Proceedings of the RAAD 2008
17th International Workshop on Robotics in Alpe-Adria-Danube Region
September 15-17, 2008, Ancona, Italy

Workpiece Drift Recognition and Adaptation for Robot Manipulation Tasks

Jan DEITERDING and Dominik HENRICH[a]

[a] *Lehrstuhl für Angewandte Informatik III, Universität Bayreuth, Bayreuth, Germany*
E-mail: {jan.deiterding, dominik.henrich}@uni-bayreuth.de –
URL: http://www.ai3.uni-bayreuth.de

Abstract. The aim of this paper is to enable a developer to easily employ external sensors for flexible robot manipulation in industrial applications. We outline a general approach to supervise robot tasks for workpiece drifts using external sensors. We describe two methods to detect workpiece drift independent from the type of sensor. Based on this, we explain strategies how a robot can adapt to or compensate a detected drift. These methods can be integrated easily into robot programs without detailed knowledge provided by the developer.

Keywords. Industrial robots, robots & automation, intelligent manipulation

1 Introduction

Industrial robots are able to perform complex tasks without symptoms of fatigue and can maintain 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 workpieces are placed in exactly the same position in each execution cycle. Additionally, all parts need to have the same dimension, orientation, etc. Only by employing external sensors, such as vision or force/torque sensors, we can enable a robot to deal with imprecisions and variations occurring in the workpieces and the environment. One specific problem, which arises when performing tasks in an industrial environment, is a drift of a workpiece position between two consecutive executions.

In (Deiterding, 2007) 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 robot's reaction to it (see Table 1).

Table 1: Classification of changes that can occur between two executions of the same program (Deiterding, 2007).

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

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 may 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, e.g. the settings of machines and tools change over time.

In this paper, we address ways of dealing with this drifts. The problem is that the drift is unintended and not a desired property of the task. Because of that it is hard - if not impossible - to model the drift in order to adapt to it. The benefits of a suitable drift recognition and adaptation are as follows: First, the robot can present a warning to the supervisor of the task if a drift occurs. Additionally, a prediction can be made when the drift will have accumulated to an error. Second, an adaptation may be performed to adjust the robots motions to the changed environment. Finally, the robot may be capable of performing a corrective motion to counterbalance or reset the drift.

Since we must employ external sensors in order to detect a drift, we face the task of filtering the information of the sensor signal for the existence of drifts and transform the information into a physical description in Cartesian space, so the robot can deal with this drift.

The aim of this paper is to show how drift recognition and adaptation can be encapsulated into a sensor independent programming paradigm which enables a developer to design an adaptive robot task

capable of dealing with workpiece drifts with minimal knowledge and effort.

The rest of this paper is organized as follows: In Section 2, we give a short overview of related work in this area. In Section 3, we define workpiece drifts and outline the fundamental requirements for its recognition. In Section 4, we explain our programming paradigm, which can be used to easily create an adaptive robot program capable of dealing with drifts. In Section 5, we describe two approaches to generate a prediction based on the data provided by the sensor. Section 6 describes how a typical task involving a drift can be solved with our approach.

2 Related work

The task of monitoring a workpiece drift in industrial applications over multiple executions is mainly covered in engineering literature. (Chiang et al., 2001) deal with general fault detection in industrial applications and include drifts in their fault scheme. (Kesavan and Lee, 1997) have given a classification of faults in industrial systems. But there appears to be no standard terminology for theses processes, one possible example is the terminology given by (Raich and Cinar, 1996).

Workpiece drift occurs only in industrial tasks which are repeated multiple times. Nowadays, the use of external sensors in these tasks is still limited to applications where this is absolutely necessary. On the other hand, autonomous robots rarely execute the same task twice in an identical environment. Because of this, drift recognition and compensation strategies are usually programmed on a per-task basis and form a detailed solution for a specific drift. (Sharma et al., 1996) have presented an approach to optimize robot motions for given stochastic models of an assembly process, but focus on motions and do not specifically deal with workpiece drifts. (LaValle, 1996) focuses on robot motions as well, and takes external sensors as input signals into account, but does not make use of a drift model.

