Improving Robustness of Task Execution Against External Faults Using Simulation Based Approach

Anastassia Küstenmacher[1] and **Paul G. Plöger**[1] and **Gerhard Lakemeyer**[2]
[1]Bonn-Rhein-Sieg University of Applied Science, Sankt Augustin, Germany
e-mail: {anastassia.kuestenmacher,paul.ploeger}@h-brs.de
[2]RWTH Aachen University, Aachen, Germany e-mail: gerhard@ksg.rwth-aachen.de

Abstract

Robots interacting in complex and cluttered environments may face unexpected situations referred to as external faults which prohibit the successful completion of their tasks. In order to function in a more robust manner, robots need to recognise these faults and learn how to deal with them in the future. We present a simulation-based technique to avoid external faults occurring during execusion releasing actions of a robot. Our technique utilizes simulation to generate a set of labeled examples which are used by a histogram algorithm to compute a safe region. A safe region consists of a set of releasing states of an object that correspond to successful performances of the action. This technique also suggests a general solution to avoid the occurrence of external faults for not only the current, observable object but also for any other object of the same shape but different size.

1 Introduction

Service robots have to perform common household tasks effectively in dynamic and partially known environments. To be truly robust, such robots must to be able to accomplish their actions in a satisfactory manner without supervision. Their performance can be degraded either by internal hardware and software faults or by unexpected *external faults*. Such external faults can occur instantaneously, are sporadic in nature and are largely unforeseeable at the design stage. As it is impossible to fully supply the robots with knowledge of all possible unexpected situations a priori, the robots must determine the causes of *external faults* on their own to avoid their occurrence in the future.

Figure 1 illustrates two typical external fault scenarios. In scenario 1, a service robot Care-O-Bot III [1], has to place an object (the green crisp can) on a table. If the robot were to release the can in its current position it would likely fall over. In scenario 2, the Care-O-Bot III successfully drops an object into a box. However, the robot may fail if the box were already full of other objects. Our following presents a technique that increases

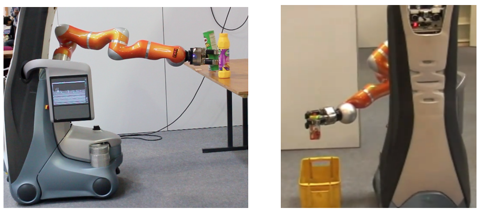
(a) Scenario 1 (b) Scenario 2

Figure 1: Examples of external faults

the ability of service robots to overcome *external faults*. This technique focuses on the actions of manipulator involved in the release of an object. It suggests the safest state space for the release of not only a particular object, but also for any scaled object of the same shape. Releasing the object in a state from this safest state space avoids the occurrence of external faults. To do so we collect three training sets from experiments. One set of samples consists of experiments with a given ob-

[1]http://www.care-o-bot-research.org

ject. The next set contains experiments with an object of the same shape but scaled down to the smallest possible size which the robot is able to grasp. The third set contains experiments with the largest graspable object of the same shape. The algorithm's task is to maintain a good estimation, namely state space, of the positive outcomes (successful execution of the releasing action) within each of the three training sets and to find the common intersection of these three state spaces. The algorithm assumes that by monitoring the post-conditions of an executed action external faults are detected. We also assume availability of a simulation that shows an example of the expected behaviour of the manipulated object for the case of successful completion of the executed action.

2 Related Work

Methods for detection, identification and handling of faults are frequently discussed in robotics literature. In his survey on execution monitoring Pettersson [7] groups the existing approaches into three classes: analytical, data-driven and knowledge-based. In robotics the terms fault diagnosis and fault detection and isolation are used as synonyms for execution monitoring [7]. Fault diagnosis techniques deal mostly with internal faults. These methods focus on model-based monitors, where the models of possible faults are available a priori. Although it is impossible to describe all interaction faults in dynamic environments, the model-based techniques allow the detection of unexpected situations by comparing the nominal/desired behaviour of a robot with its observations. Pettersson *et al.* and Mendoza *et al.* [6] [5] present work, in which external faults are detected from the robot's behaviour, rather than from a predictive model. We present an alternative technique which utilizes simulation in order to find a general solution for external faults during releasing actions. The inspiration for using simulation to handle external faults comes from the PhD work of Zickler [8] PhD in which a physics engine is used as a black-box to compute a robot motion planning in a complex environment. In another study Jorgensen *et al.* [4] use a simulator to address the problem of stable placement of objects onto both plane and complex surfaces. Similar to our approach, the authors use a simulation to produce a set of labeled samples. To compute an optimal drop pose, they suggest two methods. In the first method they fit the largest enclosed ellipsoid to the positive samples. In the second they apply kernel density estimation to estimate probability of each pose. The optimal pose corresponds to the pose with the largest success probability. The methods of Jorgensen *et al.* assume only two relevant parameters. However, for our scenarios (1,2) we need more than two parameters because of dimensional space. Jiang *et al.* [2] propose a learning approach for placing various objects in different places. The authors use a simulator to generate training and test datasets. The created data sets are then labeled manually and used as inputs for supervised learning algorithms to predict the optimal areas for a stable placement. Johnston and Williams [3] present the architecture *Comirit* for common sense reasoning, in which they combine 3D simulation with formal logic. Their framework generalizes the method of analytic tableaux to allow both logical terms and simulation objects within a single search structure. The authors represent each object as an annotated, connected set of vertices, where annotations are used to describe the local physical properties of the object. Such representation requires complex modelling for each single object.

