Improving Task Planning Knowledge Robustness for Autonomous Robots

Hadeel Jazzaa, [1] **Thomas McCluskey,** [2] **David Peebles**[3]
University of Huddersfield [1,2,3]
hadeel.jazzaa@hud.ac.uk, [1] t.l.mccluskey@hud.ac.uk, [2] d.peebles@hud.ac.uk[3]

Abstract

For a more robust robot capable of adapting to the changing environment, the goal of this work is to bridge the gap between abstract plans and robot action execution. Our platform combines planning, reasoning and learning new success values incrementally based on experience. Refinement involves reasoning over action execution failure using anomaly detection techniques. This paper reports on the embedding of a theory refinement process within a real robot. Empirical testing and evaluation has been performed using the NAO robot in kitchen scenario.

Introduction

Task planning for autonomous robots presents many challenges, in particular those stemming from changes in the environment and the mission requirements. It is infeasible to pre-program such robotic applications by predicting, at the design stage, all possible courses of actions on demand (Ingrand and Ghallab 2017). The plan execution strategy must account for the action/-plan failure, which results from ignorance or change (Cashmore et al. 2015). The planning engine itself may need updating from time to time, but this is not seen as critical as the validity of the knowledge being reasoned about.

The robot planners and designers of robot controllers, treat actions at a level of abstraction that neglects their subtle differences. The planning system considers actions as black boxes with performance independent of the prior and subsequent steps (Stulp and Beetz 2008). This abstraction is partially achieved by ignoring action parameters, which are relevant to the performance not to the action selection (planning). Our research aims at moving towards the goal of Long-term autonomy (Kunze et al. 2018). Long-term autonomy can be achieved by making the system more robust through experiential learning: failures and successes drive the improvement of the pre-programming multi-level representation. The research is based on a multi-level abstraction platform, represented in Figure 1, with the most abstract level being the *Planning Domain Definition Language* (PDDL) and all the abstractions are related to each other.

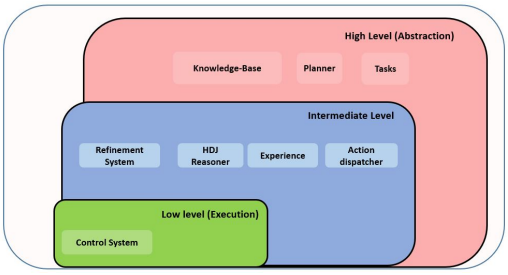

Figure 1: HDJ Intelligent Robot Hierarchical Architecture

Our hypothesis is that the gap between the abstract plan and its execution is one of the main reasons for plan execution failure and that action abstraction and ignoring of action parameters are part of this gap. The long term aim therefore to bridge this gap between actions at the high-level stage (abstracted actions) and actions at low-level stage (low-level commands). We think bridging this gap will make it possible to enable robots to adapt to changing situations and lead to more robust planning and efficient behaviour.

The action parameters and all information related to the actions and objects involved with that action are crucial to our approach and so the focus of this study is to learn from action execution where action (here represented in PDDL) are transformed from this abstract level to be executed by the control system. Examining the information and parameters of the failed action could lead to the discovery of the failure's cause. Our take on knowledge refinement is inspired by human cognitive science. When an individual fails to execute a frequently performed task, s/he will try to discover the reason and the first question that comes to mind is: "What did I do differently this time that led to failure?". The human action control is an integration

of feed-forward and feedback components (Schmidt 1975; Glover 2004). For example: picking a pan up does not need knowing its exact weight in advance; humans can determine this easily by picking it up and slightly increasing the exerted force until the pan leaves the surface. According to Schmidt (Schmidt 1975), human action control is hybrid, combining both feed-forward and feedback components. Schmidt argued that humans set action schemas by specifying the relevant attributes of that action but leave free parameters to be specified online while collecting environmental information.This indicates the integration of the off-line action planning with the online sensorimotor and knowledge update.

The novel aspect of our contribution is the addition of anomaly detection and subsequent knowledge refinement performed by Algorithm 1 on a PDDL domain model within a real functioning robot. The refinement activity starts after execution has failed and the feedback of the failed execution has been sent to the knowledge-base. Before starting the re-planning, to avoid failures in the future, the system refines the domain model that the planning engine relies on to produce plans.

In this paper we describe a hierarchical robot architecture (HDJ) which continuously stores data from planning experiences, analyses failures in operation, detects anomalies using historical data, and changes the planning knowledge in order to make plans produced in the future more robust. We evaluate the approach by investigating the result of knowledge refinement within a NAO robot in a kitchen scenario.

Related Work

Most work in anomaly detection in robotics has concentrated on condition monitoring and faults detection, and novelty detection in the environment. In addition to fewer applications that detect anomalous behaviours. However, there is little evidence that anomaly detection has been used specifically to identify inaccuracies in a robot's symbolic knowledge, initiated by the execution failure of its own generated plans, as covered in this work. It has been employed in robot condition monitoring and fault detection (e.g., (Khalastchi et al. 2015), (Hornung et al. 2014), and (Häussermann, Zweigle, and Levi 2015)) and for novelty detection in environment (e.g., (Crook, Hayes, and others 2001) and (Crook et al. 2002)). While (Kunze et al. 2018) employed anomaly detection to learn differences for object classification and (Ando et al. 2011) employed it for identifying anomalous behaviours using time scale and resolution. In addition to this, anomaly detection has been employed in healthcare applications, for Robot-Assisted feeding (Park, Hoshi, and Kemp 2018).

