2008 IEEE International Conference on
Robotics and Automation
Pasadena, CA, USA, May 19-23, 2008

Acquiring Change Models for Sensor-Based Robot Manipulation

Jan Deiterding and Dominik Henrich

Lehrstuhl für Angewandte Informatik III
Universität Bayreuth, D-95445 Bayreuth, Germany
E-Mail: *{jan.deiterding, dominik.henrich}@uni-bayreuth.de*, http://ai3.inf.uni-bayreuth.de

Abstract—**The aim of this paper is to enable a programmer to easily employ external sensors for flexible robot manipulation. We describe a general approach to determine the relation between the position deviation of an object and the resulting data from sensors used to recognize this deviation. This information can be used to employ adaptation techniques to compensate the deviation thus enabling robots to react flexibly to changes such as workspace variations or object drifts. The proposed methods are designed to be independent of the type of sensor. We describe methods to automatically determine a function describing this relation and how adaptive techniques can be integrated easily into robot programs without detailed knowledge provided by the programmer.**

Keywords: Manipulation Planning, Intelligent and Flexible Manufacturing, Programming Environments

I. Introduction

Industrial robots are able to perform complex tasks without symptoms of fatigue as well as highest precision and speed. However, these tasks are nearly always executed in a fixed environment, that is the precision is gained by ensuring that all objects are placed in exactly the same position every time. All parts need to have the same dimension, position, orientation, etc. Only by employing external sensors such as cameras or force-/torque-sensors, we can enable a robot to deal with imprecisions and variations occurring in the objects and the environment. When designing such programs for flexible robots, a programmer faces two problems: The first problem is, that the programmer must know which type of sensor may be employed to measure a specific variation. The second problem is to determine how the measured data (physically) relates to the variation.

In [5] we have classified changes that can occur between two executions of the same robot program with two characteristics: The origin of the change and the robots reaction to it (Table 1). An indeterminacy is something we are not aware of at this moment, but once we have learned about it, it will remain constant for a prolonged period of time. Variations on the other hand occur every time the robot performs the task at hand. Faults and errors occur when a sudden change in the workspace occurs. The drift is a problem caused by gradual changes within the workspace, i.e. the settings of machines and tools changes over time. In this paper, we are interested in ways of dealing with changes requiring a continuous adaptation strategy: Variations and drifts.

Table 1
Classification of changes that can occur between two executions of the same program [5].

		Origin of change	
		Caused by the task	Caused by abrasion
Reaction to change	One-step learning	Indeterminacies	Faults and Errors
	Continuous learning	Variations	Drifts

To successfully deal with both, we need external sensors to identify the change so we can compute a reaction to it. The first task is to find a function which transforms sensor values into a Cartesian description of the change. While this is straightforward for "easy" sensors, e.g. distance sensors, it proves to be a lot more difficult for complex sensors, like images from cameras etc. Additionally the sensor values are nearly always blurred by some kind of noise. The classical approach is to analytically determine a function describing this mapping. But, for complex sensors this task turns difficult very fast and sometimes finding an analytical solution is simply not possible if the underlying physical principles are unknown to the programmer. In these cases other approaches have to be taken to get an idea which change has occurred for a given sensor signal as input.

In this paper, we introduce a systematic approach to recognize continuous changes in manipulation tasks using external sensors, regardless of the type of sensor. The rest of this paper is organized as follows: In Section II, we give a short overview of related work concerning this topic. In Section III, we define the mathematical properties of continuous changes and define a change model. Based on this model we show which requirements must be fulfilled by the change and the supervising sensors to allow a successful recognition of and adaptation to the change. In Section IV, we describe methods to recognize and adapt to a change during execution of the task. In Section V, we show the validity of our approach with two experiments. In the last Section VI, we give a short summary of our work and discuss further steps.

II. Related Work

The task of infering information from noisy sensor data is covered thoroughly by various books on pattern classification, e.g. [8]. But all of these describe methods how to extract the relevant information from the sensor values, assuming that this information is somehow present in the data. Multiple papers exist dealing with the task of planning sensing strategies for robots, e.g. [15, 23]. Most of these assume a specific task [1, 10] or are aimed at employing multi-sensor strategies [3, 7]. Various papers deal with the task of setting up the sensors in the work cell to allow information retrieval [12]. [14] proposes a general platform for sensor data processing, but once more assumes that the sensors are already capable of detecting changes. Papers covering the topic of employing sensors for robot tasks from a general point of view are [9, 21]. There is one paper dealing specifically with drift in the servo motors of robots joints [13], but again, this work is geared towards a specific type of sensor.

