ONLINE CALIBRATION OF ONE-DIMENSIONAL SENSORS FOR ROBOT MANIPULATION TASKS

Jan Deiterding and Dominik Henrich
Lehrstuhl für Angewandte Informatik III
Universität Bayreuth, D-95440 Bayreuth, Germany

Keywords: Learning and adaptive systems, Architectures and programming, Compliant assembly.

Abstract: The purpose of this paper is to enable a developer to easily employ external sensors emitting a one-dimensional signal for flexible robot manipulation. To achieve this, the sensor must be calibrated using data tuples describing the relation between the positional change of the supervised object and the resulting sensor value. This information is used for adaptation methods, thus enabling robots to react flexibly to changes such as workspace variations or object drifts. We present a sensor-independent method to incrementally generate new data tuples describing this relation during multiple task executions. This method is based on the Secant method and is the only generally applicable solution to this problem. The method can be integrated easily into robot programs without detailed knowledge about its functionality.

1 INTRODUCTION

Industrial robots are able to perform complex tasks with utmost precision and at high speed without exhibiting symptoms of fatigue. However, these tasks are nearly always executed in a fixed environment, i.e. the precision is achieved by ensuring that all objects are placed in exactly the same position every time. All parts must have the same dimension, position, orientation, etc. Only by employing external sensors such as vision or force/torque sensors, a robot can deal with imprecisions and variations in objects and the environment. When designing such programs for more flexible robots, a developer faces the problem of determining the relation between the sensor value obtained and the actual physical variation of the supervised object.

The task is to find a *change function* that transforms sensor values into Cartesian descriptions of the change in order to successfully deal with these. The classical approach is to analytically determine a function describing this mapping. However, for complex sensors this task quickly becomes difficult and it is sometimes simply not possible to find an analytical solution if the underlying physical principles are unknown to the developer. In these cases data tuples describing the relation between the positional change of an object and the resulting sensor value are recorded and a selected type of function is fitted to these tuples. These approaches require a large amount of analysis and programming before the robot executes the task for the very first time. Another downside is, that this pre-calculated solution is fixed and prevents the robot from adapting to changes of the environment. For example, the robot must be stopped and re-calibrated if a drift in the workspace or the sensor system occurs. The advantage in the use of change functions is that an additional layer of abstraction is introduced. The program can be designed independent from the actual sensor because all workspace changes are described in Cartesian coordinates. Now, we may replace the sensor with a different one using another measuring principle and – as long as the change function is correct – no alterations have to be made to the program. General features of change functions are described in (Deiterding, 08) and a general outline to determine these functions is given, but no generally applicable method is presented to calibrate sensors iteratively during the execution of a robot manipulation task.

In this paper, we focus on sensors emitting one-dimensional signals, such as distance or force/torque sensors. We do not deal with imaging sensors as this class of sensors usually requires an upstream pattern matching algorithm to distinguish the relevant information from background data. We show how calibration data for a change function can be

Deiterding J. and Henrich D. (2009).
ONLINE CALIBRATION OF ONE-DIMENSIONAL SENSORS FOR ROBOT MANIPULATION TASKS.
In *Proceedings of the 6th International Conference on Informatics in Control, Automation and Robotics - Robotics and Automation*, pages 387-395
DOI: 10.5220/0002181803870395

computed iteratively during the first executions of the task and how these methods can be integrated easily into the programming environment, only requiring the developer to specify a minimum of task-dependent parameters. Additionally, we show how the robot adaptively optimizes the task with respect to execution time based on a steadily improving approximation of the function.

The rest of this paper is organized as follows: In Section 2, we give a short overview of related work concerning this topic. In Section 3, we will outline a framework with which a developer can create sensor based robot programs that automatically acquire calibration data during execution. In Section 4, we describe which algorithms are encapsulated into this framework and compare them with other approaches. Section 5 describes how a typical robot task can be solved using our approach. In the last section, we give a short summary of our work and discuss further steps.