3 Approach

This work is the updated version of our previous *simulation based approach using example simulation* (SBAES)[1]. The previous technique enabled the robot to suggest the safest releasing state of an object for successful performance of its action or to predict the behaviour of an object for a given releasing state.

Previous work

The SBAES approach was presented as a three step scheme, that requires two inputs: 1) an example simulation that showed the desired behaviour of the object for a given action and, 2) a definition of the planning operator of the action that results in a detected external fault. In its first step, the scheme used the example simulation to find a logical description of the expected behaviour of the released object. The description consists of two logical sentences, namely states of the object at the start and at the end of the simulation. In its second step, the approach found the limits of parameters of the object using this description. Each parameter corresponds to a physical property of the manipulated object and the values of these parameters define the releasing state of this object. The approach utilized them to create different examples of the releasing state of the object. With the help of the description vocabulary and the description of the sample behaviour the approach labelled these examples as either desired or undesired. In step three, these labelled examples were exploited by a learning algorithm which we referred as *N-Bins*, to suggest a safest releasing state. The results of the learning algorithm were used by the SBAES to modify the releasing action of the robot.

The main objective of the current approach is not only to find a safe releasing state of the current manipulated object but also to generate a solution for other objects with the same geometrical shape but variations in scale. This problem can not be solved using our previous approach (as described above), because *N-Bins* algorithm only suggests one optimal releasing state instead of a space of safe states. The other disadvantage of the SBAES approach is that it assumes the parameters are completely independent of each other. This assumption leads to loss connection between the parameters whereas the closer the released object is to the edge of the table, the lower the value of the releasing height should be.

Approach

This approach mainly focuses on the improvement of Step 2 of the SBAES technique, finding safe space for parameters' values to avoid the occurrence of faults. Similar to SBAES the current approach assumes the settings of plan based robotic systems, in which a robot is able to detect the occurrence of a fault at the planning level by monitoring the postconditions of an executed action. It also assumes that the causes of the external faults are limited to natural physical phenomena (there is no external agent involved in the occurrence of faults).

Additionally our technique expects three inputs: 1) a model of the domestic environment, 2) an example simulation of the actual action (the technique only needs the emulation of a given action, the release of any object over any surface) and 3) postconditions to be checked for successful completion of the action. Before applying our algorithm to compute safe regions, we need to perform necessary preparations. First, the given simulation of a domestic environment has to be updated by the later changes in the robot's current environment. For instance, the model of the table in the simulation has to be changed if some other objects are later placed on the table. The next preparation is to get the limits of parameters to generate the training set. In this work we consider six parameters which correspond to location and orientation of the object. To define the initial limits we can use the SBAES approach. The main disadvantage is its strong dependence on a given example simulation which is modelled for a particular released object and a specific surface. But originally a service robot is equipped with knowledge about the sizes of the objects in its environment, own position, dextrous workspace of the manipulator, desirable final state of the object etc. This knowledge can be applied to estimate the limits of each parameter. Later during the simulation these limits are used to randomly select values of the parameters to generate training examples. Each simulated instance is labeled based on given postconditions. We generate three sets of examples for three objects of different size. One set of samples consists of experiments with the current object. The next set includes those experiments with an object of the same shape but scaled down to the smallest possible graspable size. Similarly, the third set contains experiments with the largest graspable object of the same shape. To ensure the ability of the manipulator to grasp of an object we use a bounding box as rough estimation of its size. Below we introduce a notation and illustrate details of the technique.