Most of the current contributions for acquiring planning domain models or control knowledge are concerned with speeding up task planners; only a few consider learning while acting; even fewer are demonstrated in robotics (Jiménez et al. 2012). Indeed, many researchers have been concerned with detecting the cause of the failure. However, there is little evidence of techniques of knowledge refinement resulting from the results of AD, as covered in this work. The presented approaches exploit different methods to learn new knowledge for refining planning models. For example, LIVE (Ranasinghe and Shen 2008) employs the incremental enlargement heuristic to examine the faulty rules while EXPO (Gil 1992) employs the ORM method to refine incomplete planning knowledge. GDA (Weber, Mateas, and Jhala 2012) and OBSERVER (Wang 1996) learn from demonstration. Furthermore, Probabilistic rules of the success of actions are learned by PELA system (Jiménez, Fernández, and Borrajo 2013). The RACE system (Rockel et al. 2013) employs HTN planner to learn from experiences. While ProbCog (Karapinar, Altan, and Sariel-Talay 2012) employs Logic Programming (ILP) and Geometric Reasoning module. The work in (Lindsay et al. 2020) refines hybrid domain models. It utilises machine learning techniques to identify the serious situation and temporal features. The REWRITE's system(Upal 1999) refines plans by employs what is called "a conflicting choice point". In another approach, The work in (Stulp and Beetz 2008) predicates the effects and performance of the planed actions earlier before the execution stage.

System Overview

The HDJ hierarchical system architecture we employ is inspired by previous work on robotics architectures such as the layering used in the functional architecture within a UAV (Doherty, Kvarnström, and Heintz 2009). HDJ consists of three layers (represented in Figure 1) embedded into an environment that enables the recording of planning and acting experiences, and subsequent adaptation of knowledge.

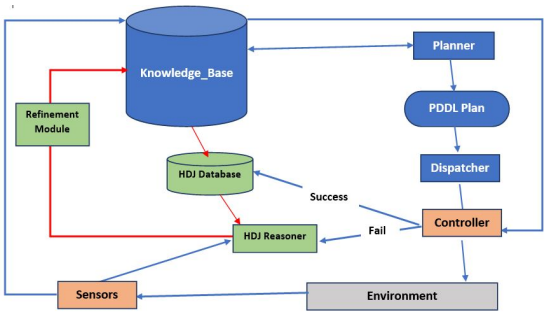

Figure 2: HDJ data flow diagram. The existing components (Blue boxes) and HDJ components (Green boxes)

Figure 2 presents the data flow through HDJ. It contains original components as well as existing components - in particular parts of the ROSPlan (Cashmore et al. 2015) are utilised in the top level of HDJ. The intermediate level is a reactive level that processes actions

to be executed and processes the execution feedback. It includes the action dispatcher, action interface and the extension components of HDJ (the green boxes in Figure 2). As seen in Figure 2, if the feedback coming from the control system returns "execution failed", the Algorithm 1 is employed to detect the anomalies in the information of the failed execution that do not appear in the robot experience.

The action execution is a bottom up process. The controller executes actions, re-actively responds to immediate changes and provides feedback on the effect of the action to KB. The controller receives this data and passes it up to the HDJ reasoner, in the intermediate layer. Sensory information is used to update part of this Knowledge-Base and hence generate updated state information such as positions of objects (Bailey and Durrant-Whyte 2006).

The Algorithms in HDJ

Algorithm 1 is the top-level algorithm which collects data and drives the changes to HDJ's knowledge (see Figure 3). This algorithm processes the feedback coming from the control system after each execution. The control system sends feedback to the Knowledge-Base and this feedback is either that the execution was successful or failed. In the case of "failed", the information of the failed execution will be tested to extract the differences (anomalies) by employing Algorithm 2, which can detect a single or group of anomalies (point and collective). The output of Algorithm 2 is a report of the extracted differences that are considered as the potential cause of failure. It is used to learn success values by Algorithm 3, and then to refine the PDDL model by executing Algorithm 4, that updates the KB. Any detected anomaly is a value that indicates parameters or pieces of information in the Knowledge Base, such as pre-conditions and states. Future failure can be avoided by updating the knowledge of the task's planner.

Training Data

The training-data TD can be defined as a history record of all previous successful executions. Let $A = a_{1,\ldots,an}$ be the set of attributes that represents all information related to an action.

TD is an $m \times n$ matrix where the columns denote n attributes and the rows maintain the values of these attributes over m execution. For example, in the domain model DM in Listing 1, consider the operator 'grip' that has n number of attributes. The distance between the robot and the targeted object represents its first attribute (X_1) while (X_2) is the angle attribute that represents the position of the targeted object. So, (X_{m1}) and (X_{m2}) are the the distance and angle values in the execution number (m). As can be seen from the Figure 4, if we recorded 100 successful execution, then (m) will take any value between 1 and 100.