2 RELATED WORK

The task of inferring information from noisy sensor data is covered thoroughly by various books on pattern classification, e.g. (Duda, 00). But all of these describe methods for extracting the relevant information from sensor values, assuming that this information is present in the data. Multiple papers dealing with the planning of sensing strategies for robots exist, e.g. (Leonhard, 98), (Rui, 06). Most of these involve a specific task (Adams, 98), (Hager, 90) or are aimed at employing multi-sensor strategies (Bolles, 98), (Dong, 04). Various papers deal with the use of sensors in the work cell to allow for information retrieval (Hutchinson, 88). In (Kriesten, 06), a general platform for sensor data processing is proposed, but once more it is assumed that the sensors are already capable of detecting changes. More general discussions of employing sensors for robot tasks can be found in (Firby, 89), (Pfeifer, 94).

Two types of sensors are typically used for manipulation tasks: Force/torque and vision sensors. When force/torque sensors are employed, maps may be created describing the measured forces with respect to the offset to the goal position. (Chhatpar, 03) describes possibilities to either analytically compute these maps or create them from samples. Based on this, (Thomas, 06) 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. If 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 how this information relates to the positional variation is once again the task of the developer and highly dependent on the nature of the task. Examples for information retrieval using vision sensors are given in (Dudek, 96), (Paragios, 99) and (Wheeler, 96).

In summary, all of the papers mentioned above either propose specific solutions for specific types of sensors and tasks or algorithms to extract the relevant information from the sensor signal. A problem is that these solutions do not outline a general approach which can be used regardless of the type of sensor. Additionally, all papers assume that the developer is capable of integrating the methods into his own robot program. Unfortunately, this is usually not the case for developers in small and medium sized enterprises, which often possess only basic knowledge about robot programming.

Here, we are interested in determining the relationship between the sensor signal and the Cartesian deviation iteratively during multiple task executions. We want to integrate this algorithm into an easy-to-use interface that will enable developers having no special knowledge in robot programming to create adaptive robot programs. We only focus on one-dimensional data, such as distance sensors or force/torque sensors. Vision sensors always require some kind of pre-processing that is highly dependent on the task.

3 INTEGRATION INTO THE PROGRAMMING ENVIRONMENT

In this chapter, we will explain how a developer with minimal knowledge about sensor data processing can easily create robot programs that employ external sensors. We will explain which considerations must be made by the developer, how the program must be structured in general, and which parameters are mandatory.

3.1 Setting Up the Workspace

The first thing a developer has to do is to decide in which way a change can occur between consecutive executions of the task. Based on this, a suitable sensor must be chosen that is capable of recognizing

this change and that satisfies the requirements imposed on change functions. Here we will only provide a short summary, see (Deiterding, 08) for a detailed explanation: A change function f describes the alteration of a sensor signal when the object supervised by the sensor has moved. It is a function that relates a Cartesian position to a sensor value. Using the inverse f^{-1} gives us the position p_{est} for the current sensor value. Note that all functions are defined in relation to a pre-set reference position p_{ref} and a corresponding reference sensor value s_{ref}. Only the difference of the current sensor signal to s_{ref} is taken into account. This is not a limitation, but rather a standardization of the function, so the only root of this function is (0,0) because there is only one reference position.

3.2 Online Computation of Change Functions

The central idea of this paper is that the change function f_{real}, which is defined by the task and the sensor, is unknown and cannot be calculated analytically or approximated beforehand. Instead, the robot will compute an approximation f_{est} of f_{real} online during the first executions of the task. Instead of two separate phases – the calibration of the sensor and the actual execution of the task – the calibration process is encapsulated in the execution (see Figure 1). The calibration may take longer now, nonetheless the program will work correctly right from the very first execution. In addition the developer will spend less time setting up the sensor and the program is capable of adapting to changes both in the workspace and in the sensor data, e.g. due to a warm-up of the sensor, without the need for a manual recalibration. The robot starts with a very rough approximation f_{est} of f_{real} and refines this approximation gradually with each execution by incorporating newly gained information.