Assume that for a given action we generated m training instances. These instances are stored as a $m \times (n+2)$ matrix $AllSamples$. $AllSamples$=$\{(State^i$, $Label^i$, $ExpSize^i)$, $i = 1, 2, \ldots, m\}$, where $State^i \in \mathbb{R}^n$ are input values of the parameters of an object (i.e. its releasing state) and n is the total number of parameters (in our example they correspond to $x, y, z, \rho, \theta, \phi$, where ρ, θ, ϕ are respectively $roll, pitch, yaw$). $Label^i \in \{-1, 1\}$ are *negative* and *positive* outputs of experiments. $ExpSize^i \in \{small, midium, large\}$ are different categories of experiments which correspond to the experiments with objects of the actual, smallest and largest graspable sizes. $threshold$ is a scalar which shows the desired probability of a positive outcome of the experiment for the safe state space of the parameters. Algorithm 1 shows the pseudocode for finding the common safe state space for given $AllSamples$ and $threshold$. The FIND-SAFE-STATE-SPACE algorithm starts with computing the safe regions SR_{size} for the training set of each experiment category $size \in \{small, middle, large\}$. Then in line '9' it calculates the intersection of these safe regions. This intersection consists of the limits of the parameters where the current action can be performed

Algorithm 1 FIND-SAFE-STATE-SPACE

```
Input: $AllSamples$, $threshold$
Output: $SR$
1: for all $size \in \{small, middle, large\}$ do
2:     $AllSamples_{size}$ ← Select instances according to experiment category $size$
3:     ($PosSamples_{size}$,$NegSamples_{size}$) ← Split instances according to their labels
4:     $Weights$ ← WEIGHT-INDEXES ($PosSamples_{size}$, $NegSamples_{size}$)
5:     $HyperGrid$ ← Partition the space of $AllSamples_{size}$
6:     $F_{size}$,$\{HR_{size}\}$←FIND-HIST($Weights$, $PosSamples_{size}$,$HyperGrid$,$threshold$)
7:     $SR_{size}$ ← Extract $HR$: $F > threshold$
8: end for
9: $SR$=$SR_{small} \bigcap SR_{middle} \bigcap SR_{large}$
```

safely.

To find SR_{size} the FIND-SAFE-STATE-SPACE algorithm splits the corresponding training set $AllSamples_{size}$ into two matrices $PosSamples_{size}$ and $NegSamples_{size}$ based on the labels of the samples, where $PosSamples_{size}$ contains only the positive instances and $NegSamples_{size}$ contains only the negative instances.

These matrices are used by the WEIGHT-INDEXES algorithm to calculate the importance of each parameter in the behaviour of the objects. This algorithm is based on the procedure described in [1], p. 36. The main difference is that WEIGHT-INDEXES uses the Kolmogorov-Smirnov test (KS-test) to exclude parameters which are not important for the outcome of an experiment. (line '3' in Algorithm 2). The remaining weight indexes are calculated using the measures of the first four statistical moments (i.e. mean μ_p, variance σ_p, skewness $skew_p$ and kurtosis $kurt_p$) of the distributions of the values of each parameter. The differences ($\Delta\mu_p$, $\Delta\sigma_p$, $\Delta skew_p$ and $\Delta kurt_p$) between statistic moments of corresponding columns of $PosSamples$ and $NegSamples$ are used in calculating the so-called importance IMP_p of each parameter. IMP_p is further utilized for computing the weight of the parameter (see line '2' Algorithm 2). For any parameter, higher value of W_p shows that the final state of the object in the simulation is more sensitive to the value of that parameter.

We associate the safe regions SR_{size} of each experiment category $AllSamples_{size}$ with the intervals on the parameter space $(x, y, z, \rho, \theta, \phi)$, where the amount of positively labeled instances reaches the given $threshold$. To find such regions we decompose the given parameter space into hyper-rectangles and count the number of positive instances that fall into each of the hyperectangles. A hyper-rectangle is defined as the Cartesian product of partitions. A well-known way to build a set of hyper-rectangles with corresponding counts is a multidimensional histogram.

Algorithm 2 WEIGHT-INDEXES

```
Input: $PosSamples$, $NegSamples$
Output: $Weights$ ← A vector composed of $W_p$ for each parameter
1: for each parameter, $p \in 1, 2, \ldots, n$ do
2:     $W_p = \frac{IMP_p}{\sum_{i=1}^{n} IMP_i}$ where:
3:     $IMP_p \leftarrow 0$ if KS-test($PosSamples_p$)>0
4:     else $IMP_p$←$|\Delta\mu_p| + |\Delta\sigma_p| + |\Delta skew_p| + |\Delta kurt_p|$
5: end for
```

The FIND-HIST function shown in Algorithm 3 presents a pseudocode to compute multidimensional histogram where the counts of positive samples occurring in certain ranges of parameters's values are equal or higher than the $threshold$. The weight vector $Weights$, the matrix $PosSamples$ (positive instances of the one of the experiment's category e.g. $small$, $middle$ or $large$), the partition of the space of $AllSamples$ of the current category in hyperrectangles $HyperGrid$ and $threshold$ are the input arguments to the algorithm. In this algorithm the multidimensional histogram is specified as a set of hyper-rectangles $\{HR^j, j = 1, \ldots, h\}$, and multi-dimensional array F showing how many instances fit within a certain hyper-rectangle.