Two types of sensors are typically used for manipulation tasks: Force-/torque sensors and cameras. When force-/torque sensors are employed, maps are created describing the measured forces with respect to the offset to the goal position. [4] describes possibilities to either analytically compute these maps and or create these maps from samples. Based on this, [24] shows how these maps can be computed using CAD data of the parts involved in the task. In both cases, the maps must be created before the actual execution of the task and are only valid if the parts involved are not subject to dimensional variations. When the information is acquired using cameras, the first step is to perform some kind of pre-processing of the data to extract the relevant information. To determine in which way this information relates to a positional variation is once more task of the programmer and highly dependent on the type of the task. Examples are given in [6, 20, 25].

In summary, all of the papers mentioned above either propose specific solutions for given types of sensors and tasks or propose algorithms to extract the relevant information from the sensor data. Here, we are interested in defining the fundamental properties a sensor must fulfill in order to allow such an information extraction. These properties should be independent from the actual task and only define requirements which must be met by the sensor in order to allow for a recognition of and adaptation to the change. Using this knowledge a programmer can decide which sensor can be employed to monitor changes during executions of a robot task and how a change in the sensor data can be mapped to a position deviation in Cartesian space.

III. Requirements for Change Detection

In this section, we will define the term *continuous change* and show how this change can be modeled using analytical terms. Based on this, we show which premises must be fulfilled in order to successfully recognize an occurring change during a manipulation task. Then, we describe methods how a compensation function can be determined that computes a position deviation for a given sensor signal.

A. Definition and properties of a continuous change

We define a *continuous change* as a spontaneous position deviation in Cartesian space between an estimated and an actual position of an object in the workspace of the robot between two consecutive executions of the same task.

This means that we will approach the position p_{obj} of an object we believe to be correct during each execution t of a task and measure its position deviation Δp_{obj} compared to the previous execution:

$$\Delta p_{obj} = p_{obj}(t) - p_{obj}(t-1)$$

This definition refers to the position deviation of the object in Cartesian space. But we will need a sensor to recognize this deviation. This sensor must not be the same which is used to approach p_{obj}, otherwise we are unable to recognize the change. This can be explained by two examples: In the first example, the position is determined using the internal sensors of the robot. If the object has moved, we cannot recognize this change solely with the internal sensors. Instead we have to employ a second, external sensor to measure if a deviation has occurred. In the second example, we use a force/torque sensor to describe a force-dependent position. In this case, we can employ the internal sensors of the robot to check if this position has moved.

We see that the position p_{obj} of an object in Cartesian space is mapped to a (vector of) sensor value(s), m_{obj}, in the measurement space of the sensor, so we have a function f_{change}: $R^6 \rightarrow R^m$, where m is the dimension of the sensor:

$$f_{change}(p_{obj}) = m_{obj}$$

B. Requirements of the drift function and sensor

To successfully adapt to the change, we must be able to infer the position deviation from the sensor values, that is build the inverse function of f_{change}, f^{-1}_{change}: $R^m \rightarrow R^6$ with

$$\exists f^{-1}_{change} \; with \; f^{-1}_{change}(f_{change}(p_{obj})) = p_{obj}$$

Based on this requirement, we can directly postulate that a physical change must modify the sensor signal. Otherwise we would not be able to recognize a change, that is

$$f_{change}(p_{obj}) = c \, \forall \, p_{obj}, c \in R$$

We can easily see that there exists no inverse for this function. There is always an inverse for all bijective functions. In addition, if f_{change} is continuous, f_{change} is strictly monotonic as well. If necessary the surjection can be guaranteed by a deliberate constraint of the measurement range of the sensor.

Another requirement is that the dimension of the sensor must be at least as high as the degrees of freedom (DOF) of the change. Otherwise there can be no inverse for f_{change}.

If we are only taking the physical effect into account, which maps a position to a set of sensor values, there is no universal solution for f_{change}. Instead the sensor values for a given position are highly dependent on the objects position in the workspace and the type of object which is to be manipulated. It should be noted though, that there are similarities of the change function to the Jacobi matrix [19].

Another thing which must be kept in mind is the signal-to-noise ratio (SNR) of the sensor for the given object. We can only recognize a change if the alteration of the sensor values for a given position deviation is significantly higher than the noise generated by the sensor.