During execution, the robot uses f_{est}^{-1} to react to Cartesian changes of the supervised object. If the object has moved away from p_{ref} by x_{change} to p_{change}, this is detected through the sensor value s_{act}:

$$s_{act} = f_{real}(x_{change}) \quad (1)$$

Thus, the robot must modify its movement by calculating:

$$x_{est} = f_{est}^{-1}(s_{act}) = f_{est}^{-1}(f_{real}(x_{change})) \quad (2)$$

The stored reference position is then modified accordingly:

$$p_{est} = p_{ref} + x_{est} \quad (3)$$

Now, the robot moves to p_{est}. If f_{est} is close enough to f_{real} then:

$$x_{est} = x_{change} \quad (4)$$

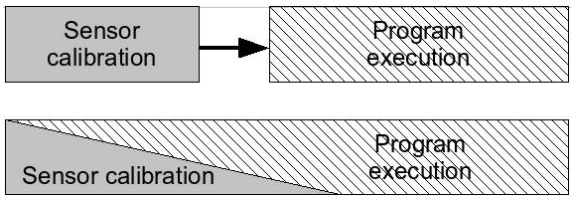

Figure 1: In the classical approach to sensor-based robot programming, the sensor is calibrated before the actual program is executed (top). In our approach, the calibration process is integrated into the execution cycle (bottom).

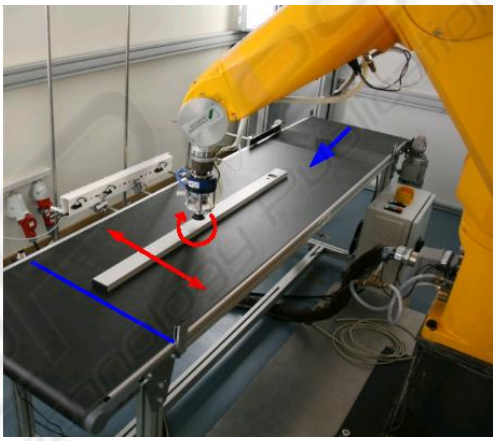

Figure 2: Experimental setup. A steel rod is delivered along a conveyor belt (blue arrow) until it reaches a light barrier (blue line). The rod can be in any position on the belt (red). Shown in this picture is the reference position of the rod in order to be picked up.

If the change was estimated correctly, this knowledge is incorporated into the change function. If the estimate was wrong, then there is not enough information stored in S to perform a reasonable correction using the current sensor value s_{act}. Thus, the correct position must be determined and f_{est} must be modified in such a way that the next estimate will be correct for the current sensor value. Initially this will often be the case since early values of f_{est} are quite inadequate.

When the robot has performed the motion defined by x_{est}, the new position is either correct or it is skewed because f_{est} was not accurate enough. In the latter case, two possibilities arise. The key point is to decide whether the robot motion will modify the sensor signal or not. This is best illustrated by an example. Consider the following task: A steel rod is delivered to the robot via a conveyor belt. The belt

stops when the rod passes a light barrier (see Figure 2). The robot shall pick up the rod using a vacuum gripper and place it in a box for transport. To solve this task, we could construct a feeding mechanism ensuring that the rod is aligned the same way every time. However, we want to allow the rod to be in any position as long as it faces upwards. So we have translational changes along the x-axis and rotational changes around the z-axis of the coordinate system of the conveyor belt. To sense this misalignments, we employ two distance sensors that are placed parallel to the y-axis of the conveyor belt. (Figure 3) The developer faced with the task to design this robot program now has to plan how the position and orientation of the rod can be recognized and how the robot should react. So, there are two cases:

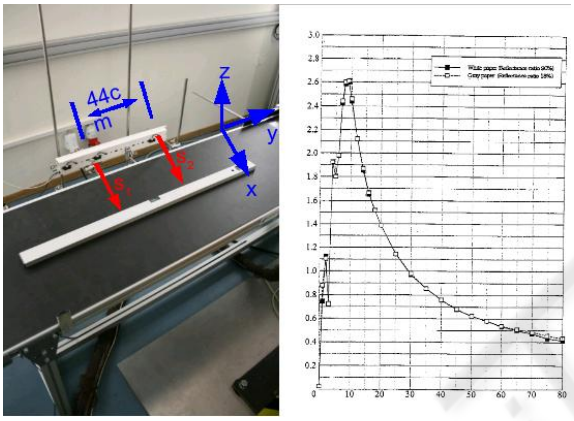

Figure 3: Left: Reference position of the rod and placement of the distance sensors to recognize the position and rotation of the rod on the conveyor belt. The distance is determined using s_1. The rotation is determined using the difference between s_1 and s_2. Right: Scan of the data sheet provided by the manufacturer describing the sensor signal for given distances (x-axis: distance, y-axis: sensor signal). The resolution of the sensor is in the range [10; 80] cm.