In summary, all of these articles are about finding a specific solution for a given problem. None of the articles use the knowledge gained in previous executions to create an adaptive drift model. Here, we are interested in outlining a general approach to drift recognition independent from the type of task and the sensors used for its supervision. Additionally, we are interested in showing how a robot can deal with a detected drift and how these strategies can be integrated into a programming environment without demanding detailed knowledge about the process from the developer.

3 Properties of drifts

In this section, we specify the basic properties of drifts. We focus on workpiece drifts, that is how a specific workpiece may change its location over multiple executions of the same task. We deal with workpiece changes only, such as position, weight, etc. We do not deal with drifts caused by the sensor itself due to temperature or lighting changes, etc.

3.1 Properties of workpiece drifts

The term *workpiece drift* describes a geometrical displacement d_t of a workpieces position p_0 between multiple executions t of the same robot task, that is

$$p_{t+1} = p_0 + d_t \tag{1}$$

The difference between a drift and a variation is that the drift is characterized by a preferred direction. Variations on the other hand fluctuate around a given position. The preferred direction may not be constant and can change over time. The extent of the drift is small compared to the size of the workpiece, so a drift is usually only recognizable after multiple executions.

This definition holds for one dimension in Cartesian space. Multiple drifts in different dimensions can be combined to model more complex drifts, but in this case we require that for all dimensions the drift in each dimension is statistically independent, that is

$$P(d_i | d_j) = P(d_i) \forall\, i \neq j \tag{2}$$

where $P(d_i)$ describes the probability of an occuring drift in the current execution.

3.2 Drift recognition

In order to detect a workpiece drift during a manipulation task, external sensors must be employed.

The change of the workpiece position between two executions is small and may not be detected by the sensor after only one consecutive execution. So the drift may be lower than the *signal-to-noise ratio* (SNR) of the sensor s

$$d_t < SNR(s) \tag{3}$$

In order to realize this drift recognition independent from the type of sensor, we use so-called *change functions* (Deiterding, 2008), which describe the relation between a change in a sensor-signal and the dislocation of a specific workpiece in Cartesian space. These functions can be preset, but may as well be computed by the robot during task execution. By transferring the sensor signal into a Cartesian description of the drift, we are capable of designing sensor independent methods to detect and predict a drift. To use a change function f, we must specify the default position p_0 and the corresponding sensor value s_0

$$d_i = f(s_i) - f(s_0) = f(s_i) - p_0 \tag{4}$$

From this point on, we will only deal with Cartesian descriptions of drifts. By measuring the

workpieces position during each execution, we build a time series $\hat{d}$ over n executions

$$\hat{d} = d_0, \ldots, d_n \quad (5)$$

where

$$d_0 = 0 \quad (6)$$

because no drift has occurred yet or the sensor has just been calibrated in the very beginning. Then, drift recognition is done by checking $\hat{d}$ after every execution and comparing the values against a threshold chosen depending on the SNR of the sensor. It is necessary to use a time series $\hat{d}$ because otherwise we cannot distinguish between a drift and a variation. We can only determine a drift by checking for a pattern in the workpieces locations. Otherwise every drift would be considered to be a variation.

Using $\hat{d}$, we can determine if a drift has occured. But, for successful adaptation to the drift, we need to make a prediction about the future motion of the workpiece. We will show how this can be achieved in Section 5.

4 Integration into the programming environment

In this section, we will show how workpiece drifts can be detected during execution of a robot task. Based on this we describe two ways of dealing with such a drift: Drift adaptation and drift correction. We explain how these ideas can be encapsulated into an easy-to-use interface in order to minimize the time required for the development of sensor-based robot programs.

We will only describe how drift recognition and adaptation can be set up for a drift occurring in one Cartesian dimension. The process is similar for multi-dimensional drifts.

During setup, the developer must determine which sensor shall be used to monitor a workpiece for drifts. In the next step, the default position d_0 of the workpiece in question is recorded. This includes the corresponding default sensor value s_0. Later on, all sensor values will be compared against this value. The last thing to do during setup is to specify a change function for the given workpiece and the sensor. Simple ways of doing this are described in (Deiterding, 2008).

