter 1

Introduction

1.1 Thesis Overview

The main goal of this master thesis project has been to correct for inaccuracies in a low cost robotic arm by a learning approach. By combining Locally Weighted Projection Regression (LWPR) [27] with visual servoing an acceptable level of accuracy as reached.

Initially an analytical closed-form inverse kinematics solution for a 5 DOF robotic arm has been developed and implemented. This analytical solution proved not to meet the accuracy required for the shape sorting puzzle setup used in the COSPAL (COgnitiveSystems using Perception-Action Learning) project [2]. The shape sorting puzzle scenario consist of a number of puzzle pieces of different shapes and holes in corresponding shapes. The correctness of the analytic model could be confirmed through a simulated ideal robot and the source of the problem was deemed to be nonlinearities introduced by weak servos unable to compensate for the effect of gravity. Instead of developing a new analytic model that takes into account the effect of gravity, which would be erroneous when the characteristics of the robotic arm changed, e.g. when picking up a heavy object, a learning approach was selected.

As learning method LWPR has been used. It is an incremental supervised learning method and it is considered a state-of-the-art method for function approximation in high dimensional spaces.

LWPR was used to learn the inverse kinematics of the above mentioned robotic arm. The learned inverse kinematics was further combined with visual servoing. This allows for an improvement in accuracy by the use of visual feedback and the problems introduced by the weak servos can be solved. By combining the trained LWPR model with visual servoing, a high level of accuracy is reached, which is sufficient for the shape sorting puzzle setup used in COSPAL.
1.2 Mathematical Notation

Throughout this thesis, bold lower case letters $\mathbf{x}$ are used to denote vectors, bold upper case letters $\mathbf{X}$ denote matrices. All scalar values $x,X$ are denoted by italic letters.

1.3 Thesis Outline

Chapter 1: Introduction.

Chapter 2: Short introduction to robotic arms.

Chapter 3: Introduction to the basic ideas behind learning systems. A detailed description of two simple supervised learning methods is given as a preparation for Locally Weighted Projection Regression described in chapter 4.

Chapter 4: The learning method Locally Weighted Projection Regression (LWPR) is presented. It is an incremental supervised learning method and it is considered a state-of-the-art method for function approximation in high dimensional spaces.

Chapter 5: The concept of visual servoing is introduced. How LWPR can be used as a base for visual servoing is described.

Chapter 6: Experimental setup and obtained results. LWPR has been used to learn the inverse kinematics and visual servoing has been used to further increase the accuracy.

Chapter 7: Discussion and conclusions. Some suggestions of further work are presented.
Chapter 2

Robotics

This section gives a brief introduction to robotics with focus on robotic arms.

2.1 Robotic Arms

A robotic arm, see figure 2.1, can be modeled as a chain of rigid bodies (links) which in general are connected to each other with lower-pair joints. There are a total of six possible types of lower-pair joints, i.e. revolute, sliding, cylindrical, spherical, screw and planar joints, although only revolute and sliding joints are common in industrial robotic arms [10]. The first link is assumed to be connected to a supporting base while the last link is connected to the end-effector, e.g. a gripper or a drilling tool. The robotic arm is moved into different configurations by manipulating the angles of revolute joints and the lengths of sliding joints. The configuration \( c \) of the end-effector is usually of interest and can fully be defined by position \( p = (x, y, z)^T \), approach vector \( a = (a_x, a_y, a_z)^T \) (i.e. the direction from the last joint to the position \( p \)) and a vector \( r = (r_x, r_y, r_z)^T \) defining rotation such that \( c = (x, y, z, a_x, a_y, a_z, r_x, r_y, r_z)^T \).

2.2 Robotic Arm Kinematics

Robot arm kinematics deals with the the motion of a robotic arm, i.e. the position, velocity and acceleration of each link, without any regards to the forces that cause the motion [10]. Two fundamental problems within this area of study are the forward position and the inverse position problem. In the forward position problem the links and joint parameters of the robotic arm are given. The task is to find the configuration of the end-effector that will be the result of the given parameters. For the inverse position problem the desired configuration of the end-effector is given and the task is to find the corresponding joint parameters.

Let the joint parameters be noted \( \theta = (\theta_1, \theta_2, \ldots, \theta_k) \) where \( k \) is the degrees of freedom (DOF) and \( \theta_i \) is the configuration of the \( i \)th joint. The forward position
problem can then be formulated as: Find the function $f$ such that

$$f(\theta) = c.$$  \hspace{1cm} (2.1)

The inverse position problem can be stated as: Find $g$ such that

$$g(f(\theta)) = \theta.$$  \hspace{1cm} (2.2)

The solution to the forward position problem is rather straightforward. One common method is to use Denavit-Hartenberg parameters, that describe the characteristics of each link and also the relationship between connecting links [9], and homogeneous transform matrices. The problem, when the Denavit-Hartenberg parameters are identified, is then reduced to simple matrix multiplication.

The inverse problem on the other hand is a more complex one. An analytical solution to the general inverse position problem does not exist even though numerical approaches exist. For 6 DOF robotic arms where kinematic decoupling is possible, e.g. the three last joints intersecting allowing the positioning and orientation problem of the gripper to be separated, closed-form analytical solutions exist [10].
It is worth noting that the solution to the forward position problem is unambiguous while the inverse position problem may result in a number of different solutions, i.e. a given configuration of the end-effector can often be reached by more than one joint configuration, see figure 2.2.

The accuracy of analytical solutions to the inverse and forward problems depends on how reliable the values of the link parameters are. A small deviation from the true value may result in significant errors for the end-effector rendering the analytical solution unusable.

Figure 2.2. The same configuration of the end-effector can often be reached in a number of ways. Here are two possible solutions illustrated for a 5 DOF robotic arm.

For a thorough description of the forward and inverse kinematics problem and the use of D-H parameters see [8], [9] or [10].
Chapter 3

Learning

This chapter introduces the basic ideas behind learning systems. A detailed description of two simple supervised learning methods (linear regression and progression regression) are given in the last two sections of this chapter as a preparation for Locally Weighted Projection Regression described in chapter 4.

3.1 Machine Learning

‘Learning is defined as any relatively permanent change in behavior resulting from past experience, and a learning system is characterized by its ability to improve its behavior with time, in some sense towards an ultimate goal.’ [18]

The field of machine learning strives towards developing systems that have the ability to learn. A strict definition of the ability to learn is hard to find although most sources seem to agree on that the ability to learn is roughly equivalent to the ability to gain knowledge or skills from experience [17][18][20].

The field of machine learning can be divided into three main classes: supervised, unsupervised and reinforcement learning.

Supervised learning

In supervised learning the system is presented sequences of corresponding inputs and outputs. That is, for each input the system is also shown the desired output. The system is expected to be able to generalize from the shown training sequences and to correctly predict the output for previously unseen inputs.

Reinforcement learning

In reinforcement learning the system is given a reward (or punishment) depending on the generated output. The aim for the system is to over time acquire as much
reward as possible. For a reinforcement learning system there will always be a trade-off between exploration (learning) and exploitation of learned knowledge. From experience the system knows which previously generated output that was effective to gain rewards; but to learn such outputs the system needs to try new outputs [24].

**Unsupervised learning**

In unsupervised learning the system is shown only the input sequence. The system has to look for patterns in the input data without any feedback at all. Unsupervised learning is mostly used to learn a certain representation of the input that can be favorable for later processing [6].

A further distinction between learning methods can be done into *batch/offline* or *incremental/online* methods. For a batch method, all the training data need to be collected before the training can start. The system will then get access to all the data at once. An incremental method will receive one training sample at the time. Batch versions of an algorithm will often perform better than an incremental version of the same algorithm but the drawback is that the entire set of training samples needs to be kept in memory.

### 3.2 Why Learning?

An autonomous system operating in an unknown changing environment encounters a large diversity of situations. When developing such a system one approach would be to supply the system with appropriate instructions for all possible situations, i.e. in the sense of traditional artificial intelligence. Despite the fact that this would be a very difficult task it would also require that the system would perform identically in the developing environment as well as in an unknown environment. This would be a very difficult, if not an impossible task [12].