Case 1) When the robot moves onto the belt to pick up the rod, this motion does not alter the sensor signal because the rod itself has not moved. In this case the correct position must be searched for. This is usually the case when preparatory sensors are used. The developer can either manually guide the robot to the correct position or use a second sensor to perform an automated search, but it is up to the developer to define a valid search algorithm, because this depends strongly on the task. The search should be kept as simple as possible. When the sensor is calibrated adequately well, the change function's estimate is accurate and always locates the object correctly. So this search is only executed in the very first iterations. Because of this it is not necessary to implement a fast, efficient search strategy, since this represents only a backup strategy in case the change function is still inadequate for a given sensor value. Once the correct position p_{change} has been reached, x_{change} is calculated as

$$x_{change} = p_{change} - p_{ref} \quad (5)$$

and the data tuple (x_{change}, s_{act}) describes a valid data point of f_{real}, because the sensor value has not changed during the search. This tuple is added to a set S describing the current knowledge about f_{real}. With increasing size of S more and more knowledge about f_{real} is collected and the more precise the next estimations will be.

Case 2) This case occurs, when the robot has located the rod and grasped it. Now, a robot motion will alter the sensor signal. In this case a corrective motion can be performed instead of a search. This is usually the case if the sensor is used concurrently. We can employ an automated search; the direction of the search is defined by the Cartesian coordinates that are altered by the sensor. The search terminates when $s_{act} = s_{ref}$. If this value has been reached, the robot has corrected the change. A detailed solution describing the motions involved is described in Section 4.

3.3 Defining the General Program Structure

When defining the program structure, the developer must decide how the adaptation strategy for the change can be integrated into the robot program. This is done at the point when robot movements are executed based on the sensor signal. The robot uses the current sensor value s_{act} and current estimate of the change function f_{est} using S to determine p_{est}.

The key point is to decide whether the robot motion will modify the sensor signal or not. This leads to the following basic program structure:

If a motion does not change the signal, the source code will look similar to this:

```
1 pos = changeFunction();
2 MOVE pos;
3 IF NOT isOkay() THEN {
4        performSearch();
5        pos = getCurrentPosition();  }
6 updateChangeFunction(pos,sensor.value());
```

The robot will calculate and move to the estimated position using the change function by calling the function `changeFunction` (Lines 1 and 2). At this point, a decision must be made if the position is correct, which is either accomplished using a second sensor or by asking the developer to check (Line 3).

If this is not the case, a search is initiated, guiding the robot to the correct position (Lines 4 and 5). Then a new data tuple is added to S improving f_{est} (Line 6) by calling `updateChangeFunction`. This must be called explicitly by the developer to update S with the new, correct position p_{real} for s_{act}.

On the other hand, if a motion does change the signal, the source code will look similar to this:

```
1 DO {
2  pos = getCorrection();
3  MOVE pos;
4 } WHILE sensor.value()!=s_ref;
```

Here, the search is realized using a do/while-loop. We estimate the current change (Line 2) and move the robot accordingly (Line 3) until we have reached the reference position (Line 4). We will describe a suitable method to calculate reasonable correction values in Section 4. Here, it is important that these methods are encapsulated in the function `getCorrection`, so they remain hidden from the developer.

All the developer must do to use these methods is to specify the following parameters:

1) The taskframe and the coordinate(s) in which the change occurs. p_{ref} and s_{ref} are calculated within this taskframe. The default sensor values are recorded when p_{ref} is stored.
2) The sensor used to supervise p_{ref}. This includes a specification of the sensor's signal-to-noise ratio (SNR).
3) A Boolean value specifying if a robot motion will alter the sensor signal. The function `getCorrection` uses this value to determine which estimation method is executed.
4) Furthermore, it makes sense to require all estimates x_{est} to be within a specific range to prevent the robot from leaving the workspace in case of an extreme estimate. However, this may increase the number of corrections necessary to reach p_{ref}.

These four parameters enable the robot to learn a change function adaptively during task execution. All other functionality is independent from the task and is integrated into the function `getCorrection`.