When this is done, the developer must decide in which way the drift shall be modelled and set the corresponding parameters for the model. We describe two possible models usable for this task in Section 5.

In the robot program itself, the developer has three options to deal with a drift: Supervision, adaptation and correction.

4.1 Drift supervision

The purpose of *drift supervision* is just to check if a workpiece is moving and inform the person monitoring the automation process when the workpiece is about to leave the workspace. In this case the developer must specify the range of the workspace $r_{workspace}$. The current drift is then extrapolated and an estimate will be formed how many more executions can be performed until the drift must be corrected.

4.2 Drift adaptation

Under certain circumstances, it may be useful to alter the motions of the robot to accommodate a detected drift. We call this process *drift adaptation.* In general, this will not be necessary because of the change function. If the sensor is used preparatory, the robot will know the current position of the workpiece straight away and can act accordingly. If the sensor is used concurrently, the robot can modify all subsequent motions as soon as the workpiece is localized. However, there are tasks where drift adaptation is useful if concurrent sensors are used. Usually this is the case, if the Cartesian range of the change function is smaller than the allowed range of the drift. An example for a task like this is given in Section 6. To perform an adaptation in the i-th execution, the default position is set to the current position of the workpiece

$$d_0 = d_i \quad (7)$$

so the robot will use the adapted position in all subsequent executions.

After we have performed an adaptation, we can still use the drift data stored in $\hat{d}$ for drift supervision, but must accommodate the fact that drifts are described in relation to the old default position. A simple way to maintain a correct description of all drifts up to now, is to subtract all entries in $\hat{d}$ by the current total drift d_i

$$\forall d_j \in \hat{d} : d_j := d_j - d_i \quad (8)$$

4.3 Drift correction

Another option when dealing with a drift is to try to correct the environment or the workpiece somehow by performing an unscheduled task, which is not performed during the normal execution of the task. We call this *drift correction*. This should happen only when the workpiece is about to leave the workspace. This corrective motion can be trivial, e.g. the workpiece is grasped and moved back to d_0, but can be quite complex as well, e.g. an adjustment of a machine involved in the task. Because of this no general corrective motion can be described. This motion must be designed by the developer instead. When this motion is performed, we must reset $\hat{d}$ because we have altered the environment, so our current time series no longer represents the actual state of the environment.

4.4 Integration into the programming environment

In the cases of drift adaptation and drift correction, the decision when to perform an adaptation or correction will be triggered by thresholds $d_{adaption}$ and $d_{correction}$ which are set by the developer.

All three options can be integrated into one general algorithm `update_Drift_Position()` as described above. Now, the developer only has to set the specific parameters $r_{workspace}$, $d_{adaptation}$, $d_{correction}$ and define a corrective motion $m_{correction}$. In the robot program itself, this algorithm must be called, as soon as the workpiece can be measured by the sensor. This may be before or after the robot has moved to the default position, depending on wether the sensor is used preparatory or concurrently. The pseudocode of `update_Drift_Position()` then looks like this:

```
function update_Drift_Position()
{
   d_mom = predict_Drift();

   // drift supervision
   n_executions = calc_Remaining_executions();
   notify_supervisor(n_executions);

   // drift adaptation
   if(d_mom > d_adaptation)
   {
      d_0 = apply_drift(d_0, d_mom);
      modify_drift_data();
      update_prediction_model();
   }

   // drift correction
   if(d_mom > d_correction)
   {
      correction_motion();
      reset_drift_data();
   }
}
```

5 Drift prediction

In this section, we describe two approaches to automatically build a model which can be used to detect the current and predict the future drift. First, we describe how Kalman filter can be used for this problem and a more general approach involving ARIMA models without the need for a movement model afterwards.

5.1 Drift prediction using Kalman filters

Kalman filters are mainly used for object tracking in mobile robots (Brookner, 1998). The task here is similar, so we will use a similar approach.

