Simulation-based approach for avoiding external faults

Naveed Akhtar
Bonn-Rhein-Sieg University
of Applied Sciences
Grantham-Allee 20, 53757
Sankt Augustin, Germany
Email: nakhta2s@inf.h-brs.de

Anastassia Kuestenmacher
and Paul G. Plöger
Bonn-Rhein-Sieg University
of Applied Sciences
Grantham-Allee 20, 53757
Sankt Augustin, Germany
Email: anastassia.kuestenmacher@h-brs.de
Email: paul.ploeger@h-brs.de

Gerhard Lakemeyer
RWTH Aachen University,
Ahornstr. 55, D-52056
Aachen, Germany
Email:
gerhard@informatik.rwth-aachen.de

***Abstract*—When a robot interacts with its environment to perform tasks, it often faces unexpected situations which render its actions unsuccessful despite perfect functioning of its components. These situations occur as deviations of properties of the objects (manipulated by the robot) from their acceptable values. Hence, they are experienced by the robot as external faults. In this work, we propose a simulation-based approach for avoiding the external faults that occur during the manipulation actions of a robot which involve releasing of objects. With the help of a single example simulation, that shows the behavior of the manipulated object for successful completion of the action, the proposed approach constructs different examples of the object's behavior and labels them as 'desired' or 'undesired'. These labelled examples are used by an algorithm, which we refer as *N-Bins*, to suggest a releasing state of the object that avoids the occurrence of external faults. Once exposed to the labelled examples, *N-Bins* can also be used for predicting the occurrence of external faults for a given releasing state of the object. These abilities of *N-Bins* are used by the approach to modify the releasing action of the robot for avoiding external faults. We present the approach as a four-step scheme that performs the above mentioned tasks completely autonomously.**

I. Introduction

A robot that manipulates its environment to accomplish its tasks, often faces unexpected situations which prohibit it from achieving its goals despite perfect functioning of its internal components (e.g. sensors and actuators). During an action of a robot, these situations occur in its environment as unpermitted deviations of properties of objects (manipulated by the robot) from their desired/acceptable values and they result in an instant failure of the robot's action. Therefore, they are termed as *external faults* [1], [24]. These faults occur very commonly in the tasks performed by robots like mobile manipulators and humanoid robots. Fig. 1 shows an example of occurrence of an external fault when a robot performs the task of placing a die on a table (i.e. the cube). In the shown situation, the die is released by the robot in an *undesired orientation* which causes it to roll on its own and immediately fall on the floor. This renders completion of the robot's task unsuccessful.

Occurrences of external faults are reported very frequently in the robotics literature. For instance, Okada et al. [3] reports

978-1-4799-2722-7/13/$31.00

Fig. 1. Occurrence of an external fault while a robot places a die on a table. During the releasing action of the robot the orientation of the die causes the die to fall on the floor instead of staying on the table.

slipping of a bottle from the hand of HRP2JSK humanoid robot despite utilization of robust software and hardware. Similarly, in his survey on faults of robots [4] Steinbauer mentions occurrences of external faults by referring to them as *interaction faults* and *unfavorable environment conditions*. In [5], [6] and [7], the authors use the terms *unforeseen events*, *exogenous events* and *errors* to refer to external faults.

Despite their frequent occurrences, external faults have mostly eluded their treatment as primary research problems in robotics literature, especially under much suited perspective of being *faults*. This work treats external faults as a primary research problem and it focuses on the most commonly occurring external faults encountered in the robotic manipulation tasks. More specifically, it proposes an approach that helps a robot in avoiding occurrences of external faults encountered in the action of releasing an object over another object. The proposed simulation-based approach enables the robot to autonomously avoid the external fault in the future if its occurrence has been detected once.

We present the proposed approach as a four-step scheme that requires the following two inputs: 1) an *example simulation* and, 2) a definition of the *planning operator* of the action that results in a detected external fault. Here, (1) provides an example of the expected behavior of the object (manipulated by the robot) for a successful completion of the action, and (2) is a classical planning operator definition [13] (for a release action). The presented approach first finds a description of the behavior of the object in the *example simulation*. This description is given as logical expressions which consist of

conjunctions of ground atomic facts. Each of such atomic fact represents a different aspect of a given state of the objects in the simulation. We refer to the atomic facts collectively as the *description vocabulary*. The *description vocabulary* is derived from the concepts in the area of *qualitative spatial representation* [10]. Henceforth, we use the term 'predicate' instead of 'ground atomic fact' as a shorthand.

The scheme creates different examples of the behavior of the object (in simulation) and uses the description of the *example simulation* to *autonomously* label these examples as desired (i.e. positive) or undesired (i.e. negative). The labelled examples are utilized by an algorithm, which we call *N-Bins*, to find the best releasing state of the object. Releasing the object in this state avoids the occurrence of external faults. Once the *N-Bins* algorithm is exposed to the labelled examples, it can also be used for predicting the desirability of a releasing state of the object. We use the abilities of the *N-Bins* algorithm to modify the *preconditions* of the releasing action of the robot. This modification is done in a way that a releasing state of the object that satisfies the modified *preconditions* of the action would result in the desired behavior of the object. Hence, it avoids the occurrence of the external faults.

We conduct experiments with the proposed approach using the simulation environment OpenRAVE [8], which uses ODE [11] (a physics engine) to simulate the dynamics of rigid bodies. Results of the experiments show that by using the proposed approach not only we can calculate a releasing state of the object that avoids external faults but we can also explicate the states which result in the undesired behavior of the object. The results of our experiments also show that for our settings *N-Bins* outperforms other popular Machine Learning (ML) algorithms (e.g. decision trees, k-nearest neighbours) in its prediction (i.e. classification) ability.