The actual implementation of f_{est} is interchangeable. The calibration data gained by the adaptation is stored in S. It is up to the developer to determine how the tuples in S are used to approximate the function. Any interpolation method can be employed, because no additional knowledge about the function type of f_{est} is necessary. Curve-fitting methods may be used as well, which will lead to a reasonable approximation of f_{est} after fewer executions compared to interpolation methods. But, as is the case with all adaptation and learning methods in general, the more information one has available right from the start, the faster the methods will work adequately.

4 SUPERVISING AND ADAPTING TO CHANGES DURING EXECUTION

In this section, we describe how corrective motions can be executed by the robot using sensor information gained during a movement. All corrective motions are used to supplement the existing knowledge about the change function. We explain how this method can be integrated into a programming environment and kept hidden from the developer.

4.1 Using the Secant Method for Corrective Motions

In principle, it is possible to use a search motion predefined by the developer even if the correction has changed the sensor signal, but this discards the information gained by the alteration of the sensor signal during the search. We can use this information to our advantage and generate corrective motions which locate p_{est} faster than a standard search motion.

Since this correction alters the sensor signal, we use it to judge the performed correction and compute subsequent corrections accordingly. Suppose we knew x_{change}, the first tuple for S would be (x_{change}, s_{act}). Here, we only know s_{act}, not x_{change}. But x_{change} is simultaneously the offset along the x-axis of (x_{change}, s_{act}) from the root, due to the monotonicity of f_{est}. If we perform multiple corrections until we reach the root, we can compute x_{change} as the sum of all corrections the robot has made. From a mathematical point of view, this is equivalent to finding the root of an unknown function.

The Secant method (Press, 92) is defined by the recurrence relation

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n) \qquad (6)$$

where f is an unknown function. As can be seen from the recurrence relation, the Secant method requires two initial values, x_0 and x_1. The values x_n of the Secant method converge to a root of f if the initial values x_0 and x_1 are sufficiently close to the

root. The order of convergence is φ, where $\varphi = (1+\sqrt{5})/2 \approx 1.62$ is the golden ratio. In particular, the convergence is superlinear. This result only holds true under some conditions, namely that f is twice continuously differentiable and the root in question is simple and may not be a repeated root. Change functions, as we have defined them, fulfill these conditions.

In our case, the f is the real change function f_{real} and the first value x_0 is simply the change we wish to calculate, x_{change}, while the second value x_1 is the first corrective motion the robot has performed, x_{est}, which is based on the current estimate of the change function f_{est}. Note that f_{est} is used only once for the initial correction, all subsequent corrections are based on the Secant method (see Figure 4) only using the current sensor values provided by f_{real}. Since the convergence of this method is superlinear, we will not need many additional corrections x_n, $n > 1$, should x_1 prove to be poor.

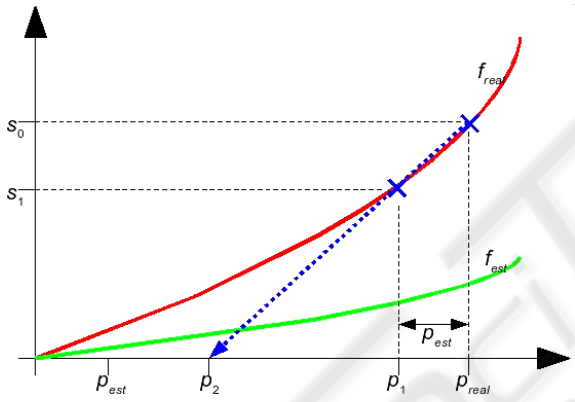

Figure 4: Illustration of the first two steps of the correction algorithm: For a given variation p_{real} we perform an estimated correction p_{est} based on the corresponding sensor value s_0, the real change function f_{real} (red) and our current estimate f_{est} (green). We move the robot to position p_1 and retrieve a new sensor value s_1. We then use the Secant method to grade the last correction and move the robot accordingly to p_2. All subsequent corrections are performed using the Secant method only.

It is important to consider the following: When the next value x_{i+1} is calculated, it must be kept in mind that we have already performed correction x_i before we could measure s_{i+1} to rate x_i. So we must subtract the impact of x_i from x_{i+1}.