C. Determination of the change function

To be able to adapt to a continuous change, we must specify a function f_{comp}: $R^m \rightarrow R^6$ with

$$f_{comp}(m_{obj}) = f^{-1}_{change}(f_{change}(p_{obj}))$$

so that we can compute an estimated change for a given sensor value.

1) Analytical computation of f_{comp}

The straightforward way to determine f_{comp} is to work out an analytical solution. But sometimes this task proves to be too complex: While the type of function may be known, it can be extremely difficult to determine a set of parameters, which fit the function well enough to the problem at hand.

2) Analytical approximation of f_{comp}

If we cannot calculate the parameters of f_{comp} analytically, but at least have some idea about the type of the function, we must create a training set T containing pairs of type <physical change, sensor values>. To build T, we systematically create artificial changes and measure the sensor values for each change. Using this training set, we can approximate f_{comp}, so that it will be close enough to f^{-1}_{change} to ensure a valid guess for an existing change. The accuracy of f_{comp} is then determined by the size of T and the accuracy of each sample in T. The minimum size of the training set is determined by the complexity of f_{change} and is equal to the number of parameters in the change function, although the size should be significantly higher to counterbalance the noise of the sensor data.

Once we have created a training set, we can use it to approximate f_{comp}, provided that f_{change} fulfills the requirements mentioned in Section III.B. To achieve this, we use analytical or iterative methods to fit a given function to the data which minimizes, e.g., the mean error of all pairs in T. Various algorithms for curve fitting exist, the most popular are the Householder algorithm [11, 18] and the Levenberg-Marquardt algorithm [16, 17]. The problem with these and other approaches is that we must have some kind of idea about the general type of f_{comp}, that is which sensor values are influenced by which DOF of the change.

3) Estimation of f_{comp} for unknown function types

The problem gets even more complex, when the type of function itself is unknown. Here, we will need the same training set as for an analytical approximation. Then we can employ series expansion, neural nets or equivalent methods to obtain a solution for T. In this case, we estimate the type of function which fits the training set best. For example, we can use multilayer perceptron (MLP) networks [2, 22] to implicitly learn f_{comp}. While all of these approaches save us the task of analytically determining the general outline of f_{comp}, the price we have to pay for this is that the size of T increases drastically, because we will need a lot more samples to train the MLP adequately.

TABLE 2
CLASSIFICATION OF APPROXIMATION METHODS TO DETERMINE THE COMPENSATION FUNCTION.

Knowledge about f_{change}	**Methods for approximation**
Thorough / Complete	Analytical computation
General type of function	Analytical approximation
Evaluated training data only	Function estimation
Unknown or no training data	Reinforcement or unsupervised learning strategies

4) Learning algorithms to determine f_{comp}

Both approaches for approximation or estimation of f_{comp} require a training set. So far, we have assumed that we create T offline before the actual execution. Another option is to create T online, so we let the robot figure out which change will produce which sensor values during the execution of the task. The actual algorithms to infer f_{comp} from T remain unchanged.

The offline approach has the advantage of providing us with very exact pairs for T, resulting in a very well approximated function f_{comp}. This can be used directly and delivers the best possible results without having to re-train at a later point. The disadvantage is, that we need to know in which DOF the change will occur beforehand, in order to create a set T which covers all possible changes. If we choose to place no external restrictions on the change, we must deal with six possible DOF, thus enlarging T drastically.

In case that there is no way to obtain some set of training data offline, we must make use of learning algorithms to obtain and classify training data during the actual task execution, called online creation of T. Every time we measure a change, we add a new pair to T. In this case, we do not have to deal with the task of artificially building the training set. But the main problem with this approach is that we do not know the correct change for a given sensor value.

We further classify learning algorithms into three groups: Supervised, reinforcement and unsupervised learning. In supervised learning, the programmer checks the estimation and corrects it, if necessary. Here, this approach is impractical, because the programmer would have to stay with the robot during execution, negating the desired time saving of online creation of T. Reinforcement learning strategies have no need for supervision. Instead they guess the change and check its correctness by measuring and evaluating after the adaptation. Based on this rating, each pair in T is modified. Unsupervised learning strategies don't even evaluate how well the change has been guessed in the last execution.