II. Related Work

Fault tolerance and diagnosis is considered a major challenge in robotics [14], therefore numerous works in robotics are dedicated to this area. However, most of them deal with the faults that are caused by a robot's internal component failures. For instance, Verma et al. [15] present a real-time fault detection and diagnosis approach for the faults caused by a robot's mechanical component failures. Similarly, researchers in [16], [17], [22] and [23] present model-based approaches for identification and diagnosis of faults occurring in robots' internal components. However, internal component failures and malfunctioning are not the only sources of faults encountered in robotics.

Some recent works in robotics have also directed their attention to external faults. For example, in [1] the authors propose an approach that uses naive physics knowledge to explicate the reasons of the occurrence of unknown external faults encountered in releasing action of a robot. In this work, the robot applies a qualitative version of physical laws on naive physics knowledge to hypothesize the causes of occurrence of unknown external faults. In [18] the authors are interested in dealing with external faults that occur in the presence of external agents. To that end, the authors propose to formalize the understanding of 'normality' about the behavior of a robot's environment in presence of the external agents and deal with the external faults as deviations in the normality.

In the area of robotic manipulation Jorgensen et al. [19] and Moll and Erdmann [20] are, in essence, concerned with avoiding external faults encountered during the robotic action of releasing objects. Both of these works use simulation data to generate distributions of the final states of the released objects and use these distributions to estimate the pose and drop regions for the object. The approach proposed in [19] suffers from the issue that it requires very dense sampling of the distributions in order to succeed. This results in a huge number of required simulations. On the other hand, [20] only focuses on achieving particular orientations of objects in their final state for successful assembly tasks. The main focus of that work is on developing strategies to orient objects with minimal sensing and manipulation.

Ueda et al. [7] uses feedback to recover from external faults[1] occurring in presence of external agents. However, this work focuses recovering from the faults at planning level and does not deal with avoiding the occurrence of the faults. Similarly, Gspandl et al. [21] presents a belief management system to detect and explain inconsistencies in the higher level beliefs of a robot which are caused by external faults. However, this work is also not concerned with avoiding the causes of inconsistencies.

III. Settings & Problem Formulation

The approach presented in this work assumes the settings of a plan-based robotic system. This system detects the occurrence of an external fault by monitoring the *effects* of the executed actions. That is, if the *effects* of an action (i.e. a planning operator) remain unsatisfied after its execution, then the system detects an external fault and isolates it to the definition of the planning operator whose *effects* remain unsatisfied. In this work we assume that the detected fault is indeed an *external* fault. Furthermore, we neglect the presence of external agents in the environment of the robot and assume that causes of external faults are limited to natural physical phenomena (e.g. gravity, friction).

This work focuses on the external faults that occur during those actions of a robot which involve releasing an object (i.e. `object1`) over another object (i.e. `object2`). Occurrence of such faults can be described as a problem in which a robot expects `object1` to be in a particular *goal state* after releasing it in an *initial state*, but the object ends up in some *final state* that is different from the *goal state* [2]. This unexpected behavior of the object, which is governed by physical laws, is a result of unpermitted deviations of physical properties (e.g. location, orientation) of the object in its *initial state*. Thus, a robot can avoid the occurrence of the external faults by careful selection of the *initial state* of the object. This work enables a robot to make this careful selection in order to perform its action reliably in the future. This work uses ODE physics engine as a black box to simulate the dynamics of the object's behavior.

IV. Approach

Fig. 2 shows the basic schematics of the proposed approach. The approach is presented as a four-step scheme. Below we give details of each of these steps.

[1] External faults are termed as external errors in [7].

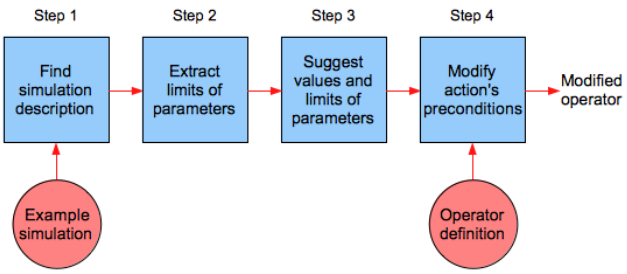


Fig. 2. Basic schematics of the proposed approach.

TABLE I. ASSOCIATION BETWEEN PREDICATES, ASPECTS OF OBJECT AND OBJECT'S PARAMETERS.

Predicate	Aspect of objects	δ
P1	Connectedness [9] of the objects	x, y, z
P2	object1's direction relative to object2	x, y
P3	Orientation of object1	ρ, θ, ϕ
P4	Linear velocity of object1	$\dot{x}, \dot{y}, \dot{z}$
P5	Angular velocity of object1	$\dot{\rho}, \dot{\theta}, \dot{\phi}$

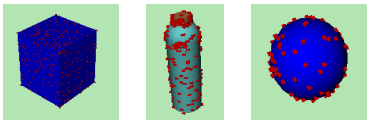


Fig. 3. Side views of models of object1 with markers (red dots).

