Improving Task Planning Knowledge Robustness for Autonomous Robots

Hadeel Jazzaa[a,d], Thomas McCluskey[a] and David Peebles[b]

[a]School of Computing and Engineering, University of Huddersfield, Huddersfield, UK [b]School of Human and Health Sciences, University of Huddersfield, Huddersfield, UK

ARTICLEINFO

Keywords:
Autonomous Robots
Anomaly detection
Reasoning
Task planning knowledge knowledge refinement

ABSTRACT

The requirement for autonomous robots to exhibit higher-level cognitive skills by planning and adapting in an ever-changing environment is indeed a great challenge for the AI community. This paper demonstrates how a physical robot can be capable of adapting its symbolic knowledge of the environment, by using experiences in robot action execution to drive knowledge refinement, and hence to improve the success rate of the task plans the robot creates. We propose a method for refining domain knowledge, encoded in the PDDL language, based on a novel method in reasoning using anomaly detection techniques. The approach used in this paper is to extend a high-level planning platform (ROSPlan) to help create more robust planning systems to improve the knowledge on which intelligent robot behaviour is based. Empirical testing and evaluation has been performed using the NAO robot in a kitchen scenario. The results show that the robot system refines its knowledge in line the result of its experience of using plans that it creates to achieve tasks. The refined knowledge leads to the future synthesis of task plans which demonstrate decreasing rates of failure over time as faulty knowledge is removed or adjusted.

1. Introduction

Autonomous robots operating in partially known and dynamic environments (e.g., exploration robotics) require systems to create and execute plans deliberately [1]. Gathering knowledge, and acting on that knowledge, in such environments is a major demand for robust autonomous robot systems.

A robot may encounter a range of failures while executing its planned actions. The plan execution strategy must account for the action/plan failure, which results from ignorance or change [2]. The robot could overcome such failures by gathering updated information about its current environment, and replanning to generate a new plan. However, in some cases re-planning does not solve the problem because the robot's knowledge of the conditions and effects of its own actions on the environment may be faulty. Equipping autonomous robots with reasoning mechanisms can be useful when dealing with such issues. To overcome failure, the robot must reason about the cause of the failure to refine its knowledge using his experience. When execution of an action in a plan fails, the robot must realise why it has happened. Then, to avoid any future failure, an update to its knowledge should be made by the refinement system.

Systems with pre-engineered knowledge of their environment often provide an abstraction hierarchy of types of knowledge for domain experts to use in crafting knowledge. At an abstract level parameterised actions are key to an efficient approach in bridging the gap between the abstract plan and action execution. Actions include parameters that accept changed values to enable the implementation of the same action in different situations [3, 4]. For example, the action GoTo(Pos) is more flexible to program than GoTo-Centre, where 'Pos' is a parameter that accepts different values, including the centre [3].

But however much hand-crafted engineering is facilitated, incomplete action models that are missing some aspects may cause action execution failure and result in failures in achieving tasks. Hence, ultimately, plan construction and execution can only be successful through a *combination* of human-specified and robot learned knowledge [3] on which behaviour is based. For example, incorrectly specified actions requiring new attributes or relationships, in the face of changing environment needs to be updated regularly without relying always on human hand-crafting. Additionally, an engineer may miss preconditions and/or effects or correct knowledge, or entire operators may be absent from the model [5]. Our work furthers the hypothesis that accurate models can be learned online while data gathering during operation [6]. This is aligned with the move towards long-term autonomy [7], in that making the system more robust through experiential learning will increase its effective life. Here failures and successes drive the improvement of pre-specified domain knowledge by

hadeel.jazzaa@hud.ac.uk (H. Jazzaa); t.l.mccluskey@hud.ac.uk (T. McCluskey); d.peebles@hud.ac.uk (D. Peebles)

ORCID(s): 0000-0000-0000-0000 (H. Jazzaa); 0000-0000-0000-0000 (T. McCluskey); 0000-0003-1008-9275 (D. Peebles)

connecting the robot architecture's abstraction levels in order to investigate the cause of execution failure and use this to drive updates to the robot's knowledge as a result.

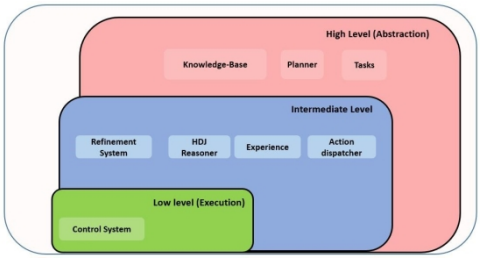

Figure 1: HDJ Intelligent Robot Hierarchical Architecture

To do so, we developed the HDJ robotics architecture (shown in Figure 1) which is based on a "conventional" task planning architecture. This contains knowledge of its environment encoded within a knowledge base referred to as KB below. The KB contains important components such as facts representing the environment's state, and expressions forming the robot's model of its action capabilities, referred to as the "domain model". HDJ but has extra components that reason with experience-based data, leading to the refinement of the KB held in a PDDL model. The HDJ software includes the anomaly detection and knowledge refinement subsystem called ADKRA, which manages the creation of learned values used to update the KB. The robot's operational experience is stored in datastores, which constitutes the ADKRA's training data (TD). TD is the history of successful executions, typically maps of action model parameters, and contains all the relevant information required in the implementation of ADKRA. The novel aspects of our contribution include the method for anomaly detection (AD) and subsequent knowledge refinement performed by Algorithm 1, detailed below.

Anomaly detection involves identifying data that deviates from the norm and raises concerns about a particular issue or problem [8]. It has been utilised in various fields such as detecting bank fraud, identifying medical issues, monitoring systems, detecting ecological disruptions, ensuring network security, and recognising events [9, 10]. Outliers can present themselves in different forms, and the anomaly detection survey [11] highlights several anomaly detection methods based on different categories, mainly point anomalies, contextual anomalies, and collective anomalies. Several anomaly detection approaches exist, including density-based techniques, machine learning methods, and neural network approaches. Additionally, multiple anomaly detection techniques can be used on the same dataset. The latest trend in anomaly detection systems is the use of hybrid techniques, which combines multiple methods to overcome the weaknesses and limitations of each approach and maximise the benefits of each one [9].

The refinement activity starts after execution has failed and the feedback of the failed execution has been sent to the KB. Action parameters and all related information are crucial to our approach, so the focus of this research is to use the execution of actions as the basis of learning, initiated by action plan failure. The focus is on developing a reasoning method that enables the robot to recognise the cause of action failure (the anomaly), when it occurs, and to use the anomaly to direct autonomous refinements and knowledge correction. The refinements then make future generated plans more likely to succeed.

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 [12, 13]. 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 [12], 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.

This paper's contributions are summarised as follows:

- The detailed description of the anomaly detection and knowledge refinement algorithms (ADKRA) which identify and repair faulty taskoriented domain knowledge in a robot that has previously led to execution failures;
- An empirical evaluation of HDJ when using an implementation of the ADKRA within a realworld humanoid robot.
- An overview of the HDJ architecture which is capable of operating the robot and its repair components.

We have experimented with an implementation of HDJ using the domain and running example explained in next section, Section 2. Section 3 presents an overview of research related work. Section 4 describes the HDJ hierarchical system and its framework, the system mechanism and algorithms. Section 5 describes the evaluation of the HDJ system through designing a series of experiments. Finally, this article concludes in Section 7.

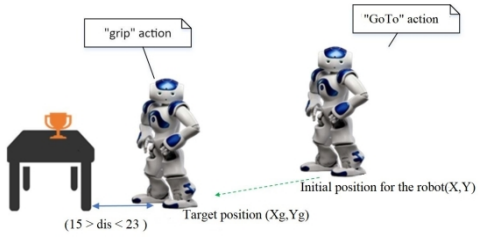

Figure 2: Kitchen scenario: gripping task.

Figure 3: Successful execution: the NAO robot can reach for the cup.

2. Running Example

We will use a running example, based on the kitchen scenario [14], to motivate, illustrate and evaluate the techniques introduced. As shown in Figure 2, the tasks performed by the robot included gripping a cup and moving it from one table to another. As
Figure 3 illustrates, a successful execution of the 'grip' operator is when the robot can carry out its actions; in this specific case, this means the robot can reach for and grip the cup.

The grasping task requires several functions, such as vision recognition, object detection, distance calculation and motion design. Distance measurement is of great importance in this example. The accuracy of the distance data affects the performance of the robot, which may lead to execution failure.

In Figure 2, we can see that for a successful execution, the distance between the NAO robot and the table must be from 15 to 23 centimetres. If this range is specified incorrectly, e.g., 15 to 27 centimetres, it will result in some failed executions. Hence, if the robot is equipped with an autonomous reasoning function that enables it to realise this fault in the KB and justify and correct the model's description, this will make for more robust and longer term autonomy.

In this research, we use the NAO robot in our experiments. The NAO robot can learn, recognise and track objects, as well as perform various motion tasks.

To perform the grasping task, the planning system dispatches two actions,'goto' and 'grip', to the controller of the NAO robot. Domain model definitions and action parameters, residing in the KB, are provided in the following subsections.

2.1. Action Parameters and Attributes

Action parameters refer to a data vector of all information related to an action. Let $X = x_1,...,x_n$ where n is the number of attributes of the action.

X1	X2	X3	X4	X....	Xn
10	17	14	22	...	15
11	10	15	16	...	16
12	10	16	17	..	17
13	16	14	22	...	20
..	..	..	..	..	..
X_{m1}	X_{m2}	X_{m3}	X_{m4}	X_m...	X_{mn}

Figure 4: Action experience table

For example, the action 'grip' has n number of attributes. The distance between the robot and the table represents its first attribute (X_1), while (X_2) is the angle attribute that represents the position of the targeted object. Furthermore, the relationship between the 'distance' and the 'angle' attributes is another factor, a correlated attribute [15], must be considered as an attribute of the grip action that can be represented as (X_3). These action parameters can be learned and justified based on online execution. Figure4 shows an action table where each record is the value of the action attributes for each execution.

2.2. The PDDL Model

To be able to do Task Planning in our architecture, we utilise the PDDL description language v. 2.1 [16], a simple (non-temporal) PDDL variant. The PDDL model of the gripping task is represented in Figure 5. Figure 6 shows the task problem.

For task planning, a task is defined to be a pair of a domain model, and a problem instance of the form *(Os, Init, G)*. This is a triple consisting of the objects *Os* in the domain model, the initial state *Init* and the goal *G* specification. Given operator *P* in the domain model, each *P* consists of a pair *(pre(P), efc(P))* where *pre(P)* represents the action preconditions, and *efc(P)* represents the effects of the action.

In the domain model, shown in Figure 5, the functions **dist-to** and **hwangle** are associated with the pre-conditions of the operator **grip**. These fluents are employed to represent the distance rule (**dis**) and the angle rule (**HeadYawAngle**).

Incorrectly specified preconditions will cause the action to fail. This, in turn, will affect the constraints on

the ‘goto’ action preceding. Doing so could lead to failure to execute a seemingly valid abstract plan.

```
( define ( domain NAO)
( : requirements : s t r i p s : typing : f l u e n t s )
( : types waypoint thing robot gripper )
( : predicates
( atrobby ? r – robot ?x – waypoint )
( pos ?o – thing ?x – waypoint ) ( f r e e ? r –
robot ?g – gripper )
( carry ? r – robot ?o – thing ?g
– gripper ))

( : functions
( dist_to ?x1 – waypoint ?x2 – waypoint )
( maxdis ? grp – gripper )
( mindis ? grp – gripper ) ( hwangle ?
r – robot )
( maxhwangle ? r – robot )
( minhwangle ? r – robot ))

( : action goto
: parameters (? r – robot ?from – waypoint ? to
– waypoint )
: precondition ( and
( atrobby ? r ?from ))
: e f f e c t ( and
( atrobby ? r ? to )
( not ( atrobby ? r ?from ) ) ) )

( : action grip
: parameters (? r – robot ? obj – thing
? waypoint1 – waypoint
? waypoint2 – waypoint ?g – gripper )

: precondition ( and
( atrobby ? r ? waypoint1 )
( pos ? obj ? waypoint2 )
( f r e e ? r ?g)
(> ( dist_to ? waypoint1 ? waypoint2 )
( mindis ?g ))
(< ( dist_to ? waypoint1 ? waypoint2 )
( maxdis ?g ))
(< ( hwangle ? r ) ( maxhwangle ? r ))
(> ( hwangle ? r ) ( minhwangle ? r )))
: e f f e c t ( and
( carry ? r ? obj ?g)
( not ( f r e e ? r ?g ) ) ) ) )
```

Figure 5: A segment of the KB’s PDDL Model implemented in our experiments

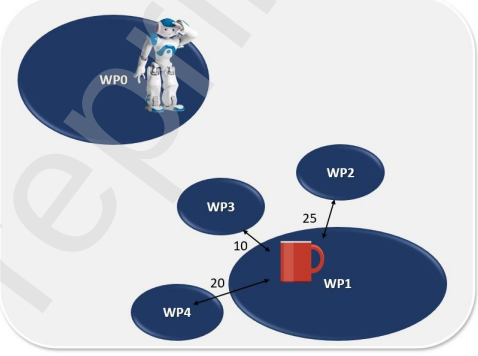

Figure 6: Problem representation of the gripping task.

Our example illustrates the possibility that plan rules and pre-conditions could be updated to avoid failure of future executions. This can be done by repairing incorrectly specified actions and recognising new attributes or relationships [17].

```
( define ( problem task )
( : domain NAO)
( : objects
wp0 wp1 wp2 wp3 wp4 – waypoint
nao – robot grp –
gripper
redcup – thing )
( : i n i t
( atrobby nao wp0)
( pos redcup wp1)
( f r e e nao grp )
(= ( dist_to wp2 wp1) 25)
(= ( dist_to wp3 wp1) 10)
(= ( dist_to wp4 wp1) 20)
(= ( mindis grp ) 15)
(= ( maxdis grp ) 27)
(= ( hwangle nao ) 0.0)
(= ( maxhwangle nao ) 0.2) )
( : goal ( and
( carry nao redcup grp ) ) ) )
```

Figure 7: The problem file of failed execution

```
( define ( problem task )
( : domain NAO)
( : objects
wp0 wp1 wp2 wp3 wp4 – waypoint
nao – robot grp –
gripper
redcup – thing )
( : i n i t
( atrobby nao wp0)
( pos redcup wp1)
( f r e e nao grp )
(= ( dist_to wp2 wp1) 25)
(= ( dist_to wp3 wp1) 10)
(= ( dist_to wp4 wp1) 20)
(= ( mindis grp ) 15)
(= ( maxdis grp ) 23)
(= ( hwangle nao ) 0.0)
(= ( maxhwangle nao ) 0.2))
( : goal ( and
( carry nao redcup grp )) )
```

Figure 8: The problem file of successful execution (after knowledge refinement)

```
;  Cost :  0.001
;  Time  0.00
0.000:  ( goto  nao wp0 wp2)  [ 0 . 0 0 1 ]
0.001:  ( grip  nao  redcup  wp2 wp1  grp )  [ 0 . 0 0 1 ]
```

Figure 9: The plan generated from faulty domain knowledge resulting in execution failure.

Figures 7 and 8 show the problem file before and after refinements that repair incorrect parts of the model. The constraint (27cm>dis>15cm) is refined to

```
;       Cost :      0.001
;       Time      0.00
0.000:          ( goto       nao wp0 wp4)      [ 0 . 0 0 1 ]
0.001:          ( grip     nao     redcup      wp4 wp1       grp )  [ 0 . 0 0 1 ]
```

Figure 10: The plan generated from refined, more accurate encoding of domain knowledge resulting in successful execution.

be (23cm>dis>15cm). The plan generated with the faulty constraint (which leads to execution failure), is presented in Figure 9. While the plan generated with the refined constraint (which leads to execution success) is presented in Figure 10.