A different approach would be to supply the system with the ability to learn and hence adapt to changes. Such a system would change its behavior depending on the current situation and the task to be performed. A learning system would be more flexible to changes and able to function in a wider range of situations.

### 3.3 The Curse of Dimensionality

The curse of dimensionality is an often encountered term in learning and especially in high dimensional problems. It refers to the exponential growth of the volume of the hypercube with the number of dimensions [4]. To illustrate the problem, assume the training space is made up of the unit cube in $nD$. To cover half of the training space volume in $nD$ we need to cover a distance of $l = 0.5\sqrt{n}$ units in
3.4 Local Learning

each direction. This becomes cumbersome in as few dimensions as 3 and nearly impossible for 10 dimensions, see table 3.1.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Distance to cover</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.500</td>
</tr>
<tr>
<td>2</td>
<td>0.707</td>
</tr>
<tr>
<td>3</td>
<td>0.794</td>
</tr>
<tr>
<td>4</td>
<td>0.841</td>
</tr>
<tr>
<td>5</td>
<td>0.871</td>
</tr>
<tr>
<td>10</td>
<td>0.933</td>
</tr>
<tr>
<td>25</td>
<td>0.973</td>
</tr>
<tr>
<td>50</td>
<td>0.986</td>
</tr>
<tr>
<td>100</td>
<td>0.993</td>
</tr>
</tbody>
</table>

Table 3.1. The growth of the volume of the hypercube. The table shows the distance in each dimension that needs to be covered to span a hypercube with half the volume of the unit hypercube.

Assume that we are acquiring training points by sampling half the training space along each dimension. One of the problems due to the curse of dimensionality becomes apparent when we realise that for \( n^D \) we are effectively sampling \( 0.5^n \) of the volume of the training space, e.g. for \( n = 10 \) we are acquiring training samples for less than 0.1% of the training space.

### 3.4 Local Learning

In global learning methods one global model is fitted to all the training data while in local learning methods a number of local models are fitted to different parts of the training data. All the different models have a position, e.g. spatially, and a region of validity centered around the location of the individual model. Instead of region of validity the biologically inspired term receptive field (RF) is sometimes used. When training the individual local model, samples near, in whatever metric used, the center of the local model will influence the model more than samples far away. The simplest local learning method is perhaps the nearest neighbor method. Nearest neighbor approximates the output by using the output data of the nearest training sample, i.e. only the nearest training sample will affect the local model.

The use of RFs requires a way to decide which training samples that will contribute, and with how much, to the training process of each model. The weight, or activation, associated with a certain sample can be calculated by using different weighting kernels. The weight is then used as a measurement of how much that particular sample will contribute to the overall characteristics of the local model. Two different weighting kernels can be seen in figure 3.1. The left plot shows a
Figure 3.1. Left: A uniform weighting kernel. A sample that is less than 5 units from the model center is given weight 1. Samples further away are given weight 0.
Right: A Gaussian weighting kernel. The weight, $w$, of a training sample is given as $w = e^{-\frac{D^2}{2}}$. Here $D = 0.1$.

uniform weighting kernel. All samples within a certain distance from the model center is given weight 1 while samples further away are given weight, and thus influence, 0. The kernel to the right weights the influence of training data depending on the distance to the model center.

Fitting a global model to data that is intrinsically different in different parts of the
space may result in a poor fit. See figure 3.2 where noisy training samples have been drawn from two different straight lines valid in different regions. The least-squares method has been used globally in the left plot, resulting in a poor fit, and locally in the right plot where two models have been used. The global approach minimizes the residual sum of squares in a global aspect causing the local fit to be very poor. The local approach minimizes the residual sum of squares on a local scale resulting in better performance.

![Graph showing local and global least-squares fits.](image)

**Figure 3.2.** The intrinsic nature of the training data is represented by the solid line. Noisy training samples are marked with + and the fitted model is represented by the dashed line. **Left:** A global least-squares method has been used resulting in a poor fit to the training data. **Right:** A local least-squares method has been used resulting in a good fit. Two least-squares models, centered in $X = 5$ and $X = 15$, have been used with a uniform weighting kernel with width of 5 units.

When it comes to generalizing and predicting it is by no means sure that the local approach outperforms the global one. If the intrinsic nature of the training data is the same throughout the training space, local approaches may perform worse, see figure 3.3. Noisy training samples are drawn from a straight line and sufficient training data is only available in the left part of the training space. Fitting a global least-squares model to the data works well while fitting two local least-squares models to the same data gives a better fit but the model is specific to the region of the training data available.
squares models performs badly.

Figure 3.3. The intrinsic nature of the training data is represented by a solid line. Noisy training samples are marked with + and the fitted model is represented by the dashed line. Left: A global least-squares method has been used resulting in a good fit. Right: A local least-squares method. Two least-squares models, centered in \( X = 5 \) and \( X = 15 \), have been used with a uniform weighting kernel (width of 5 units).

3.5 Linear Regression

Let \( \mathbf{x} = (x_1, x_2, \ldots, x_q)^T \) represent a \( q \)-dimensional input vector and \( y \) represent a scalar output such that \( y = f(\mathbf{x}) + \epsilon \) where \( \epsilon \) is noise of some kind. Multiple linear regression (MLR), or univariate linear regression if \( q = 1 \), makes a prediction \( \hat{y} \) of \( y \) as a linear combination of the input variable \( \mathbf{x} \). Let \( \mathbf{\beta} = (\beta_1, \beta_2, \ldots, \beta_q, \beta_0)^T \) be the regression parameters and \( \tilde{\mathbf{x}} = (x_1, x_2, \ldots, x_q, 1)^T \). The MLR can then be written as

\[
\hat{y} = x_1\beta_1 + x_2\beta_2 + \ldots + x_q\beta_q + \beta_0 = \tilde{\mathbf{x}}^T\mathbf{\beta}.
\] (3.1)

The problem in MLR is how to choose \( \mathbf{\beta} \) such that \( \hat{y} \) describes \( y \) as well as possible. Usually the norm of the residual vector \( e = y - \hat{y} \) is to be minimized.
3.6 Projection Regression

Assuming the number \( N \) of input-output observations \((y_n, x_n)\) is larger than the number of regression parameters then in general no exact solution can be found [19]. By introducing and minimizing a suitable cost function \( J \) that depends on the residual \( e \) a solution can be found. The cost function used by the least-squares method is

\[
J = \sum_{n=1}^{N} (y_n - \hat{y}_n)^2 = (Y - X\beta)^T(Y - X\beta)
\]  

(3.2)

where \( Y = (y_1 \ y_2 \ \ldots \ y_n)^T \) and \( X = (\tilde{x}_1 \ \tilde{x}_2 \ \ldots \ \tilde{x}_n)^T \). \( J \) is minimized by choosing \( \beta \) as the pseudo-inverse [13]

\[
\beta = (X^T X)^{-1} X^T Y.
\]  

(3.3)

The discussion above can easily be generalized to the output being a vector instead of a scalar, e.g. for an \( m \) dimensional output consider \( m \) independent univariate regression problems [13].

3.6 Projection Regression

The computational cost of function approximation by ordinary MLR becomes a problem in high dimensional input spaces [23]. Also, if the input consists of redundant and/or correlated dimensions the solution may become numerically unstable and very sensitive to noise. Using dimensionality reducing techniques, such as projection regression, has been proven to be a useful approach [23].

The term projection regression refers to the procedure of replacing the ordinary MLR by a number of univariate regressions performed in a subspace of the original data. The original data is projected onto a subspace spanned by orthogonal basis vectors which will act as regression directions. The original MLR can be replaced by computationally cheaper univariate regressions since the regression directions are chosen to be orthogonal. The final MLR is then given by simply adding the univariate regressions [13]. How to choose this subspace is a non-trivial task. In section 3.6.1 and 3.6.2 two different methods for projection regression are described.