Another advantage of this approach is that all corrections x_i and corresponding sensor values $s_i = f_{real}(x_{i-1})$ are known. We can store these as pairs (x_i, s_i) in a temporary stack. When we have reached p_{ref}, we can use this information to create multiple new data tuples for S. If we have performed i corrections until the robot reaches p_{ref}, the topmostpair (x_i, s_i) on the stack already describes a valid data tuple for S. The next pair on the stack (x_{i-1}, s_{i-1}) describes a correction to p_{ref} altered by x_i. So ($x_i + x_{i-1}$, s_{i-1}) is another valid data tuple for the set. Subsequent processing of the stack provides us with a valid data tuple for every correction performed, so we add i new data tuples to S. This leads to an accurate approximation of f_{est} after fewer executions compared to the addition of only one tuple to S in every execution.

The Secant method only works for one-dimensional functions. It is possible to combine multiple sensors to obtain an n-dimensional signal. In this case, the Broyden method (Broyden, 65) can be used, which is similar to the Secant method.

This method is only applicable if a robot motion alters the sensor signal, as is described in Case 2 in Section 3.2. In the first case of that section, there is no other option as to use either a manual guidance method or an automated search.

4.2 Possible Utilization of other Approaches

The Secant method is not the only method to determine the root of a function. Some other methods are Newton's method, fixed point iteration, and the bisection method. We will now compare the Secant method with these and show why the Secant method is the best choice for this task.

Newton's method and fixed point iteration both use the derivative of the function to calculate the next correction. But, as we have explained in Section 1, it is not always possible to find an analytical solution. Additionally, if this solution was known, it would be more sensible to record a number of examples before setting up the main program and use the examples to determine the function parameters.

The bisection method does not rely on the function's derivative, but has another drawback: To find the root of a function f in an interval [a, b], both $f(a) < 0$ and $f(b) > 0$ must hold, or vice versa. If both values are negative or positive, this method cannot be employed. This is a serious drawback for this case, since we cannot ensure that the first correction we have performed will result in a new sensor value which has the inverse sign of the first value.

In summary, we can say that to our knowledge the Secant method is the only applicable method that enables a robot to perform a series of corrective motions without any need for backtracking until the

root of an unknown change function is reached.

5 EXPERIMENTS

In this section, we show the validity of our approach and explain the interaction of all components described in Sections 3 and 4.

We have implemented the task described in Section 3.2. The sensors used are distance sensors GP2D12 made by SHARP with a measurement range of [10; 80] cm. The first sensor supervises the position where the robot is supposed to pick up the rod and measures the translation along the x-axis. The second is located 44 cm away from the first along the y-axis of the belt (Figure 3, left). The difference between the two sensor values describes the rotation around the z-axis.

The data sheet for the sensors shows that the sensor signal is not linear with respect to the physical distance (Figure 3, right), so it is not possible to use a simple linear conversion to determine the translation or the rotation of the rod. In theory, the change function describing the rotation can be derived as an Arcus-Tangens function, but the parameters for this function are unknown. Therefore, the robot shall learn both functions adaptively during task execution. A reference position pref is set up (Figure 3, left), describing the ideal position and orientation the rod should have. This position would be identical with the position of the rod in case a feeding mechanism is employed. It is important to measure the sensor values for p_{ref} as well. Later on, all measurements are compared against these values and if the difference exceeds the SNR of the sensor in question, a change is recognized. The developer now sets up two mappings describing the changes (Table 1).

The robot program for a single task execution is now short and relatively simple:

```
1 PROGRAM pickupRod() {
2   offset_est = getCorrection(Distance);
3   MOVE offset_est;
4   IF (force_z-axis() < force_contact) THEN
5      searchRod();
6   update(Distance, HERE);
7   graspRod();
8   MOVE p_ref;
9   DO
10     rotation_est = getCorrection(Rotation);
11     MOVE rotation_est;
12  WHILE (rotation_est != p_ref)
13  MOVE p_dropoff;
14  releaseRod();
15 }
```

Table 1: Change function mappings used for the experiment.