Algorithm 1: Discrimination Process Algorithm

Data: Training data, failed action information
Result: KB Update
// **Pseudo Code:**

```
1  while plan execution do
2      if not (Feedback.Success) then
3          TD ← Training-Data
4          FD ← Failed-Data
5          Anomaly Detection(FD, TD)
6              // Implementation of Algorithm 2
7          if QueryResult.count > 0 then
8              // If Outliers exists
9              Out ← Detected Outlier
10             Return Out
11         Learning-Processing (Out)
12             // Learn new success values,
                  Algorithm 3
13             LV ← Learned Value
14             Return LV
15         Refinement-model(LV);
16             // Refine the KB, Algorithm 4
17     end
18     else
19         SD ← Successful action Data
20         Add-Training-Data (SD);
21     end
22 end
```

```
(define (domain NAO)
(:requirements :strips :typing :fluents)
(:types waypoint object robot gripper)
(:predicates (at-robby ?r - robot ?x - waypoint)
             (at ?o - object ?x - waypoint)
             (free ?r - robot ?g - gripper)
             (carry ?r - robot ?o - object ?g - gripper))
(:functions (dist_to ?r - robot ?x - waypoint)
            (maxdis ?r - robot ?x - waypoint)
            (mindis ?r - robot ?x - waypoint)
            (hwangle ?r - robot)
            (maxhwangle ?r - robot))

(:action goto :parameters (?r - robot ?from ?to - waypoint)
  :precondition (and
                 (at-robby ?r ?from))
  :effect (and
                        (at-robby ?r ?to)
                        (not(at-robby ?r ?from))))
(:action grip :parameters (?r - robot ?obj - object ?waypoint
-waypoint ?g - gripper)

 :precondition
     (and (at-robby ?r ?waypoint)
     (at ?obj ?waypoint)
     (free ?r ?g)
     (> (dist_to ?r ?waypoint) (mindis ?r ?waypoint))
     (< (dist_to ?r ?waypoint) (maxdis ?r ?waypoint))
     (< (hwangle ?r) (maxhwangle ?r)))

 :effect (and
     (carry ?r ?obj ?g)
     (not (at-robby ?r ?waypoint))
     (not (free ?r ?g)) ))
```

Listing 1: A segment of DM

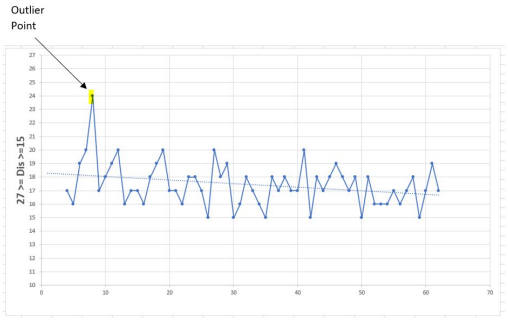

Figure 3: Anomaly detection of a failed execution. The outlier 'distance' value appears far a way from its nearest neighbour in TD

Algorithm 2 embodies our approach to anomaly detection (the approach is referred to as ADHDJ below). This is the problem of searching for patterns in data that differ and raise suspicions about a specific problem or issue (Zimek and Schubert 2017). We use the same technique to compare the values of the failed action execution with the training data. The purpose is to discover the suspicious values in the failed execution, and we suppose that the outlier value is responsible for that failure.

A standard approach for anomaly detection in datasets is to create a normal data model and compare/test records against normality (Zimek and Schubert 2017). First, we define normality for the given data. Because the TD represents the values of successful executions, we assume that historical records of successful execution contain data values falling within a normal distribution.

In our anomaly detection approach, the failed execution record is compared and tested against TD by extracting the anomalous values of the action execution features. For our running example, in the Evaluation Section, the grip action, Figure 3 shows the draw profile of the distance attribute's history, where the outlier point represents the value of the distance feature of the failed action.

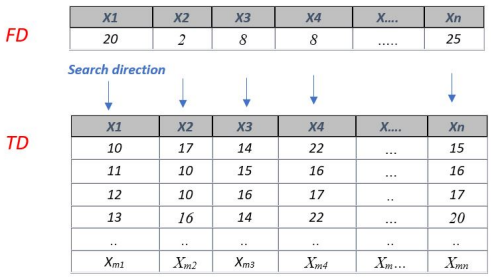

Figure 4: ADHDJ search. Each attribute instance (X) of FD is tested against its X vector in TD. n is number of attributes. M number of records in TD

ADHDJ can detect both a point or a group of anomalies. In the latter case more than one anomaly value that belongs to different attributes is found. It deals with multiple vectors that represents the action attributes. The failed execution information (FD) is a record of a failed complete action execution that is given online. FD is an input vector $x = x_1,...,x_n$ where n is number of attributes of FD. The online anomaly detection problem is to decide for each given x, whether or not x is anomalous with respect to TD. ADHDJ approach tests each attribute of FD and searches for the same attribute that does not appear in TD. As seen in Figure 4, ADHDJ considers rules of form X where X attribute takes particular values. We seek the value of $X \in$ FD that does not appear in the expected distribution of TD; Where $X \in$ TD, that $X = X_1,...,X_m$. Each action attribute is represented by X vector where the X vector is an $m \times 1$ matrix where m is number of successful executions in TD.

Algorithm 2: Anomaly Detection

```
//  TD-X: Attribute value in the
    Training-Data
//  FD-X: Attribute value in the
    Failed-Data
//  FD-X.label: Attribute name in
    the Failed-Data
//  n = Number of attributes
//  m = Record number of TD
Open TD Data-set;
for ( i ← 1 to n ) {
    for ( j ← 1 to m ) {
        Select FD-X(i) ∉TD-X(j,i) ;
        next j
    }
    Next i
}
if QueryResult.count > 0 then
    // if an anomaly detected
    Insert into report_table FD-X(i).value,
     FD-X(i).label;
end
```

HDJ Refining Process and Generalisation.

Algorithm 3 implements the learning-processing procedure called from Algorithm 1. This algorithm proposes the LV (learned value) based on the value of Out. LV is calculated by adding $Unit$ value to the detected outlier. The size of each $Unit$ is determined by a parameter that we call the learning rate LR. $Unit$ is considered as a step toward the $NNhg$ and reduces the gap between the predicted value (initial DM value) and success values (the experience). In ML and statistics, the step size or learning rate is a tuning parameter in an optimisation algorithm that determines the step size at each iteration to minimise the cost. It makes

steps down the cost function in the direction with the steepest descent (Murphy 2012). Figure 5 is a visualised representation of LP for 'dis' rule.