A. Finding simulation description (step 1)

The first step of the scheme takes an *example simulation* as an input and finds logical expressions that describe the behavior of the objects in the simulation (i.e. *simulation description*). The behavior shown by the objects in the *example simulation* is expected to be exhibited by them when the performed action is completed successfully. The *example simulation* is passed as a construct comprising 1) Δ: a list of parameters, 2) $\boldsymbol{\mu}$: a vector of values of the parameters and 3) CAD models of the objects involved in the action. Each parameter δ in Δ corresponds to a physical property of the manipulated object and the values of these parameters (in $\boldsymbol{\mu}$) define the *initial state* of this object. The δs considered in this work are shown in the last column of table I. In this table x, y, z denote the coordinates of the geometric center of object1, whereas ρ, θ and ϕ denote the roll, yaw and pitch of object1. Other six parameters denote the components of the linear and angular velocities of the object.

The *example simulation* shows a single execution of the desired behavior of the objects without taking into account physical constraints of the robot or other objects in the environment. In order to find the *simulation description*, *step 1* places markers on the surfaces of the models of the objects. A marker $\mathcal{M} = [\mathbf{v}, d]$, where: $\mathbf{v}$ is a vector representing coordinates of a point on the surface of the object and d is a logical symbol (i.e. a constant) such that $d \in D$, where: $D = \{$*top, bottom, right, left, front, back, none*$\}$. For the case of object1 the point represented by $\mathbf{v}$ is a random point on the surface of the object, whereas for object2 this point lies on the boundary of the concerned surface (e.g. top) of the object (see fig. 3 and 4). Using the markers, this step autonomously constructs two logical expressions, $\mathtt{S}_{init}$ and $\mathtt{S}_{goal}$, which describe the *initial* and the *goal* states of the objects in the *example simulation*. A general representation of these expressions is given below.

$$\mathtt{S}_s \equiv \mathtt{P1} \wedge \mathtt{P2} \wedge \mathtt{P3} \wedge \mathtt{P4} \wedge \mathtt{P5}.$$

In the above expression each Pj (where, j$\in\{1,2,..,5\}$) is a predicate that represents a particular aspect of a given state of the objects in the simulation. The second column of table I shows the aspects represented by each Pj (given in the first column of the table). We characterize the behavior of the objects by their states at the start and end of the simulation. We do so because the dynamics of objects is taken care of by the physics engine. Thus, $\mathtt{S}_{init}$ and $\mathtt{S}_{goal}$ serve as the description of the behavior of the objects. In these logical expressions each predicate is selected from a collection of predicates, which we refer as *description vocabulary* (see section V). A predicate gets selected in these expressions from the *description vocabulary* when the given state of the objects satisfies the (logical) definition of that predicate.

Notice that for object1, using markers enables the approach to autonomously attach geometric semantics to different models for object1. Therefore, in our experiments (section VI) we consider those models for object1 which can be modeled from a cube, a cylinder, a sphere or any combination of these. For object2 the markers enable us to experiment with the objects who's convex planner top surface can be constructed from a rectangle, a semi-circle or any combination of these. Needless to say, using markers instead of complete models also provide computational advantages in mathematical calculations.

B. Extracting limits of parameters (step 2)

This step uses the predicates in $\mathtt{S}_{init}$ to find the limits of the parameters δs in Δ. These limits correspond to rough estimates of the extreme values of δs using which the expression $\mathtt{S}_{init}$ can be reconstructed. That means, if a vector $\boldsymbol{\mu}_{new}$ is created by selecting values of δs within these limits and $\mathtt{S}_{init_{new}}$ is constructed using $\boldsymbol{\mu}_{new}$, then for some $\boldsymbol{\mu}_{new}$ it should be possible that $\mathtt{S}_{init_{new}} \equiv \mathtt{S}_{init}$. In order to find such values we exploit the fact that each parameter δ is associated with a predicate Pj in $\mathtt{S}_{init}$. This association is shown in table I, where the third column of the table shows the parameters associated with each Pj in the first column.

Assume that for a given Object1 (e.g. a bottle) P3 $\equiv$ StraightAlong-z(Object1) in $\mathtt{S}_{init}$, where StraightAlong-z/1 is defined to be *true* if the orientation of Object1 is straight along the z-axis of a given reference frame. Then, rotating the object around z-axis (of the same reference frame) by any angle does not affect the truth value of StraightAlong-z/1. As we denote such a rotation by ϕ, therefore we can say that for this example $-\pi \leq \phi \leq \pi$. From these limits we choose a subinterval $[-\pi/4, \pi/4]$ as the limits of ϕ. Since rotating the object along x or y-axis of the reference frame can change the value of P3, therefore we choose limits of the other two orientation parameters (i.e. ρ, θ) to be defined by the interval $[-\pi/16, \pi/16]$. This interval is much smaller than that for ϕ because the value of P3 is more sensitive to the changes in the values of ρ and θ. The aforementioned intervals are only rough empirical estimates of the extreme values of the parameters that are supported by basic knowledge of mechanics. We also determine the limits of the other parameters in table I in a similar manner

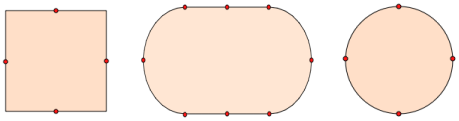


Fig. 4. Top views of models of `object2` with markers (red dots).

and associate them with the corresponding predicates in the *description vocabulary*. Further details on finding these limits and their association with the predicates can be found in [12].