Because we are only dealing with Cartesian drifts, we can construct a Kalman filter, which is capable of predicting the future drift independent from the type of sensor used for supervision. Here, we will use the nomenclature of (Kalman, 1960):

The input-vector $\hat{x}_t$ is made up from the current position of the workpiece d_t in regard to its default position d_0 as well as the current drift between two executions, which can be regarded as the current speed of the workpiece v_{d_t}. So

$$\hat{x}_t = \begin{pmatrix} d_t \\ v_{d_t} \end{pmatrix} \tag{9}$$

We assume that the current drift is statistically independent from the current position. Then, the covariance matrix Σ_t is defined as

$$\Sigma_t = \begin{pmatrix} \sigma_{d_t d_t} & 0 \\ 0 & \sigma_{v_{d_t} v_{d_t}} \end{pmatrix} \tag{10}$$

where $\sigma_{d_t d_t}$ and $\sigma_{v_{d_t} v_{d_t}}$ describe the accuracy of the estimates.

The transition matrix A_t describes the alteration from $\hat{x}_t$ to $\hat{x}_{t+1}$ and can be computed as follows: We assume that the transition is determined by the current speed v_{d_t}, the acceleration a and the time Δt elapsed between the two measurements:

$$\begin{pmatrix} d_t \\ v_{d_t} \end{pmatrix} = \begin{pmatrix} d_{t-1} + v_{d_{t-1}} \Delta t + \frac{1}{2} a \Delta t^2 \\ v_{d_{t-1}} + a \Delta t \end{pmatrix} \tag{11}$$

We can rewrite this to

$$\hat{x}_t = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \hat{x}_{t-1} + \begin{pmatrix} \frac{1}{2} a \Delta t^2 \\ a \Delta t \end{pmatrix} \tag{12}$$

Then

$$A_t = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \tag{13}$$

and

$$\epsilon_t = \begin{pmatrix} \frac{1}{2} a \Delta t^2 \\ a \Delta t \end{pmatrix} \tag{14}$$

Using ϵ_t we can compute the gain matrix R_t as

$$R_t = \frac{d\,\epsilon_t}{da} \sigma_a^2 \frac{d\,\epsilon_t^T}{da} = \begin{pmatrix} \frac{1}{4} \Delta t^4 \sigma_a^2 & \frac{1}{2} \Delta t^3 \sigma_a^2 \\ \frac{1}{2} \Delta t^3 \sigma_a^2 & \Delta t^2 \sigma_a^2 \end{pmatrix} \tag{15}$$

where σ_a^2 is the variance of a. Here, Δt can be set to 1 as the time between two consecutive executions is constant, as long as the drift only occurs while executing the task.

Finally, we need a vector z_t which describes how we perceive a drift from one position to the next. This is achieved by using the external sensor. But, if

we would use the sensor signal directly, the developer would have to specify this vector for every type of sensor. By employing a change function which maps the sensor signal to Cartesian space, we can compute a general form of z_t, which is applicable for all types of sensors. In this case the perceived position is exactly the current position blurred by the SNR δ_t of the sensor, so

$$z_t = C_t \hat{x}_t + \delta_t = \begin{pmatrix} 1 & 0 \end{pmatrix} \hat{x}_t + \delta_t \quad (16)$$

These are all the vectors and matrices necessary for a Kalman filter to compute a prediction of the next drift. The exact calculations are described in (Kalman, 1960) and will not be repeated here.

This Kalman filter is parametrized by Σ_t, σ_a^2 and δ_t, which must be set by the developer. It is possible to set basic values for these, which work adequately well, but for optimization purposes, these should be tuned by the developer.

Using this Kalman filter we get an estimation of the current drift of the workpiece. This estimate describes how the workpiece will change its current position in the very next execution.

There are two drawbacks when this Kalman filter is used for drift prediction: First, we can only predict the very next drift, but no drifts in the further future. Second, if the drift between two executions is lower than the noise of the sensor, we cannot predict any drift at all, because the Kalman filter only uses the most recent values to update its internal state.

5.2 Drift prediction using ARIMA models

An alternative to Kalman filters is to use an *auto-regressive integrated moving-average* (ARIMA) model. These models make use of bigger parts of $\hat{d}$ allowing for predictions of more than the very next execution. An ARIMA model is actually a combination of three models, which can be described by one parameter each: An auto-regressive (p_{ar}), an integrated (p_i) and a moving-average (p_{ma}) model.