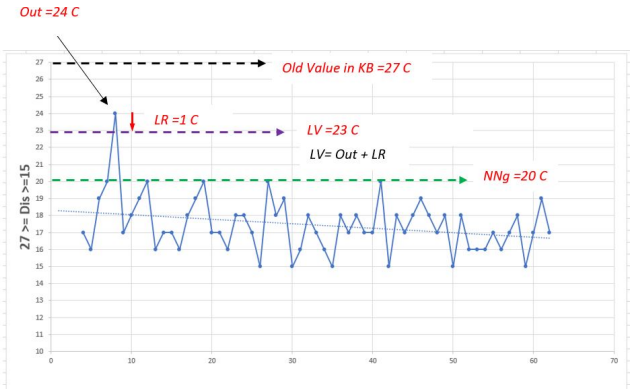

Figure 5: Learning process: The calculation of the new value *LV* for the 'dis' rule.

In the presented example, in Figure 5, the *Unit* size is **one** centimetre; This because *Out* is a distance value that is measured by the centimetre unit.

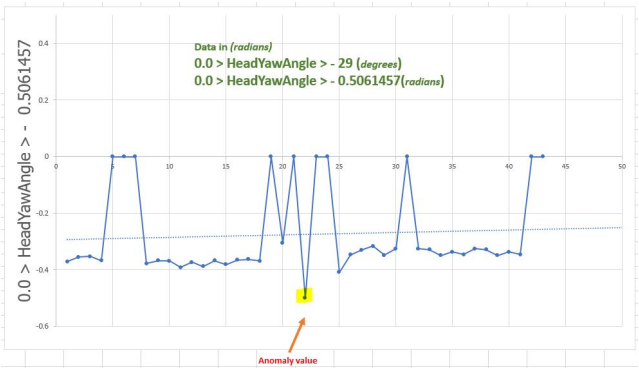

Figure 6: Failed execution with anomaly detection . 'HeadYawAngle ' value is less than its nearest neighbour(*Out* < *NNhg*)

The direction in which the step is taken is determined by the comparison of *Out* and *NNhg*. LPA considers two comparisons of outliers. The first comparison is represented in Figure 3, and Figure 5 where the outlier value is greater than its nearest neighbour. While the second comparison type is represented in Figure 6, and Figure 6 where the outlier value is less than its *NNhg*. Figure 7 shows how to specify *LV* for the two type of anomaly in the 'distance' attribute.

In summary, Algorithm 3 derives a new success value (*LV*) based on the detected anomaly by Algorithm 2. If the (*Out*) greater than its nearest neighbour (*NNg*) in TD, the new value *LV* will be (*Out* minus one *unit*). Else, the new value (*LV*) will be (*Out* plus one *unit*), Figure 5 and Figure 7. Then *LV* will be passed to *Refinement-model* to refine the related value in KB.

Learning Process

Out ← Outlier value
NNhg ← Nearest Neighbor in TD
LV ← Learned value
Fn ← Fluent name

If Out > NNgh ← ***Example 1*** **(24 > 20)**
LV= Out - Unit ← **LV = 24 -1**
Fn= flaunt Name ← **Example 1 (maxdis)**
Else

If Out < NNgh ← ***Example 2*** **(13 < 15)**
LV= Out + Unit ← **LV = 13 + 1**
Fn ← Fluent name ← **Example 2 (mindis)**

End Process

Figure 7: learning process for two type of anomalies in the 'distance' attribute. *LV = Out - Unit* this when *Out > NNhg*. while if *Out < NNhg*, *LV = Out + Unit*

.

Algorithm 3: Learning Process Algorithm

```
1  While Anomaly Detection
2  if Anomaly-Detection.count > 0 then
       // If Outliers exists
3      Out ← Detected Anomaly
4      Att ← Attribute name
5      Att : get-attribute-name (Out)
       // Get the name of the attribute in KB that
           represents the outlier value
6      LV ← Learned value
7      NNhg ← Nearest-Neighbour in TD
8      Unit ← One measurement unit
       // For example one centimetre
9      if Out > NNhg then
10         LV = Out - Unit;
11     else
12         LV = Out + Unit;
           // If anomaly detection less than its
               nearest neighbour
13     end
14     Return LV
15     Call Refinement-model(LV, Att);
16 end
```

Algorithm 4 implements the knowledge refinement process, called in line number 15 of Algorithm 1. The refinement process is in charge of updating the KB, which holds instances values of the PDDL model. After a success value (*LV*) is learned from Algorithm 3 (*LV*), it is passed to Algorithm 4's refinement process to update the relevant value in KB. This sets the update as a temporary update to be confirmed or decline by next executions, as Figure 8 illustrates.