3. Related Work

The related work in this paper covers diverse areas of research, with a focus on the most relevant previous work. It is divided into four main parts. The first part is a brief survey of anomalies in general robotics applications. The second part is a brief survey of related works that perform refinements of the planning domain based on anomalies but using different methods than the one implemented in this research. The running example, explained in Section 2, of this paper combines task and motion planners to execute the plan. For this reason, the third part covers related works that implement task and motion planning approaches. Finally, the fourth part presents planning systems that deal with numeric constraints, similar to those used in the evaluation presented in Section 5.

While many existing robotics techniques use anomaly detection (AD) for condition monitoring, fault detection, novelty detection in the environment, and identifying anomalous behaviors of robots, there is little evidence that AD 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. AD has been applied to condition monitoring and fault detection in robots in previous works, such as [18], [19], and [20]. Novelty detection in the environment has been achieved using AD, as shown in the works of [21] and [22]. In addition, AD has been used to learn differences in object classification, as demonstrated in [7], and to identify anomalous behaviours using time scale and resolution, as in [23]. Robot-assisted feeding in healthcare applications has also utilised AD, as described in [24]. Finally, [25] modeled the spatiotemporal dynamics of the environment's continuous processes through its frequency spectrum using AD.

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, and, even fewer are demonstrated in robotics [26]. Indeed, many exist works perform the refinement of a planning domain or knowledge. Many of these works have concerned detecting the cause of failure as a method to drive change. However, these works often use different methods and are limited to specific functions that differ from the method proposed in this paper. For instance, LIVE [27] employs prediction rules and the incremental enlargement heuristic to identify relation differences in faulty rules, while EXPO [5] uses the ORM method to refine incomplete planning knowledge. The AUV [28] employs a progression algorithm to detect execution failure, but our architecture focuses on inferring the cause of action execution failure. Similarly, CRAM framework [29] employs plans in specific languages and a knowledge processing system, and GDA [30] and OBSERVER [31] use learning from demonstration. Our proposed method uses AD as a reasoning method to learn from failed executions, which is different from PELA [32] that learns probabilistic knowledge about the success of actions and predictions of execution dead-ends. RACE system [33] employs HTN planner and semantic descriptions, while HDJ experience is stored in a rational database that records action execution information. ProbCog [34] uses logic programming and geometric reasoning, whereas our method employs AD for reasoning. Other works, such as [35], refine hybrid domain models using ML techniques, and PBR system [36] employs conflicting choice points to refine plans. The work in [3] predicates the effects and performance of planned actions before execution, and STRANDS project [37] monitors system behavior and navigation for long term autonomy. Additionally, [38] developed PLEXIL, an execution language for richer representations of executable plans, and [39] proposed the PNPs programming language for robot and multi-robot behavior design. Learn PNP tool [40] combines Petri Net Plans formalism with reinforcement learning, and [41] proposes an approach for scenarios that involve teams of humans and robots with parallel planning and dispatching features to reduce re-planning waiting times.

Several works have been proposed to address the problem of planning with geometric constraints using task and motion planning approaches. One such work is the HPN hierarchy [42], which employs an A* search to generate low-level plans while ensuring optimal and valid conditions. It constructs plans at an abstract level and generates sub-goals recursively backwards from the goal, allowing for efficient planning. Another approach, the hybrid approach to motion, manipulation, and task planning [43], deals with planning knowledge involving multiple robots and objects by considering the static geometric environment and incrementally synthesising environment constraints during planning when needed. It

allows the task planner to operate in an abstract state space

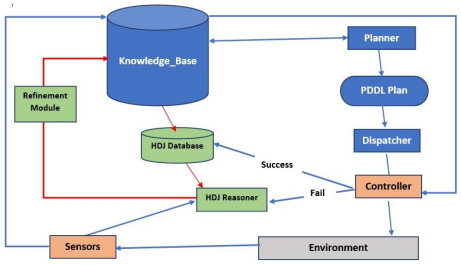

Figure 11: HDJ data flow diagram. The existing components (Blue boxes) and HDJ components (Green boxes)

without considering geometry constraints. Additionally, [44] proposed an approach that integrates off-theshelf task planners and motion planners with minimal assumptions on each layer. This approach requires failures to be expressed and represented in logical predicates at the task level.

In our domain model implementation (Section 2), we utilised numeric constraints. Several planning systems also use numeric constraints, such as cqScotty [45], a heuristic forward search planner that reasons with control variables using SOCP. Another approach is the compilation of the full PDDL+ language into SMT [46], which is designed to model mixed discrete continuous planning domains and process the set of PDDL+ features. This encoding is based on the SAT encoding of planning problems by [47] and [48]. Metric-FF [49], a domain-independent planning system, is an extension of the algorithms used in the heuristic planning system FF for linear tasks. It deals with PDDL 2.1 level 2 combined with ADL, which allows for a finite number of numerical state variables. Lastly, POPF [50] is a forward-chaining temporal planner built on grounded forward search and linear programming. POPF can handle domains with continuous linear numeric change and effects dependent on the duration of actions and supports a significant portion of PDDL 2.1 level 5.

4. 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 [28]. 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 11 presents the data flow through HDJ. It contains original components as well as existing components - in particular parts of the ROSPlan [2] 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 11).

As seen in Figure 11, if the feedback coming from the control system returns 'execution failed', the HDJ reasoner combines the information of the failed execution with the TD extracted from a successful execution. The reasoner suspects the anomalies of causing the failure. It employs Algorithm 2 to extract the anomalies. The refinement model employs Algorithm 3 and Algorithm 4. It uses that cause/s to learn new success value and uses these values to refine the KB. This is to avoid a future failed execution for the same situation context

4.1. The Algorithms in HDJ

Algorithm 1 is the top-level algorithm which collects data and drives the changes to HDJ's knowledge (see Figure 11). 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 KB by executing Algorithm 4, which updates the KB. Any detected anomaly is a value that indicates parameters or pieces of information in the KB, such as pre-conditions and states. Future failure can be avoided by updating the knowledge of the task's planner.

4.2. 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 *mxn* matrix where the columns denote *n* attributes and the rows maintain the values of these attributes over *m* execution, check action parameters in our running example Section 2.

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 13, if we recorded 100 successful execution, then (*m*) will take any value between 1 and 100.

Algorithm 2 embodies our approach to AD. It is the problem of searching for patterns in data that differ and raise suspicions about a specific problem or issue [8]. We use the same technique to compare the values of the

Algorithm 1: Anomaly Detection and Knowledge Refinement Algorithm

```
Data: Training data, failed action information
Result: KB Update
// Pseudo Code:
while plan execution do
    if not (Feedback.Success) then
        TD ← Training-Data
        FD ← Failed-Data
        Anomaly Detection(FD, TD)
            // Implementation of Algorithm 2
        if QueryResult.count > 0 then
                // If Outliers exists
                Out ← Detected Outlier
                Return Out
            Learning-Processing (Out)
                // Learn new success values, Algorithm 3
                LV ← Learned Value
                Return LV
            Refinement-process (LV);
                // Refine the KB, Algorithm 4
        end
    else
        SD ← Successful action Data
        Add-Training-Data (SD);
    end
end
```

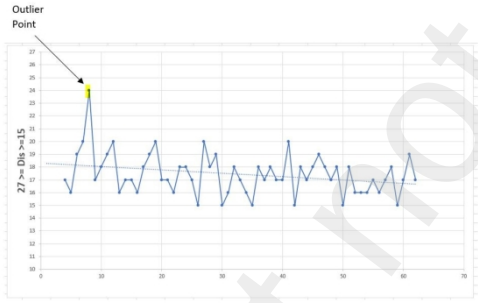

Figure 12: AD of a failed execution. The outlier 'distance' value appears far a way from its nearest neighbour in TD

failed action execution with the training data. The purpose is to discover the suspicious values in the failed execution, focussing on the outlier values which are often responsible for that failure.

A standard approach for AD in datasets is to create a normal data model and compare/test records against normality [8]. 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 AD approach, the failed execution record is compared and tested against records in TD by

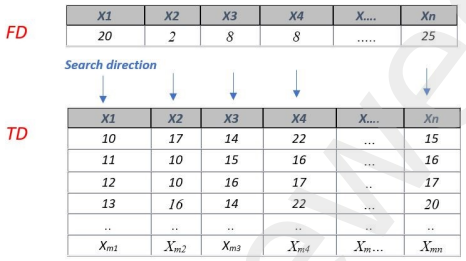

Figure 13: Algorithm 2 search. Each attribute instance (*X*) of *FD* is tested against its *X* vector in TD. *n* is number of attributes. *M* number of records in TD extracting the anomalous values of the action execution features. In our running example, Section 2, the grip action draws a profile of the distance attribute's history, Figure 12, where the outer point is the distance feature value of the failed action.

Algorithm 2 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 represent the action attributes. The failed execution information (FD) is a record of a failed complete action execution that is given during operation. FD is an input vector $x = x_1,...,x_n$ where n is number of attributes of FD. The online AD problem is to decide for each given x, whether or not x is anomalous with respect to TD. Our approach is to test each attribute of FD and search for the same attribute that does not appear in TD. As seen in Figure 13, Algorithm 2 considers expressions of form X, it seeks the value of X in FD that does not appear in the corresponding distribution of TD. Each action attribute is represented by a vector X where X is an m * 1 matrix where m is the number of successful executions recorded in the 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

```
1 Open TD Data-set;
2 for i ← 1 to n do
3     for j ← 1 to m do
4         Select FD-X(i) not-in TD-X(j,i) ;
5     end
6     if QueryResult.count > 0 then
          // if an anomaly detected
7         Insert into report_table FD-X(i).value,
            FD-X(i).label;
8     end
9 end
```

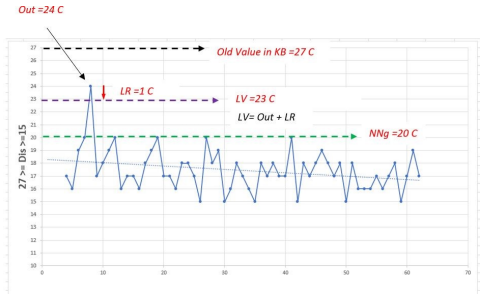

Figure 14: Learning process: The calculation of the new value *LV* for the 'dis' rule.

Figure 15: Failed execution with anomaly detection. 'HeadYawAngle' value is less than its nearest neighbour(*Out < NNhg*)

4.3. The Refinement 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 value in the KB) 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 [51]. Figure 14 is a visualised representation of LP for 'dis' rule.

In the presented example, in Figure 14, the *Unit* size is **one** centimetre; This because *Out* is a distance value that is measured by the centimetre unit.

The direction in which the step is taken is determined by the comparison of *Out* and *NNhg*. Algorithm 3 considers two comparisons of outliers. The first comparison is represented in Figure 12, and Figure 14 where the outlier value is greater than its nearest neighbour. While the second comparison type is represented in Figure 15 where the outlier value is less than its *NNhg*. Figure 16 shows how to specify *LV* for the two types of anomaly in the 'distance' attribute.

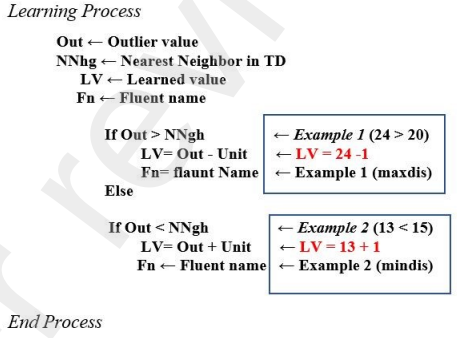

Figure 16: learning process for two type of anomalies in the 'distance' attribute. *LV = Out - Unit* this when *Out > NNhg*. while if *Out < NNhg*, *LV = Out + Unit*
.

Algorithm 3: Learning Process Algorithm

```
    // LV: Learned value.
1 if  report_table.count > 0 then
      // If Outliers exists
2     while !report_table.EOF do
          // While not end of file
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         NNhg ← Nearest-Neighbour in TD
7         Unit ← One measurement unit
          // For example one centimetre
8         if Out > NNhg then
9             LV = Out - Unit;
10        else
11            LV = Out + Unit;
              // If anomaly detection less than its nearest
                 neighbour
12        end
13    end
14 end
```

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 14 and Figure 16. Then *LV* will be passed to *Refinement-process* to refine the related value in KB.

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 language. 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

Algorithm 4: Refinement Process Algorithm.

```
// LV: Learned value
// Att: Attribute name
// P: Action (operator)
1  while LV Exist do
       // If new success value is learned
2      P ← Operator
3      LV ← Learned-Value
4      Att ← Attribute name
5      Select (Att)
6      Update KB (Att, LV, P)
       // Temporary Update the KB
7  end
```

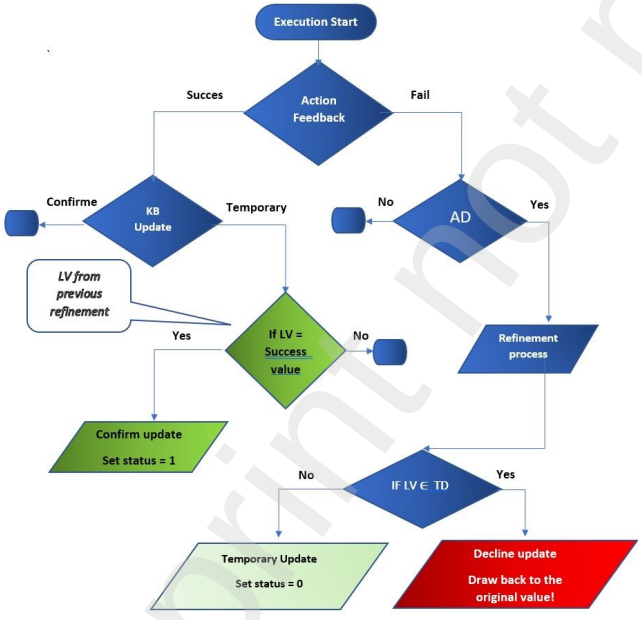

Figure 17: Refinement validation flowchart

as a temporary update to be confirmed or decline by next executions, as Figure 17 illustrates.

Refinement Rules

The number of KB instances to refine in each iteration is domain specific. Although *Algorithm 2* 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 action attributes. For example, in the grasping task of Nao robot, Section 2, the 'distance' (X_{m1}) and the 'headyawangle' (X_{m2}) attributes are have an inverse relationship ((X_{m1}) (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 17, 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* into *TD* update will be reject, draw back to last confirmed update.

5. Evaluation

A set of experiments has been designed to test the effectiveness of the HDJ architecture in dealing with a flawed model of its environmental capabilities. We explore its behaviour by implementing our running example, explained in Section 2. Solutions to robotic tasks are generated using an automated planning engine, and executed in the real world. We will assume that the robot's model is sufficiently developed to allow the generation of some plan to achieve a task (there has been much research and related techniques available in the automated planning area on fixing flawed or partial models in order to generate plans. Here plan generation is considered generally *without* taking into account plan execution in a real scenario; e.g. see the review paper [26]). In our experiments, the model containing faulty domain knowledge means some of the plans generated will fail when they are executed. We test the effectiveness of HDJ software in reducing the rate of failed executions as a result of knowledge refinement, and can claim a baseline success when this measure reduces monotonically over a period of use.

Each experiment investigates action parameter/s that were incorrectly specified, or ignored, in the KB. Typically,

such incorrect parameters and attributes tend to form types of anomalies. The experiments investigate:

- Distance failures: the distance between the robot and the table.
- Angle failures: the angle of the position of the cup on the table to the robot position.
- Combinations of attributes failures: where specific combinations of distance and angle attributes cause failure.

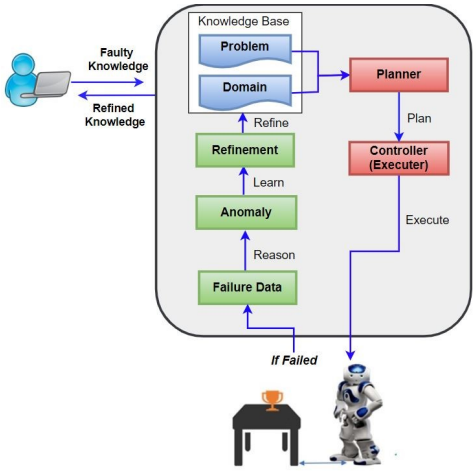

Figure 18: Conceptual model of the gripping task refinement.