	Distance	**Rotation**
Position	p_{pickup}	p_{pickup}
Dimension	Translation along *x*	Rotation around *z*
Sensors	Sensor1	Sensor1 - Sensor2
SNR of Sensor	5.0	10.0
Movement modifies sensor signal	FALSE	TRUE
Range of Correction	[-240; 240] mm	[-10; 10] mm

In Lines 2 and 3 the function `getCorrection` receives a reference to a mapping structure defined in Table 1 as parameter and moves the robot to the estimated position of the rod. We use a force/torque sensor to check whether the rod was grasped correctly (Line 4). If this is not the case, we employ a basic search motion probing the conveyor belt in fixed intervals for the rod (Line 5). When the rod is located, we manually update *S*, grasp the rod and move it to the reference position (Lines 6 to 8). At this point the rod may still be rotated by an unknown amount. In Lines 9 to 12 we correct this rotation by repeatedly calling `getCorrection` until the reference position is reached. Then we move the rod to $p_{dropoff}$ and release it (Lines 13 and 14). Note that the program itself does not contain any sensor data processing. Additionally, it is neither necessary for the developer to determine the type of the change functions nor any parameters for these functions. To calculate the Cartesian change for an unknown sensor value, we use a simple linear interpolation over all data tuples in *S*.

We executed the program 100 times. Every time the translation and rotation of the rod was chosen randomly. The initial estimate of both change functions was deliberately chosen badly as a bisecting line (Figure 5). For the change function describing the distance of the rod, we could have also created data tuples using the data sheet of the sensor (Figure 4, right). We have chosen not to do this, for two reasons: Firstly, the data tuples would have to be measured manually by the developer in the figure and modified by the distance of the rod's default position, which is a cumbersome task. Secondly, the data sheet is rather small and the resolution is low so it is difficult to determine exact values. Here, it is easier to just use a bad approximation for the very first executions, because this will change after a few executions. Because of this, the robot was unable to grasp the rod correctly

during the very first executions and also needed multiple corrections to compensate the rod's rotation. After 10 executions the estimations of the change function look similar to the one in Figure 3, and an Arcus-Tangens function respectively (Figure 6). After 100 executions we obtained a precise interpolation of both change functions (Figure 7), allowing the robot to grasp the rod 20 out of 20 times (100%) without the need for a search motion. The rotation was corrected successfully with just one rotation in 14 out of 20 cases (75%). In the other cases, the robot had to perform more than one rotation to align the rod correctly.

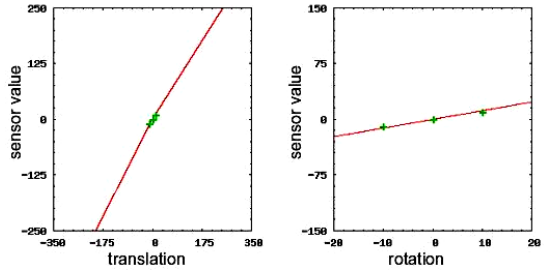

Figure 5: Initial estimates of the change functions used to compute the translation (left) and rotation of the rod (right).

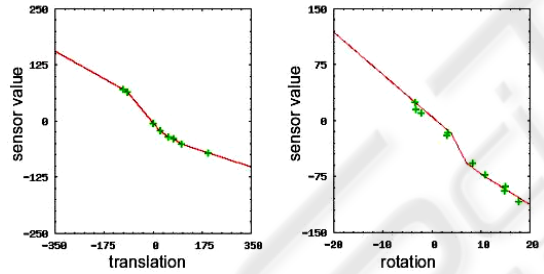

Figure 6: Estimates of the change functions used to compute the translation (left) and rotation of the rod (right) after 10 executions.

The accuracy of the estimated change functions in locating and rotating the rod during the adaptation process is shown in Figures 8 and 9. In both figures we show whether the robot was able to grasp the rod and rotate it correctly using the estimates of the change functions (red). A value of 0 means that the robot had to search for the rod or perform multiple rotational corrections, respectively, while a value of 1 means that the estimate was correct. The green lines show the overall accuracy of the robot over all task executions up to that point, while the blue lines show the accuracy over the last 20 executions. We can see that the robot was capable of grasping the rod correctly nearly all the time after 50 executions,

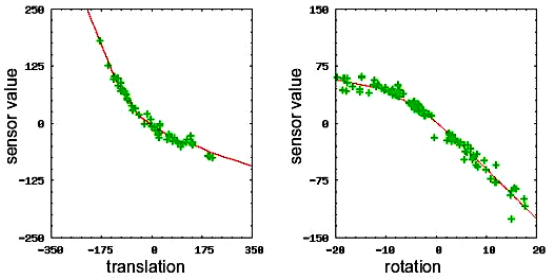

Figure 7: Estimates of the change functions used to compute the translation (left) and rotation of the rod (right) after 100 executions.