The purpose of finding the limits of the parameters is to enable the robot to create different examples of the behavior of the objects in the simulation. Each of these examples is created by constructing a $\boldsymbol{\mu}_{new}$ vector with randomly selected values of δs within the limits associated with each predicate in $\mathtt{S}_{init}$. Once this vector is created, a simulation is run using this vector as the *initial state* of `object1`. After completion of the simulation, a description of the *final state* of the simulation is found. In this description, if `P1` and `P3` remain the same as corresponding predicates in $\mathtt{S}_{goal}$, then the behavior of the object is considered *desired* otherwise it is considered *undesired* [2]. Here, we only consider the predicates `P1` and `P3` because only they represent the aspects of interest for the robot's action that involves a simple release. It should be noticed that the existence of other predicates is not redundant as they are used in constraining the *initial state* of the objects[3]. We store the information on each example by creating a row vector whose first $n-1$ elements consists of the elements of $\boldsymbol{\mu}_{new}$ and the n^{th} element is the *label* of the example. The value of label is 1 for the desired behavior of `object1` in the simulation and 0 for the undesired behavior. By repeating the above mentioned procedure a robot can autonomously create as many labelled examples as required (using only one *example simulation*). Henceforth, we refer to such examples as *training instances*.

C. Suggest values and limits of parameters (step 3)

Assume that for a given action *step 2* generates m training instances. We store these instances in an $m \times n$ matrix **I** in which each row represents a single instance. *Step 3* divides **I** into two matrices **C0** and **C1**, such that **C0** consists of undesired (i.e. negative) instances and **C1** consists of desired (i.e. positive) instances. We evaluate the measures of *mean* (μ_f), *variance* (σ_f), *skewness* ($skew_f$) and *kurtosis* ($kurt_f$) of each column[4] of **C0** and **C1** and find the difference between the corresponding measures of the corresponding columns of these matrices (i.e. $\Delta\mu_f, \Delta\sigma_f, \Delta skew_f$ and $\Delta kurt_f$). We use these differences in calculating the so-called importance (IMP_f) of each parameter, which is further used in calculating the weight (W_f) of each parameter. In above notations and notations to follow the subscript f denotes 'feature'. Formulae for IMP_f and W_f are given in line '4' and '3' in fig. 5. In this figure and the figures to follow, we use the word *feature* in place of *parameter*. Henceforth, we also do the same in the textual description in this subsection (because of ML literature convention).

[2] `On(object1, object2)` is an example of `P1`. This predicate is *true* when `object1` is *on* `object2`.

[3] The predicates `P2`, `P4` and `P5` also remain available for more complicated actions of the robot. For instance, releasing an object to achieve a particular configuration in its *final state*.

[4] Except the n^{th} columns, which contain the labels of the instances.

1. **function** CALCULATE-FEATURE-WEIGHTS (**C0, C1**) **returns w**
2. *for each feature, f*
3. $W_f = \frac{IMP_f}{\sum_f IMP_f}$, where:
4. $IMP_f = |\Delta\mu_f| + |\Delta\sigma_f| + |\Delta skew_f| + |\Delta kurt_f|$
5. **w** = A vector composed of W_f for each feature
6. return **w**

Fig. 5. Definition of the function CALCULATE-FEATURE-WEIGHT.

For any feature, higher value of W_f shows that the final state of the object in the simulation is more sensitive to the value of that feature. This step divides the limits (found in *step 2*) of each feature into smaller bins in a way that features with larger W_f have more bins. For each feature the number of bins (NB_f) is calculated using the equation given in line '4' in Fig. 6. In this equation, $binMul$ and $augBin$ are constants whose values decide the maximum and the minimum number of bins for the features. For each bin of each feature, its $BinStrength$ is calculated (see line '11 - 15' in Fig. 6) with the help of p values of the feature (in the training instances) that fall in the bin. $BinStrength$ of a bin increases when a value that falls inside it belongs to a positive (i.e. $C1$) instance and it decreases when the value belongs to a negative (i.e. $C0$) instance. Thus, for a given bin of a feature, higher values of $BinStrength$ means that the bin contains more of those values of the feature which are suitable for the desired behavior of `object1`.

Algorithms in Fig. 5 and 6 show the functions for calculating W_f and specifying the bins of each feature. Here, specification of the bins implies calculation of NB_f, $BinStrength$ and limits of each bin of each feature. Both of the above mentioned functions are called by another function given in Fig. 7. This function utilizes the information on the bins specifications to suggest values of the features which are most suitable for achieving the desired behavior of the object manipulated by the robot. For each feature, the *suggestedValue* is calculated as the mean value of the limits of the bin with maximum $BinStrength$ [5]. These values are stored in a vector **sVal**, which (as a whole) represents the suggested *initial state* of the object. Limits of the bins of the *suggestedValue*s are stored in a matrix **SValLim**.

Lines '11 - 18' in Fig. 7 show the procedure of binary classification in which the algorithm predicts the label of any unseen instance (i.e. a test instance) as positive (i.e. 1) or negative (i.e. 0). Here, we pick the values of the features in the test instance and find the bins for each feature in which these values fall. Then, the $CumulativeStrength$ of the test instance is found with the help of the $BinStrength$s of the found bins. The formula for $CumulativeStrength$ calculation is given in line '15' of Fig.7. If the $CumulativeStrength$ of the instance is greater than 0.0, then the label of the test instance is predicted to be 1, otherwise it is predicted 0.

D. Modify action's preconditions (step 4)

Step 3 of the scheme provides us with three important components. 1) An *initial state* of the object (**sVal**) that is most

[5] We can also calculate the worst values of the features from the bins with minimum $BinStrength$s, in a similar manner.