- Group of attribute failures: where more than one attribute caused the failure. e,g.,the distance and angle attributes.

5.1. The Testing Procedure

When execution fails to execute an action within a plan that has been generated to solve a given task, the robot needs to realise why it has happened. Then, to avoid any future failure, an update to the KB should be made by the refinement system. Software that implements the anomaly detection and refinement processes above, called ADKRA, reasons about the cause of the failure and then refines the KB based on the outcome, to repair the cause of the failure.

To test this and evaluate ADKRA, and hence HDJ's effectiveness in reducing failure rates, we used the following procedure in each of the four kinds of tests referred to in Section 5:

1. Apply a change to the KB to make it faulty (e.g., give it the wrong maxdis).
2. Generate 100 problem files where the parameter/s in the KB are *randomly* generated to give a different problem each time (e.g., different distances between way points, or/and different initial states of the robot and cup)
3. Generate a plan for each of the 100 problems. To do this we use a general PDDL+ planning engine ENHSP [52], which has been embedded in HDJ.
4. Execute each of the 100 plans.
    - If the execution succeeds, add the execution data to TD.
    - If the execution fails, and we are not using ADKRA, then record the failure.
    - If the plan fails, and we are intending to use ADKRA, then record the failure; run ADKRA to do anomaly detection and knowledge refinement on the KB. Store the newly refined knowledge, for use in later tests.
5. If we are using ADKRA, then after the 100 tests, REPEAT STEP 3, and execute each of the 100 plans generated, recording the total number of failures.

Executing this procedure for each of the four aspects above, we can determine how much using the ADKRA process helps reduce failure rates within the execution of generated planning solutions. This is done by comparing the number of failures without ADKRA, with the number of failures in step 5 above, where we have used ADKRA.

5.2. Development and Technical Setup

The development setup of the experiments includes the developmental aspects of HDJ architecture (the integration of new and existing components) and the design of the current state facts and domain model. We first describe the some aspects of the PDDL language used in our experiments (Section5.3). Then a description of the software development and Middleware integration is provided in Section 5.4.

5.3. The PDDL Version Used

To capture both the symbolic and numerical aspects of robot domains, we chose to employ PDDL version 2.1 which allows numeric fluents, represented in Figure 5. It is inspired by the domain model of 'pouring water between jugs", in PDDLv2.1, by [16].

Numeric fluents are used in the pre-conditions of the 'grip' operator, and the metrics are described in the problem file (Figure 7). The numeric expressions are values associated with tuples of domain objects by domain functions. They allow the setting of different values that construct different solutions to problems of the same domain [16]. Given operator *P* from the domain model, *PNE* is a primitive numeric expression (*fluent*) of a task 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 PDDL2.1 models contain objects, and actions are represented as parameterised operations which change the status (attribute-values, relations) of objects. Numbers are used as values of attributes of objects and are manipulated through their connections with the objects that are identified in the initial state [16]. In the model of our running example, the functions **dist-to**, **mindis**, and **maxdis**, are referred to as numeric fluents, and are associated with the pre-conditions of the operator **grip**. These fluents are

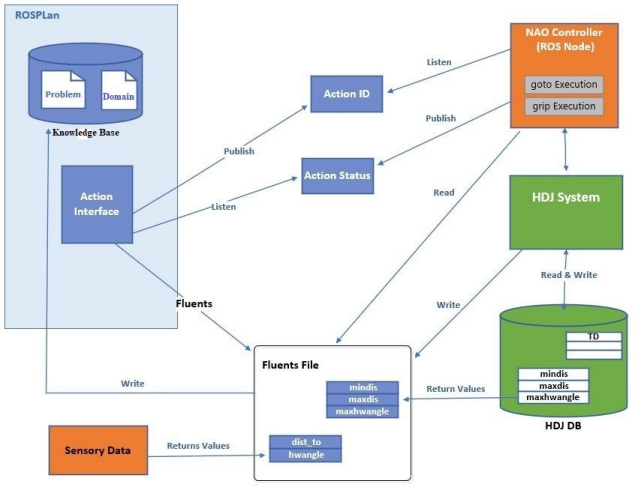

Figure 19: Software and Middleware communications

employed to represent the distance attribute and its permitted range of values. We use a prefix syntax (comparison) for the **grip** operator and its precondition (>=(dist-to ?r ?waypoint) (mindis ?r ?waypoint)). This forms a plan rule we call **dis** which is similar to the distance rule. We also have a **HeadYawAngle** rule where we employ numeric fluents in the functions **hwangle** and **maxhwangle**.

5.4. Software and Middleware HDJ system development entails implementing a new programme integrated with the existing components of ROS. In this research, we utilise ROS and ROSPLan as components of HDJ architecture.The controller of NAO was developed as a ROS node; this is because the controller communicates with ROSPlan to execute the planned actions, as represented in Figure 19. The ENHSP planner [52] was employed in our experiments to generate plans. The reasoning and refinement algorithms proposed by this research (ADKRA) have been developed and embedded in the context of NAO robot and HDJ architecture.

The HDJ system utilises the SQLite database to store and retrieve knowledge that is used to construct the initial state of the problem instance. This database is created to include structured information of the KB such as the actions parameters and all related information including the relationships between the attributes and correlated attributes. This includes the operators' pre-conditions, plan rules and even the representations of the targeted object. This provides the flexibility to customise the fluents values before passing them to the system. Figure 20 shows the representation of the relationship between the **HeadYawAngle** rule and the **dis** rule as in the database structure. The sensory data holds values of action attributes such as the current distance to the object and its location. Furthermore, the information of the targeted object,

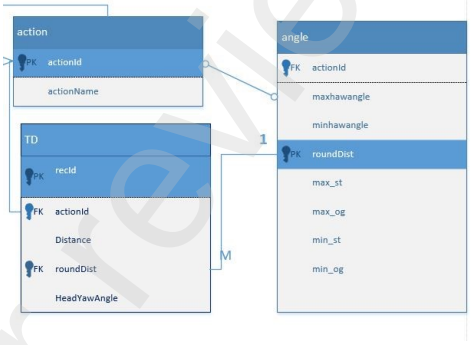

Figure 20: HDJ database : rounDist attribute and Angle elationship

the red cup, is also processed during the instanstiation process, [53], as it is used to detect the object and calculate the distance and the angle attributes. The Python programming language was used for controller development.

In the domain model, represented in Figure 5, we employ numeric fluents in the new pre-conditions of the 'grip' operator. As represented in Figure 19, all values of fluents are saved in an external file by the HDJ system and are passed to the problem file through the instantiation process and to the system through the execution stage. The initial values of **mindis** and **maxdis** are calculated and processed by the HDJ system and are stored as an external file to be read and retrieved by the system. Meanwhile, the values of **dist-to**, and **hwangle** are sensory data (actual distance and angle values) that are measured online during execution stages.

5.5. Experimental Setup

This section describes the experimental setup for four types of experiments. Each setup is described based on investigations conducted on each failure type, and the implementation setup for each experiment.

Experiments Investigating Distance Failures

These experiments investigate the distance parameter as a cause of failure, particularly the distance between the robot and the cup as seen in Figure 2. For the successful execution of gripping, the distance must be in the range of 15 to 23 centimetres (see Figure 2). If this

range is extended (e.g., 27cm>dis>15cm), some failures would be caused during execution because these extended values (24, 25, 26 and 27) of the distance will cause the execution to fail.

In order that some failures would be caused during execution, faulty knowledge was given to the system in this set of experiments. This consisted of an erroneous distance precondition as follows: the correct range of (23cm>dis>15cm) was extended to (27cm>dis>15cm). For each experimental episode the

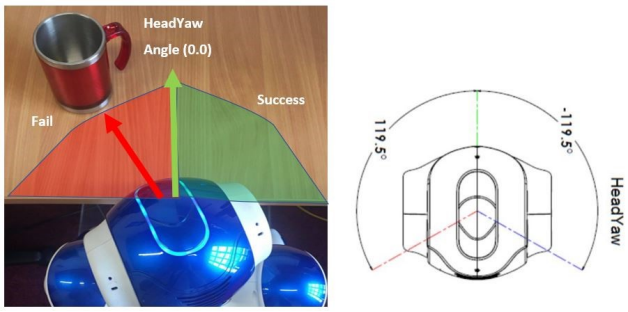

Figure 21: Success and fail ranges of 'HeadYawangle', to grip using the right hand of the NAO robot.

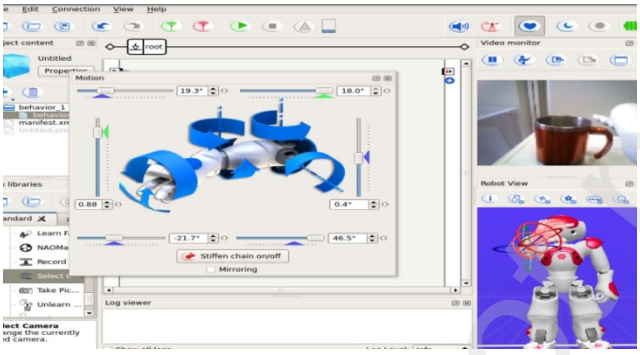

Figure 22: The joint values of NAO's arm in grasping position at the maximum degree towards the body.

initial position of the robot is changed, and then the generated solution plan for this new gripping task is checked for success or failure in execution. If the task plan fails, the KB is refined using ADKRA. Figure 28, in Appendix 7, describes the starting input and the final output (which is produced after a number of refinements) of these experiments.

Experiments Investigating Angle Failures

These experiments investigate the angle parameter as a cause of failure. For a successful execution of the gripping task, by the right hand, this angle should be in a range of 0.0° to 25° (see Figure 21). If this range is extended to a wider one (e.g., 25°>HeadYawAngle>25°), some failures would be caused during execution because these extended values (greater than 0°) will cause the execution to fail.

The NAO robot has 25 DOF of which 14 are for the upper part including its trunk, arms, and head. The rotation axis of its arms, and head joints is inclined at 45◦ towards the body. 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 DOF for the hand's grasping.

The available DOF of the arms of the NAO robot limits its abilities in grasping. The angle of the object's position is a critical condition for successful grasping. As seen in Figure 22, the maximum rotation of the 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.

In order to cause some failures during execution, faulty knowledge was given to the system in this set of experiments. This knowledge contained an erroneous HeadYawAngle precondition as follows: The correct range of (0.0° > HeadYawAngle > -25°) was extended to (0.0° > HeadYawAngle > -29°). For each experimental episode, the initial position of the robot was not changed, but the angle of the cup position on the table was randomly changed (see Figure 21) and the distance was fixed to 18 cm; Then the generated solution plan for this new gripping task was checked for success or failure in execution. If the task plan failed, the KB is refined using ADKRA. Figure 29 , in Appendix 7, describes the starting input and the final output (which is produced after a number of refinements) of these experiments. Figure 20 shows the success data of 'HeadYawangle' when the distance = 18 centimetres.

Experiments Investigating Attribute Combinations Failures

These experiments investigate the particular combinations of action attributes (Distance and HeadYawAngle) that cause executions to fail. For successful grasping, our experiments have shown that each distance point accepts a specific range of angle. The error in the KB is to ignore the associated attribute, which is the inverse relationship between the distance and angle parameters of the 'grip' action. As seen in Figure 23, when the **Distance** increases, the **HeadYawAngle** range is narrowed and goes towards a **0.0** angle. Figure 24 represents the inverse relationship between the **HeadYawAngle** attribute and the **Distance** attribute. For example, the 15 centimetre distance accepts the full range (0.0>HeadYawAngle>25), for the right hand. However, the 20 centimetre distance accepts a shorter range of angle of about (0.0>HeadYawAngle>-12).

For this reason, particular combinations (of the **Distance** and the **HeadYawAngle** attributes) cause execution failure. For example, despite the fact that the ranges (0.0>HeadYawAngle>-25) and (23cm>dis> 15cm) are considered successful ranges of distance and angle

attributes, some values within these ranges, if combined, will cause execution to fail. A 'Collective' anomaly is a group of attribute instances that appear isolated compared to the rest of the data-set. A specific case of 'Collective' is when each single attribute is not an anomaly but as a group appears together as an anomaly [11]. These experiments test such cases.

In order to cause some failures during execution, faulty knowledge was given to the system in this set of experiments. In the domain model part

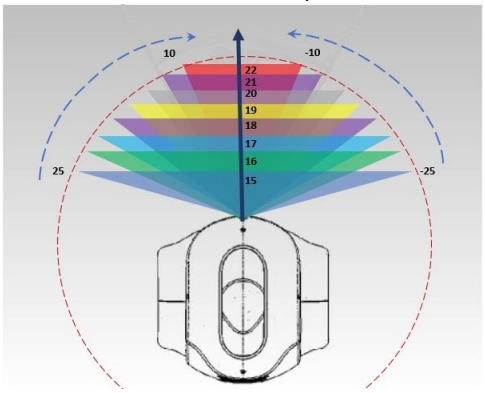

Figure 23: The relationship of the distance and angle attributes.

of the KB, represented in Figure 5, we have two preconditions that accept a range of values of the **Distance** and **HeadYawAngle** attributes. We set the ranges of both rules **HeadYawAngle** and **dis** to (0.0>HeadYawAngle>-25) and (23cm>dis>15cm); These ranges are considered to be the range needed for success. However, we know that within this range specific cross values will fail the execution; this is due to ignoring the inverse relationship between the **Distance** and **HeadYawAngle** attributes, which is considered as a fault in the KB. For each episode, we changed both the initial position of the robot to the table, and the angle of the cup position on the table as well. Then the generated solution plan for this new gripping task was checked for success or failure in execution. If the task plan failed, knowledge is refined according to Algorithm 1. Figure 30 , in Appendix A, describes the starting input and the final output (which is produced after a number of refinements) of these experiments

The correlation between the angle and the distance attributes is considered an attribute of the action 'goto' that needs to be checked as well when execution failed. To detect such an anomaly, we may have to create a new attribute (a correlated attribute [15]). Such that, *X3 = X1 X2 + X1*, where *X1* is the value of the **Distance** attribute and *X2* is the value of **HeadYawAngle** attribute. This approach can help detect unusual combinations of values that would not have possibly been detected by earlier steps of Algorithm 1.

For each failed execution, the ADKRA processes (Algorithm 1) learns values *LV* and used them to refine the module. We stated in Section 7 that the relationships of the domain's attributes determine which KB attribute is to be refined. So, for this particular instance, the KB attribute (**maxhwangle**) to refine is based on its relationship with the **Distance** attribute. The angle is classified as a 'slave' relationship, as seen in Figure 20, with the distance attribute. The

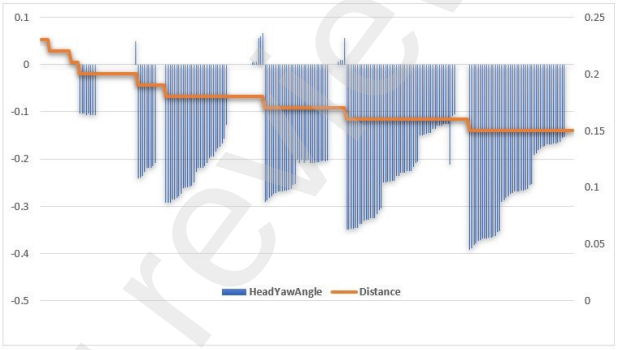

Figure 24: TD distribution. Distance and angle data

refinement system is set to only learn and refine the **HeadYawAngle** attribute.

Experiments Investigating Failure of a Group of Attributes

These experiments investigate the failure of multiple attributes (more than one attribute specified incorrectly) e.g, angle and distance attributes, which cause executions to fail. For example, a distance of 24 centimetres is considered a failure value. -27 degree of the angle is another failure value. A 'Collective' anomaly is where a group of attributes - instances appears isolated compared to the rest of the data-set [11]. This experiment is the general case of 'Collective' anomalies, when detecting two or more attributes as being anomalies.