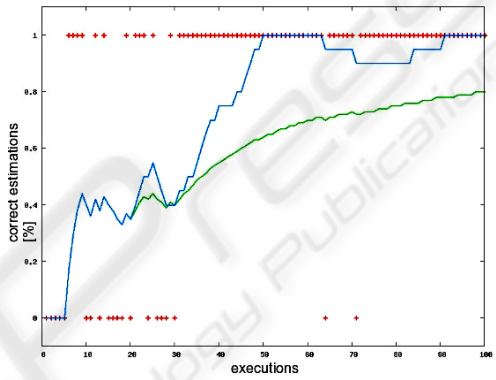

Figure 8: Overall (green) and averaged (blue) percentage of correct estimations of the rod's translation on the conveyor belt using the change function for 100 executions. A red dot with a value of 0 indicates that the robot could not locate the rod with the given change-function, but had to perform a search instead. A value of 1 indicates that the rod was found without the need for a search motion.

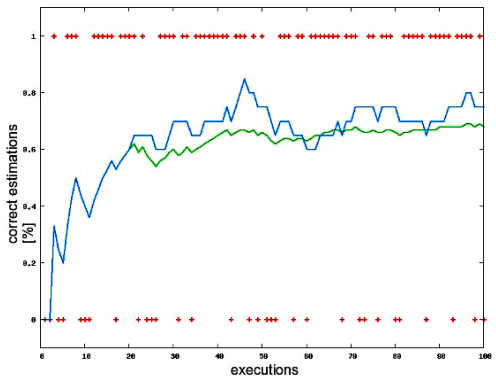

Figure 9: Overall (green) and averaged (blue) percentage of correct estimations of the rod's rotation on the conveyor belt using the change function for 100 executions. A red dot with a value of 0 indicates that the first correction of the rotation did not align the rod perfectly and further corrections were necessary. A red dot with a value of 1 indicates that the rod was aligned correctly with only one motion.

and had an overall accuracy of 80%. Due to the fact that two sensors are necessary to measure the rotation, the SNR of this combined sensor is relatively high, so the correction could not be performed in one motion every time. In spite of this, the robot was still capable of performing a perfect correction in 75% of all cases.

6 CONCLUSIONS

The aim of this work is to enable a developer to easily employ external sensors for flexible robot programs. The focus of this study was to show that data tuples describing the connection between sensory data and positional variations can be acquired automatically by the robot independent of the task and without the need for intricate calculations by the developer. We have presented a method to determine this data online during multiple executions of the task. The intention was to keep the requirements and methods independent from the type of sensor and make them universally applicable so they can be easily incorporated into a robot program. Finally, we presented an experiment to validate our research. We showed that it is possible to employ the proposed methods to successfully determine two change functions for a pick-and-place task.

In the next step our aim is to integrate time stamps into the data set *S*. Then we are able to deal with drifts in the sensor data due to heating processes of the sensor itself by discarding the older data tuples which do not reflect the current state of the system any more.

REFERENCES

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

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

Broyden, C.G., “A Class of Methods for Solving nonlinear Simultaneous Equations”, Mathematics of Computation, Vol. 19, No. 92. (Oct., 1965), pp. 577-593, Jstor.

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

Deiterding, J., Henrich, D. “Acquiring Change Models for Sensor-Based Robot Manipulation”, Int. Conf. o. Robotics and Automation 2008.

Dong, M., Tong, L., Sadler, B.M., “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.

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

Dudek, G., Zhang, C. “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.

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

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

Hutchinson, S.A., Cromwell, R.L. and Kak, A.C., “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.

Kriesten, D., Rößler, M., 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.

Leonhardt, U., Magee, J., “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.

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

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

Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling W.T. "Secant Method, False Position Method, and Ridders' Method." §9.2 in Numerical Recipes in FORTRAN: The Art of Scientific Computing, 2nd ed. Cambridge, England: Cambridge University Press, pp.347-352, 1992.

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

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

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