Refinement Rules: The number of KB instances to refine in each iteration is domain-specific. Despite AD-

Algorithm 4: Refinement Process Algorithm).

```
1  Refienment Process
2  while LV Exist do
       // If new success value is learned
3      P ← Operator
4      LV ← Learned-Value
5      Att ← Attribute name
6      Select (Att)
7      Update KB (Att, LV, P)
8      if Update is valid then
9          Temporary Update
10     else
11         Undo Update
           // (If LV∈ TD) and execution continued
               to fail, then reject update. check
               Figure 8
           // ( Check Figure 8)
12     end
13 end
```

HDJ can detect more than one anomaly the rational relationships of the domain model should be considered when refining its knowledge. Refining a specific knowledge attribute is specified based on the relationships between the attributes in the domain. This is due to the mutual dependency between the

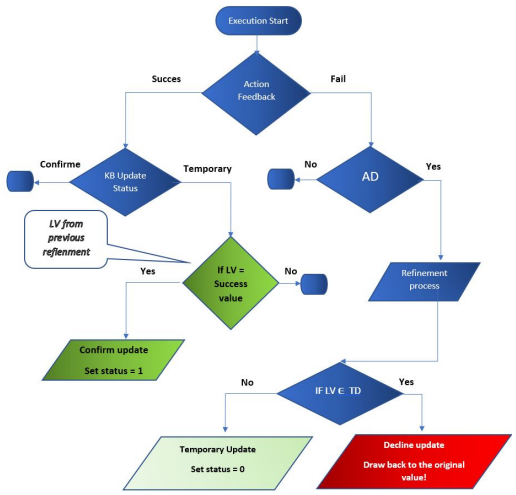

Figure 8: Refinement validation flowchart

action attributes. For example, In the grasping task of Nao robot, in the Evaluation Section, the 'distance' (X_{m1}) and the 'headyawangle' (X_{m2}) attributes are have an inverse relationship ((X_{m1}) $\propto$ (X_{m2})) where the 'headyawangle' values follows the 'distance' values. For this reason we selected the 'headyawangle' attribute to be refine it when appearing combained with the 'distance' attribute, as 'Collective' anomalies.

Initially, the refined knowledge is set as a temporary update and waits for confirmation to be set as permanent value. Figure 8, is a flowchart representing the process. The refinement model updates KB incrementally based on experience. The update is set as a temporary update to be confirmed as permanent when it is validated. The refinement will continue to process in iterations until successful execution is achieved. Each time the model reduces the gap between the outlier and its nearest neighbour in the TD, the new solution is validated before confirming the update as valid. The validation checks and compares LV against TD. For the current fail, if $LV \in$ TD update will be reject, draw back to last confirmed update.

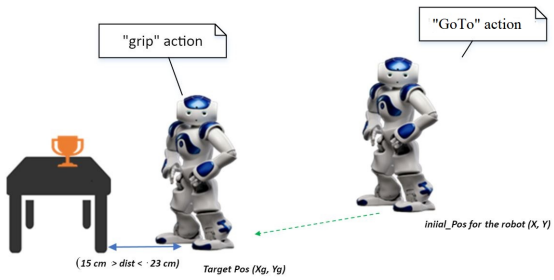

Figure 9: Kitchen Scenario.

Evaluation

A set of experiments have been designed to test the effectiveness of the HDJ architecture by exploring its behaviour within a kitchen scenario (Müller and Beetz 2006). We tested the effectiveness of HDJ software in reducing the fail executions percentage as a result of knowledge refinement. For this reason we performed and recorded a series of experiments without employing the HDJ system, using the NAO robot. Then, we performed another series of experiments employing the HDJ system.

As represented in Figure 9, the kind of tasks given to the robot are to do with gripping and moving a cup from a table to another one. Plans that will fail are executed in order to test the robot abilities in identifying the cause of execution failure and then to update and justify KB. Each of the experiments is used to investigate a particular type of anomaly:

- The distance between the robot to the cup.
- The angle of the position of the cup on the table to the robot position.
- 'Collective' anomalies, where the combination of attributes is detected as being the anomaly.
- 'Collective' anomalies, the detection of group of anomalies.

Experimental Setup

We use PDDLv2.1 for the Domain Model, as illustrated in the Listings 1. Given operator P in DM, PNE is a

primitive numeric expression (*fluent*) of a planning instance related to the precondition of *P*. The execution of Algorithm 1 detects the cause of a failure, learns success values *LV*, and then uses the learned value to update the KB, by replacing the original value of expression *PNE* with the learned value *LV*.

The functions **dist-to**, **mindis**, and **maxdis** are associated with the pre-conditions of the operator **grip**. These fluents are employed to represent the distance rule (**dis**). As seen in Listing 1 , we use a prefix syntax (comparison) for the **grip** operator and its precondition (>=(dist-to ?r ?waypoint) (mindis ?r ?waypoint)). Similar to the distance rule, the functions **hwangle** and **maxhwangle** are associated with the **HeadYawAngle** rule.

The values of fluents are passed and processed by HDJ system which passes them to the planner, specifically to the problem interface in ROSPlan, Figure 2 is an example of the PDDL problem file implemented our experiments. In contrast the value of dist-to is a sensory data (distance value) that measured online and it is used to update KB. For each failed execution, HDJ system implements Algorithm 1 to update and refine the knowledge base of the task planner to avoid future failure.

```
(define (problem task)
 (:domain NAO)
 (:objects
     wp0 wp1 - waypoint
     nao - robot
     grp - gripper
     redCup - object)
 (:init
      (at-robby nao wp0)
      (at redcup wp1)
      (free nao grp)
      (= (dist_to nao wp1) 20)
      (= (mindis nao wp1) 12)
      (= (maxdis nao wp1) 23)
      (= (hwangle nao) 0.0)
      (= (maxhwangle nao ) 0.2) )
(:goal (and
         (at-robby nao wp1)
         (carry nao redcup grp))
 ))
```

Listing 2: Problem File

Experiments Investigating Distance Failures: These experiments investigate distance as a cause of failure, in particular the distance between the robot and the cup. The environment is not changed for each execution (episode), but the initial position of the robot to the cup is randomly changed.

We intentionally set incorrect ranges for the distance rule to cause failure and investigate the effectiveness of the HDJ to correct the distance constraints. The correct range of (23cm>dis>15cm) was extended to range (27cm>dis>15cm) in order that some failures would be caused during execution.