3.6.1 Principal Component Regression

Principal component regression uses the principal components, i.e. the eigenvectors of the covariance matrix, of the input data as the subspace for the univariate regressions [13]. Limiting the number of principal components used, e.g. only using principal components corresponding to sufficiently large eigenvalues, works as dimensionality reduction. Using \( k \) principal components guarantees that the
spanned subspace is the subspace of $k$-dimensions that are describing the distribution of input data as well as possible in a least-squares sense. This does not necessarily imply that they also describe the correspondence between input and output as well as possible.

### 3.6.2 Partial Least-Squares Regression

Projection regression using partial least-squares (PLS) performs univariate regression along the directions that describe the correspondence between input and output as well as possible. The procedure of choosing the regression direction as well as deciding the regression parameters is described in algorithm 1. Algorithm 2 shows how to predict the output for a query. The input and output in algorithm 1 are assumed to be mean centered. If this is not the case then it can easily be obtained through preprocessing of the data. For a more complete source on PLS see [19]

---

**Algorithm 1** PLS pseudo code

```plaintext
1: Initialize $X_{res} = X, y_{res} = y$
2: for $r = 1 : R$ (For each projection) do
3:   $u_r = X_{res}^T y_{res}$
4:   $s_r = X_{res} u_r$
5:   $\beta_r = \frac{s_r^T y_{res}}{s_r^T s_r}$
6:   $p_r = \frac{X_{res}^T}{s_r^T s_r} s_r$
7:   $X_{res} = X_{res} - s_r p_r^T$
8:   $y_{res} = y_{res} - s_r \beta_r$
9: end for
```

---

**Comments on algorithm 1:**

Line 2. The number of projections $R$ could be prespecified or a stopping criteria for adding new projections may be used, e.g. if the residual $y_{res}$ is small enough no further projections are used.

Line 3. The direction $u_r$ of highest correlation between $X$ and $y$ is computed.

Line 4. $X_{res}$ is projected onto $u_r$ to obtain the coordinates of $X_{res}$ along $u_r$.

Line 5-6. $X_{res}$ and $y_{res}$ is regressed on $s_r$. $\beta_r$ and $p_r$ is given as the solution to the ordinary least-squares problem.

Line 7-8. The input space (line 7) and the output space (line 8) is reduced by subtracting the contributions already accounted for. This is done to ensure that
all following regression directions are orthogonal to previous ones.

**Algorithm 2** Predicting output with PLS for query point $x_q$

1. $x_{res} = x_q$
2. for $r = 1 : R$ (For each projection) do
3. \[ s_r = x_{res}u_r \]
4. \[ \hat{y} = \hat{y} + \beta_s r \]
5. \[ x_{res} = x_{res} - s_r p_r \]
6. end for
Chapter 4

Locally Weighted Projection Regression

Locally weighted projection regression (LWPR) is an incremental local learning algorithm for nonlinear function approximation in high dimensional spaces [26]. The key concept in LWPR is to approximate the underlying function by local linear models. The LWPR model automatically updates the number of receptive fields (RFs), i.e. local models, used as well as the location (which is decided by the RF center $c$) of each RF. The size and shape of the region of validity (decided by the distance metric $D$) of each RF is updated continuously based on the performance of each model. As stated in section 3.6, function approximation by ordinary MLR becomes cumbersome in high dimensions. To deal with this problem, within each local model an incremental version of weighted PLS is used.

In this chapter the main pseudo code for LWPR is recapitulated. The output is assumed to be a scalar but generalization to a vector is as straightforward as for multiple linear regression, e.g. by creating an LWPR model for each dimension in the output. In table 4.1 and table 4.2 the notations and parameters used in this chapter are summarized. The main pseudo-code for LWPR is given in algorithm 3 followed by an in-depth description of the involved steps, section 4.1 to 4.5. The procedure of predicting the output given a query point is described in section 4.6.

The algorithms presented in this chapter are basically the same as described in [25] except from some minor simplifications.
Internal variables for each RF

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q$</td>
<td>The number of input dimensions</td>
</tr>
<tr>
<td>$w$</td>
<td>Activation by current input sample</td>
</tr>
<tr>
<td>$W$</td>
<td>Discounted sum of weights</td>
</tr>
<tr>
<td>$c$</td>
<td>Center of the RF</td>
</tr>
<tr>
<td>$x_c$</td>
<td>Discounted weighted mean of inputs.</td>
</tr>
<tr>
<td>$\beta_r$</td>
<td>Regression parameters (for predicting output)</td>
</tr>
<tr>
<td>$p_r$</td>
<td>Regression parameters (for reducing input space)</td>
</tr>
<tr>
<td>$R$</td>
<td>Number of projection directions currently used</td>
</tr>
<tr>
<td>$N$</td>
<td>Number of training samples seen</td>
</tr>
<tr>
<td>$x_{\text{res},r}$</td>
<td>The remaining part of the input before the $r$:th projection.</td>
</tr>
<tr>
<td>$y_{\text{res},r}$</td>
<td>The residual before the $r$:th projection.</td>
</tr>
<tr>
<td>$MSE_r$</td>
<td>Mean squared error after $r$ projections</td>
</tr>
<tr>
<td>$u_r$</td>
<td>Discounted direction of maximal correlation between inputs and outputs, i.e. the $r$:th projection direction.</td>
</tr>
<tr>
<td>$s_r$</td>
<td>The coordinates of $x_{\text{res},r}$ when projected onto $u_r$</td>
</tr>
<tr>
<td>$ss_r$</td>
<td>Memory term to enable the incremental PLS</td>
</tr>
<tr>
<td>$sx_r$</td>
<td>Memory term to enable the incremental PLS</td>
</tr>
<tr>
<td>$sy_r$</td>
<td>Memory term to enable the incremental PLS</td>
</tr>
<tr>
<td>$D$</td>
<td>Distance metric defining shape and size of the RF</td>
</tr>
<tr>
<td>$C_r$</td>
<td>Memory term to enable distance metric update</td>
</tr>
<tr>
<td>$E_r$</td>
<td>Memory term to enable distance metric update</td>
</tr>
<tr>
<td>$H_r$</td>
<td>Memory term to enable distance metric update</td>
</tr>
<tr>
<td>$R_r$</td>
<td>Memory term to enable distance metric update</td>
</tr>
<tr>
<td>$T_r$</td>
<td>Memory term to enable distance metric update</td>
</tr>
</tbody>
</table>

**Table 4.1.** Internal variables for each RF.

Parameters used by the LWPR model

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{\text{def}}$</td>
<td>Initial value of distance metric</td>
</tr>
<tr>
<td>$w_{\text{add}}$</td>
<td>Used to decide when to add a new RF</td>
</tr>
<tr>
<td>$w_{\text{prune}}$</td>
<td>Used to decide when to prune RFs</td>
</tr>
<tr>
<td>$w_{\text{act}}$</td>
<td>Used to decide whether to update a RF</td>
</tr>
<tr>
<td>$\eta$</td>
<td>Used to decide whether to add a new projection direction</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Controlling the gradient decent rate in the distance metric update</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Balancing the shrinking in the distance metric update</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>Forgetting factor used in iPLS</td>
</tr>
</tbody>
</table>

**Table 4.2.** Parameters used by the LWPR model.
4.1 Calculating the Activation

LWPR uses a Gaussian weighting kernel (see section 3.4) to calculate the activation of each RF according to

\[ w_k = e^{-\frac{(c_k - x)^T D_k (c_k - x)}{2}}, \]

(4.1)

where \( x \) denotes the input and \( c_k \) denotes the center of RF \( k \). The matrix \( D_k \), called distance metric, is used to adjust the size and shape of the RF. The distance metric of each RF is updated according to section 4.3.

4.2 Locally Weighted Incremental PLS

LWPR performs function approximation by replacing the original problem of multiple linear regression by a number of univariate linear regressions by using an incremental locally weighted version of PLS (algorithm 4). A forgetting factor \( \lambda \in [0, 1] \) and a number of memory terms are introduced to enable the incremental update. The pseudo code can be found in algorithm 4. The subscript \( k \) (indicating that we are dealing with RF \( k \)) for all variables in algorithm 4 has been dropped to simplify the notation.