In order to cause some failures during execution, the faulty knowledge given to the system in this set of experiments contained erroneous distance and HeadYawAngle preconditions as follows: The correct range of (23cm>dis>15cm) was extended to (25cm>dis>15cm) and the correct range of (0.0>Head YawAngle>-25) was extended to (0.0>HeadYawAngle >-27). These ranges are both incorrect and lead to failure during the execution. For each experimental episode, the distance of the robot to the cup was randomly changed, while the angle of the cup position was fixed at -20°. Then the generated solution plan for this new gripping task was checked for success or failure in execution. If the task plan failed, the knowledge was refined using ADKRA. Figure 31 , in Appendix A, describes the starting input and the final

output (produced after a number of refinements) of these experiments.

5.6. Experimental Process and Results

The test results shows the results of running the Testing Procedure specified above. The rate of failure decreases over time as the parameters are adjusted by the ADKRA subsystem. The overall results of these tests are represented in Figure 25. Table 1 represents the accuracy of predictions, false negative rate (FNR), and true positive rate (TPR) made by robot.

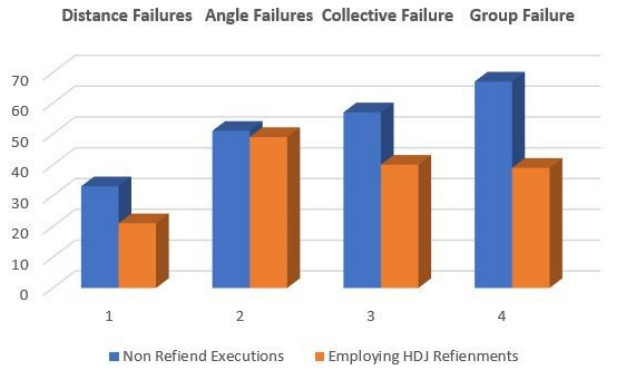

Figure 25: Field executions with and without knowledge refinement by ADKRA

Table 1

The Overall Accuracy of Predictions by HDJ Reasoner

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%

The experiments demonstrate the efficiency of our approach in refining the KB of the task planner, which reduces plan execution failures after repeated use. For example, by the end of the distance experiment, the ADKRA successfully corrected the values of the distance rule. The instance value of the maximum allowed distance (maxdis ?r ?waypoint) was corrected to be 23 centimetres (Figure 8). ADKRA reasons and detects anomalies efficiently and can refine the knowledge for successful future executions.

They also demonstrate the efficiency of the update validation process, which was demonstrated in the angle failure experiments. The validation process rejected all predictions made by ADKRA because they relied on faulty knowledge that neglected the correlated attribute of distance and angle parameters. The domain model neglected the rational relationship between the 'Distance' and the 'HeadYawangle' attributes, where each distance point accepts a specific range of angle values, which causes incorrect predictions by ADKRA.

As the relationships between the operator's parameters have to be represented, missing such representation can cause action executions to fail. In particular, in the angle failure experiment, the refinement process behaved like a domain validator, which is considered an extra advantage of using AD. The refinement process uncovers incorrect predictions that were caused by inaccurate kmowledge.

In other words, our experiments demonstrate that accurate knowledge can overcome issues related to inaccurate domain models. However, despite this fault, the HDJ (specifically in the group attribute failures) 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 was because the HDJ database considers the relationship between these attributes.

Furthermore, the specific case of the 'Collective' anomaly (whre 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 where such a type of anomaly can be used to develop a tool to discover such hidden relationships in domain models. Employing the 'Collective' AD type gives the HDJ the ability to discover hidden relationships that the current domain model lacks. Thus, it can indicate faults in the domain representation.

Moreover, employing the 'Collective' AD method to over come the model demonstrated that employing multiple methods of AD is beneficial and can overcome the problems or weaknesses of each method.

Furthermore, the experiments also demonstrated that real application brings about complications related to the real environment and sensor issues. The experiments indicated that NAO does not measure the distance to objects accurately 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 [54], in addition to the hardware specifications issue and the method that estimates the distance, is another factor of inaccurate distance measurements. Different techniques for depth estimation exist in which each has different outcomes and drawbacks.

The accuracy of sensory data is critical for successful predictions. This is due to the problem of overlapped edges between the accumulated data of the successful executions and the data of failed executions. The inaccuracy of the distance data led to overlapped data edges between the successful execution data and the fail data, as seen in Figure 26. For example the distance value of 23 centimetre appears in both datasets. While it appears as a successful attribute value in TD at the same time the same value can cause the execution to fail. Its

appearance in both sets prevents the robot from making successful predictions, as it does not consider it as an anomaly in TD.

6. Future Work

In this study, we utilised a simple (non-temporal) PDDL model and the efficiency of ADKRA was tested and approved with 'Point' and 'Collective' anomaly types. We also employed density based AD technique to detect anomalies in the action attribute values.

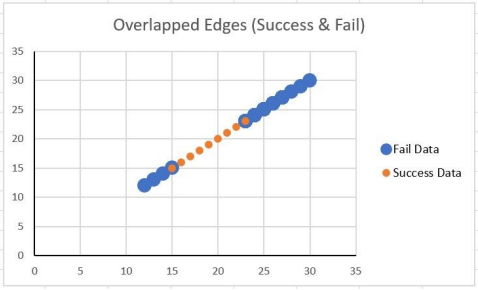

Figure 26: The distance datasets.

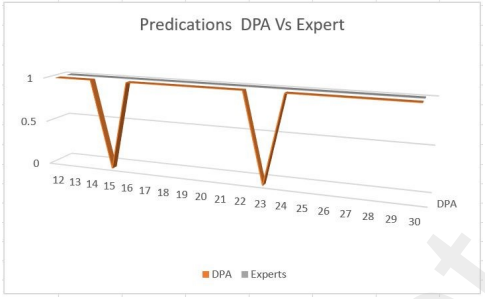

Figure 27: Graphical representation of the correct predictions made by the robot, in the distance experiment.

However, the low quality of NAO cameras led to overlapped data edges, which reduces the performance of the reasoning system.

Going forward, we aim to broaden the scope of knowledge that HDJ can refine and conduct more experiments across various scenarios. This includes using a temporal-PDDL model with "Contextual" AD, which is usually employed for time series models, in addition to the "Point" and "Collective" AD types. We plan to examine other AD approaches and techniques, potentially combining multiple methods to overcome each one's limitations and take advantage of their strengths.

A hybrid AD technique, such as using vision recognition AD in conjunction with density-based techniques, is also being considered. For instance, the vision recognition AD could identify differences in the features of a red cup while the density-based techniques detect differences in the object's geometric information.

Additionally, we plan to improve the system by utilising high-quality sensory systems, which can enhance the prediction accuracy. Research into methods such as ANN-based methods to detect and define correlations between action attributes will also be conducted. This can help detect "Collective" anomalies, address inaccurate domain models, and lead to more autonomous refinement systems.

7. Conclusions

In this paper, we have described the HDJ architecture, which we have implemented on a physical NAO robot. The original contribution of the paper is the description, implementation and evaluation of the components making up ADKRA. This embodies anomaly detection to find the cause of action failure of a real robot, and then knowledge refinement to resolve the anomaly by replacing faulty knowledge, with more accurate knowledge, within the robot's action-related knowledge base. We have demonstrated it to be capable of autonomously detecting an anomaly, and refining knowledge to overcome it. We can conclude that ADKRA contributes to bridging that gap between planning and execution. It reasons and detects anomalies efficiently and refines the PDDL model, so that in future plans a robots plan are less likely to fail.

The refinement process effectively validates the system predicates and the refined knowledge before setting it as permanent. It is capable of exposing inaccuracy in the domain model, and extracting accurate knowledge from inaccurate domain models; this is due to its ability to specify and customise knowledge with the help of its rational database.

This research 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 or knowledge refinement approach, and we can implement more than one approach as the latest trend in anomaly 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.

Appendix A. I/O

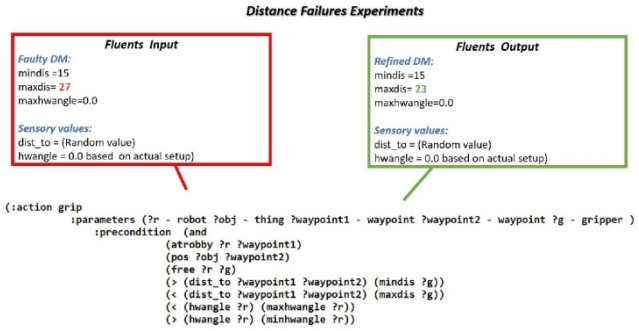

Figure 28: Example of an I/O for a distance failure experiment

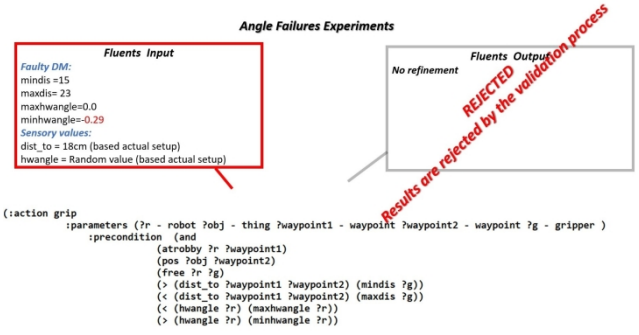

Figure 29: Example of an I/O for an angle failure experiment, The validation process rejected predictions made by reasoning system because the correlation attribute is neglected in the domain model

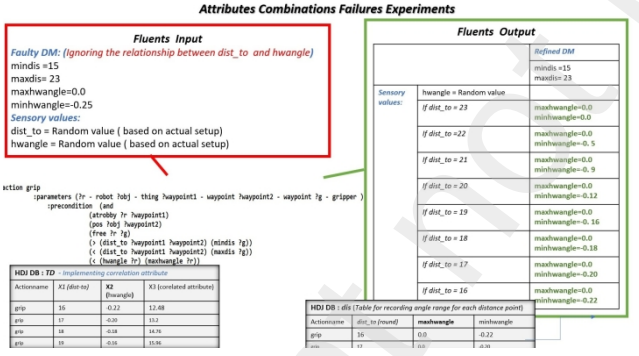

Figure 30: Example of an I/O for a combination failure experimen

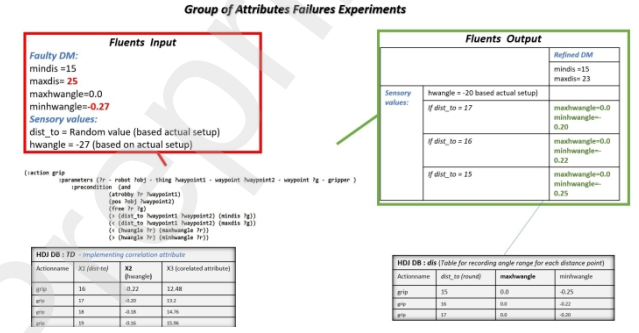

Figure 31: Example of an I/O for a group of attributes failure experiment.

References

[1] F. Ingrand, M. Ghallab, Deliberation for autonomous robots: A survey, Artificial Intelligence 247 (2017) 10–44.

[2] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtos, M. Carreras, Rosplan: Planning in the robot operating system., in: ICAPS, 2015, pp. 333–341.

[3] F. Stulp, M. Beetz, Refining the execution of abstract actions with learned action models, Journal of Artificial Intelligence Research 32 (2008) 487–523.

[4] R. C. Arkin, Behavior-based robotics, MIT press, 1998.

[5] Y. Gil, Acquiring domain knowledge for planning by experimentation, Tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE (1992).

[6] T. Munzer, M. Toussaint, M. Lopes, Efficient behavior learning in human–robot collaboration, Autonomous Robots 42 (5) (2018) 1103–1115.

[7] L. Kunze, N. Hawes, T. Duckett, M. Hanheide, T. Krajník, Artificial intelligence for long-term robot autonomy: A survey, IEEE Robotics and Automation Letters 3 (4) (2018) 4023–4030. [8] A. Zimek, E. Schubert, Outlier Detection, Springer New York, New York, NY, 2017, pp. 1–5. doi:10.1007/978-1-4899-7993-3_80719-1.

URL https://doi.org/10.1007/978-1-4899-7993-3_80719-1

[9] V. Hodge, J. Austin, A survey of outlier detection methodologies, Artificial intelligence review 22 (2) (2004) 85–126.

[10] M. R. Smith, T. Martinez, Improving classification accuracy by identifying and removing instances that should be misclassified, in: The 2011 International Joint Conference on Neural Networks, IEEE, 2011, pp. 2690–2697.

[11] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM computing surveys (CSUR) 41 (3) (2009) 15.

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

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

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

[15] T. Næs, E. Risvik, Multivariate analysis of data in sensory science, Elsevier, 1996.

[16] M. Fox, D. Long, Pddl2. 1: An extension to pddl for expressing temporal planning domains, Journal of artificial intelligence research 20 (2003) 61–124.

[17] A. Kirsch, M. Beetz, Training on the job—collecting experience with hierarchical hybrid automata, in: Annual Conference on Artificial Intelligence, Springer, 2007, pp. 473–476.

[18] E. Khalastchi, M. Kalech, G. A. Kaminka, R. Lin, Online datadriven anomaly detection in autonomous robots, Knowledge and Information Systems 43 (3) (2015) 657–688.

[19] R. Hornung, H. Urbanek, J. Klodmann, C. Osendorfer, P. Van Der Smagt, Model-free robot anomaly detection, in: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2014, pp. 3676–3683.

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

[21] P. Crook, G. Hayes, et al., A robot implementation of a biologically inspired method for novelty detection, in: Proceedings of the Towards Intelligent Mobile Robots Conference, 2001.

[22] P. A. Crook, S. Marsland, G. Hayes, U. Nehmzow, A tale of two filters-on-line novelty detection, in: Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292), Vol. 4, IEEE, 2002, pp. 3894–3899.

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

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

[25] T. Krajnik, J. P. Fentanes, G. Cielniak, C. Dondrup, T. Duckett, Spectral analysis for long-term robotic mapping, in: 2014 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2014, pp. 3706–3711.

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

[27] W.-M. Shen, Discovery as autonomous learning from the environment, Machine Learning 12 (1) (1993) 143–165.

[28] P. Doherty, J. Kvarnström, F. Heintz, A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems, Autonomous Agents and Multi-Agent Systems 19 (3) (2009) 332–377. doi:10.1007/s10458-009-9079-8.

[29] M. Beetz, L. Mösenlechner, M. Tenorth, Cram—a cognitive robot abstract machine for everyday manipulation in human environments, in: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2010, pp. 1012–1017.

[30] B. G. Weber, M. Mateas, A. Jhala, Learning from demonstration for goal-driven autonomy., in: AAAI, 2012.

[31] X. Wang, Learning planning operators by observation and practice, Ph.D. thesis, Carnegie Mellon University (1996).

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

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

[34] S. Karapinar, D. Altan, S. Sariel-Talay, A robust planning framework for cognitive robots, in: Workshops at the TwentySixth AAAI Conference on Artificial Intelligence, 2012.

[35] A. Lindsay, S. Franco, R. Reba, T. L. McCluskey, Refining process descriptions from execution data in hybrid planning domain models, in: Proceedings of the International Conference on Automated Planning and Scheduling, Vol. 30, 2020, pp. 469–477.

[36] J. L. Ambite, C. A. Knoblock, Planning by rewriting: Efficiently generating high-quality plans., Tech. rep., UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFORMA-
TION SCIENCES INST (1997).

[37] N. Hawes, C. Burbridge, F. Jovan, L. Kunze, B. Lacerda, L. Mudrova, J. Young, J. Wyatt, D. Hebesberger, T. Kortner, et al., The strands project: Long-term autonomy in everyday environments, IEEE Robotics & Automation Magazine 24 (3) (2017) 146–156.