Experiments Investigating Angle Failures: The available DOF of the arms of NAO robot limits its abilities in grasping. The head can rotate about yaw and pitch axes. Each arm has 2 DOF at the shoulder, 2 DOF at the elbow, 1 DOF at the wrist, and 1 additional

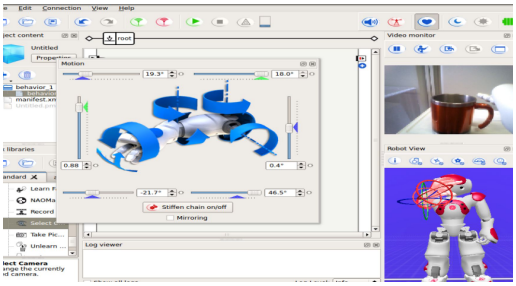

Figure 10: The joints values of NAO's arm in grasping position at the maximum degree toward the body.

DOF for the hand's grasping. The angle of the object's position is a critical condition for successful grasping. As seen in Figure 10, the maximum rotation of shoulder joint towards the body is only able to bring the arm in front of the robot's cameras. The ideal position for grasping an object is when the **HeadYawAngle**=0.0.

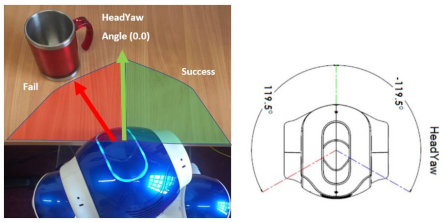

Figure 11: Success and fail ranges of 'HeadYawangle, to grip by the right hand of NAO robot.

In this experiment the environment and the initial position of the robot to the cup is not changed but the angle of the cup position on the table is randomly changed (see Figure 11). The distance to the cup was fixed to 18 cm. The **HeadYawAngle** rule was intentionally set wrong values. Instead of (0.0>HeadYawAngle>-25) we set it to (0.0>HeadYawAngle>-29), which leads to fail the execution.

Experiments Investigating Collective Failures: A 'Collective' anomaly is a group of attribute instances that appear isolated compared to the rest of the Dataset. A specific case of 'Collective' is when each single attribute is not an anomaly but as a group appears together as anomaly. For this reason, this experiment is set to test such cases. In DM, Listing 1, we have two preconditions that accept a range of values of the **Distance** and **HeadYawAngle** attributes. For successful grasping, our experiments have showed that each distance point accepts a specific range of the **HeadYawAngle**. As seen in Figure 12, when the distance increases, the **HeadYawAngle** range is narrowed and goes toward **0.0** angle. For example, the 15cm distance accepts the full range (0.0>HeadYawAngle>-25), for the right hand. How-

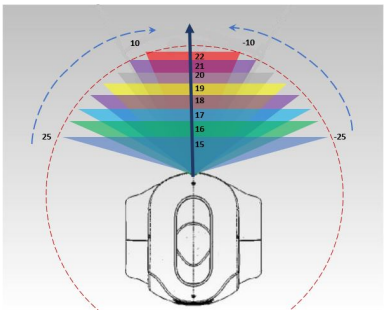

Figure 12: The relationship of the ranges of distance and angle attribute.

ever, the 20 cm distance accepts a shorter range of angle about (0.0>HeadYawAngle>-12).

For this reason, particular combinations (of the **Distance** and the **HeadYawAngle** attributes) cause execution failure. Even if each attribute by itself is not an anomaly but their combination forms such a specific case of 'Collective' anomalies.

In this experiment we set the ranges of the both rules **HeadYawAngle** and **dis** to (0.0>HeadYawAngle>-25) and (23cm>dis>15cm) in order. Those ranges are considered as the success range. However, we know that within those range a specific cross values will fail the execution. For each episode, we change both of the initial position of the robot to the cup, and the angle of the cup position on the table as well.

Experiments Investigating Failure of a Group of Anomalies This experiment tests the general case of 'Collective' anomalies, when detecting two or more attributes as being anomalies: In this experiment we randomly change the initial position of the robot to the cub but the angle of the cup position on the table is not changed (see Figure 11). The angle of the cup position fixed to -27.

We set the ranges of the both **HeadYawAngle** and **dis** rules to (0.0>HeadYawAngle>-27) and (25cm>dis>15cm) in order. These ranges are both incorrect and lead to failure during the execution. The correct range of (23cm>dis>15cm) was extended to range (25cm>dis >15cm). Same for the **HeadYawAngle**, instead of (0.0>HeadYawAngle>-25) we set it to (0.0>HeadYawAngle>-27), which leads to fail the execution.

Experimental Process and Results

The test results showed that the rate of failure decreases over time as the parameters are adjusted by HDJ system. The overall results of this test represented in Figure 13. Table 1 represents the accuracy of predictions made by robot.