Comments on algorithm 4:

Line 2. If the activation \( w \) is higher than a fixed value \( w_{\text{act}} \) the local regression model should be updated. This limitation is done to gain computational efficiency.
Algorithm 4: Incremental LWPLS for receptive field $k$ centered in $c$

1: for each new training point $(x, y)$ do
2: \hspace{1em} if $w > w_{\text{act}}$ then
3: \hspace{2em} $W^{n+1} = \lambda W^n + w$
4: \hspace{2em} $x_{n+1}^c = (\lambda W^n x_n^c + w x)/W^{n+1}$
5: \hspace{2em} $\beta_{0}^{n+1} = (\lambda W^n v \beta_{0}^n + w y)/W^{n+1}$
6: \hspace{2em} $x_{\text{res}} = x - x_{n+1}^c$
7: \hspace{2em} $y_{\text{res}} = y - \beta_{0}^{n+1}$
8: \hspace{2em} for $r = 1 : R$ (For all projections) do
9: \hspace{3em} $u_{n+1}^r = \lambda u_n^r + w x_{\text{res}} y_{\text{res}}$
10: \hspace{3em} $s_r = (x_{\text{res}}^T u_n^r + 1T u_n^r + 1)$
11: \hspace{3em} $s_{x_{\text{res}}}^{n+1} = \lambda s_{x_{\text{res}}}^n + w s_r$
12: \hspace{3em} $y_{\text{res}}^{n+1} = \lambda y_{\text{res}}^n + w x_{\text{res}}$
13: \hspace{3em} $s_{x_{\text{res}}}^{n+1} = \lambda s_{x_{\text{res}}}^n + w s_{x_{\text{res}}}$
14: \hspace{3em} $p_{n+1}^r = s_{x_{\text{res}}}^{n+1} / s_{s_{y_{\text{res}}}}^{n+1}$
15: \hspace{3em} $y_{\text{res}}^{n+1} = y_{\text{res}}^n - s_r \beta_{0}^{n+1}$
16: \hspace{3em} $s_{x_{\text{res}}}^{n+1} = s_{x_{\text{res}}}^n - s_r p_{n+1}^r$
17: \hspace{3em} $MSE_{n+1 = \lambda MSE_{n} + w y_{\text{res}}^2}$
18: \hspace{2em} end for
19: \hspace{2em} if $MSE_{n+1} / MSE_{n-1} > \eta$ then
20: \hspace{3em} $R = R + 1$
21: \hspace{2em} end if
22: end if
23: end for

Line 3-7. Update the mean of input and output for the local model. The current training sample is then mean centered.

Line 8-19 are the locally weighted incremental version of PLS. To enable the incremental PLS a number of memory terms, $ss_r, sy_{\text{res}}$, and $sx_{\text{res}}$ are introduced. By comparison with the batch version of PLS in section 3.6.2 the memory terms can be identified as:

$$ ss_r^{(n+1)} = \sum_{i=1}^{n+1} \lambda^{n+1-i} w^{(i)} s_r^{(i)} \approx s_r^T s_r $$

$$ sy_{\text{res}}^{(n+1)} = \sum_{i=1}^{n+1} \lambda^{n+1-i} w^{(i)} s_r^{(i)} y_{\text{res}}^{(i)} \approx s_r^T y_{\text{res}} $$

$$ sx_{\text{res}}^{(n+1)} = \sum_{i=1}^{n+1} \lambda^{n+1-i} w^{(i)} s_r^{(i)} x_{\text{res}}^{(i)} \approx X_{\text{res}}^T s_r, $$
4.3 Distance Metric Update

where the superscript \((i)\) denote the variable corresponding to the \(i\):th input.

Line 18 computes the squared sum of errors. It is used later at line 20 as a measurement of how many projections that should be used.

Line 20 If \(\frac{MSE_{t+1}}{MSE_t} > \eta\), \(\eta \in [0, 1]\) being a predefined parameter, then increase the number of projections \(R\) by one.

4.3 Distance Metric Update

The update rule for the distance metric \(D\) is based on stochastic gradient descent. Let \(M\) be the Cholesky decomposition of \(D\) and \(\alpha\) a parameter regulating the learning rate. The update rule can then be written as:

\[
D = M^T M \tag{4.5}
\]

\[
M^{n+1} = M^n - \alpha \frac{\partial J}{\partial M} \tag{4.6}
\]

\[
J = \sum_{n=1}^{N} w_n \frac{\|y_n - \hat{y}_{n,-n}\|^2}{\sum_{n=1}^{N} w_n}. \tag{4.7}
\]

The cost function \(J\) is the weighted leave-one-out cross-validation error. \(\hat{y}_{n,-n}\) stands for the estimate of \(y_n\) when the model has been trained on all training points except the \(n\)th. The definition of \(D\) guarantees that \(D\) will be positive definite which is required for equation 4.1 to result in a Gaussian weighting kernel.

If the number of training samples grows, minimizing equation (4.7) would lead to receptive fields of very small size [21]. This is avoided by introducing a penalty term \(\gamma \sum_{i,j=1}^{Q} D_{ij}^2\), where \(0 < \gamma \ll 1\) and \(Q\) being the number of input dimensions. A small receptive field would correspond to large squared sum of the elements of \(D\) which would induce a large penalty. Hence, we consider the cost function

\[
J = \sum_{n=1}^{N} w_n \frac{\|y_n - \hat{y}_{n,-n}\|^2}{\sum_{n=1}^{N} w_n} + \gamma \sum_{i,j=1}^{Q} D_{ij}^2. \tag{4.8}
\]

Equation (4.8) can be written as (see appendix A)

\[
J = \frac{1}{\sum_{n=1}^{N} w_n} \sum_{n=1}^{N} w_n \frac{\|y_n - \hat{y}_n\|^2}{\|1 - w_n x_n^T (X^T X)^{-1} x_n\|^2} + \gamma \sum_{i,j=1}^{Q} D_{i,j}^2, \tag{4.9}
\]

which means that the cost function can be calculated without the need for N-fold training. Equation 4.9 can (see [28]\(^1\)) furthermore be written as

\(^1\)It should be noted that the proof in [28] includes an error. However, a correct proof can easily be derived from the in [28] presented outline.
\[ J = \frac{1}{\sum_{n=1}^{N} w_n} \sum_{n=1}^{N} w_n \| y_n - \hat{y}_n \|^2 + \gamma \sum_{i,j=1}^{Q} D_{i,j}^2 = \sum_{n=1}^{N} J_{1,n} + J_2 \] 

(4.10)

under the assumption that \( N = R \), and \( S = (s_1s_2 \ldots s_R) \).

Due to the incremental nature of the LWPR algorithm the true derivative (with respect to \( M \)) of the cost function can not be calculated. Instead a stochastic gradient approximation is used. The update rules for this approach are summarized below (for a full derivation see [21]).

The stochastic gradient approximation can be written as

\[ \frac{\partial J}{\partial M} \approx \frac{\partial w}{\partial M} \left( \sum_{n=1}^{N} \frac{\partial J_{1,n}}{\partial w} \right) + \frac{w}{W^{n+1}} \frac{\partial J_2}{\partial M} = \frac{\partial w}{\partial M} Z + \frac{w}{W^{n+1}} \frac{\partial J_2}{\partial M} \]  

(4.11)

\( Z \) is calculated incrementally according to algorithm 5 and the other terms are calculated as:

\[ \frac{\partial J_2}{\partial M_{kl}} = 2 \gamma \sum_{i,j=1}^{Q} D_{ij} \frac{\partial D_{ij}}{\partial M_{kl}} \]  

(4.12)

\[ \frac{\partial w}{\partial M_{kl}} = -\frac{1}{2} w(x - x_c)^T \frac{\partial D}{\partial M_{kl}} (x - x_c) \]  

(4.13)

\[ \frac{\partial D_{ij}}{\partial M_{kl}} = \delta_{ij} M_{kj} + \delta_{ji} M_{ki} \]  

(4.14)

\[ \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \]  

(4.15)

4.4 Adding Receptive Fields

Adding a new RF is done when the maximum activation of the already existing RFs are less then \( w_{\text{add}} \). A high \( w_{\text{add}} \) means more overlap between RFs but also that more computing resources are needed. When adding a new RF all terms specific for the new RF that need to be initialized are summarized in algorithm 6. All other terms, e.g. memory terms for the incremental weighted partial least-squares, are set to zero.