1. **function** SPECIFY-BINS (**C0, C1, w**) **returns BL, bs, nb**
2. **Set** binMul (a positive real number), augBin (a positive integer).
3. *for each feature, f*
4. $NB_f = floor(\frac{binMul*(n-1)}{R_f}) + augBin$, where:
5. R_f = Position of a feature when the features are arranged in decreasing order of W_f
6. **nb** = A vector composed of NB_f for each feature
7. *for each feature, f*
8. Divide the range of the feature in equal NB_f bins
9. $\mathbf{BL}_f$ = A matrix containing the limits of each bin
10. *for each bin*
11. $BinStrength = \sum_p S(Ci) * W_f * k(Ci)$, where:
12. p = Total number of feature values falling in the bin
13. $S(Ci) = 0.9 + \frac{\#\text{of } Ci \text{ instances}}{5*\text{Total instances}}$ for $i \in \{0, 1\}$
14. $k(Ci) = 1$ for $i = 1$ and $= -\frac{\#\text{of } C1 \text{ instances}}{\#\text{of } C0 \text{ instances}}$ for $i = 0$
15. $\mathbf{bs}_f$ = A vector composed of $BinStrength$ of each bin
16. **BL** = A matrix formed by joining **BLf** of each feature
17. **bs** = A vector formed by joining $\mathbf{bs}_f$ of each feature
18. return **BL, bs, nb**

Fig. 6. Definition of the function SPECIFY-BINS.

1. **function** N-Bins (**I, T**)
2. **input: I**, an $m * n$ matrix composed of labelled instances
T, an $r * n$ matrix composed of test instances
3. **C0, C1** ← Split instances according to their labels
4. **w** ← CALCULATE-FEATURE-WEIGHTS (**C0, C1**)
5. **BL, bs, nb** ← SPECIFY-BINS (**C0, C1, w**)
6. *for each feature, f*
7. Select the bin from **BL** corresponding to maximum value in **bs**
8. $suggestedValue$ = mean of the limits of the selected bin
9. **sVal** = Vector formed by $suggestedValue$s of each feature
10. **SValLim** = Matrix formed by the corresponding limits of the selected bins
11. *for each test instance* **t**, *in* **T**
12. *for each feature, f of* **t**
13. Select the bin from **BL** in which the value of the feature falls
14. Select the corresponding $BinStrength$ from **bs**
15. $CumulativeStrength = \sum_f BinStrength * \frac{NB_f}{\sum_f NB_f}$
16. *if* $CumulativeStrength > 0.0$
17. *then* $predictedLabel = 1$
18. *else* $predictedLabel = 0$

Fig. 7. N-Bins algorithm.

suitable for achieving the *goal state* of `object1`, 2) limits of the bins of each feature/parameter (**SValLim**) from which the $suggestedValues$ of each parameter is calculated, and 3) the ability of predicting the desirability of an *initial state* (given as a test instance to *N-Bins* algorithm.). *Step 4* of the scheme uses these components to modify the releasing action of the robot in a way that the robot can avoid the occurrence of the detected external fault in the future. This is done by placing a predicate `Allowed/3` in the *preconditions* of the releasing action of the robot. This predicate is defined as following:
`Allowed(action, object1, object2)` $\Longleftrightarrow$
`Condition-1` $\vee$ `Condition-2` $\vee$ `Condition-3`.

According to the above logical definition, the predicate `Allowed/3` is evaluated *true* only if the `action` involving `object1` and `object2` satisfies at least one of the following three `Conditions`:
`Condition-1`: The *initial state* of `object1` is same as the state suggested by *N-Bins* algorithm as **sVal**.
`Condition-2`: Values of all the parameters in the *initial state* `object1` are chosen within the limits of the corresponding bins, given in **SValLim**.
`Condition-3`: A test instance representing the *initial state* of `object1` is predicted positive' by the *N-Bins* algorithm.

The `Allowed/3` predicate is placed in the *preconditions* of the planning operator of the releasing action. Definition of this operator is received as an input to the scheme, as shown in Fig. 2 in section IV. If a robot releases the `object1` in an *initial state* that satisfies the new *preconditions* then it can avoid the occurrence of the external faults. The above mentioned `Condition-1, 2` and `3` are stated in 'strict to relax' order, with `1` being the most strict. This is also the order of preference for a robot to satisfy these conditions. That is, a robot must first attempt to satisfy `Condition-1`, and if it is not possible then it must attempt to satisfy `Condition-2`, and so on.

V. Description Vocabulary

The approach uses *description vocabulary* to find the *simulation description* in *step 1* and to autonomously label the training instances in *step 3* of the approach. The *description vocabulary* consists of 67 predicates which represent the concepts related to five aspects (shown in table I) of the objects involved in the simulation. We formalize the definition of each concept (i.e. the predicate) as a logical expression that evaluates to *true* when the concept is true in a given state of the simulated objects. For instance, the expression given below defines a predicate `Over/2` (related to *connectedness* of the objects) that is *true* only when the simulation shows that an `object1` is *over* an `object2`.

```
Over(object1, object2) ⟺
(z-coord(lowest(object1)) > height(object2))
∧ NTPP(xy-proj(object1), xy-proj(object2)).
```

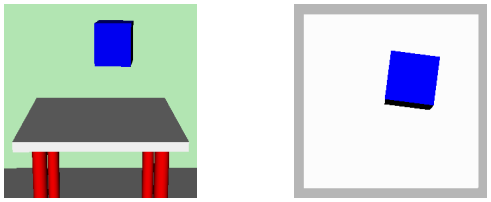
In the above expression `z-coord/1` is a function that refers to the z-coordinate of the point in its argument. Similarly, `lowest/1` refers to the lowest point of the object in its argument, `height/1` refers to the height of the object in its argument and `xy-proj/1` refers to the xy-plane projection of the markers on the object in its argument. `NTPP(pp1, pp2)` is a predicate that evaluates to *true* when the convex hull of the points in the set of points `pp1` is a *non-tangential proper part* [9] of the convex hull of the points in set `pp2`.