First, $\hat{d}$ is differentiated p_i times, resulting in a time series $\hat{d}'$. The calculation for the next prediction is then

$$\hat{d}_{n+1} = \epsilon_t + \sum_{j=1}^{p_{ar}} p_{ar_j} \hat{d}'_{n-j} + \sum_{k=1}^{p_{ma}} p_{ma_k} \hat{d}'_{n-k} \quad (17)$$

Note, that - unlike the Kalman filter - no assumption about the movement of the drift (that is its velocity and acceleration) are made.

A good choice of the parameters p_{ar}, p_i and p_{ma} is usually difficult, but in this case, we can make some basic assumptions which will help us choosing suitable parameters. The weighting parameters p_{ar_j} and p_{ma_k} can be fitted automatically for given methods (Box et al., 1994).

The auto-regressive parameter p_{ar} must be zero, because the current drift is independent from the prediction of the last drift; otherwise the act of making a prediction would already alter the environment.

We need to differentiate $\hat{d}$ exactly once, so p_i can be set to one. This is because we store the total drift from d_0 up to the current execution d_i in $\hat{d}$. To predict the next drift(s), we are interested in the alteration from one execution to the next. So, we must differentiate our time series exactly once.

The moving-average parameter p_{ma} describes how many drifts from the immediate past are used to approximate the current drift. This parameter can be chosen by the developer. The choice of p_{ma} is dependant from the SNR of the sensor and the stability of the drift. If p_{ma} is chosen too low, the noise of the sensor will corrupt the prediction of the current drift. If the drift changes its preferred direction relatively often, a high value for p_{ma} will take drifts into account which are no longer adequate .

Note that, if we set

$$p_{ma} = s(\hat{d}) \quad (18)$$

and

$$p_{ma_i} = \frac{1}{s(\hat{d})} \forall i \quad (19)$$

where $s(\hat{d})$ gives us the size of $\hat{d}$, the ARIMA model is a simple linear extrapolation using the whole time series $\hat{d}$.

6 Experiments

In this section we will show how all three methods of dealing with drifts from Section 4 can be integrated easily into a robot task using Kalman filters and ARIMA models.

6.1 Experimental setup

Consider the following task: A robot places a workpiece on the entry side of a conveyor belt. The workpiece is then processed by some kind of machine. When the workpiece leaves the machine, the robot picks it up again and performs some task with it, without releasing it. Afterwards this workpiece is placed onto the conveyor belt once more (see Figure 2). In this experiment, we will use a round disk with a size of 150 mm in diameter.

The robot program for this task looks like this:

```
1  repeat
2  {
3   MOVE p_dropoff
4   RELEASE
5   // wait for machine to finish processing
6   WAIT
7   MOVE TRANS_y(l_belt):p_dropoff
8   GRASP
9  // perform some other task with workpiece
10 }
```

We place the center of the robots coordinate system into its base and the center of the conveyor

belts coordinate system into the position where the robot places the disk (see Figure 3). The problem with this implementation is, that if the x-axes of both coordinate systems are not exactly parallel, a drift d_y along the conveyor belts y-axis will occur. In theory the resulting drift is calculated as

$$d_y = \tan(\alpha) \bullet l_{belt} \tag{20}$$

where α is the angle by which the coordinate systems are rotated in relation to each other and l_{belt} is the distance between the dropoff and the pickup position.

To supervise this drift, we use two different sensors, one force/torque sensor (*FTS*) and one distance sensor (*DS*) which are positioned in the wrist of the robot and at the pickup position, respectively (see Figure 2). We will try to adapt to this drift by measuring the torque of the disk around the x-axis and by measuring the distance of the disk when it is to be picked up. So, the FTS is used in a concurrent way, while the DS is used preparatory.

To allow for a flexible execution of the given program, we add a line calling *update_Drift_Position()* directly before moving the robot to the pickup position in line 7.