[38] V. Verma, T. Estlin, A. Jónsson, C. Pasareanu, R. Simmons, K. Tso, Plan execution interchange language (plexil) for executable plans and command sequences, in: International symposium on artificial intelligence, robotics and automation in space (iSAIRAS), Citeseer, 2005.

[39] V. A. Ziparo, L. Iocchi, P. U. Lima, D. Nardi, P. F. Palamara, Petri net plans, Autonomous Agents and Multi-Agent Systems 23 (3) (2011) 344–383.

[40] M. Leonetti, L. Iocchi, Learnpnp: A tool for learning agent behaviors, in: Robot Soccer World Cup, Springer, 2010, pp. 418–429.

[41] S.-O. Bezrucav, B. Corves, Improved ai planning for cooperating teams of humans and robots, in: Proceedings of the Planning and Robotics (PlanRob) Workshop—ICAPS, 2020.

[42] L. P. Kaelbling, T. Lozano-Pérez, Hierarchical task and motion planning in the now. in 2011 ieee icra (2011).

[43] S. Cambon, R. Alami, F. Gravot, A hybrid approach to intricate motion, manipulation and task planning, The International Journal of Robotics Research 28 (1) (2009) 104–126.

[44] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, P. Abbeel, Combined task and motion planning through an extensible planner-independent interface layer, in: 2014 IEEE international conference on robotics and automation (ICRA), IEEE, 2014, pp. 639–646.

[45] E. Fernandez-Gonzalez, E. Karpas, B. Williams, Mixed discrete-continuous planning with convex optimization, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 31, 2017.

[46] M. Cashmore, M. Fox, D. Long, D. Magazzeni, A compilation of the full pddl+ language into smt, in: Workshops at the Thirtieth AAAI Conference on Artificial Intelligence, 2016.

[47] H. Kautz, B. Selman, Pushing the envelope: Planning, propositional logic, and stochastic search, in: Proceedings of the national conference on artificial intelligence, 1996, pp. 1194– 1201.

[48] J. Rintanen, Madagascar: Efficient planning with sat, The 2011 International Planning Competition 61 (2011).

[49] J. Hoffmann, The metric-ff planning system: Translating"ignoring delete lists"to numeric state variables, Journal of artificial intelligence research 20 (2003) 291–341.

[50] A. Coles, A. Coles, M. Fox, D. Long, Forward-chaining partialorder planning, in: Proceedings of the International Conference on Automated Planning and Scheduling, Vol. 20, 2010, pp. 42–49.

[51] K. P. Murphy, Machine learning: a probabilistic perspective, MIT press, 2012.

[52] Enrico Scala, The enhsp planning system, https://sites.google.com/view/enhsp/, accessed: 2022-05-06 (n.d.).

[53] kCL planning, Rosplan overview, *ROSPlan*(online), accessed: 2021-01-03 (n.d.).
URL http://kcl-planning.github.io/ROSPlan/documentation/

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

Improving Task Planning Knowledge Robustness for Autonomous Robots

Hadeel Jazzaa[a,d], Thomas McCluskey[a] and David Peebles[b]

[a]School of Computing and Engineering, University of Huddersfield, Huddersfield, UK
[b]School of Human and Health Sciences, University of Huddersfield, Huddersfield, UK

ARTICLE INFO

Keywords:
Autonomous Robots
Anomaly detection
Reasoning
Task planning knowledge
knowledge refinement

ABSTRACT

The requirement for autonomous robots to exhibit higher-level cognitive skills by planning and adapting in an ever changing environment is indeed a great challenge for the AI community. This paper demonstrates how a physical robot can be capable of adapting its symbolic knowledge of the environment, by using experiences in robot action execution to drive knowledge refinement, and hence to improve the success rate of the task plans the robot creates. We propose a method for refining domain knowledge, encoded in the PDDL language, based on a novel method in reasoning using anomaly detection techniques. The approach used in this paper is to extend a high-level planning platform (ROSPlan) to help create more robust planning systems to improve the knowledge on which intelligent robot behaviour is based. Empirical testing and evaluation has been performed using the NAO robot in a kitchen scenario. The results show that the robot system refines its knowledge in line the result of its experience of using plans that it creates to achieve tasks. The refined knowledge leads to the future synthesis of task plans which demonstrate decreasing rates of failure over time as faulty knowledge is removed or adjusted.

1. Introduction

Autonomous robots operating in partially known and dynamic environments (e.g., exploration robotics) require systems to create and execute plans deliberately [1]. Gathering knowledge, and acting on that knowledge, in such environments is a major demand for robust autonomous robot systems.

A robot may encounter a range of failures while executing its planned actions. The plan execution strategy must account for the action/plan failure, which results from ignorance or change [2]. The robot could overcome such failures by gathering updated information about its current environment, and re-planning to generate a new plan. However, in some cases re-planning does not solve the problem because the robot's knowledge of the conditions and effects of its own actions on the environment may be faulty. Equipping autonomous robots with reasoning mechanisms can be useful when dealing with such issues. To overcome failure, the robot must reason about the cause of the failure to refine its knowledge using his experience. When execution of an action in a plan fails, the robot must realise why it has happened. Then, to avoid any future failure, an update to its knowledge should be made by the refinement system.

Systems with pre-engineered knowledge of their environment often provide an abstraction hierarchy of types of knowledge for domain experts to use in crafting knowledge. At an abstract level parameterised actions are key to an efficient approach in bridging the gap between the abstract plan and action execution. Actions include parameters that accept changed values to enable the implementation of the same action in different situations [3, 4]. For example, the action GoTo(Pos) is more flexible to program than GoTo-Centre, where 'Pos' is a parameter that accepts different values, including the centre [3].

But however much hand-crafted engineering is facilitated, incomplete action models that are missing some aspects may cause action execution failure and result in failures in achieving tasks. Hence, ultimately, plan construction and execution can only be successful through a *combination* of human-specified and robot-learned knowledge [3] on which behaviour is based. For example, incorrectly specified actions requiring new attributes or relationships, in the face of changing environment needs to be updated regularly without relying always on human hand-crafting. Additionally, an engineer may miss preconditions and/or effects or correct knowledge, or entire operators may be absent from the model [5]. Our work furthers the hypothesis that accurate models can be learned online while data gathering during operation [6]. This is aligned with the move towards long-term autonomy [7], in that making the system more robust through experiential learning will increase its effective life. Here failures and successes drive the improvement of pre-specified domain knowledge by connecting the robot architecture's abstraction levels in order to investigate the cause of execution failure and use this to drive updates to the robot's knowledge as a result.

hadeel.jazzaa@hud.ac.uk (H. Jazzaa); t.l.mccluskey@hud.ac.uk (T. McCluskey); d.peebles@hud.ac.uk (D. Peebles)
ORCID(s): 0000-0000-0000-0000 (H. Jazzaa); 0000-0000-0000-0000 (T. McCluskey); 0000-0003-1008-9275 (D. Peebles)

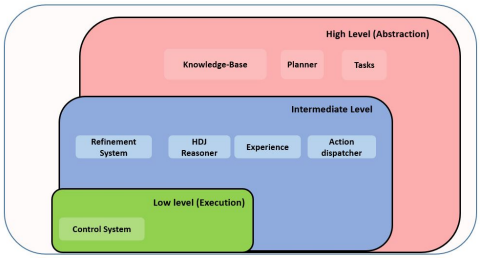

Figure 1: HDJ Intelligent Robot Hierarchical Architecture

To do so, we developed the HDJ robotics architecture (shown in Figure 1) which is based on a "conventional" task planning architecture. This contains knowledge of its environment encoded within a knowledge base referred to as KB below. The KB contains important components such as facts representing the environment's state, and expressions forming the robot's model of its action capabilities, referred to as the "domain model". HDJ but has extra components that reason with experience-based data, leading to the refinement of the KB held in a PDDL model. The HDJ software includes the anomaly detection and knowledge refinement subsystem called ADKRA, which manages the creation of learned values used to update the KB. The robot's operational experience is stored in datastores, which constitutes the ADKRA's training data (TD). TD is the history of successful executions, typically maps of action model parameters, and contains all the relevant information required in the implementation of ADKRA. The novel aspects of our contribution include the method for anomaly detection (AD) and subsequent knowledge refinement performed by Algorithm 1, detailed below.

Anomaly detection involves identifying data that deviates from the norm and raises concerns about a particular issue or problem [8]. It has been utilised in various fields such as detecting bank fraud, identifying medical issues, monitoring systems, detecting ecological disruptions, ensuring network security, and recognising events [9, 10]. Outliers can present themselves in different forms, and the anomaly detection survey [11] highlights several anomaly detection methods based on different categories, mainly point anomalies, contextual anomalies, and collective anomalies. Several anomaly detection approaches exist, including density-based techniques, machine learning methods, and neural network approaches. Additionally, multiple anomaly detection techniques can be used on the same dataset. The latest trend in anomaly detection systems is the use of hybrid techniques, which combines multiple methods to overcome the weaknesses and limitations of each approach and maximise the benefits of each one [9].

The refinement activity starts after execution has failed and the feedback of the failed execution has been sent to the KB. Action parameters and all related information are crucial to our approach, so the focus of this research is to use the execution of actions as the basis of learning, initiated by action plan failure. The focus is on developing a reasoning method that enables the robot to recognise the cause of action failure (the anomaly), when it occurs, and to use the anomaly to direct autonomous refinements and knowledge correction. The refinements then make future generated plans more likely to succeed.

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 [12, 13]. 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 [12], 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.

This paper's contributions are summarised as follows:

- The detailed description of the anomaly detection and knowledge refinement algorithms (ADKRA) which identify and repair faulty task-oriented domain knowledge in a robot that has previously led to execution failures;
- An empirical evaluation of HDJ when using an implementation of the ADKRA within a real-world humanoid robot.
- An overview of the HDJ architecture which is capable of operating the robot and its repair components.

We have experimented with an implementation of HDJ using the domain and running example explained in next section, Section 2. Section 3 presents an overview of research related work. Section 4 describes the HDJ hierarchical system and its framework, the system mechanism and algorithms. Section 5 describes the evaluation of the HDJ system through designing a series of experiments. Finally, this article concludes in Section 7.

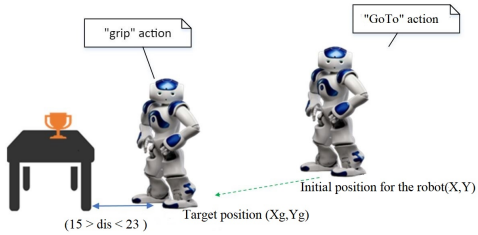

Figure 2: Kitchen scenario: gripping task.

Figure 3: Successful execution: the NAO robot can reach for the cup.

2. Running Example

We will use a running example, based on the kitchen scenario [14], to motivate, illustrate and evaluate the techniques introduced. As shown in Figure 2, the tasks performed by the robot included gripping a cup and moving it from one table to another. As Figure 3 illustrates, a successful execution of the 'grip' operator is when the robot can carry out its actions; in this specific case, this means the robot can reach for and grip the cup.

The grasping task requires several functions, such as vision recognition, object detection, distance calculation and motion design. Distance measurement is of great importance in this example. The accuracy of the distance data affects the performance of the robot, which may lead to execution failure.

In Figure 2, we can see that for a successful execution, the distance between the NAO robot and the table must be from 15 to 23 centimetres. If this range is specified incorrectly, e.g., 15 to 27 centimetres, it will result in some failed executions. Hence, if the robot is equipped with an autonomous reasoning function that enables it to realise this fault in the KB and justify and correct the model's description, this will make for more robust and longer term autonomy.

In this research, we use the NAO robot in our experiments. The NAO robot can learn, recognise and track objects, as well as perform various motion tasks. To perform the grasping task, the planning system dispatches two actions,'goto' and 'grip', to the controller of the NAO robot. Domain model definitions and action parameters, residing in the KB, are provided in the following subsections.

2.1. Action Parameters and Attributes

Action parameters refer to a data vector of all information related to an action. Let $X = x_1,...,x_n$ where n is the number of attributes of the action.

X1	X2	X3	X4	X....	Xn
10	17	14	22	...	15
11	10	15	16	...	16
12	10	16	17	..	17
13	16	14	22	...	20
..	..	..	..	..	..
X_{m1}	X_{m2}	X_{m3}	X_{m4}	$X_{m}...$	X_{mn}

Figure 4: Action experience table

For example, the action 'grip' has n number of attributes. The distance between the robot and the table represents its first attribute (X_1), while (X_2) is the angle attribute that represents the position of the targeted object. Furthermore, the relationship between the 'distance' and the 'angle' attributes is another factor, a correlated attribute [15], must be considered as an attribute of the grip action that can be represented as (X_3). These action parameters can be learned and justified based on online execution. Figure4 shows an action table where each record is the value of the action attributes for each execution.

2.2. The PDDL Model

To be able to do Task Planning in our architecture, we utilise the PDDL description language v. 2.1 [16], a simple (non-temporal) PDDL variant. The PDDL model of the gripping task is represented in Figure 5. Figure 6 shows the task problem.

For task planning, a task is defined to be a pair of a domain model, and a problem instance of the form *(Os, Init, G)*. This is a triple consisting of the objects *Os* in the domain model, the initial state *Init* and the goal *G* specification. Given operator *P* in the domain model, each *P* consists of a pair *(pre(P), efc(P))* where *pre(P)* represents the action preconditions, and *efc(P)* represents the effects of the action.

In the domain model, shown in Figure 5, the functions **dist-to** and **hwangle** are associated with the pre-conditions of the operator **grip**. These fluents are employed to represent the distance rule (**dis**) and the angle rule (**HeadYawAngle**).

Incorrectly specified preconditions will cause the action to fail. This, in turn, will affect the constraints on the 'goto' action preceding. Doing so could lead to failure to execute a seemingly valid abstract plan.

```
(define (domain NAO)
(:requirements :strips :typing :fluents)
(:types waypoint thing robot gripper)
(:predicates
      (atrobby ?r - robot ?x - waypoint)
      (pos ?o - thing ?x - waypoint)
      (free ?r - robot ?g - gripper)
      (carry ?r - robot ?o - thing ?g
      - gripper))

(:functions
        (dist_to ?x1 - waypoint ?x2 - waypoint)
        (maxdis ?grp - gripper)
        (mindis ?grp - gripper)
        (hwangle ?r - robot)
        (maxhwangle ?r - robot)
        (minhwangle ?r - robot))

(:action goto
  :parameters (?r - robot ?from - waypoint ?to
  - waypoint)
  :precondition (and
                (atrobby ?r ?from))
  :effect (and
             (atrobby ?r ?to)
             (not(atrobby ?r ?from))))

(:action grip
  :parameters (?r - robot ?obj - thing
  ?waypoint1 - waypoint
  ?waypoint2 - waypoint ?g - gripper )

   :precondition  (and
       (atrobby ?r ?waypoint1)
       (pos ?obj ?waypoint2)
       (free ?r ?g)
           (> (dist_to ?waypoint1 ?waypoint2)
           (mindis ?g))
       (< (dist_to ?waypoint1 ?waypoint2)
          (maxdis ?g))
       (< (hwangle ?r) (maxhwangle ?r))
       (> (hwangle ?r) (minhwangle ?r)))
   :effect (and
       (carry ?r ?obj ?g)
       (not (free ?r ?g)))))
```

Figure 5: A segment of the KB's PDDL Model implemented in our experiments

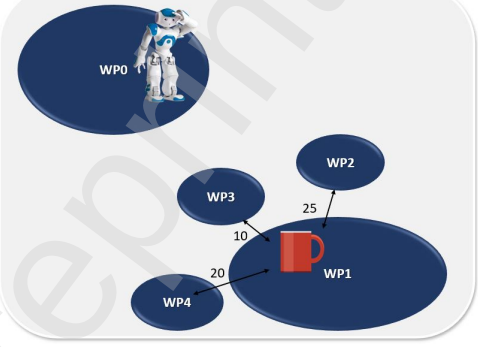

Figure 6: Problem representation of the gripping task.