`Over/2` is only one of the 67 predicates of the *description vocabulary*. We formalize the definitions of all the predicates in a similar manner. We also formalize the functions (e.g. `z-coord/1`) used by these definitions. All the predicates in the *description vocabulary* and the functions used by these predicates have been created systematically within the scope of this work. It is not possible to discuss all the details about the *description vocabulary* in this paper because of the limitations of the space. Complete details on this topic can be found in chapter 6 of [12].

VI. Results and Discussion

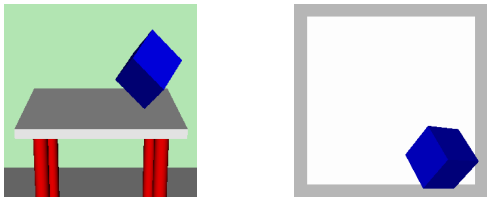
A. Results

We performed various experiments with the proposed approach in which different objects were released over other objects. Here, we present results of three of these experiments.



(a) Front view (b) Top view

Fig. 8. Suggested initial state of the die for experiment 1.



(a) Front view (b) Top view

Fig. 9. Estimated worst initial state of the die for experiment 1.

1) Experiment 1 (release a die over a table): In this experiment the *example simulation* shows that a die is dropped on a table. From this single simulation the approach calculates the (approximate) safest *initial state* (i.e. **sVal**) of the die. This state is shown in Fig. 8. In Fig. 9 we also show the (approximate) worst *initial state* of the die calculated by the approach. For a plan-based robotic system the qualitative descriptions of both the shown states are the same (i.e. the die is being dropped on the table). However, the proposed approach is clearly able to distinguish the desirability of the two states for a successful completion of the action. Similarity between the initial states of the die shown in Fig. 9 and Fig. 1 is also noticeable. In our experiments, we also test the prediction ability of *N-Bins* algorithm. We do this by creating 1000 unseen test instances which comprise 500 positive and 500 negative instances. We test how accurately the algorithm predicts the labels of these test instances after it is exposed to different numbers of training instances. Results of these tests, for this experiment, are shown in Fig. 10. This figure also shows the prediction accuracy of other popular ML algorithms with the same training and test instances. Models for these algorithms were selected using five folds cross validation. Further details on the model selection procedure and working principles of the algorithms can be found in [12]. We do not discuss these details here because they are not directly relevant to the main focus of this work. Discussion on the reasons for better performance of *N-Bins* is also deferred to section VI-B.

2) Experiment 2 (release a carton over a table): As mentioned in section I, the proposed approach expects *example simulation* to demonstrate the desired behavior of the objects for a given action. Therefore, the *example simulation* is expected to be created at the time when the planning operator for the action is first defined. Hence, the models of the objects in the *example simulation* can become outdated by later changes in the robot's environment. For example, the model of the table in the *experiment 1* changes if some other objects are later placed on the table. At first, it may appear that such changes will break up the approach because of its dependence on the *example simulation*. However, this is not true. This fact

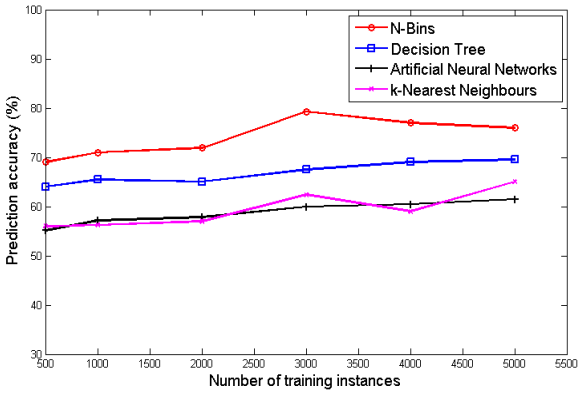
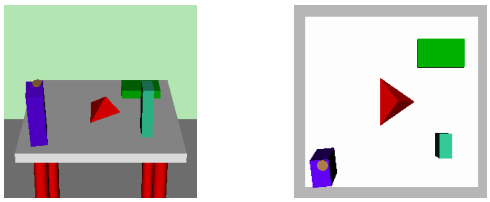


Fig. 10. Comparison of prediction accuracies of learning algorithms for experiment 1.



(a) Front view (b) Top view

Fig. 11. Suggested initial state of the carton for experiment 2.

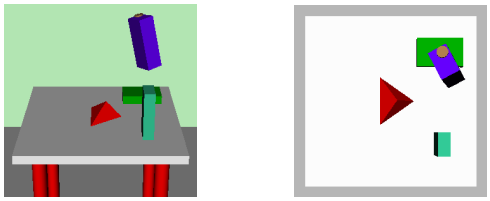
is illustrated in this experiment.

For this experiment the *example simulation* shows that a carton is dropped over the center of an empty table such that it stands tall on the table. However, before generating the training instances we update this model by placing different solid objects on the table (see the models of the table in Fig. 11 or Fig. 12). This update corresponds to the later changes in the robot's environment [6]. We simply let the approach to generate the training instances using the updated models. This automatically adjusts the results according to the update. This fact is visible in Fig. 11 and Fig. 12, which respectively show the best and the worst *initial states* calculated by the approach for this experiment.