6.2 Parametrization during setup

During setup, we record the position and the corresponding sensor values of the disk when it leaves the machine for the very first time. This defines our default position d_0 with the sensor values fts_0 and ds_0.We use the approaches described in (Deiterding, 2008) to determine appropriate change functions f_{fts} and f_{ds} for both sensors and the SNR of the sensors SNR_{fts} and SNR_{ds}. These are the parameters of the Kalman filters. For ARIMA prediction, we set

$$p_{ma} = \frac{1}{10} \bullet s(\hat{d}) \tag{21}$$

assuming that the drift will remain constant.

To parametrize the recognition and adaptation process, we measure the width of the conveyor belt w_{belt} and divide it by two, because the ideal position of the disk will be in the middle of the belt. This value will be used for drift supervision.

To adapt to the drift, we set $d_{adaptation}$ to the half of the disks width. So when the drift exceeds this value, the robot will modify the pickup position accordingly.

Finally, when we get to close to the edge of the conveyor belt, the robot shall pickup the disk and perform a corrective motion to position it in the middle of the belt once more. So, we set

$$d_{correction} = \frac{2}{3} \bullet \frac{w_{belt}}{2} \tag{22}$$

and define a corresponding correction motion $m_{correction}$.

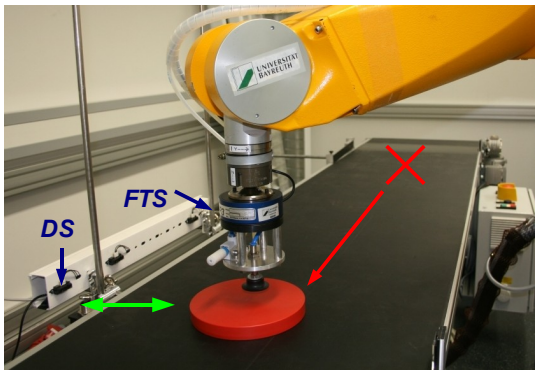


Figure 2: Experimental setup of the task with sensors DS and FTS described in Section 6.1

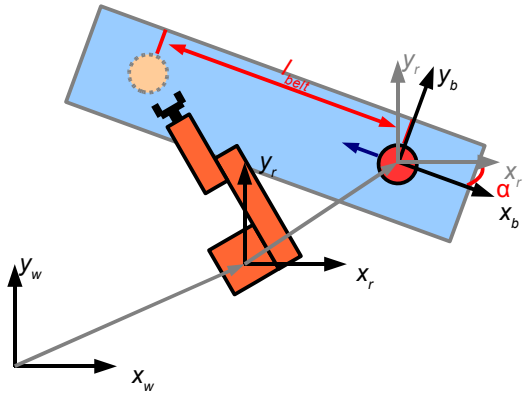


Figure 3: Coordinate systems of the robot and the conveyor belt in relation to the world coordinate system for the task described in Section 6.1

6.3 Drift recognition

We have executed the task 20 times and recorded the sensor values during each execution (see Figure 4), describing the total drift of the disk. The resolution of the DS is relatively low, so we can only measure distances in sizes of 1 cm.

In Figures 5 *a* and *b* we have plotted the predicted drift for the next execution, the actual drift as measured by the sensor and the accuracy of the prediction for each combination of sensor and prediction method. We can see that after 20 executions, the combination of a Kalman filter and a FTS provides the most accurate predictions. But, in general the Kalman filter tends to oscillate while ARIMA models take longer to adapt to changes in the drift.

If a distance sensor is used to monitor the task, the predictions of the Kalman filter are worse than those of the ARIMA method. This is because the Kalman filter starts to oscillate when measurable drifts occur rarely. The ARIMA method on the other hand adapts relatively fast to this type of drifts.

If a force/torque sensor is used, it is the other way round. Here, the sensor is more accurate, recognizing drifts in every execution. Because of this, the Kalman filter adapts faster, but the difference to the ARIMA prediction is less significant.