Our example illustrates the possibility that plan rules and pre-conditions could be updated to avoid failure of future executions. This can be done by repairing incorrectly specified actions and recognising new attributes or relationships [17].

```
(define (problem task)
 (:domain NAO)
 (:objects
     wp0 wp1 wp2 wp3 wp4 - waypoint
     nao - robot
     grp - gripper
     redcup - thing)
 (:init
       (atrobby nao wp0)
      (pos redcup wp1)
      (free nao grp)
      (= (dist_to wp2 wp1) 25)
      (= (dist_to wp3 wp1) 10)
      (= (dist_to wp4 wp1) 20)
      (= (mindis grp) 15)
      (= (maxdis grp) 27)
      (= (hwangle nao) 0.0)
      (= (maxhwangle nao ) 0.2) )
  (:goal (and
     (carry nao redcup grp)) ))
```

Figure 7: The problem file of failed execution

```
(define (problem task)
 (:domain NAO)
 (:objects
     wp0 wp1 wp2 wp3 wp4 - waypoint
     nao - robot
     grp - gripper
     redcup - thing)
 (:init
      (atrobby nao wp0)
      (pos redcup wp1)
      (free nao grp)
      (= (dist_to wp2 wp1) 25)
      (= (dist_to wp3 wp1) 10)
      (= (dist_to wp4 wp1) 20)
      (= (mindis grp) 15)
      (= (maxdis grp) 23)
      (= (hwangle nao) 0.0)
      (= (maxhwangle nao ) 0.2))
  (:goal (and
         (carry nao redcup grp)) )
```

Figure 8: The problem file of successful execution (after knowledge refinement)

```
; Cost: 0.001
; Time 0.00
0.000: (goto nao wp0 wp2)  [0.001]
0.001: (grip nao redcup wp2 wp1 grp)  [0.001]
```

Figure 9: The plan generated from faulty domain knowledge resulting in execution failure.

Figures 7 and 8 show the problem file before and after refinements that repair incorrect parts of the model. The constraint (27cm>dis>15cm) is refined to

```
; Cost: 0.001
; Time 0.00
0.000: (goto nao wp0 wp4) [0.001]
0.001: (grip nao redcup wp4 wp1 grp) [0.001]
```

Figure 10: The plan generated from refined, more accurate encoding of domain knowledge resulting in successful execution.

be (23cm>dis>15cm). The plan generated with the faulty constraint (which leads to execution failure), is presented in Figure 9. While the plan generated with the refined constraint (which leads to execution success) is presented in Figure 10.

3. Related Work

The related work in this paper covers diverse areas of research, with a focus on the most relevant previous work. It is divided into four main parts. The first part is a brief survey of anomalies in general robotics applications. The second part is a brief survey of related works that perform refinements of the planning domain based on anomalies but using different methods than the one implemented in this research. The running example, explained in Section 2, of this paper combines task and motion planners to execute the plan. For this reason, the third part covers related works that implement task and motion planning approaches. Finally, the fourth part presents planning systems that deal with numeric constraints, similar to those used in the evaluation presented in Section 5.

While many existing robotics techniques use anomaly detection (AD) for condition monitoring, fault detection, novelty detection in the environment, and identifying anomalous behaviors of robots, there is little evidence that AD 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. AD has been applied to condition monitoring and fault detection in robots in previous works, such as [18], [19], and [20]. Novelty detection in the environment has been achieved using AD, as shown in the works of [21] and [22]. In addition, AD has been used to learn differences in object classification, as demonstrated in [7], and to identify anomalous behaviours using time scale and resolution, as in [23]. Robot-assisted feeding in healthcare applications has also utilised AD, as described in [24]. Finally, [25] modeled the spatio-temporal dynamics of the environment's continuous processes through its frequency spectrum using AD.

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, and, even fewer are demonstrated in robotics [26]. Indeed, many exist works perform the refinement of a planning domain or knowledge. Many of these works have concerned detecting the cause of failure as a method to drive change. However, these works often use different methods and are limited to specific functions that differ from the method proposed in this paper. For instance, LIVE [27] employs prediction rules and the incremental enlargement heuristic to identify relation differences in faulty rules, while EXPO [5] uses the ORM method to refine incomplete planning knowledge. The AUV [28] employs a progression algorithm to detect execution failure, but our architecture focuses on inferring the cause of action execution failure. Similarly, CRAM framework [29] employs plans in specific languages and a knowledge processing system, and GDA [30] and OBSERVER [31] use learning from demonstration. Our proposed method uses AD as a reasoning method to learn from failed executions, which is different from PELA [32] that learns probabilistic knowledge about the success of actions and predictions of execution dead-ends. RACE system [33] employs HTN planner and semantic descriptions, while HDJ experience is stored in a rational database that records action execution information. ProbCog [34] uses logic programming and geometric reasoning, whereas our method employs AD for reasoning. Other works, such as [35], refine hybrid domain models using ML techniques, and PBR system [36] employs conflicting choice points to refine plans. The work in [3] predicates the effects and performance of planned actions before execution, and STRANDS project [37] monitors system behavior and navigation for long-term autonomy. Additionally, [38] developed PLEXIL, an execution language for richer representations of executable plans, and [39] proposed the PNPs programming language for robot and multi-robot behavior design. LearnPNP tool [40] combines Petri Net Plans formalism with reinforcement learning, and [41] proposes an approach for scenarios that involve teams of humans and robots with parallel planning and dispatching features to reduce re-planning waiting times.

Several works have been proposed to address the problem of planning with geometric constraints using task and motion planning approaches. One such work is the HPN hierarchy [42], which employs an A* search to generate low-level plans while ensuring optimal and valid conditions. It constructs plans at an abstract level and generates sub-goals recursively backwards from the goal, allowing for efficient planning. Another approach, the hybrid approach to motion, manipulation, and task planning [43], deals with planning knowledge involving multiple robots and objects by considering the static geometric environment and incrementally synthesising environment constraints during planning when needed. It allows the task planner to operate in an abstract state space

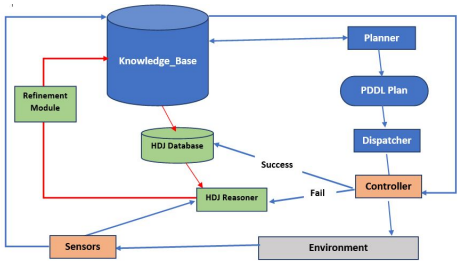

Figure 11: HDJ data flow diagram. The existing components (Blue boxes) and HDJ components (Green boxes)

without considering geometry constraints. Additionally, [44] proposed an approach that integrates off-the-shelf task planners and motion planners with minimal assumptions on each layer. This approach requires failures to be expressed and represented in logical predicates at the task level.

In our domain model implementation (Section 2), we utilised numeric constraints. Several planning systems also use numeric constraints, such as cqScotty [45], a heuristic forward search planner that reasons with control variables using SOCP. Another approach is the compilation of the full PDDL+ language into SMT [46], which is designed to model mixed discrete-continuous planning domains and process the set of PDDL+ features. This encoding is based on the SAT encoding of planning problems by [47] and [48]. Metric-FF [49], a domain-independent planning system, is an extension of the algorithms used in the heuristic planning system FF for linear tasks. It deals with PDDL 2.1 level 2 combined with ADL, which allows for a finite number of numerical state variables. Lastly, POPF [50] is a forward-chaining temporal planner built on grounded forward search and linear programming. POPF can handle domains with continuous linear numeric change and effects dependent on the duration of actions and supports a significant portion of PDDL 2.1 level 5.

4. 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 [28]. 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 11 presents the data flow through HDJ. It contains original components as well as existing components - in particular parts of the ROSPlan [2] 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 11).

As seen in Figure 11, if the feedback coming from the control system returns 'execution failed', the HDJ reasoner combines the information of the failed execution with the TD extracted from a successful executions. The reasoner suspects the anomalies of causing the failure. It employs Algorithm 2 to extract the anomalies. The refinement model employs Algorithm 3 and Algorithm 4. It uses that cause/s to learn new success value and uses these values to refine the KB. This is to avoid a future failed execution for the same situation context

4.1. The Algorithms in HDJ

Algorithm 1

is the top-level algorithm which collects data and drives the changes to HDJ's knowledge (see Figure 11). 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 KB by executing Algorithm 4, which updates the KB. Any detected anomaly is a value that indicates parameters or pieces of information in the KB, such as pre-conditions and states. Future failure can be avoided by updating the knowledge of the task's planner.

4.2. 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 mxn matrix where the columns denote n attributes and the rows maintain the values of these attributes over m execution, check action parameters in our running example Section 2.

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 13, if we recorded 100 successful execution, then (m) will take any value between 1 and 100.

Algorithm 2

embodies our approach to AD. It is the problem of searching for patterns in data that differ and raise suspicions about a specific problem or issue [8]. We use the same technique to compare the values of the

Algorithm 1: Anomaly Detection and Knowledge Refinement Algorithm

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

```
while plan execution do
    if not (Feedback.Success) then
        TD ← Training-Data
        FD ← Failed-Data
        Anomaly Detection(FD, TD)
            // Implementation of Algorithm 2
        if QueryResult.count > 0 then
                // If Outliers exists
            Out ← Detected Outlier
            Return Out
            Learning-Processing (Out)
                // Learn new success values, Algorithm 3
            LV ← Learned Value
            Return LV
            Refinement-process (LV);
                // Refine the KB, Algorithm 4
        end
    else
        SD ← Successful action Data
        Add-Training-Data (SD);
    end
end
```

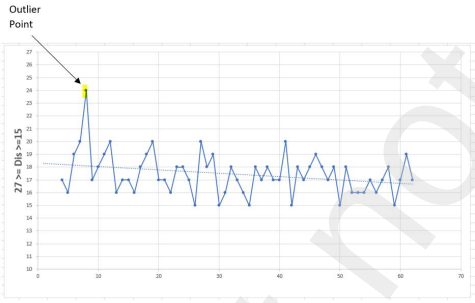

Figure 12: AD of a failed execution. The outlier 'distance' value appears far a way from its nearest neighbour in TD

failed action execution with the training data. The purpose is to discover the suspicious values in the failed execution, focussing on the outlier values which are often responsible for that failure.

A standard approach for AD in datasets is to create a normal data model and compare/test records against normality [8]. 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 AD approach, the failed execution record is compared and tested against records in TD by

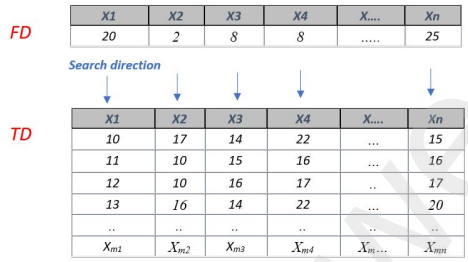

Figure 13: Algorithm 2 search. Each attribute instance (X) of FD is tested against its X vector in TD. n is number of attributes. M number of records in TD

extracting the anomalous values of the action execution features. In our running example, Section 2, the grip action draws a profile of the distance attribute's history, Figure 12, where the outer point is the distance feature value of the failed action.

Algorithm 2 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 represent the action attributes. The failed execution information (FD) is a record of a failed complete action execution that is given during operation. FD is an input vector $x = x_1,\ldots,x_n$ where n is number of attributes of FD. The online AD problem is to decide for each given x, whether or not x is anomalous with respect to TD. Our approach is to test each attribute of FD and search for the same attribute that does not appear in TD. As seen in Figure 13, Algorithm 2 considers expressions of form X, it seeks the value of X in FD that does not appear in the corresponding distribution of TD. Each action attribute is represented by a vector X where X is an m * 1 matrix where m is the number of successful executions recorded in the 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 do
    for j ← 1 to m do
        Select FD-X(i) not-in TD-X(j,i) ;
    end
    if QueryResult.count > 0 then
        // if an anomaly detected
        Insert into report_table FD-X(i).value,
          FD-X(i).label;
    end
end
```

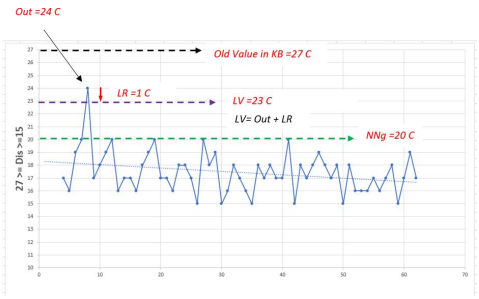

Figure 14: Learning process: The calculation of the new value *LV* for the 'dis' rule.

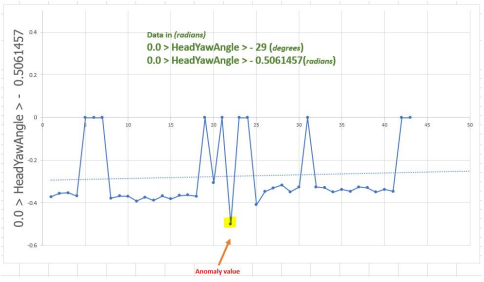

Figure 15: Failed execution with anomaly detection. 'HeadYawAngle' value is less than its nearest neighbour(*Out* < *NNhg*)

4.3. The Refinement 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 value in the KB) 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 [51]. Figure 14 is a visualised representation of LP for 'dis' rule.

In the presented example, in Figure 14, the *Unit* size is **one** centimetre; This because *Out* is a distance value that is measured by the centimetre unit.

The direction in which the step is taken is determined by the comparison of *Out* and *NNhg*. Algorithm 3 considers two comparisons of outliers. The first comparison is represented in Figure 12, and Figure 14 where the outlier value is greater than its nearest neighbour. While the second comparison type is represented in Figure 15 where the outlier value is less than its *NNhg*. Figure 16 shows how to specify *LV* for the two type of anomaly in the 'distance' attribute.

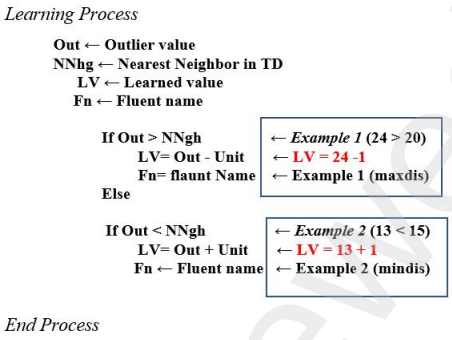

Figure 16: learning process for two type of anomalies in the 'distance' attribute. *LV* = *Out - Unit* this when *Out* > *NNhg*. while if *Out* < *NNhg*, *LV* = *Out* + *Unit*

Algorithm 3: Learning Process Algorithm

```
   // LV: Learned value.
1  if report_table.count > 0 then
       // If Outliers exists
2      while !report_table.EOF do
           // While not end of file
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          NNhg ← Nearest-Neighbour in TD
7          Unit ← One measurement unit
           // For example one centimetre
8          if Out > NNhg then
9              LV = Out - Unit;
10         else
11             LV = Out + Unit;
               // If anomaly detection less than its nearest
                  neighbour
12         end
13     end
14 end
```

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 14 and Figure 16. Then *LV* will be passed to *Refinement-process* to refine the related value in KB.

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 language. 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

Algorithm 4: Refinement Process Algorithm.