The initial number of projection directions \( R \) is set to 2 to enable the comparison at line 20 in algorithm 4.
4.5 Pruning Receptive Fields

Algorithm 5 Calculation of \( Z = \sum_{r=1}^{R} (\sum_{n=1}^{N} \frac{\partial J_{1,n}}{\partial w}) \)

1: \( Z = 0 \)
2: for \( r = 1 : R \) (For all projections) do
3: \( C_r^{n+1} = \lambda C_r^n + w s_r x_{res}^T \)
4: \( k = \frac{1}{1 - ws / ss} \)
5: \( e_{cv,r} = k y_{res,r} \)
6: \( E_r^{n+1} = \lambda E_r^n + w e_{cv,r} \)
7: \( H_r^{n+1} = \lambda H_r^n + kwe_{cv,r} s_r \)
8: \( R_r^{n+1} = \lambda R_r^n + k w^2 s_r^2 e_{cv,r} \)
9: \( T_r^{n+1} = \lambda T_r^n + k w(2w e_{cv,r} s_r / ss - e_{cv,r} / \beta_r^{n+1}) x_{res}^T \)
10: \( Z = Z + k e_{cv,r} e_{cv,r} / \beta_r^{n+1} - 2(\lambda e_{cv,r} / \beta_r^{n+1}) H_r^n - 2(\lambda s_r^2 / ss_{n+1}) R_r^n - (E_r^{n+1} / \beta_r^{n+1}) + (T_r^{n+1} - 2R_r^{n+1} C_r^{n+1}) \)
11: end for

The initial value of the distance metric \( D_{def} \) should be chosen so the result is a rather too large RF than a too small RF. A large RF will quickly see a large number of training points resulting in quick adjustment towards the correct size. A very small RF will require a long time before seeing enough training points to compensate for the initial poor initialization.

Algorithm 6 Adding a new receptive field \( k \)

1: Initialize:
2: \( x_c = x \)
3: \( R = 2 \)
4: \( D = D_{def} \)
5: \( \beta_0 = y \)

4.5 Pruning Receptive Fields

Pruning a RF occurs for two reasons; overlapping and poor performance. The first reason, overlapping, occurs when two or more RFs are simultaneously activated with more than \( w_{\text{prune}} \), where \( w_{\text{prune}} \in [0, 1] \) is a parameter that defines the amount of tolerated overlap. When pruning for this reason the RF with largest region of validity (smallest value of the determinant of \( D \)) is kept. Pruning due to overlapping does not increase the performance of the LWPR model. It is only done to gain computational advantages.

The second reason, poor performance, occurs when the regularized weighted mean
squared error

\[ wMSE = \frac{e_{cv}^2}{W} + \gamma \sum_{i,j=1}^{N} D_{i,j}^2 \]  

in a RF is too large compared to that of other existing RFs. A check for poor performance is necessary to avoid degenerated cases when the RF of a local model grows to infinity. This will lead to a global learning approach but is avoided since this would also mean that the wMSE would be large [22].

### 4.6 Computing the Output

The procedure for computing the output \( \hat{y}_k \) for receptive field \( k \) given input query \( \mathbf{x}_q \) is described in algorithm 7. The final output \( \hat{y} \) is ultimately given as the weighted average of all receptive fields

\[ \hat{y} = \frac{\sum_{k=1}^{K} w_k \hat{y}_k}{\sum_{k=1}^{N} w_k} \]  

**Algorithm 7** Predicting the output for receptive field \( k \) for query point \( \mathbf{x}_q \)

1: Initialize \( \hat{y} = \beta_0 \), \( \mathbf{x}_{\text{res}} = \mathbf{x}_q - \mathbf{x}_c \)
2: for \( r = 1 : R \) do
3: \( s_r = \mathbf{u}_r^T \mathbf{x}_{\text{res}} \)
4: \( \hat{y} = \hat{y} + \beta_r s_r \)
5: \( \mathbf{x}_{\text{res}} = \mathbf{x}_{\text{res}} - s_r \mathbf{p}_r \)
6: end for
Chapter 5

Visual Servoing

This chapter begins with an introduction to visual servoing, section 5.1, where the nomenclature is adapted from [14] [15]. How an inverse kinematics model learned by LWPR can be used for visual servoing is described in section 5.3.

5.1 Introduction

The use of visual information for robot control, i.e. to reach a desired configuration with the end-effector, can be divided into two different system classes depending on approach; open-loop systems and closed-loop systems. The term visual servoing refers to the latter approach.

5.1.1 Nomenclature

Open-loop systems
In an open-loop system the extraction of information from the visual part of the system is separated from the task of operating the robot. Information, e.g. the position of the object to be gripped, is extracted from the image(s). This information is then fed to a robot control system that moves the robot arm blindly. This requires an accurate inverse kinematic model for the robot arm as well as an accurately calibrated camera system. Also, the environment needs to remain static between the assessment phase and the movement phase.

Visual servoing
In a system based on visual servoing visual information is continuously used as feedback to update the control signals. This gives a system that is less dependent on static environment than open-loop systems, calibrated camera(s) etc. Visual servoing systems are further divided into two subclasses, dynamic look-and-move systems and direct visual servo systems depending on the method of transforming
information into robot action.

Dynamic look-and-move systems use image extracted information as input to a robot controller that computes the desired joint configurations and then uses joint feedback to internally stabilize the robot. This means that once the desired lengths and angles of the joints are decided, these configurations will be reached. This does not necessarily mean that the desired configuration of the end-effector is reached.

Direct visual servo systems use the extracted information to directly compute the input to the robot. This renders the use of a robot controller redundant.

![Flowchart for a position based dynamic look-and-move system. $\Delta x$ denotes the deviation between desired ($x_w$) and reached ($x$) configuration of the end-effector.](image)

Both the dynamic look-and-move and the direct visual servoing approach may be used in a *position based* or *image based* way, or in a combination of both. In a position based approach the images are processed such that the relevant 3D information is retrieved in world (or robot or camera) coordinates. The process of positioning the robotic arm is then defined in the appropriate 3D coordinate system. In an image based approach, 2D information is directly used to decide how to position the robot, i.e. the robotic arm is to be moved to a position defined by image coordinates. See figure 5.1 and 5.2 for flowcharts describing the different system architectures.
5.2 The Visual Servoing Task

The task in visual servoing when used in robot control is to minimize the norm of the deviation vector $\Delta x = x_w - x$, where $x$ denotes the reached configuration and $x_w$ denotes the desired configuration of the end-effector.

The configuration $x$ is said to lie in the task space and the control signal $y$ that generated this configuration is located in the joint space. The image Jacobian\(^1\) $J_{\text{img}}$ is the linear mapping that maps changes in joint space $\Delta y$ to changes in task space $\Delta x$ such that:

$$\Delta x = J_{\text{img}} \Delta y. \quad (5.1)$$

Let furthermore $J$ denote the inverse image Jacobian, i.e. a mapping from changes in task space to changes in joint space such that:

$$\Delta y = J \Delta x \quad (5.2)$$

---

\(^1\)The term image Jacobian is used since the task space is often the acquired image(s). The configuration vector is then the position of features in these images. The term interaction matrix may sometimes be encountered instead of image Jacobian.
\[ J = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}. \]  

(5.3)

The term inverse image Jacobian does not necessarily mean that \( J \) is the mathematical inverse to \( J_{\text{img}} \). In fact, the mapping \( J_{\text{img}} \) does not need to be injective and hence not invertible (see figure 2.2 in section 2.2). In our case, this problem is avoided by the used test scenario (see chapter 6). The word inverse simply denotes that the inverse image Jacobian is describing changes in joint spaces given wanted changes in task space while the image Jacobian is describing changes in task space given changes in joint space.