The FIND-HIST algorithm finds the index/-es of the parameters corresponding to the first largest weight values from the weight vector $Weights$ and returns the $MostImportantDims$ vector in line '3'. Then it selects the $CurrentData$ sub-matrix of the $PosSamples$ matrix consisting of the $MostImportantDims$ columns and the $CurrentDim$ sub-set of the $HyperGrid$. $CurrentDim$ consists of partitions (bins) which correspond to the $MostImportantDims$ dimensions. Using $CurrentDim$ and $CurrentData$ the Algorithm 3 in line '6' computes the counts F and the hyper-rectangles $\{HR\}$. If the termination criteria in line '2' is satisfied, the current F and $\{HR\}$ are the solutions. Otherwise the algorithm extends the $MostImportantDims$ vector with the index of next parameter according to the next highest weight, constructs a new $CurrentData$ matrix and $CurrentDim$ array and computes a new F and $\{HR\}$.

The output of the FIND-HIST algorithm (F, $\{HR\}$) is used in FIND-SAFE-STATE-SPACE to extract the safe region SR_{size} for the current training set. By repeating the described procedure for the training set of every experiment, we get a collection of safe regions $\{SR_{small}$, SR_{middle}, $SR_{large}\}$. In general, the intersection of these safe regions SR=$SR_{small} \bigcap SR_{middle} \bigcap SR_{large}$ is the particular solution for computing limits of the parameters in which the current action can be performed safely with the desired probability for any graspable object of the given shape.

Algorithm 3 FIND-HIST

Input: $Weights, PosSamples, HyperGrid, threshold$
Output: F - counts on relevant dimensions in total parameter space, HR - Cartesian product of partitions

```
1: $dimF \leftarrow 1, maxCount \leftarrow 0$
2: **while** $maxCount \leqslant threshold$ **and** $dimF \leqslant$ number of non-zero entries of $Weights$ **do**
3:     $MostImportantDims \leftarrow$ Indexes of the first $dimF$-th elements with the largest weights from $Weights$ array
4:     $CurrentData \leftarrow PosSamples$ projected to $MostImportantDims$
5:     $CurrentDim \leftarrow HyperGrid$ projected to $MostImportantDims$
6:     $F$, $\{HR\} \leftarrow$ build histogram for $CurrentData, CurrentDim$
7:     $maxCount \leftarrow$ maximal value of $F$
8:     Increment $dimF$ by 1
9: **end while**
```

The main challenge in the construction of a histogram is the selection of $HyperGrid$, that is an optimal number of bins or intervals for hyper-rectangles. According to the FIND-HIST algorithm, it first computes a histogram for one dimension (one parameter). If the resulting histogram does not satisfy termination conditions, the algorithm gradually increases the number of dimensions until the desired histogram is found. Hence the number of dimensions vary from 1 to the total number of non-zero entries of the $Weights$ vector. On the other hand the probability distribution of a given training set based on a random sample is uniform. The histogram constructed from them should have in each bin/hyper-rectangle about the same number of elements in average. By experimentation we show that for a successful performance the histogram of the highest dimension, which is the number of all parameters, should have a minimum 50 instances in average. This value is used to compute the number of bins for each parameter.

4 Results

In this section we report the results of applying the described algorithm on three experiments in which different objects were released over other objects or over an open container. The releasing states of different objects are calculated using $47{\cdot}10^6$ instances for each experiment category[2].

Experiment 1 (release a die on a table):
The example simulation shows a die being dropped on a table. From its knowledge the robot extracts the specifications of these objects[3]. Using these specifications we create three training sets for the releasing action. Each instance of these sets is labelled using postconditions' constrains[4].