All of the described approaches are summarized and ordered by their complexity in Table 2. If the underlying physical relation between sensor and position deviation is known, the accuracy of the approximation is only dependent on the signal-to-noise ratio of the sensor.

In case of online creation of T, estimating f_{comp} with neural nets or similar methods may not be reasonable: In the very first executions T will be too small for adequate training. Later on, we have to keep in mind that every pair in T describes a guess of a change for a given sensor value. The quality of T is determined by the quality of every guess. When we have to guess a lot of values, this will deteriorate the quality of T thus reducing the net's ability to correctly calculate the drift for a given sensor value.

In all cases, f_{comp} must be continuous (or will be, should a neural network be used), otherwise it cannot be approximated by the methods described above.

IV. Supervising and Adapting to Changes During Execution

In this section, we will explain how an adaptive change compensation can be integrated easily into a robot program. We will use the basic principles described in Section III and show how these can be combined to form a powerful, yet easy method to deal with manipulation changes.

For all applications, one can sub-classify between the tasks which must be accomplished during initialization and the supervising and compensation process during execution.

A. Setup and Initialization

The first thing to do is decide which positions are to be supervised for changes. This is usually easy: The positions were tools are placed, parts are delivered to the robot, etc. When a position is chosen, one must consider the possible DOF of the change. This task is not as easy as it may sound: Variations are usually straightforward, because they represent the flexibility the robot is supposed to cope with. Unfortunately, the drift is an unintentional factor. But usually it is easy to make some assumptions about it. For example, if an object is placed on a table, one can assume that the position and orientation of it may move along the x/y-plane of table and that the object may rotate around its z-axis. Drifts (or variations) along the z-axis and rotations around the x- and y-axis may happen, but are exceedingly rare. So here, one can limit the change which is to be supervised to three DOF: The x- and y-axis for translations and the z-axis for rotation.

The next thing to be done is to decide which sensor will be used to supervise the change. It must be kept in mind, that the sensor signal must alter when a change occurs and that the dimension of the sensor values is at least as high as the DOF of the change. Additionally one has to set a change threshold c_{change} for each dimension of the sensor. The purpose of this treshold is to prevent the robot from recognizing false or minimal changes due to the SNR of the sensor. The thresholds should be chosen at least as high as the SNR of the sensor.

The last thing to do during the setup is to move the robot to the desired position p_{obj} and measure the initial sensor values m_{init}. This set of values defines the target values against which the robot checks for an occurring change during execution.

If the training set is to be acquired offline, this step is extended to that point that either the object itself is moved deliberately by a preset, known deviation or the robot moves by a preset distance. In both cases the resulting pairs of position deviation and corresponding sensor values (p_i - p_{obj}, m_i - m_{init}) are recorded and added to the training set. The subtraction of p_{obj} and m_{init} from all measured values yields the advantage that $f_{change}(0)=0$, so the input will be zero if no change has occurred. Then one can use one of the methods described in Section III.C to derive f_{comp}. If f_{comp} shall be acquired online, one has to make an initial guess about the nature of f_{comp}, that is set some initial parameters which roughly specify f^{-1}_{change}. Otherwise the robot will not be able to perform a valid adaptation right from the beginning.

Additionally, the approximation can be used to extract a measure of the SNR of the sensor, which is simply the mean error of all samples compared to f_{comp}. Then, the accuracy σ_{error} of the adaptation is given by

$$\sigma_{error} = f_{comp}\left(\frac{1}{n}\sum_{i=1,\ldots,n} m_i - f_{comp}(p_i)\right)$$

for n samples in T. It is impossible to compensate a change more exactly than this measure.

B. Adaptation to changes during execution

Each time the robot approaches the position in question in execution i, it will record a new set of sensor values m_i and subtract m_{init}. If the resulting value exceeds c_{change} for this dimension of the sensor, a significant change has occurred. The estimated change p_{guess} is obtained by applying f_{comp} to m_i. Note, that p_{guess} describes the deviation of the object in question in regard to its original position and is not a position in world coordinates. In which way the robot deals with this deviation depends on the type of change: If we deal with a variation the robot should adapt to it and modify the following movements and operations to take care of the fact that the object has moved. If we deal with a drift, we can either choose to try to compensate the drift, that is put the drifted object back to its original position or simply adapt to it and treat it like a variation.

In summary, the proposed method is independent of the type of sensor and can be applied to all situations if the sensor is capable of detecting a change.