We have summarized the results in Table 2 showing the mean and standard deviation of the drift as measured by the sensors and as predicted by a

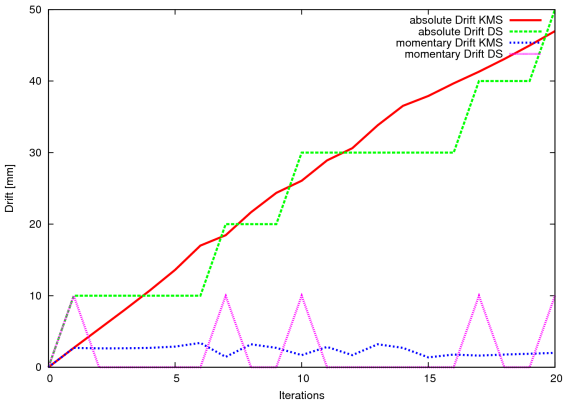


Figure 4: Recorded drift (absolute and current) during 20 executions of the task.

Table 2: Mean and standard deviation of occuring and predicted drift for both types of sensor

		Mean	Standard deviation
Real drift	*KMS*	2.35	0.65
	DS	2.5	4.44
Kalman prediction	*KMS*	2.35	3.23
	DS	1.97	17.48
ARIMA prediction	*KMS*	2.32	1.52
	DS	0.05	10.17

Kalman filter and an ARIMA interpolation respectively.

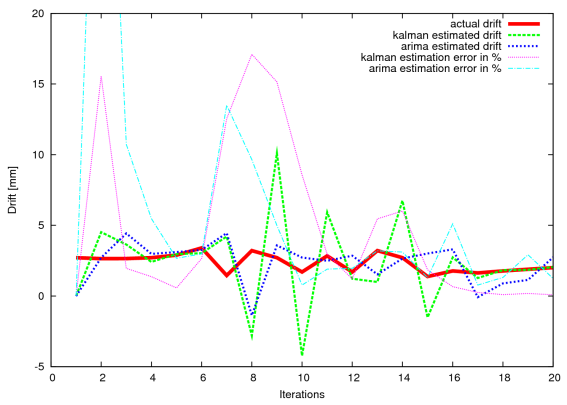
In general we can say that the SNR of the sensor is more important than the method chosen for the prediction.

6.4 Drift adaptation and correction

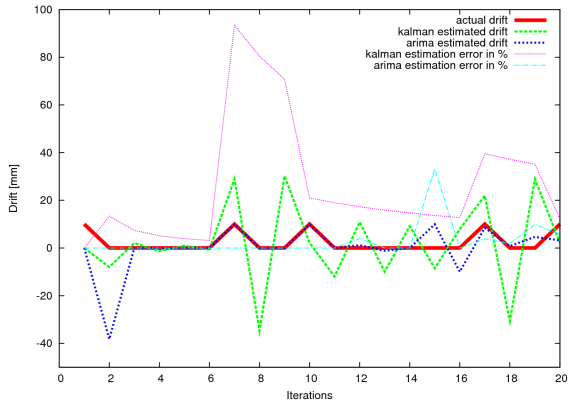
After setting up the prediction models and evaluating the drift, we have set the thresholds $d_{adaptation}$ and $d_{correction}$ to 20 mm and 50 mm, respectively. Now, we have executed the task 100 times, measured the real drift using a Kalman filter and the force/torque sensor and logged all adaptations and corrections (see Figure 6). We can see that the robot is now capable to keep the current drift below the adaptation threshold by modifying the pickup position according to the prediction approximately every 10 executions. To prevent the disk from falling of the edge of the conveyor belt, the robot automatically re-centers it in the middle of the belt approximately every 20 iterations.

In summary, although there is a significant drift inherent in this task, in theory we can execute this task infinitely. The robot automatically detects the drift and adapts its motion to the shifting position of the disk, as well as performing a corrective motion from time to time to reset the disk to the center of the conveyor belt.

a)



b)



Figures 5:Actual and predicted drift with estimation error for Kalman and Arima models using a (a) FTS sensor, (b) DS sensor.