```
// LV: Learned value
// Att: Attribute name
// P: Action (operator)
while LV Exist do
    // If new success value is learned
    P ← Operator
    LV ← Learned-Value
    Att ← Attribute name
    Select (Att)
    Update KB (Att, LV, P)
    // Temporary Update the KB
end
```

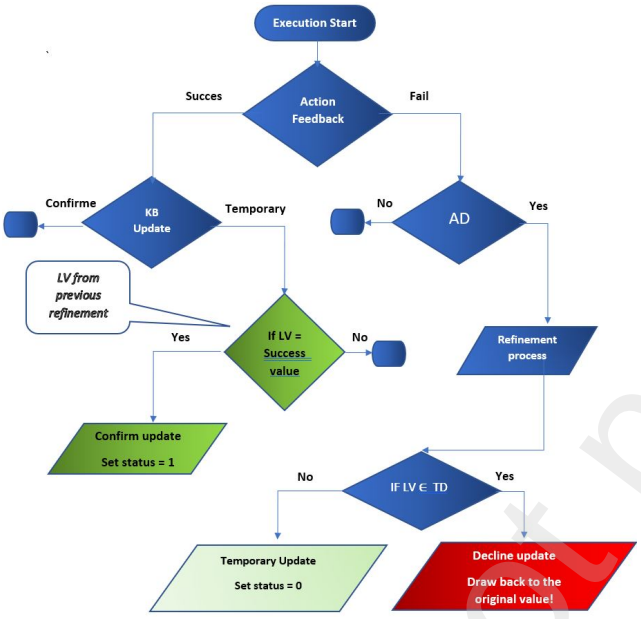

Figure 17: Refinement validation flowchart

as a temporary update to be confirmed or decline by next executions, as Figure 17 illustrates.

Refinement Rules

The number of KB instances to refine in each iteration is domain-specific. Although *Algorithm 2* 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 action attributes. For example, in the grasping task of Nao robot, Section 2, the 'distance' (X_{m1}) and the 'headyawangle' (X_{m2}) attributes are have an inverse relationship ((X_{m1}) (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 17, 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* into *TD* update will be reject, draw back to last confirmed update.

5. Evaluation

A set of experiments has been designed to test the effectiveness of the HDJ architecture in dealing with a flawed model of its environmental capabilities. We explore its behaviour by implementing our running example, explained in Section 2. Solutions to robotic tasks are generated using an automated planning engine, and executed in the real world. We will assume that the robot's model is sufficiently developed to allow the generation of some plan to achieve a task (there has been much research and related techniques available in the automated planning area on fixing flawed or partial models in order to generate plans. Here plan generation is considered generally *without* taking into account plan execution in a real scenario; e.g. see the review paper [26]). In our experiments, the model containing faulty domain knowledge means some of the plans generated will fail when they are executed. We test the effectiveness of HDJ software in reducing the rate of failed executions as a result of knowledge refinement, and can claim a baseline success when this measure reduces monotonically over a period of use.

Each experiment investigates action parameter/s that were incorrectly specified, or ignored, in the KB. Typically, such incorrect parameters and attributes tend to form types of anomalies. The experiments investigate:

- Distance failures: the distance between the robot and the table.
- Angle failures: the angle of the position of the cup on the table to the robot position.
- Combinations of attributes failures: where specific combinations of distance and angle attributes cause failure.

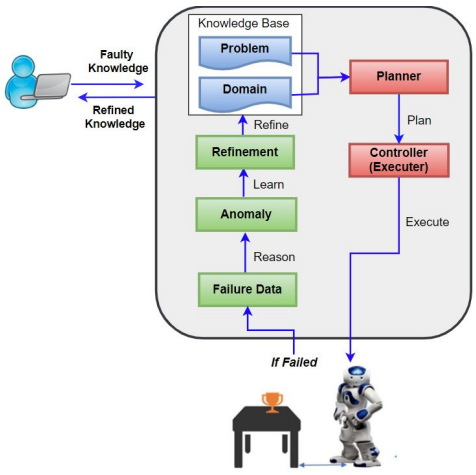

Figure 18: Conceptual model of the gripping task refinement.

- Group of attribute failures: where more than one attribute caused the failure. e,g.,the distance and angle attributes.

5.1. The Testing Procedure

When execution fails to execute an action within a plan that has been generated to solve a given task, the robot needs to realise why it has happened. Then, to avoid any future failure, an update to the KB should be made by the refinement system. Software that implements the anomaly detection and refinement processes above, called ADKRA, reasons about the cause of the failure and then refines the KB based on the outcome, to repair the cause of the failure.

To test this and evaluate ADKRA, and hence HDJ's effectiveness in reducing failure rates, we used the following procedure in each of the four kinds of tests referred to in Section 5:

1. Apply a change to the KB to make it faulty (e.g., give it the wrong maxdis).
2. Generate 100 problem files where the parameter/s in the KB are *randomly* generated to give a different problem each time (e.g., different distances between way points, or/and different initial states of the robot and cup)
3. Generate a plan for each of the 100 problems. To do this we use a general PDDL+ planning engine ENHSP [52], which has been embedded in HDJ.
4. Execute each of the 100 plans.
   - If the execution succeeds, add the execution data to TD.
   - If the execution fails, and we are not using ADKRA, then record the failure.
   - If the plan fails, and we are intending to use ADKRA, then record the failure; run ADKRA to do anomaly detection and knowledge refinement on the KB. Store the newly refined knowledge, for use in later tests.
5. If we are using ADKRA, then after the 100 tests, REPEAT STEP 3, and execute each of the 100 plans generated, recording the total number of failures.

Executing this procedure for each of the four aspects above, we can determine how much using the ADKRA process helps reduce failure rates within the execution of generated planning solutions. This is done by comparing the number of failures without ADKRA, with the number of failures in step 5 above, where we have used ADKRA.

5.2. Development and Technical Setup

The development setup of the experiments includes the developmental aspects of HDJ architecture (the integration of new and existing components) and the design of the current state facts and domain model. We first describe the some aspects of the PDDL language used in our experiments (Section5.3). Then a description of the software development and Middleware integration is provided in Section 5.4.

5.3. The PDDL Version Used

To capture both the symbolic and numerical aspects of robot domains, we chose to employ PDDL version 2.1 which allows numeric fluents, represented in Figure 5. It is inspired by the domain model of 'pouring water between jugs", in PDDLv2.1, by [16].

Numeric fluents are used in the pre-conditions of the 'grip' operator, and the metrics are described in the problem file (Figure 7). The numeric expressions are values associated with tuples of domain objects by domain functions. They allow the setting of different values that construct different solutions to problems of the same domain [16]. Given operator P from the domain model, *PNE* is a primitive numeric expression (*fluent*) of a task 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 PDDL2.1 models contain objects, and actions are represented as parameterised operations which change the status (attribute-values, relations) of objects. Numbers are used as values of attributes of objects and are manipulated through their connections with the objects that are identified in the initial state [16]. In the model of our running example, the functions **dist-to**, **mindis**, and **maxdis**, are referred to as numeric fluents, and are associated with the pre-conditions of the operator **grip**. These fluents are

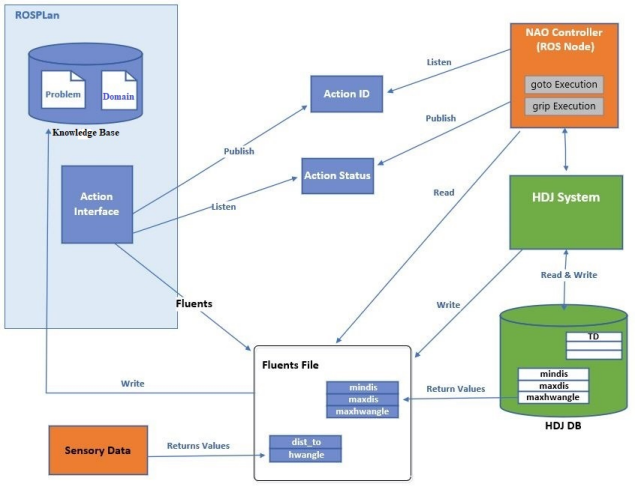

Figure 19: Software and Middleware communications

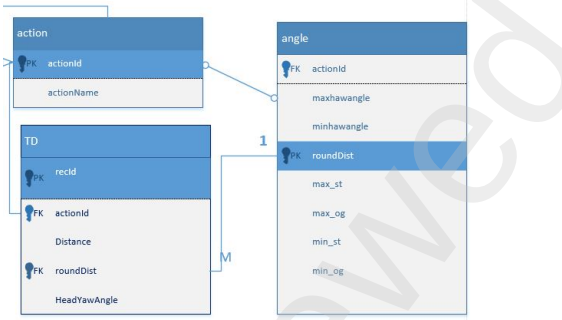

Figure 20: HDJ database : **rounDist** attribute and **Angle** elationship

employed to represent the distance attribute and its permitted range of values. We use a prefix syntax (comparison) for the **grip** operator and its precondition (>=(dist-to ?r ?waypoint) (mindis ?r ?waypoint)). This forms a plan rule we call **dis** which is similar to the distance rule. We also have a **HeadYawAngle** rule where we employ numeric fluents in the functions **hwangle** and **maxhwangle**.

5.4. Software and Middleware

HDJ system development entails implementing a new programme integrated with the existing components of ROS. In this research, we utilise ROS and ROSPLan as components of HDJ architecture.The controller of NAO was developed as a ROS node; this is because the controller communicates with ROSPlan to execute the planned actions, as represented in Figure 19. The ENHSP planner [52] was employed in our experiments to generate plans. The reasoning and refinement algorithms proposed by this research (ADKRA) have been developed and embedded in the context of NAO robot and HDJ architecture.

The HDJ system utilises the SQLite database to store and retrieve knowledge that is used to construct the initial state of the problem instance. This database is created to include structured information of the KB such as the actions parameters and all related information including the relationships between the attributes and correlated attributes. This includes the operators' pre-conditions, plan rules and even the representations of the targeted object. This provides the flexibility to customise the fluents values before passing them to the system. Figure 20 shows the representation of the relationship between the **HeadYawAngle** rule and the **dis** rule as in the database structure. The sensory data holds values of action attributes such as the current distance to the object and its location. Furthermore, the information of the targeted object, the red cup, is also processed during the instanstiation process, [53], as it is used to detect the object and calculate the distance and the angle attributes. The Python programming language was used for controller development.

In the domain model, represented in Figure 5, we employ numeric fluents in the new pre-conditions of the 'grip' operator. As represented in Figure 19, all values of fluents are saved in an external file by the HDJ system and are passed to the problem file through the instantiation process and to the system through the execution stage. The initial values of **mindis** and **maxdis** are calculated and processed by the HDJ system and are stored as an external file to be read and retrieved by the system. Meanwhile, the values of **dist-to**, and **hwangle** are sensory data (actual distance and angle values) that are measured online during execution stages.

5.5. Experimental Setup

This section describes the experimental setup for four types of experiments. Each setup is described based on investigations conducted on each failure type, and the implementation setup for each experiment.

Experiments Investigating Distance Failures

These experiments investigate the distance parameter as a cause of failure, particularly the distance between the robot and the cup as seen in Figure 2. For the successful execution of gripping, the distance must be in the range of 15 to 23 centimetres (see Figure 2). If this range is extended (e.g., 27cm>dis>15cm), some failures would be caused during execution because these extended values (24, 25, 26 and 27) of the distance will cause the execution to fail.

In order that some failures would be caused during execution, faulty knowledge was given to the system in this set of experiments. This consisted of an erroneous distance precondition as follows: the correct range of (23cm>dis>15cm) was extended to (27cm>dis>15cm). For each experimental episode the

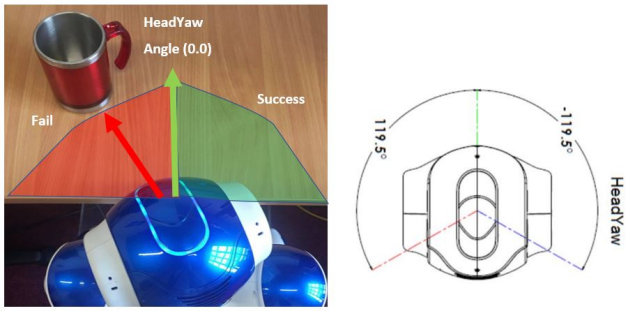

Figure 21: Success and fail ranges of 'HeadYawangle', to grip using the right hand of the NAO robot.

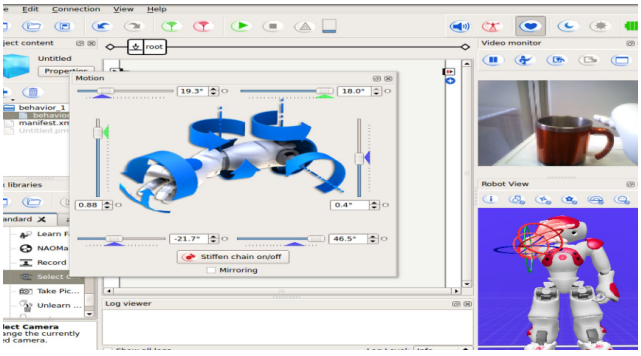

Figure 22: The joint values of NAO's arm in grasping position at the maximum degree towards the body.

initial position of the robot is changed, and then the generated solution plan for this new gripping task is checked for success or failure in execution. If the task plan fails, the KB is refined using ADKRA. Figure 28, in Appendix 7, describes the starting input and the final output (which is produced after a number of refinements) of these experiments.

Experiments Investigating Angle Failures

These experiments investigate the angle parameter as a cause of failure. For a successful execution of the gripping task, by the right hand, this angle should be in a range of 0.0° to 25° (see Figure 21). If this range is extended to a wider one (e.g., 25°>HeadYawAngle>-25°), some failures would be caused during execution because these extended values (greater than 0°) will cause the execution to fail.

The NAO robot has 25 DOF of which 14 are for the upper part including its trunk, arms, and head. The rotation axis of its arms, and head joints is inclined at 45○ towards the body. 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 DOF for the hand's grasping.

The available DOF of the arms of the NAO robot limits its abilities in grasping. The angle of the object's position is a critical condition for successful grasping. As seen in Figure 22, the maximum rotation of the 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.

In order to cause some failures during execution, faulty knowledge was given to the system in this set of experiments. This knowledge contained an erroneous HeadYawAngle precondition as follows: The correct range of (0.0° > HeadYawAngle > -25°) was extended to (0.0° > HeadYawAngle > -29°). For each experimental episode, the initial position of the robot was not changed, but the angle of the cup position on the table was randomly changed (see Figure 21) and the distance was fixed to 18 cm; Then the generated solution plan for this new gripping task was checked for success or failure in execution. If the task plan failed, the KB is refined using ADKRA. Figure 29 , in Appendix 7, describes the starting input and the final output (which is produced after a number of refinements) of these experiments. Figure 20 shows the success data of 'HeadYawangle' when the distance = 18 centimetres.

Experiments Investigating Attribute Combinations Failures

These experiments investigate the particular combinations of action attributes (Distance and HeadYawAngle) that cause executions to fail. For successful grasping, our experiments have shown that each distance point accepts a specific range of angle. The error in the KB is to ignore the associated attribute, which is the inverse relationship between the distance and angle parameters of the 'grip' action. As seen in Figure 23, when the **Distance** increases, the **HeadYawAngle** range is narrowed and goes towards a **0.0** angle. Figure 24 represents the inverse relationship between the **HeadYawAngle** attribute and the **Distance** attribute. For example, the 15 centimetre distance accepts the full range (0.0>HeadYawAngle>-25), for the right hand. However, the 20 centimetre distance accepts a shorter range of angle of about (0.0>HeadYawAngle>-12).