V. Experiments

In this section, we will show the validity of our approach by two experiments. In the first experiment, we implement a recognition of rotational variations around an objects z-axis using distance sensors. We determine the change function first analytically and then approximate it using a training set generated offline. Afterwards, we compare both approaches. In the second experiment, we train the robot to react to translational deviations along the x- and y-axis of a disk placed on a table. First, we fit a given function to our training set and afterwards train a neural network to learn f_{change} and then compare both approaches.

A. Measuring the rotation of a steel ruler

In the first experiment, we want to find a measure for the rotation of a steel ruler lying on a table, so we do not have to ensure that the ruler is orientated correctly every time we want to grasp it. For this purpose we use three Sharp GP2D120 distance sensors set up in a straight line facing the ruler (Figure 1, left). Each sensor has a resolution of 1 cm in the range from 4 to 30 cm. The sensors are set up 20 cm apart from each other. While it is a straightforward task to determine the rulers distance from the sensors, which can be solved analytically quite easily, we are interested in allowing rotational variations of the ruler. In theory, we can measure this variation by subtracting two sensor values from each other and comparing this value to one of the original sensor values. Then the rotation of the ruler is simply

$$f_{comp}(s_i - s_j) = \arctan\left(\frac{s_i - s_j}{d}\right)$$

where s_i and s_j describe distance measurements of two sensors i, j. The parameter d describes the distance that the two sensors are set up apart from each other. Note, that this is the only parameter in the change adaptation function. In theory, we would determine this parameter and set up an appropriate algorithm to calculate the rotational drift. But here, we will try to determine this parameter experimentally. To achieve this, we have grasped the ruler, rotated it counterclockwise in steps of one degree and measured the sensor values. This compromises our training set (Figure 1, right) of 30 samples for angles from 0 to 30°. To approximate this function we have used a computational approach described in Section III.C.2 employing the Levenberg-Marquardt algorithm. We have used two general types of functions for the approximation. The first is of the same type as the theoretical drift function, an arctan-function with one free parameter, d_{guess}. The second function is a simple linear function of type $a_{guess} + b_{guess}*x$, with two parameters to approximate. In the case, where we used the sensors spaced 40 cm apart, we could only use the first 22 of the 30 training pairs, as for rotations bigger than 22°, the measured distance of the ruler to the third sensor exceeded the sensor range. The functions are displayed in Figure 2 and the results are summarized in Table 3. We have calculated the mean error of each sample to the theoretical value, giving us an impression about the SNR of the sensors. This is relatively low, so we can only recognize changes which are larger than 2°, but the width of the gripper is big enough to deal with this tolerance. To evaluate how well the approximated functions compare to the theoretical function, we have analytically computed the integral error of the difference of the two functions on a range from 0 to 10 cm. We have chosen this range because it is significantly higher than the SNR of the sensors and a

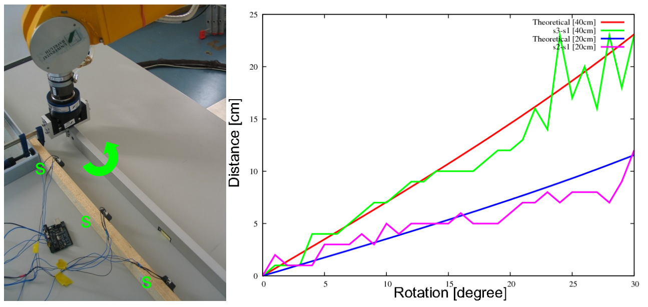

Figure 1: Left: Setup of experiment A. Three distance sensors are used to determine the rotation of the steel ruler. Right: Measured distances for given angles and theoretical values.

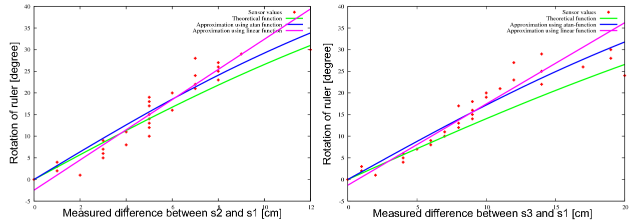

Figure 2: Theoretical and approximated functions for a rotational drift compensation function. The training data is shown as red dots. The theoretical function is shown in green. The approximated functions are shown in blue (arctan) and purple (linear).