The experiments demonstrate the efficiency of our approach in refining the knowledge base of task planner, in which reducing the plan execution failure for long term autonomy. For example, by the end of the

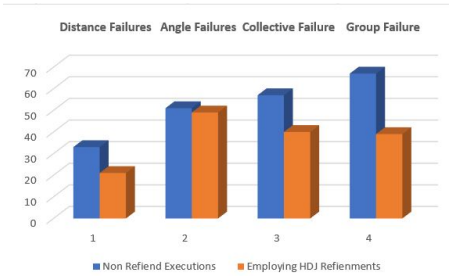

Figure 13: Field executions with and without knowledge refinement by HDJ

Attribute	Obs.	TP	TF	Preci.	Accu.	FNR	TPR
Distance	65	61	2	96.8%	93.8%	3.17%	96.8%
Angle	32	18	2	46.8%	44.1%	5.8%	44.12%
Collective	39	25	7	71.8%	64%	21.88%	82%
Collective (Group)	12	8	3	72.72%	66.6%	25%	75%

Table 1: The Overall Accuracy of Predictions by HDJ Reasoner

distance experiment, the HDJ refinement system successfully corrected the values of the distance rule. The instance value of the maximum allowed distance (maxdis ?r ?waypoint) was corrected to be 23 Cm, Figure 14 represents the refined DM while Figure 15 shows plan before and after refinements.

```
tion grip :parameters (?r -
      robot ?obj - object ?
      waypoint
point ?g - gripper)
econdition
 (and (at-robby ?r ?waypoint)
 (at ?obj ?waypoint)
 (free ?r ?g)
 (> (dist_to ?r ?waypoint) 15)
 (< (dist_to ?r ?waypoint) 27)
```

(a) The 'distance' rule before update

```
ction grip :parameters (?r -
      robot ?obj - object ?
      waypoint - waypoint ?g -
      gripper)
condition
 (and (at-robby ?r ?waypoint)
 (at ?obj ?waypoint)
 (free ?r ?g)
 (> (dist_to ?r ?waypoint) 15)
 (< (dist_to ?r ?waypoint) 23)
```

(b) The 'distance' rule after update

Figure 14: An example of PDDL domain before and after refinement. It is based on DM represented in Listing 1 where fluents are instantiated with numeric values. The 'maxdis' is a "constant" part of the problem description but since it is referenced in the preconditions of the 'grip' action, the refinement updates have changed the action preconditions.

They also demonstrate the efficiency of the update validation process, which was proved by the angle failure experiments. The validation process rejected all predictions made by reasoning system because they relayed on a faulty DM. The domain model neglects the rational relationship between the 'Distance' and the 'HeadYawAngle' attributes, where each distance point accepts specific range of angle values, which causes incorrect predictions by HDJ reasoner.

As the relationships between the operator's parameters have to be represented. The missing of such rep-

```
(goto nao wp0 wp2)
(grip nao redcup wp2 wp1 grp)
```

(a) The plan of failed execution.

```
(goto nao wp0 wp4)
(grip nao redcup wp4 wp1 grp)
```

(b) The plan of successful execution.

Figure 15: Given the plan rule (27cm>dis>15cm) which refined to be (23cm>dis>15cm). The plan B generated from refined, more accurate encoding of domain knowledge resulting in successful execution.

resentation can fail action executions. In particular, in the the angle failure experiment, the refinement process behaved like a domain validator, which is considered to be an extra advantage of using AD. The refinement process uncovers incorrect predictions that were caused by an inaccurate DM that neglects the relationship between the distance and the angle attributes. This lead to expose the fault in our DM.

In another words, our experiments demonstrates that accurate knowledge can overcome issues related to inaccurate domain models. However, Despite this fault, the HDJ (specifically in the Collective' anomalies experiments) was able to refine the range of the angle ('HeadYawAngle') rule. It successfully set the correct range for each distance value of the distance ('dis') rule. This because the HDJ database considers the relationship between these attributes.

Furthermore, the specific case of 'collective' anomaly (when only the combination of attributes is the anomaly and each single attribute by itself is not an anomaly), reveals a mutual relationship between the attributes in which such a type of anomaly can be used to develop a tool to discover such hidden relationships in domain models. Employing the 'collective' anomaly detection type gives HDJ an ability to discover hidden relationships that the current domain model lacks. So, it can indicate faults in the domain representation.

Moreover, employing the 'collective' anomaly detection method to over come the model proved that employing multiple method of anomaly detection is beneficial and can overcome the problems or weaknesses of each method.

The last experiment demonstrates that the refinements system implements the RPA rules as it selects one anomaly if a group of anomalies are detected. Furthermore, the experiments also demonstrate that real application brings the complications related to the real environment and sensors issues. The experiments indicated that Nao measures the distance to objects is not accurate enough at the moment, which is the main source of the false predictions made by NAO in our experiments. The low resolution of the NAO robot's camera (Zhang et al. 2019). In addition to the hardware specifications issue, the method that estimates the distance, is another factor of inaccurate distance measures. Different techniques for depth estimation are exist in which each has different outcomes and drawbacks.

The inaccuracy of the distance data leads to producing overlapped data edges between the successful executions data and the fail data, as seen in Figure 16. For example the distance value '23 cm' appears in both datasets. it appears as successful attribute value in TD but in same time same value can fail the execution. It appearance in both sets prevents the robot making successful predicates, as it is not considers as anomaly in TD. Figure 17 represents the predictions made by human and robot.

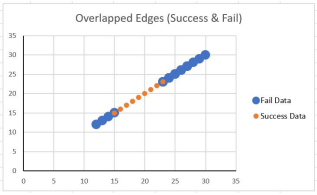

Figure 16: The distance datasets.

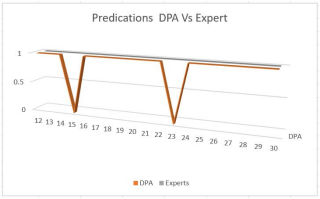

Figure 17: Graphical representation of the correct predictions made by the robot, in the distance experiment.