If the inverse image Jacobian, or an estimate, has been acquired, the task of correcting for an erroneous control signal is in theory rather simple. If the current position with deviation \( \Delta x \) were originated from the control signal \( y \) the new control signal is then given as

\[ y_{\text{new}} = y - J\Delta x. \]  

(5.4)

In a non-ideal situation, the new control signal will most likely not result in the desired position. The process of estimating the Jacobian and to update the control signal needs to be repeated until a stopping criterion is met, e.g. the deviation is sufficiently small or the number of iterations has reached a prespecified number.

### 5.3 Combining Visual Servoing and LWPR

We have been using LWPR to learn the inverse kinematics of a robotic arm. Input to the LWPR model was the 3D position \( x \) and output was the corresponding control signals \( y \). Using LWPR as a base for visual servoing is a straightforward approach. LWPR gives a number of local linear models from which the inverse image Jacobian (further on just denoted Jacobian) can be estimated.

The derivation of the LWPR Jacobian starts by demonstrating that the output from LWPR can be written as a weighted sum of linear mappings. An algorithm describing how to calculate the linear mapping matrix for each local model is given in section 5.3.1. This is then followed by the derivation of the LWPR Jacobian in section 5.3.2.

#### 5.3.1 LWPR Output as a Linear Mapping

Algorithm 7 in section 4.6 describes the procedure of predicting the output \( \hat{y}_k \) for receptive field \( k \) given the input \( x \). After dropping the subscript \( k \) to simplify the notation, the result can be written as

\[ \hat{y} = \beta_1^T s_1 + \beta_2^T s_2 + \ldots + \beta_R^T s_R + \beta_0 \]  

(5.5)
where $\beta_r$ is known and $s_r$ is given as linear function of $x, x_c, u_r$ and $p_r$ and $x_c, u_r$ and $p_r$ are known from the considered RF. Hence, $\hat{y}$ is a linear mapping of $x$ and $(5.5)$ can for a vector output $\hat{y}$ be written as

$$\hat{y} = A(x - x_c) + \beta_0. \quad (5.6)$$

Algorithm 8 describes how to calculate $A$. This algorithm can easily be derived by manipulating the expression for $s_i$ given by algorithm 2.

Algorithm 8 Calculation of $A$ for a single receptive field

1: Initialize $A_0 = 0$, $C = I_d$, $d = \text{dim}(x)$
2: for $r = 1 : R$ do
3: $A_r = A_{r-1} + \beta_r u_r^T C$
4: $C = (I_d - p_r u_r^T) C$
5: end for

5.3.2 The LWPR Jacobian

At first glance it would be tempting to approximate the Jacobian with the weighted sum of the Jacobians of all individual RFs such that

$$J = \frac{d\hat{y}}{dx} = \frac{\sum_{k=1}^{K} w_k A_k}{\sum_{k=1}^{K} w_k}. \quad (5.7)$$

However, this would neglect the fact that $w_k$ is a function of $x$.

According to (5.6) each $\hat{y}_k$ can be written as

$$\hat{y}_k = A_k(x - x_{k,c}) + \beta_{k,0} \quad (5.8)$$

leading to

$$w_k \hat{y}_k = e^{-\frac{1}{2}(x - c_k)^T D_k (x - c_k)} (A_k(x - x_{k,c}) + \beta_{k,0}). \quad (5.9)$$

The derivatives $\frac{dw_k}{dx}$ and $\frac{d(w_k \hat{y}_k)}{dx}$ are

$$\frac{dw_k}{dx} = -(x - c_k)^T D_k w_k \quad (5.10)$$

$$\frac{d(w_k \hat{y}_k)}{dx} = -\hat{y}_k (x - c_k)^T D_k w_k + w_k A_k. \quad (5.11)$$

By setting $g = \sum_{k=1}^{K} w_k \hat{y}_k$ and $h = \sum_{k=1}^{K} w_k$ and by using the quotient rule $\frac{dy}{dx}$ can be written as

$$\frac{d\hat{y}}{dx} = \frac{d \frac{g}{h}}{dx} = \frac{\frac{dg}{dx} h - g \frac{dh}{dx}}{h^2} = \frac{\frac{dg}{dx} - \hat{y} \frac{dh}{dx}}{h} \quad (5.12)$$
giving

\[
\frac{d\hat{y}}{dx} = \sum_{k=1}^{K} \left( -\hat{y}_k(x - c_k)^T D_k w_k + w_k A_k \right) - \frac{\hat{y}}{h} \sum_{k=1}^{K} \left( -(x - c_k)^T D_k w_k \right)
\]  \hspace{0.5cm} (5.13)

ultimately leading to the expression

\[
J = \frac{d\hat{y}}{dx} = \frac{1}{h} \sum_{k=1}^{K} w_k (A_k + (\hat{y}_k - \hat{y})(x - c_k)^T D_k).
\]  \hspace{0.5cm} (5.14)

5.3.3 Updating the Jacobian

After the Jacobian has been estimated, the control signal is updated. The new control signal is used to move the robot and the resulting configuration is observed. The new error \(\Delta x_{\text{new}}\) is calculated and unless any of the stopping criteria have been met, the visual servoing is repeated. This causes a problem when calculating the Jacobian through the learned LWPR model. As can be seen in (5.14), we need for the Jacobian update the current weights \(w_k\), the outputs \(\hat{y}_k\) and also the corresponding input \(x\) that generated the predicted signal \(\hat{y}_{\text{new}}\). But since \(\hat{y}_{\text{new}}\) was obtained by using the Jacobian (and is no longer an output of the LWPR model) it is a new non-trivial problem to calculate the corresponding input. Two different solutions have been used for dealing with this problem: a static approach and an approximative updating approach.

\textit{Static approach:} The simplest solution is the static approach. The Jacobian is simply not updated and the Jacobian used in the first step is (still) used in the following steps.

\textit{Approximative updating approach:} The somewhat more complex solution uses the (unrealistic) assumption that the LWPR model is flawless. This means that if we use the (after one VS iteration) reached position as input to the LWPR model we would get predicted control signals that would result in the observed position. This in turn means that we may update the Jacobian by estimating the Jacobian as if the query was the reached position. The first two iterations using the approximative updating approach is described in figure 5.3 to figure 5.8. The green dotted line represents the LWPR approximation of the real function, which in turn is represented by the solid black line.
5.3 Combining Visual Servoing and LWPR

Figure 5.3. The wanted configuration $x_w$ and the LWPR prediction $y_1$

Figure 5.4. The robotic arm is moved using the control signal $y_1$ and the reached configuration $x_1$ is observed, resulting in the deviation $\Delta x_1$. 
Figure 5.5. For the first visual servoing iteration, the Jacobian $J_1$ is estimated and the control signal is updated. Resulting in $y_2$.

Figure 5.6. The robotic arm is moved using the control signal $y_2$ and the reached configuration $x_2$ is observed, resulting in the deviation $\Delta x_2$. 
To further update the control signal we would need the Jacobian $J_2$, corresponding to unknown input $x_2$.

Instead of the real LWPR Jacobian $J_2$, the approximative Jacobian $\tilde{J}_2$ is used to estimate the new control signal $y_3$.

$y_3 = y_2 - \Delta x_2 \tilde{J}_2$
Chapter 6

Experiments

Experiments have been done both on an ideal robotic arm (simulator) and a real robotic arm. The inverse kinematics of the robotic arm was learned by LWPR and the performance with and without visual servoing was evaluated. The obtained results are presented in section 6.2.1 and section 6.3.1.

6.1 Test Scenario

The test scenario used to evaluate the performance of the visual servoing based on the learned inverse kinematics is the reduced 3D scenario, denoted 2D+, used in the COSPAL project. 2D+ refers to the fact that the end-effector can be positioned in two different planes, the grip- and the movement-plane. The approach vector of the end-effector is to be perpendicular to the ground plane. The task space of the robotic arm is restricted (by physical and practical constraints) to a half circle with radius of 240 mm.

6.2 Simulator Setup

The simulator was constructed from an ideal model of the Lynx-6 robotic arm implemented in Matlab. To imitate the real world conditions, Gaussian noise was added to the acquired training points as well as to the obtained positions after each visual servoing iteration. This was done in order to make the comparison between the simulated world and the real world as straightforward as possible.