TABLE 3:
ACCURACY OF APPROXIMATED CHANGE FUNCTIONS COMPARED TO THEORETICAL FUNCTION FOR TWO DIFFERENT SENSOR DISTANCES.

Sensor distance [in cm]	Number of samples used	Mean error of raw data to theoretical model [in °]	Integral error of arctan approx. to theoretical model [0-10cm]	Integral error of linear approx. to theoretical model [0-10cm]
20	30	2.63	1.64	17.83
40	22	2.28	0.92	11.46

difference of 10 cm in two sensors values would mean a rotation of 26° and 21° for a sensor distance of 20 cm and 40 cm respectively. This gives us an idea of how well we can estimate the variation adaptation function with the given training data. We can see, that if we know that the change function is an arctan function, we can form an estimate with a relatively low error. The error is significantly higher if we do not know the type of change adaptation function and guess it to be linear.

In summary, even with the relatively low resolution of the sensor we are able to enable the robot to flexibly grasp the steel ruler, even if it is displaced by up to 13 cm and rotated by up to 21°. All we had to to is build a training set by deliberately rotating the ruler and measuring the resulting sensor values.

B. Measuring the drift of a disk on a table

In this experiment, we use a force/torque sensor to recognize the deviation of a round disk along the x- and y-axis of a table. The use of this sensor has the advantage, that it is mounted at the robots tool-tip, so no additional sensors have to placed in the robots workspace (Figure 3).

The idea is, that if the disk drifts along the table, we can measure a significant moment along the x- and y-axis because we will not grasp the disk in the center. While it is still possible to analytically determine a drift compensation function, this is highly dependent on the weight of the disk, its position, etc. Because of this, we try to determine f_{comp} experimentally.

We have chosen to acquire the training set T with an offline approach and have set up a simple algorithm which moves along a grid of a specified size around the center of the disk, picks it up at given intervals and measures the resulting moments.

We know that the relation between the measured moments and the drift is linear and have set up the general drift compensation function in that way:

$$f_{comp}\begin{pmatrix} m_X \\ m_Y \end{pmatrix} = \begin{pmatrix} a_1 + b_1 * m_X + c_1 * m_Y \\ a_2 + b_2 * m_X + c_2 * m_Y \end{pmatrix}$$

We have assumed that a deviation along the x- and y-axis influences the moments along both axes. The sensor values along the grid and the approximated functions for a grid size of 9*9 are shown in Figure 4.

We have used the same training sets to train a MLP network to learn the same drift compensation function. The MLP consists of three layers with two input and output neurons and three neurons in the hidden layer. Each MLP was trained for a maximum of 100000 epochs with an desired error of 0.001. The functions learned by the MLP network for a grid size of 15*15 are shown in Figure 5.

It can be seen that a deviation along the x-axis mainly influences the moment along the y-axis and vice versa. We have calculated the mean error of the approximated function and the training set for various grid sizes, giving us an idea about the SNR of the sensor. To test our drift compensation function, we have deliberately moved the disk by a random offset p_{real} and recorded the corresponding sensor values m_{real}. Afterwards, we have entered m_{real} into our compensation function and compared the estimated drift p_{guess} against the real drift p_{real}. We have repeated this with 100 different offsets for every grid size. The results are summarized in Table 4.

We can see that due to the SNR of the sensor, which is about 3.5 Nm, we need at least 16 samples in our training set to obtain a reasonable approximation for f_{comp}. Below this value the noise of the sensor inhibits a reasonable approximation. On the other hand, it is unnecessary to create excessively large training sets with hundreds of samples. Above 81 values, there is no significant improvement of the approximation if we increase the number of samples.

In case we do not know the type of function and use a neural network to approximate the function we need more than twice as many samples to approximate the change function with the same accuracy. But for training sets of this size, an equally good approximation is possible.

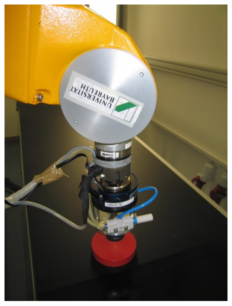

Figure 3: Setup of the second experiment. The vacuum gripper grasps the disk at specified offsets from the center and measures the forces and torques for each position. Based on this information the drift compensation function is approximated using a Householder approximation and a MLP neural network.