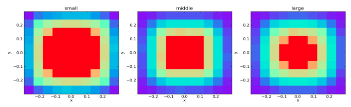

Figure 2: Safe regions of the die for experiment 1

Figure 2 shows the estimated safe regions (red areas) for each experiment category. WEIGHT-INDEX algorithm estimates the following importance order of the parameters: $x, y, z, pitch, yaw, roll$, where the weight of $roll$=0 because according to KS-test $roll$ does not have an impact on the outcome of the experiment ($roll, pitch, yaw$ correspond to ρ, θ, ϕ). The termination conditions are satisfied for the two

[2]With the help of parallelization on 48-core PC (4 $\times$ AMD Opteron^{TM}6174 12-core 2.2 GHz) the average time per instance is less then 0.8ms

[3]Table's dimensions: 0.47m(h)$\times$0.6m(l)$\times$0.6m(w), die's dimensions: small 0.005m$\times$0.005m$\times$0.005m, middle 0.01m$\times$0.01m$\times$0.01m, large 0.015m$\times$0.015m$\times$0.015m

[4]The experiment performed successfully if the die lies on the table: $Xdie_{final}$∈[-0.3,0.3], $Ydie_{final}$∈[-0.3,0.3] and $Zdie_{final} \approx$ 0.74

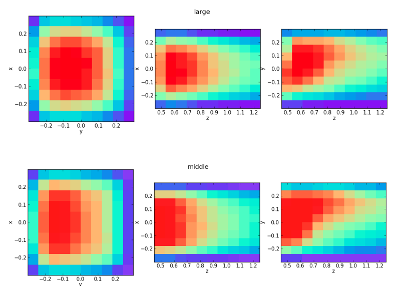

Figure 3: Safe regions of the die for experiment 2

dimensional histogram built for x and y parameters. The intersection of these safe regions is the safe region for releasing the die of the largest size.

Experiment 2 (release a die on an inclined table):
The experiment set up is similar to experiment 1, the only difference is the top of the table tilted at 45 degree. The safe region for the *small* die consists of two (x, z) parameters, but the safe regions for the *middle* and *large* objects include three dimensions (x, y, z), see Figure 3.

Experiment 3 (release a pencil into a box):
The example simulation shows a pencil being placed into an empty container. If the initial inputs for generating the training set are spread too far like in the previous experiments, where $\rho, \theta, \phi \in [-\pi, \pi]$, the algorithm requires more instances to cover all possible behaviours of the object. Hence, it may be necessary to refine them using additional techniques. For example by knowing the object's symmetry groups we can utilize the Euclidian transformation to reduce the limits of ρ, θ, ϕ to $[0, \frac{\pi}{2}]$ interval. Taking into consideration the information about the final orientation of the object (it should stand in the container) we can restrict ρ, θ even more. With the new parameters the algorithm is able to find six dimension safe space.

5 Conclusions and Future Work

The main contribution of this work is to enable a robot using an example of its action to learn how to execute this action in the safest way. We also show that a robot estimates safe performance of an action not only for the given object but also for any graspable object of the same shape in general. There still remain several issues for improving the generalization process, e.g. extending the given algorithm to different objects of similar shape categories. It is obvious that objects from the same category often share similar placing patterns. For example, two cylindrical objects (e.g. a crisp can and a big ketchup bottle) will probably have similar safe regions. It is important to consider a stable release with respect to other objects as well. The object should be placed in its preferred location and orientation in such a way that the other objects on the table will stay at the same positions. If the robot learns the safest placement once, it can still be able to perform this action successfully if the number of objects or their positions on the table are changed. In our algorithm we select a number of bins to compute a histogram empirically, but a more efficient way of such a selection is required. The next step is to extend this work for other robotic actions.

References

[1] N. Akhtar. Improving reliability of mobile manipulators against unknown external faults. Technical report, BRS University of Applied Sciences, Sankt Augustin, Germany, 2012.

[2] Y. Jiang, M. Lim, C. Zheng, and A. Saxena. Learning to place new objects in a scene. *I. J. Robotic Res.*, 31(9):1021–1043, 2012.

[3] B. Johnston and M. Williams. Comirit: Commonsense reasoning by integrating simulation and logic. In *Artificial General Intelligence*, pages 200–211, 2008.

[4] J. A. Jorgensen, L. Ellekilde, and H. G. Petersen. Handling uncertainties in object placement using drop regions. *Proceedings of ROBOTIK*, pages 1 –6, 2012.

[5] J.P. Mendoza, M. Veloso, and R. Simmons. Motion interference detection in mobile robots. In *International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[6] O. Pettersson, L. Karlsson, and A. Saffiotti. Model-free execution monitoring in behavior-based robotics. *IEEE Trans. on Systems, Man and Cybernetics, Part B*, 37(4):890–901, 2007.

[7] O. Pettersson. Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems*, 53:73–88, 2005.

[8] S. Zickler. *Physics-Based Robot Motion Planning in Dynamic Multi-Body Environments*. PhD thesis, Carnegie Mellon University, May 2010.