For the simulator training points in both planes were acquired by using the ideal model, but with Gaussian noise added, since we can not expect to obtain perfect training data in the real world setup either. That means, we add noise to the position we want to reach by the ideal model (since we can not position the real robot exactly in the planes) and then add additional noise to simulate noise in the process of estimating the reached position.
6.2.1 Results

Table 6.1 and 6.2 contain the results from simulations with and without Gaussian noise added in the visual servoing process. LWPR denotes the mean deviation with just the trained LWPR model while I-LWPR denotes the mean error when LWPR model has been used incrementally. That is, for each position 100 attempts to reach it are done. The reached position after each attempt was used as a training sample to the LWPR model and the presented result is the reached position after the 100th attempt. J stands for the correct Jacobian of the LWPR model and A for the initial naive approximation of the Jacobian. Static/Update denotes whether the static or the updating approach has been used. The stopping criteria for the visual servoing was set to 20 iterations or deviation of less than 0.1 mm from the desired position.

<table>
<thead>
<tr>
<th>Training points:</th>
<th>500</th>
<th>1000</th>
<th>5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>LWPR</td>
<td>9.71</td>
<td>7.82</td>
<td>6.42</td>
</tr>
<tr>
<td>I-LWPR</td>
<td>6.41</td>
<td>5.88</td>
<td>5.76</td>
</tr>
<tr>
<td>A Static</td>
<td>2.55</td>
<td>1.95</td>
<td>1.36</td>
</tr>
<tr>
<td>A Update</td>
<td>2.59</td>
<td>2.06</td>
<td>1.32</td>
</tr>
<tr>
<td>J Static</td>
<td>0.78</td>
<td>0.27</td>
<td>0.16</td>
</tr>
<tr>
<td>J Update</td>
<td>0.62</td>
<td>0.22</td>
<td>0.15</td>
</tr>
</tbody>
</table>

Table 6.1. Evaluation on simulated 2D+ scenario when trained with 500, 1000 and 5000 training samples. The numbers presented are the mean deviation in mm from desired position. 1000 test points were used. No noise has been used in the iterations of the visual servoing.

<table>
<thead>
<tr>
<th>Training points:</th>
<th>500</th>
<th>1000</th>
<th>5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>LWPR</td>
<td>11.16</td>
<td>8.81</td>
<td>7.78</td>
</tr>
<tr>
<td>I-LWPR</td>
<td>8.43</td>
<td>7.43</td>
<td>7.25</td>
</tr>
<tr>
<td>A Static</td>
<td>3.69</td>
<td>2.32</td>
<td>2.01</td>
</tr>
<tr>
<td>A Update</td>
<td>3.54</td>
<td>2.19</td>
<td>2.00</td>
</tr>
<tr>
<td>J Static</td>
<td>2.80</td>
<td>1.97</td>
<td>1.84</td>
</tr>
<tr>
<td>J Update</td>
<td>2.42</td>
<td>1.95</td>
<td>1.85</td>
</tr>
</tbody>
</table>

Table 6.2. Evaluation on simulated 2D+ scenario when trained with 500, 1000 and 5000 training samples. The numbers presented are the mean deviation in mm from desired position. 1000 test points were used. Gaussian noise with standard deviation of 2.6 mm was used in the iterations of the visual servoing.
6.2.2 Discussion

The results from the simulator shows that by combining LWPR with visual servoing a higher level of accuracy is reached than by using LWPR alone. The visual servoing approach outperforms the I-LWPR even when the number of iterations made by I-LWPR is 5 times as many (100 vs 20).

The results from the noise contaminated case may appear too good, because the mean error after visual servoing is less than the mean error of perfect positioning with just the Gaussian noise added. This can be explained by the fact that the visual servoing got 10 iterations to improve the position. When the reached position is close to the desired position after just a few iterations, the remaining iterations may result in a better observed position than just one perfectly positioned with added noise would.

The obtained results show that the correct Jacobian compares favorable to the approximation. In the noise free scenario, the true Jacobian outperforms the approximative by an order of magnitude.

The choice between a static approach or an updating approach is not as clear as the choice between true Jacobian or approximative Jacobian. The real world experiments will focus on this question.

6.3 Real World Setup

The real world experimental setup consisted of a low cost robotic arm of Lynx-6 type [3] (see figure 6.1) and a calibrated stereo rig. The end-effector of the robotic arm was equipped with three spherical markers in distinct colors. The configuration of the end-effector was estimated through a four step process:

1. Detection of the 2D positions of the spherical markers in left and right image.
2. By stereo triangulation acquire the 3D positions of each marker relative the left camera.
3. Map the acquired 3D positions to the reference frame centered at the base of the robotic arm.
4. Estimate the configuration of the end effector based on the position of markers.

The entire process could as easy (if not easier) be done in a image based way, i.e. by using image coordinates. This would avoid the extra uncertainties introduced by step 2 and 3 described above. The reason for doing the real world experiments in a position based way and not in an image based way was purely due to practical issues, e.g. the stereo rig being moved would render the collected training data untrustworthy. The real world experimental setup is described in more detail in Appendix B.
Simulator results showed that the correct Jacobian outperformed the naive approximation. Real world experiments are therefore done just using the correct Jacobian. Training points for the real world evaluation were acquired by using the analytical model to move the robotic arm. The analytical model was instructed to position the end-effector in the grip-plane or in the movement-plane and the reached position was estimated and later used as a training sample.

### 6.3.1 Results

Table 6.3 contain the results from real world evaluation. \( \text{LWPR} \) denotes the mean deviation with just the trained LWPR model. \( J \) stands for the correct Jacobian of the LWPR model and \( \text{Static/Update} \) denotes whether the static or the updating approach has been used. The stopping criteria for the visual servoing was set to 20 iterations or deviation of less than 1 mm from the desired position.

The noise in estimated positions due to e.g. the robotic arm shaking, noise in captured images and non-perfect segmentation of markers was assumed to be Gaussian with zero mean. The standard deviation was estimated to 2.6 mm.

The performance of the analytical model was evaluated on 100 random positions in the 2D+ space. The mean error was estimated to 15.87 mm. However, it should be noted that no time was spent on fine tuning the analytical model. Also, the analytical model is heavily dependent on the calibration of the stereo rig and the

![Figure 6.1. The experimental setup used for the experiments.](image)
6.3 Real World Setup

mapping from camera frame to robot frame while the learned inverse kinematics was trained with data acquired with these imperfections.

No evaluation of the visual servoing methods was done for the 10k case. The level of accuracy reached for 1k and 5k is as accurate as the noise level permits.

<table>
<thead>
<tr>
<th>Training points:</th>
<th>100</th>
<th>500</th>
<th>1000</th>
<th>5000</th>
<th>10 000$^1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>LWPR</td>
<td>16.89</td>
<td>12.83</td>
<td>7.53</td>
<td>8.78</td>
<td>5.86</td>
</tr>
<tr>
<td>J Static</td>
<td>9.83</td>
<td>5.41</td>
<td>1.79</td>
<td>1.64</td>
<td>-</td>
</tr>
<tr>
<td>J Update</td>
<td>9.07</td>
<td>4.32</td>
<td>1.65</td>
<td>1.65</td>
<td>-</td>
</tr>
<tr>
<td>Analytic solution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15.87</td>
</tr>
</tbody>
</table>

Table 6.3. Evaluation on real world 2D+ scenario. The numbers presented are the mean deviation in mm from desired position. 50 test points were used for evaluation except from in the 10k case and in the analytical case were 100 test positions were used. Stopping criteria for the visual servoing was 10 iterations or a deviation less than 1 mm.

6.3.2 Discussion

The result from the real world experiments supports the results obtained through simulations. This consistency between simulations and the real world, strengthens the confidence to be placed in the obtained results, even though the exact numbers can be questioned.

The open question from the simulator, regarding a static or an updating approach does seems to have an answer. The main difference between the two approaches can best be seen when the inverse kinematics has been learned from few training samples. In this case the updating approach works better than the static. This can be explained by the fact that the updating approach got a chance to correct for an initial erroneous Jacobian, which is more likely to be the case when the LWPR model has been trained on fewer points. This leads to the conclusion that the updating approach is to be preferred.