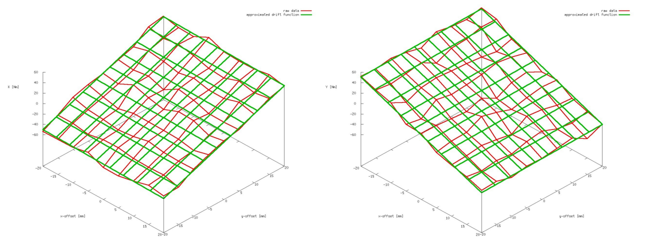

Figure 4: Measured moments mX and mY (red) of the disc for given deviations and approximated drift functions using the Householder algorithm (green) for supervised acquisition with a training set of 81 samples.

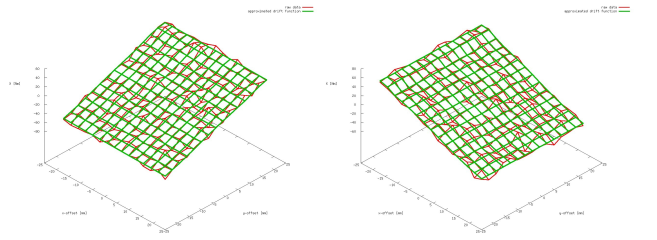

Figure 5: Measured moments mX and mY (red) of the disc for given deviations and approximated drift functions (green) for supervised acquisition using a neural network with a training set of 225 samples.

TABLE 4:
COMPARISON OF APPROXIMATION WITH A GIVEN TYPE OF FUNCTION AND A MLP NEURAL NETWORK.

Number of samples used to approximate drift function	Mean error of raw data to approximated model using the analytical function [in mm]	Mean accuracy for 100 random drifts using analytical function [in mm]	Mean error of raw data to approximated model using the MLP network [in mm]	Mean accuracy for 100 random drifts using the MLP network [in mm]
4	3.13	10.99	0.31	12.47
9	3.28	6.33	1.23	8.99
16	3.57	4.03	1.97	7.92
25	3.61	3.57	3.10	4.88
81	4.01	3.62	4.55	4.05
121	4.09	3.60	4.65	4.03
225	4.05	3.62	4.04	3.50

The accuracy of the approximated change functions determined with the Householder algorithm and a neural net for various sizes of supervised training data. The accuracy is determined by testing both functions with samples which were not used for the approximation.

In both cases the low mean error of the approximations for low grid sizes is caused exactly by this fact: There are only a few training samples, so it is possible to find a very good approximation with a low cumulative error for all samples, but the mean error for samples not used for the approximation is relatively high. After a certain sample size, the mean error of the approximation remains constant, thus enabling us to determine the SNR of the sensor.

With both approximations, we are able to automatically check if and how far the disk has moved along the table and use the result to adapt to the environment accordingly. The approximations are precise enough to allow for an adaptation without the need to re-grasp the disk.

In summary, we have shown that it is possible to easily approximate change functions independent of the type of sensor with relatively low effort. The less knowledge we have entered into the approximations, the bigger the training set must be in order to successfully approximate the function.

VI. Conclusion

The aim of this paper is to enable a programmer to easily employ external sensors for flexible robot programms. The focus of this work is to show that the function describing the connection between sensory data and positional variations can be acquired automatically and task independent by the robot without the need for intricate calculations by the programmer. We have outlined the basic requirements on both the sensor and the physical relation of sensor data and position deviation that must be met to compute a change from a sensor signal. We have presented methods to determine this function automatically and have ordered them by the amount of knowledge necessary for the approximation. The presented requirements and methods are independent from the type of sensor and can easily be incorporated into a robot program. Finally, we have presented two experiments to validate our research. We have shown that is possible to employ the proposed methods to successfully approximate the change function for two given applications using different sensors.

Further work needs to be in finding ways to generate the training set online and develop a rating function which is capable of telling how good an estimated change was guessed depending on sensor data gained in the very next executions. Additionally, we want to develop approaches how this change function can be employed to deal with both variations and drifts adaptively in a systematic way, that is independent from the actual type of application. Finally, we want to check if fuzzy rule based sets describing the change function may be employed.

References

[1] M. Adams, “Sensor Modelling, Design and Data Processing for Autonomous Navigation”, World Scientific Publishing, 1998, ISBN 9810234961

[2] C.M. Bishop, “Neural Networks for Pattern Recognition”,Oxford University Press., 1995, ISBN 0-19-853849-9

[3] B. Bolles, H. Bunke, H. Christensen, H. Noltemeier, “Modelling and Planning for Sensor-Based Intelligent Robot Systems”, Seminar on, Schloß Dagstuhl, 1998, http://www.dagstuhl.de/Reports/98391.pdf

[4] S. R. Chhatpar, M. S. Branicky. “Localization for robotic assemblies with position uncertainty”. Proc. IEEE/RSJ Intl. Conf. Intelligent Robots and Systems, Las Vegas, NV, October, 2003.