The approach is able to find the correct results despite outdated models of the objects in the *example simulation* because it was able to correctly label the training instances. The correct labeling of the training instances, in turn, is made possible by the fact that the approach uses only qualitative information in the *simulation description* of the *example simulation*. This information (encoded as predicates in the *simulation description*) remains unaffected by the quantitative changes caused by the updates in the models of the objects. Hence, the approach is able to find the correct results.

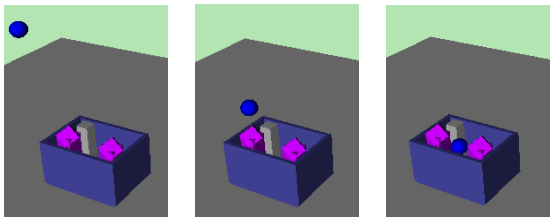
3) Experiment 3 (throw a ball into a basket): In this experiment the *example simulation* shows that a ball is *thrown* towards an empty basket, such that it falls and stays inside the basket. Similar to the case of experiment (2), we update the model of the basket in this experiment. The approach generates

[6]It is assumed that the updated models of the objects are available to the robot.



(a) Front view (b) Top view

Fig. 12. Estimated worst initial state of the carton for experiment 2.



(a) Frame 1 (b) Frame 2 (c) Frame 3

Fig. 13. Motion of the ball, for experiment 3, when it is thrown according to the initial state suggested by the approach.

the instances of throwing the ball using this updated model which contains different solid objects inside the basket. Based on these instances, the approach calculates the best way of throwing the ball into the updated model of the basket. The motion of the ball, according to its *initial state*, suggested by the approach, is shown in Fig. 13. Considering the fact that under the given conditions the *final state* of the ball can vary dramatically with slight changes in the location and/or velocity components of the ball in its *initial state*, this result shows how well the approach performs for such a complex action.

B. Discussion

In the results reported above, the *initial states* of different objects (shown in the figures) are calculated using 3000 training instances in each experiment[7]. These instances comprise different numbers of positive and negative instances. For example, in experiment (1) among 3000 training instances 2,771 are positive and 229 are negative. Notice, here negative instances represent the case of 'die falling from the table despite being released over the table', therefore in the simulation process the frequency of its occurrence is very low. This results in highly skewed training data set. Our experiments suggest that this is true in general. Usually, a classification algorithm's prediction accuracy is not seriously affected by the skewness of the training data if it is tested over a data set with similar distribution. However in our settings we can not assume that the test instances will have same skewed distribution as training instances. There are two major reasons behind that. First, in a real world application the values of features in a test instance are governed by the physical constraints of the environment and the robot. Under different circumstance this can result in distributions of test instances which can be different from the distribution of the training instances autonomously generated by the simulator. Second, in our settings a robot generates a test instance once its action had previously failed because of *unexpected* or *unforeseen* situation. Thus, we should not assume that the robot posses any a priori knowledge about the distribution of the test instances.

Based on the reasons stated in the previous paragraph we tested the accuracy of *N-Bins* on the test data with equal number of positive and negative instances and compared it with other popular classification algorithms. Results of experiment (1) and further experiments (not reported here) show that under such conditions *N-Bins* generally shows better prediction accuracies than the other learning algorithms found in the legend in Fig. 10 [8]. Complete details of these experiments can be found in [12]. For some of the experiments in [12] *artificial neural networks* (ANNs) show better accuracies, however *N-Bins* also shows comparable accuracies in those experiments. In the experiments reported here we choose $binMul = 1.8$ and $augBin = 4$ in line '2' in Fig. 6. These values divide the limits of each parameter of `object1` into 5 to 25 bins (depending upon the importance of the parameter).

From the results of the experiments it is clear that releasing an `object1` in the *initial state* suggested by the approach always results in its desired behavior. Thus, if a robot releases an `object1` in a state that satisfies `Condition-1` of `Allowed/3` (in section IV-D) then it can avoid the occurrence of external faults. To test `Condition-2` of `Allowed/3` we created 100 *initial states* (for one experiment) that satisfy `Condition-2` and simulated them to see the behavior of the objects. We did this for five different experiments. In all these simulations the objects showed the desired behaviors. In the nine experiments in [12] (including above three) the *mean* of *N-Bins*' prediction accuracy is 75.11% with *standard deviation* of 5.36 (ANNs are second best with *mean* = 70.5% and *std. dev.* = 13.88). This shows how `Condition-3` (the most relaxed condition) of `Allowed/3` increases the chances of avoiding the external faults in an action.

VII. Conclusion and Future Work

In this paper we proposed a simulation-based approach for avoiding external faults which occur in the releasing action of robotic manipulation tasks. The proposed approach enables a robot to estimate the safest releasing state of an object for successful completion of its action. It also enables the robot to predict the behavior of an object for a given releasing state. This ability of the approach comes from an algorithm, referred as *N-Bins*, that uses labelled instances of simulated behavior of the manipulated object. These instances are labelled with the help of an *example simulation* that shows the desired behavior of the object for the performed action. The labeling process is made autonomous by capturing qualitative description of objects' behavior in the *example simulation* and using it for labeling the instances. We performed different experiments with the proposed approach in a simulation environment. Results of these experiments clearly show that with the help of the approach proposed in this work, a robot can not only avoid external faults by selecting a safe releasing state for the

[7] With the help of parallelization on a 48-core PC (4 × AMD Opteron™6174 12-core 2.2 GHz) 3000 instances per experiment are generated in an average time of less than 0.7 seconds.