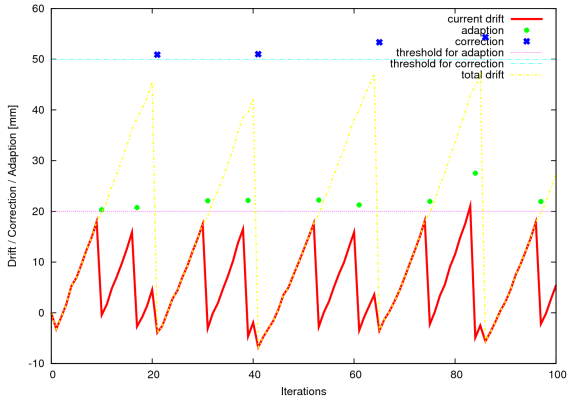


Figure 6: Current (red) and absolute (yellow) drift for 100 executions of the task. When the prediction of the drift exceeds the set thresholds for adaptation (pink) or correction (light blue), the robot either modifies the pickup position (green dots) or re-centers the disk (blue dots).

7 Conclusion

Our goal is to enable a programmer to easily employ external sensors for flexible robot programs. The focus of this work is to show that workpieces can be monitored automatically for drifts which occur due to an imprecise setup of the workspace or abrasion without the need for intricate calculations by the

developer. We have defined the term workpiece drift and have described two methods to detect and predict a drift for a specific workpiece and given sensor by examining a time series describing the workpiece position over multiple executions. The presented requirements and methods are independent from the type of sensor. We have shown that these methods can be integrated into a programming environment, so that the developer only has to specify basic parameters and modify an existing robot program by a minimal amount. Finally, we have presented an experiment to validate our findings. We have shown that it is possible to employ the proposed methods to successfully detect and adapt to a workpiece drift during an automation task.

Further work needs to be done in finding ways to automatically determine reasonable parameters for the Kalman filter and the ARIMA model. Simple estimates work well, but by tuning these parameters the prediction process might be optimized.

If an ARIMA model is used, the choice of p_{ma} can be optimized along the following idea: We increment p_{ma} until a significant change in the current drift is encountered. This can be done using the methods described in (Schlechter, 2006). If this happens, p_{ma} is set back to a default size. So the ARIMA model will only use values of $\hat{d}$ which are significant for the next prediction.

8 References

Box, George, Jenkins, Gwilym M., and Reinsel, Gregory C.. Time Series Analysis: *Forecasting and Control*, third edition. Prentice-Hall, 1994.

Brookner, Eli. 1998. Tracking and Kalman Filtering made easy, *Wiley Interscience, ISBN 0-471-18407-1*

Chiang, L.H. Russell, E.H. And Bratz, R. D., 2001, *Fault Detection and Diagnosis in Industrial Systems*, Springer Verlag, ISBN 1-85233-327-8

Deiterding, Jan and Henrich Dominik. 2007. Automatic optimization of adaptive robot manipulators, *2007 IEEE International Conference on Intelligent Robots and Systems,* San Diego / USA

Deiterding, Jan and Henrich Dominik. 2008. Acquiring change models for sensor-based robot manipulation, *2008 IEEE International Conference on Robotics and Automation,* Pasadena / USA

Kalman, Rudolph Emil. 1960. A New Approach to Linear Filtering and Prediction Problems, *Transactions of the ASME--Journal of Basic Engineering*, Vol. 82-D, pp.35-45

Kesavan, P. and Lee, J. H. 1997, *Diagnostic tools for multivariable model-based control systems*. Ind. Eng. Chem. Res. 36:2725-2738

LaValle, Steven , 1996, Robot Motion Planning: A Game Theoretic Foundation, Algorithmica ISSN 0178-4617 , 2000, vol. 26, pp. 430-465

Raich, A.C. and Cinar, A. 1996, *Statistical process monitoring and disturbance diagnosis in multivariate continous processes*, AIChE J. 42:995-1009

Schlechter, Antoine and Henrich, Dominik. 2006. Discontinuity detection for force-based manipulation, *2006 IEEE International Conference on Robotics and Automation*

Sharma, Rajeev, LaValle, Steven and Hutchinson, Seth, 1996, *Optimizing Robot Motion Strategies for Assembly with Stochastic Models of the Assembly Process*, IEEE Transactions on Robotics and Automation, 12:160-174