It can be noted that the accuracy when trained on 1000 points (or more) is high enough for the COSPAL shape sorting puzzle.

$^1$The LWPR model was trained on a total of 6k unique points. The first 1000 points were shown (in random order) 5 times and then the additional 5k points were used.
Chapter 7

Conclusions

This chapter contains conclusions and discussions. Some suggestions for further work are presented.

7.1 Conclusions

From the results obtained, both from the simulator and the real world, we conclude that by combining inverse kinematics learned by LWPR with visual servoing, a high level of accuracy can be reached. The trained systems shows a performance with sufficient accuracy for the shape sorting puzzle setup used in COSPAL.

Results, both from the simulator and the real world, showed that the correct Jacobian of the LWPR model (5.14) combined with the proposed updating approach (5.3.3) seems to be the best approach.

Furthermore we may draw the conclusion that the actual limit of the accuracy is decided by the noise levels when estimating the position. To further increase the performance, work should be done to improve the segmentation routine.

7.2 Future Work

To further improve the accuracy, the noise in acquired training points has to be reduced. However, the next step to make the system more in line with the COSPAL philosophy would be to remove the spherical markers from the end-effector and also include learning to recognize the end-effector. This leads to the decision not to spend time on perfecting the current procedure of estimating the configuration of the end-effector.

One problem with LWPR is that if a position can be reached in two ways, say with the base servo $= \pm \pi$, and training points are drawn equally from both these
constellations, the output would be the average of these two positions, i.e. base servo = 0. One potential way to solve this problem would be to use the channel representation [11]. By channel encoding the input and the output this unwanted averaging property could hopefully be avoided. If this would introduce other problems need to be investigated.
Appendix A

Leave-One-Out Cross-Validation Without Retraining

A.1 The Sherman-Morrison-Woodbury Theorem

For the derivation of the expression for leave-one-out cross-validation without retraining the Sherman-Morrison-Woodbury [5] theorem will be used. The theorem is restated here for completeness.

The Sherman-Morrison-Woodbury theorem: 1 Let \( A \) be a nonsingular matrix and \( u \) and \( v \) be two column vectors such that \( 1 - v^T A^{-1} u \neq 0 \). Then

\[
(A - uv^T)^{-1} = A^{-1} + \frac{A^{-1}uv^T A^{-1}}{1 - v^T A^{-1} u} \tag{A.1}
\]

A.2 Leave-One-Out Cross-Validation Without Retraining

This section recapitulates how the leave-one-out cross-validation cost function \( J \) can be computed without retraining. \( J \) is given as

\[
J = \frac{\sum_{n=1}^{N} w_n \| y_n - \hat{y}_{n,-n} \|^2}{\sum_{n=1}^{N} w_n} \tag{A.2}
\]

where \( \hat{y}_{n,-n} \) denotes the estimate of \( y_n \) when the model has been trained on all training points except the \( n \):th. Straightforward calculation of \( J \) would require the training data to be kept in memory (to enable the needed \( N \)-fold training). Fortunately this can be avoided [7].
Assume for the moment that we are dealing with ordinary least-squares regression where all $w_n = 1$ and $y_n$ is a scalar. The estimate of $\hat{y}_n$ would then (using the same notation as in chapter 3) be given as $\hat{y}_n = \beta^T x_n$, where $\beta = (X^T X)^{-1} X^T Y$.

Let $\beta = C^{-1} d$ where $C^{-1} = (X^T X)^{-1}$ and $d = X^T Y$. Let further the subscript $-n$ denote that the $n$:th training point is omitted. This means that

$$C_{-n} = C - x_n x_n^T$$  \hfill (A.3)

$$d_{-n} = d - x_n y_n$$  \hfill (A.4)

$$\beta_{-n} = C_{-n}^{-1} d_{-n}.$$  \hfill (A.5)

Assuming that $x_n^T C^{-1} x_n \neq 1$ and applying the Sherman-Morrison-Woodbury theorem (see appendix A.1) to equation A.5 results in

$$C_{-n}^{-1} = (C^{-1} + \frac{C^{-1} x_n x_n^T C^{-1}}{1 - x_n^T C^{-1} x_n}).$$  \hfill (A.6)

Hence, after some algebraic manipulation (A.6) can be written as

$$\beta_{-n} = \beta + \frac{C^{-1} x_n (x_n^T \beta - y_n)}{1 - x_n^T C^{-1} x_n}.$$  \hfill (A.7)

This in turn means that

$$\|y_n - \hat{y}_{n,-n}\|^2 = \|y_n - \beta_{-n} x_n\|^2 = \|y_n - \hat{y}_n\|^2.$$  \hfill (A.8)

Doing the same calculations for the case where not all $w_n = 1$ leads to

$$J = \frac{\sum_{n=1}^{N} w_n \|y_n - \hat{y}_{n,-n}\|^2}{\sum_{n=1}^{N} w_n} = \frac{1}{\sum_{n=1}^{N} w_n} \sum_{n=1}^{N} \frac{w_n (y_n - \hat{y}_n)^2}{1 - w_n x_n^T C^{-1} x_n}.$$  \hfill (A.9)
Appendix B

Experimental Setup

B.1 Experimental Setup

The experimental real world setup consisted of a Lynx-6 robotic arm [3] and a calibrated stereo rig. The end-effector, in our case a simple two fingered gripper, of the robotic arm was equipped with three spherical markers in distinct colors.

B.2 Data Acquisition

The robotic arm was moved and the configuration of the end-effector was estimated through a four step process:

1. Detection of the 2D positions of the spherical markers in left and right image.
2. By stereo triangulation acquire the 3D positions of each marker relative the left camera.
3. Map the acquired 3D positions to the reference frame centered at the base of the robotic arm.
4. Estimate the configuration of the end effector based on the position of markers.

B.2.1 Detection of 2D Positions

The gripper of the robotic arm was equipped with spherical markers in distinct colors. The center of each marker was detected by a simple segmentation routine (see figure B.1) based on thresholding in the CIE 1979 L*a*b* colorspace. The CIE 1976 L*a*b* color space separates intensity information, L*, and color information, a* and b*. This will in theory give a representation of color insensitive to moderate changes in light intensity [29].
Experimental Setup

Figure B.1. The spherical markers segmented in the image captured by the left camera. The calibration pattern used in step 3 can be seen placed in front of the robotic arm.

B.2.2 Stereo Triangulation

Given the pixel coordinate for each marker in the left and right image the 3D position was obtained through stereo triangulation. For this purpose the Camera
B.2 Data Acquisition

Calibration Toolbox for Matlab [1] was used.

B.2.3 Mapping Positions Into Robot Frame

A calibration pattern, figure B.2, was placed in front of the robotic arm. The 3D position of each crossing was acquired in the camera frame (by stereo triangulation) and in robot frame (known by placement of the calibration pattern). Solving the least-squares problem that arises gives the mapping between the two coordinate frames. Since it is known beforehand that the sought homogeneous transform matrix is a mapping between two orthogonal coordinate system the true transform matrix can be estimated even though all calibration points are in the same plane [16]. The estimated mapping is then used to map the positions of the spherical markers into the robotic coordinate frame, see figure B.3.

Figure B.2. The calibration pattern. The L-shaped dots are used to distinguish each crossing by relating them to a coordinate frame attached to the L.

B.2.4 Estimating the Configuration of the End-Effector

Given the 3D position in the robot frame of each marker, the end position of the gripper was estimated as the point between the two markers at the end of the fingers of the gripper. The approach vector was given as the vector from the third marker to the previously estimated end position. The rotation vector was estimated as the vector between the two markers at the end of the fingers of the gripper.
Figure B.3. The positions of the spherical markers and the calibration pattern are given in the coordinate frame centered at the base of the robotic arm.
Bibliography


49


Upphovsrätt

Detta dokument hålls tillgängligt på Internet — eller dess framtidiga ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

Copyright

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for his/her own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/

© Fredrik Larsson