[8] Accuracies of other algorithms improve if the test instances has distribution similar to that of training instances. For such cases the accuracy of N-Bins also improves and remains comparable to other algorithms.

object but it can also predict desirability of a given releasing state of the object with considerable accuracy.

Broadly speaking, the main idea behind the proposed approach for avoiding external faults can be summarized as, *give a robot an example of the action to be executed and let it find the safest way to do it*. This paper presented the tools (e.g. *N-Bins*, *description vocabulary*) that realized this idea for the robotic action of releasing objects. With promising results shown by the proposed approach, obvious future direction is to extend this work for other robotic actions (e.g. picking objects).

ACKNOWLEDGMENT

This work was supported by Bonn-Aachen International Center for Information Technology (B-IT). The authors would like to thank Alexander Asteroth and Iman Awaad for their helpful comments and suggestions.

REFERENCES

[1] N. Akhtar and A. Kuestenmacher. Using naive physics for unknown external faults in robotics. In 22nd International Workshop on Principles of Diagnosis (DX- 2011), 2011.

[2] N. Akhtar. Fault reasoning based on naive physics. Technical report, Hochschule Bonn-Rhein-Sieg, Sankt Augustin, February 2011. URL http://opus.bib.hochschule-bonn-rhein-sieg.de/opus-3.3/volltexte/2011/5/

[3] K. Okada, M. Kojima, S. Tokutsu, Y. Mori, and M. Inaba. Task guided attention control and visual verication in tea serving by the daily assistive humanoid hrp2jsk. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008.

[4] G. Steinbauer. Survey on faults of robots used in robocup, 30.03.2011. URL http://www.ist.tugraz.at/rfs/

[5] P. Pastor, L. Righetti, M. Kalakrishnan and S. Schaal. Online movement adaptation based on previous sensor experiences. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011.

[6] G. Steinbauer and F. Wotawa. On the Evaluation and Certication of the Robustness of Autonomous Intelligent Systems. In 22nd International Workshop on Principles of Diagnosis (DX- 2011), 2011.

[7] R. Ueda, Y. Kakiuchi, S. Nozawa, K. Okada, and M. Inaba. Anytime error recovery by integrating local and global feedback with monitoring task states. In Proceedings of the 15th international conference on advanced robotics, 2011.

[8] R. Diankov. Automated Construction of Robotic Manipulation Programs. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[9] D. A. Randell, Z. Cui, and A. G. Cohn. A spatial logic based on regions and connection. In Proceedings of 3rd international conference on knowledge representation and reasoning, 1992.

[10] A. G. Cohn and S. M. Hazarika. Qualitative spatial representation and reasoning: An overview. Fundam. Inf., 46(1-2):129, 2001.

[11] R. Smith. Open dynamic engine, May 2007. URL www.ode.org.

[12] N. Akhtar. Improving reliability of mobile manipulators against unknown external faults. Technical Report, BRS University of Applied Sciences, Sankt Augustin, Germany. February 2012.
URL http://nbn-resolving.de/urn:nbn:de:hbz:1044-opus-91

[13] M. Ghallab, D. Nau and P. Traverso. Automated planning theory and practice. Morgan Kaufmann Publishers Inc., 2004.

[14] P. M. Frank, R. J. Patton. Issues for fault diagnosis of dynamic systems. Springer-Verlag, 2000.

[15] V. Verma, G. Gordon, R. G. Simmons, and S. Thrun. Real time fault diagnosis. Robotics and Automation Magazine, 11(1):56 66, June 2004.

[16] G. M. Coghill and H. Liu. A model-based approach to robot fault diagnosis. Knowledge-Based Systems, 2005.

[17] V. Verma and R. G. Simmons. Scalable robot fault detection and identication. Robotics and Autonomous Systems, 54(2):184191, 2006.

[18] M. Karg, M. Sachenbacher, and A. Kirsch. Towards expectation-based failure recognition for human robot interaction. In Proceedings of 22nd International Workshop on Principles of Diagnosis (DX-2011), 2011.

[19] J. A. Jorgensen, L. P. Ellekilde and H. G. Petersen. Handling Uncertainties in Object Placement using Drop Regions. In Proceedings of ROBOTIK 2012 - 7th German Conference on Robotics, 2012.

[20] M. Moll and M. A. Erdmann. Manipulation of pose distributions. International Journal of Robotics Research, 21(3):277292, 2002.

[21] S. Gspandl, I. Pill, M. Reip, G. Steinbauer, and A. Ferrein. Belief management for high-level robot programs. In Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, 2011.

[22] A. Monteriu, P.Asthana, K. Valavanis and S. Longhi, Model-based sensor fault detection and isolation system for unmanned ground vehicles: theoretical aspects (part I) and experimental validation (part II), IEEE International Conference on Robotics and Automation, Rome, Italy, 2007.

[23] A. De Luca and R. Mattone, An adapt-and-detect actuator FDI scheme for robot manipulators, IEEE International Conference on Robotics and Automation, New Orleans, LA, USA, 2004.

[24] A. Kuestenmacher, N. Akhtar, P. G. Ploeger and G. Lakemeyer, Unexpected Situations in Service Robot Environment: Classication and Reasoning Using Naive Physics, Robocup Symposium, 2013.