[5] J. Deiterding, D. Henrich “Automatic adaptation of sensor-based robots”, Int. Conf. o. Intelligent Robots and Systems 2007

[6] G. Dudek, C. Zhang, “Vision-based robot localization without explicit object models” Int. Conf. On Robotics and Automation, 22-28 Apr 1996, ISBN 0-7803-2988-0, pages 76-82 vol.1

[7] M. Dong, L. Tong, B.M. Sadler, “Information retrieval and processing in sensor networks: deterministic scheduling vs. random access”, Proc. o.t. Int. Symp. on Information Theory, 2004. ISIT, pages 79 - 85

[8] R. Duda, P. Hart and D. Stork, “Pattern Classification”, Wiley & Sons, 2000, ISBN 0471056693

[9] R.J. Firby, “Adaptive execution in complex dynamic worlds”, Dissertation, Yale university, 1989, www.uchicago.edu/users/firby/thesis/thesis.pdf

[10] G. Hager, “Task-Directed Sensor Fusion and Planning: A Computational Approach”, Springer, 1990, ISBN 079239108X

[11] A. S. Householder, “Unitary Triangularization of a Nonsymmetric Matrix” Journal ACM, 5 (4), 1958, 339-342.

[12] S.A. Hutchinson, R.L. Cromwell and A.C. Kak, “Planning sensing strategies in a robot work cell with multi-sensor capabilities”, in. Proc. IEEE Int. Conf. On Robotics and Automation, 1988, pages 1068-1075

[13] J. W. Jeon, S. Park, S. Kim, “Compensation for servo drift in industrial robots”, Industrial Electronics, Control, Instrumentation, and Automation - IECON, (2), 1992, pp 589-594

[14] D. Kriesten, M. Rößler, et al., “Generalisierte Plattform zur Sensordatenverarbeitung”, Dresdner Arbeitstagung Schaltungs- und Systementwurf , 2006, http://www.eas.iis.fhg.de/events/workshops/dass/2006/dassprog/pdf12_kriesten.pdf

[15] U. Leonhardt, J. Magee, “Multi-sensor location tracking”, Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking, Dallas, USA, 1998, ISBN 1-58113-035-X , pages: 203 – 214

[16] K. Levenberg, “A Method for the Solution of Certain Non-Linear Problems in Least Squares." Quart. Appl. Math. 2, 164-168, 1944.

[17] D. Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters." SIAM J. Appl. Math. 11, 431-441, 1963.

[18] D. Morrison, “Remarks on the Unitary Triangularization of a Nonsymmetric Matrix”, Journal ACM, 7 (2), 1960, 185-186

[19] D. Orin, W. Schrader, “Efficient Jacobian determination for robot manipulators”, Rob. Research: The 1st int. symposium, M. Brady and P. R.P. Paul (ed.), MIT Press, Cambridge, MA, 1984

[20] N. Paragios, G. Tziritas. “Adaptive Detection and Localization of Moving Objects in Image Sequences” Signal Processing: Image Communication, 14:277--296, 1999.

[21] R. Pfeifer, C. Scheier, “From perception to action: The right direction”, Proc. ``From Perception to Action'' Conference", IEEE Computer Society Press, Los Alamitos, 1994, pages = "1-11"

[22] R. Rojas, “Neural Networks. A Systematic Introduction”, Springer, 1996, ISBN 978-3540605058

[23] K. Rui, M. Yoshifumi, M. Satoshi, “Information Retrieval Platform on Sensor Network Environment”, IPSJ SIG Technical Reports, 2006, No. 26, ISSN 0919-6072, pages 37-42

[24] U. Thomas, A. Movshyn, F. Wahl, “Autonomous Execution of Robot Tasks based on Force Torque Maps”, Proc. o. t. Jnt. Conf. on Robotics. 2006, Munich, Germany, May 2006

[25] M. Wheeler, “Automatic modeling and localization for object recognition”, Carnegie Mellon University, Computer Science Technical Report CMU-CS-96-118, 1996.