Conclusions

In this paper, we presented the concept of using anomaly detection to find the cause of action failure of a real robot, in order to improve the knowledge of a robot's task planner. We have demonstrated HDJ to be capable of autonomously detecting an anomaly, and refining knowledge to overcome it, so that in future plans a robots plan are less likely to fail. We can conclude that HDJ contributes to bridging that gap between planning and execution. The existence of a variety of AD techniques are great beneficial to our methodology. It opens the door for a variety of implementations of different planning problems and action models, and it increases the automation aspects of our reasoning methodology. It offers a high level of flexibility because it is not limited to a single AD approach, and we can implement more than one approach as the latest trend in outer detection systems is the utilising of multiple techniques; this to overcome the deficiencies and weaknesses of each approach and to exploit the advantages of each technique. In the future we plan to extend the types of knowledge that can be refined by

HDJ, and perform more experimentation involving a range of scenarios.

References

Ando, S.; Thanomphongphan, T.; Hoshino, D.; Seki, Y.; and Suzuki, E. 2011. Ace: anomaly clustering ensemble for multi-perspective anomaly detection in robot behaviors. In *Proceedings of the 2011 SIAM International Conference on Data Mining*, 1–12. SIAM.

Bailey, T., and Durrant-Whyte, H. 2006. Simultaneous localization and mapping (slam): Part ii. *IEEE Robotics & Automation Magazine* 13(3):108–117.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. Rosplan: Planning in the robot operating system. In *ICAPS*, 333–341.

Crook, P. A.; Marsland, S.; Hayes, G.; and Nehmzow, U. 2002. A tale of two filters-on-line novelty detection. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 4, 3894–3899. IEEE.

Crook, P.; Hayes, G.; et al. 2001. A robot implementation of a biologically inspired method for novelty detection. In *Proceedings of the Towards Intelligent Mobile Robots Conference*.

Doherty, P.; Kvarnström, J.; and Heintz, F. 2009. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Autonomous Agents and Multi-Agent Systems* 19(3):332–377.

Gil, Y. 1992. Acquiring domain knowledge for planning by experimentation. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE.

Glover, S. 2004. Separate visual representations in the planning and control of action. *Behavioral and brain sciences* 27(1):3–24.

Häussermann, K.; Zweigle, O.; and Levi, P. 2015. A novel framework for anomaly detection of robot behaviors. *Journal of Intelligent & Robotic Systems* 77(2):361–375.

Hornung, R.; Urbanek, H.; Klodmann, J.; Osendorfer, C.; and Van Der Smagt, P. 2014. Model-free robot anomaly detection. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3676–3683. IEEE.

Ingrand, F., and Ghallab, M. 2017. Deliberation for autonomous robots: A survey. *Artificial Intelligence* 247:10–44.

Jiménez, S.; De la Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *The Knowledge Engineering Review* 27(4):433–467.

Jiménez, S.; Fernández, F.; and Borrajo, D. 2013. Integrating planning, execution, and learning to improve plan execution. *Computational Intelligence* 29(1):1–36.

Karapinar, S.; Altan, D.; and Sariel-Talay, S. 2012. A robust planning framework for cognitive robots. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Khalastchi, E.; Kalech, M.; Kaminka, G. A.; and Lin, R. 2015. Online data-driven anomaly detection in autonomous robots. *Knowledge and Information Systems* 43(3):657–688.

Kunze, L.; Hawes, N.; Duckett, T.; Hanheide, M.; and Krajník, T. 2018. Artificial intelligence for long-term robot autonomy: A survey. *IEEE Robotics and Automation Letters* 3(4):4023–4030.

Lindsay, A.; Franco, S.; Reba, R.; and McCluskey, T. L. 2020. Refining process descriptions from execution data in hybrid planning domain models. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 469–477.

Müller, A., and Beetz, M. 2006. Designing and implementing a plan library for a simulated household robot. In *Cognitive Robotics: Papers from the AAAI Workshop, Technical Report WS-06-03*, 119–128.

Murphy, K. P. 2012. *Machine learning: a probabilistic perspective*. MIT press.

Park, D.; Hoshi, Y.; and Kemp, C. C. 2018. A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder. *IEEE Robotics and Automation Letters* 3(3):1544–1551.

Ranasinghe, N., and Shen, W.-M. 2008. Surprise-based learning for developmental robotics. In *2008 ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems (LAB-RS)*, 65–70. IEEE.

Rockel, S.; Neumann, B.; Zhang, J.; Dubba, K. S. R.; Cohn, A. G.; Konečnỳ, Š.; Mansouri, M.; Pecora, F.; Saffiotti, A.; Günther, M.; et al. 2013. An ontology-based multi-level robot architecture for learning from experiences. In *Designing Intelligent robots: Reintegrating AI II. AAAI Spring Symposium-Technical Report*, 52–57. AAAI Press.

Schmidt, R. A. 1975. A schema theory of discrete motor skill learning. *Psychological review* 82(4):225.

Stulp, F., and Beetz, M. 2008. Refining the execution of abstract actions with learned action models. *Journal of Artificial Intelligence Research* 32:487–523.

Upal, M. A. 1999. Learning rewrite rules to improve plan quality. In *AAAI/IAAI*, 984.

Wang, X. 1996. *Learning planning operators by observation and practice*. Ph.D. Dissertation, Carnegie Mellon University.

Weber, B. G.; Mateas, M.; and Jhala, A. 2012. Learning from demonstration for goal-driven autonomy. In *AAAI*.

Zhang, L.; Zhang, H.; Yang, H.; Bian, G.-B.; and Wu, W. 2019. Multi-target detection and grasping control for humanoid robot nao. *International Journal of Adaptive Control and Signal Processing* 33(7):1225–1237.

Zimek, A., and Schubert, E. 2017. Outlier detection. In Liu, L., and özsu, M. T., eds., *Encyclopedia of Database Systems*, 1–5. New York, NY: Springer New York.