For this reason, particular combinations (of the **Distance** and the **HeadYawAngle** attributes) cause execution failure. For example, despite the fact that the ranges (0.0>HeadYawAngle>-25) and (23cm>dis> 15cm) are considered successful ranges of distance and angle attributes, some values within these ranges, if combined, will cause execution to fail. A 'Collective' anomaly is a group of attribute instances that appear isolated compared to the rest of the data-set. A specific case of 'Collective' is when each single attribute is not an anomaly but as a group appears together as an anomaly [11]. These experiments test such cases.

In order to cause some failures during execution, faulty knowledge was given to the system in this set of experiments. In the domain model part

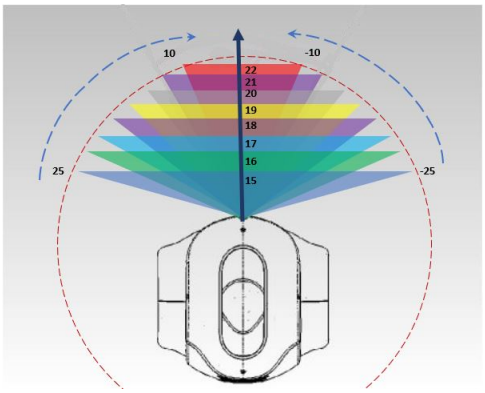

Figure 23: The relationship of the distance and angle attributes.

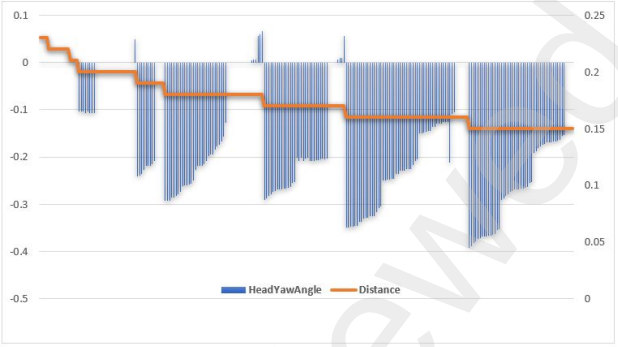

Figure 24: TD distribution. Distance and angle data

of the KB, represented in Figure 5, we have two preconditions that accept a range of values of the **Distance** and **HeadYawAngle** attributes. We set the ranges of both rules **HeadYawAngle** and **dis** to (0.0>HeadYawAngle>-25) and (23cm>dis>15cm); These ranges are considered to be the range needed for success. However, we know that within this range specific cross values will fail the execution; this is due to ignoring the inverse relationship between the **Distance** and **HeadYawAngle** attributes, which is considered as a fault in the KB. For each episode, we changed both the initial position of the robot to the table, and the angle of the cup position on the table as well. Then the generated solution plan for this new gripping task was checked for success or failure in execution. If the task plan failed, knowledge is refined according to Algorithm 1. Figure 30 , in Appendix A, describes the starting input and the final output (which is produced after a number of refinements) of these experiments

The correlation between the angle and the distance attributes is considered an attribute of the action 'goto' that needs to be checked as well when execution failed. To detect such an anomaly, we may have to create a new attribute (a correlated attribute [15]). Such that, $X3 = X1\ X2 + X1$, where $X1$ is the value of the **Distance** attribute and $X2$ is the value of **HeadYawAngle** attribute. This approach can help detect unusual combinations of values that would not have possibly been detected by earlier steps of Algorithm 1.

For each failed execution, the ADKRA processes (Algorithm 1) learns values LV and used them to refine the module. We stated in Section 7 that the relationships of the domain's attributes determine which KB attribute is to be refined. So, for this particular instance, the KB attribute (**maxhwangle**) to refine is based on its relationship with the **Distance** attribute. The angle is classified as a 'slave' relationship, as seen in Figure 20, with the distance attribute. The refinement system is set to only learn and refine the **HeadYawAngle** attribute.

Experiments Investigating Failure of a Group of Attributes

These experiments investigate the failure of multiple attributes (more than one attribute specified incorrectly) e.g, angle and distance attributes, which cause executions to fail. For example, a distance of 24 centimetres is considered a failure value. -27 degree of the angle is another failure value. A 'Collective' anomaly is where a group of attributes - instances appears isolated compared to the rest of the data-set [11]. This experiment is the general case of 'Collective' anomalies, when detecting two or more attributes as being anomalies.

In order to cause some failures during execution, the faulty knowledge given to the system in this set of experiments contained erroneous distance and HeadYawAngle preconditions as follows: The correct range of (23cm>dis>15cm) was extended to (25cm>dis>15cm) and the correct range of (0.0>Head YawAngle>-25) was extended to (0.0>HeadYawAngle >-27). These ranges are both incorrect and lead to failure during the execution. For each experimental episode, the distance of the robot to the cup was randomly changed, while the angle of the cup position was fixed at -20°. Then the generated solution plan for this new gripping task was checked for success or failure in execution. If the task plan failed, the knowledge was refined using ADKRA. Figure 31 , in Appendix A, describes the starting input and the final output (produced after a number of refinements) of these experiments.

5.6. Experimental Process and Results

The test results shows the results of running the Testing Procedure specified above. The rate of failure decreases over time as the parameters are adjusted by the ADKRA subsystem. The overall results of these tests are represented in Figure 25. Table 1 represents the accuracy of predictions, false negative rate (FNR), and true positive rate (TPR) made by robot.

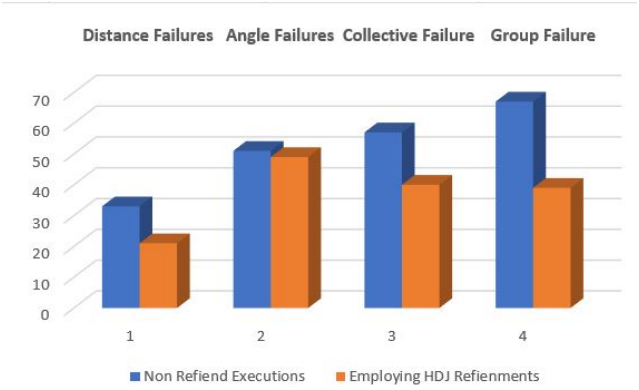

Figure 25: Field executions with and without knowledge refinement by ADKRA

Table 1

The Overall Accuracy of Predictions by HDJ Reasoner

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%

The experiments demonstrate the efficiency of our approach in refining the KB of the task planner, which reduces plan execution failures after repeated use. For example, by the end of the distance experiment, the ADKRA successfully corrected the values of the distance rule. The instance value of the maximum allowed distance (maxdis ?r ?waypoint) was corrected to be 23 centimetres (Figure 8). ADKRA reasons and detects anomalies efficiently and can refine the knowledge for successful future executions.

They also demonstrate the efficiency of the update validation process, which was demonstrated in the angle failure experiments. The validation process rejected all predictions made by ADKRA because they relied on faulty knowledge that neglected the correlated attribute of distance and angle parameters. The domain model neglected the rational relationship between the 'Distance' and the 'HeadYawangle' attributes, where each distance point accepts a specific range of angle values, which causes incorrect predictions by ADKRA.

As the relationships between the operator's parameters have to be represented, missing such representation can cause action executions to fail. In particular, in the angle failure experiment, the refinement process behaved like a domain validator, which is considered an extra advantage of using AD. The refinement process uncovers incorrect predictions that were caused by inaccurate kmowledge.

In other words, our experiments demonstrate that accurate knowledge can overcome issues related to inaccurate domain models. However, despite this fault, the HDJ (specifically in the group attribute failures) 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 was because the HDJ database considers the relationship between these attributes.

Furthermore, the specific case of the 'Collective' anomaly (whre 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 where such a type of anomaly can be used to develop a tool to discover such hidden relationships in domain models. Employing the 'Collective' AD type gives the HDJ the ability to discover hidden relationships that the current domain model lacks. Thus, it can indicate faults in the domain representation.

Moreover, employing the 'Collective' AD method to over come the model demonstrated that employing multiple methods of AD is beneficial and can overcome the problems or weaknesses of each method.

Furthermore, the experiments also demonstrated that real application brings about complications related to the real environment and sensor issues. The experiments indicated that NAO does not measure the distance to objects accurately 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 [54], in addition to the hardware specifications issue and the method that estimates the distance, is another factor of inaccurate distance measurements. Different techniques for depth estimation exist in which each has different outcomes and drawbacks.

The accuracy of sensory data is critical for successful predictions. This is due to the problem of overlapped edges between the accumulated data of the successful executions and the data of failed executions. The inaccuracy of the distance data led to overlapped data edges between the successful execution data and the fail data, as seen in Figure 26. For example the distance value of 23 centimetre appears in both datasets. While it appears as a successful attribute value in TD at the same time the same value can cause the execution to fail. Its appearance in both sets prevents the robot from making successful predictions, as it does not consider it as an anomaly in TD.

6. Future Work

In this study, we utilised a simple (non-temporal) PDDL model and the efficiency of ADKRA was tested and approved with 'Point' and 'Collective' anomaly types. We also employed density based AD technique to detect anomalies in the action attribute values.

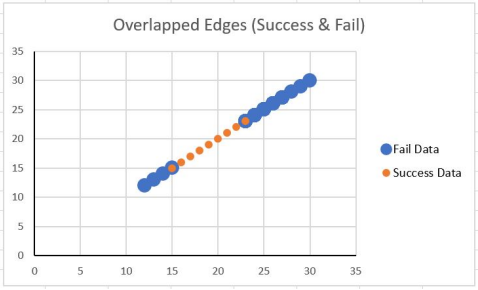

Figure 26: The distance datasets.

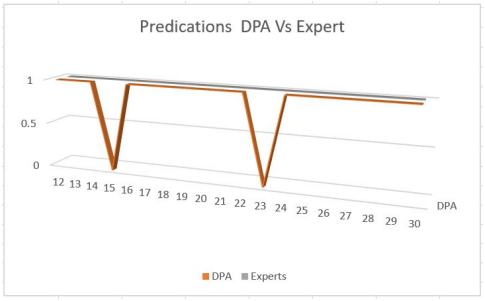

Figure 27: Graphical representation of the correct predictions made by the robot, in the distance experiment.

However, the low quality of NAO cameras led to overlapped data edges, which reduces the performance of the reasoning system.

Going forward, we aim to broaden the scope of knowledge that HDJ can refine and conduct more experiments across various scenarios. This includes using a temporal-PDDL model with "Contextual" AD, which is usually employed for time series models, in addition to the "Point" and "Collective" AD types. We plan to examine other AD approaches and techniques, potentially combining multiple methods to overcome each one's limitations and take advantage of their strengths.

A hybrid AD technique, such as using vision recognition AD in conjunction with density-based techniques, is also being considered. For instance, the vision recognition AD could identify differences in the features of a red cup while the density-based techniques detect differences in the object's geometric information.

Additionally, we plan to improve the system by utilising high-quality sensory systems, which can enhance the prediction accuracy. Research into methods such as ANN-based methods to detect and define correlations between action attributes will also be conducted. This can help detect "Collective" anomalies, address inaccurate domain models, and lead to more autonomous refinement systems.

7. Conclusions

In this paper, we have described the HDJ architecture, which we have implemented on a physical NAO robot. The original contribution of the paper is the description, implementation and evaluation of the components making up ADKRA. This embodies anomaly detection to find the cause of action failure of a real robot, and then knowledge refinement to resolve the anomaly by replacing faulty knowledge, with more accurate knowledge, within the robot's action-related knowledge base. We have demonstrated it to be capable of autonomously detecting an anomaly, and refining knowledge to overcome it. We can conclude that ADKRA contributes to bridging that gap between planning and execution. It reasons and detects anomalies efficiently and refines the PDDL model, so that in future plans a robots plan are less likely to fail.

The refinement process effectively validates the system predicates and the refined knowledge before setting it as permanent. It is capable of exposing inaccuracy in the domain model, and extracting accurate knowledge from inaccurate domain models; this is due to its ability to specify and customise knowledge with the help of its rational database.

This research 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 or knowledge refinement approach, and we can implement more than one approach as the latest trend in anomaly 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.

Appendix A. I/O

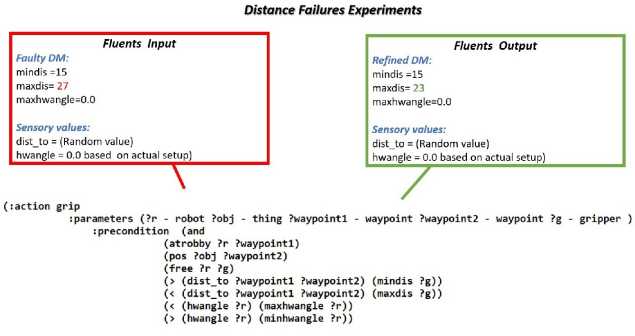

Figure 28: Example of an I/O for a distance failure experiment

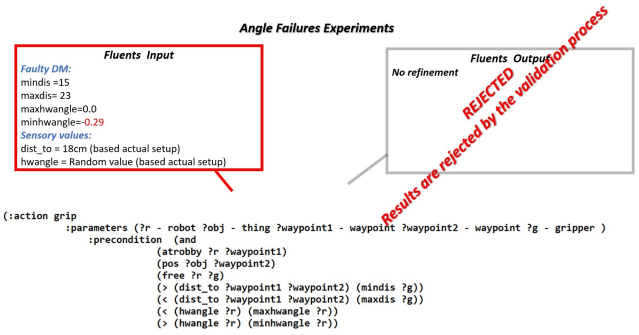

Figure 29: Example of an I/O for an angle failure experiment, The validation process rejected predictions made by reasoning system because the correlation attribute is neglected in the domain model

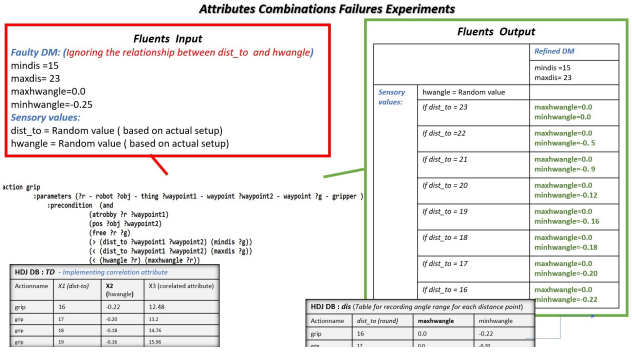

Figure 30: Example of an I/O for a combination failure experimen

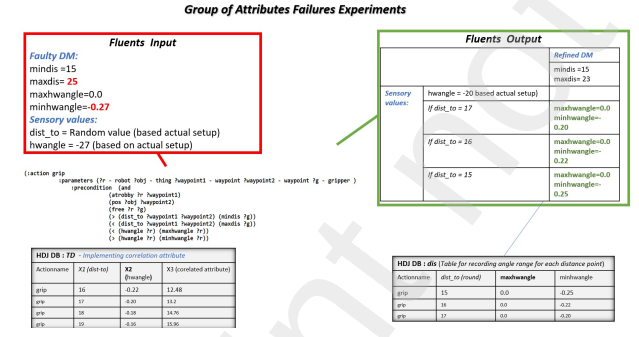

Figure 31: Example of an I/O for a group of attributes failure experiment.

References

[1] F. Ingrand, M. Ghallab, Deliberation for autonomous robots: A survey, Artificial Intelligence 247 (2017) 10–44.

[2] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtos, M. Carreras, Rosplan: Planning in the robot operating system., in: ICAPS, 2015, pp. 333–341.

[3] F. Stulp, M. Beetz, Refining the execution of abstract actions with learned action models, Journal of Artificial Intelligence Research 32 (2008) 487–523.

[4] R. C. Arkin, Behavior-based robotics, MIT press, 1998.

[5] Y. Gil, Acquiring domain knowledge for planning by experimentation, Tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE (1992).

[6] T. Munzer, M. Toussaint, M. Lopes, Efficient behavior learning in human–robot collaboration, Autonomous Robots 42 (5) (2018) 1103–1115.

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

[8] A. Zimek, E. Schubert, Outlier Detection, Springer New York, New York, NY, 2017, pp. 1–5. doi:10.1007/978-1-4899-7993-3_80719-1.
URL https://doi.org/10.1007/978-1-4899-7993-3_80719-1

[9] V. Hodge, J. Austin, A survey of outlier detection methodologies, Artificial intelligence review 22 (2) (2004) 85–126.

[10] M. R. Smith, T. Martinez, Improving classification accuracy by identifying and removing instances that should be misclassified, in: The 2011 International Joint Conference on Neural Networks, IEEE, 2011, pp. 2690–2697.

[11] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM computing surveys (CSUR) 41 (3) (2009) 15.

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

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

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

[15] T. Næs, E. Risvik, Multivariate analysis of data in sensory science, Elsevier, 1996.

[16] M. Fox, D. Long, Pddl2. 1: An extension to pddl for expressing temporal planning domains, Journal of artificial intelligence research 20 (2003) 61–124.

[17] A. Kirsch, M. Beetz, Training on the job—collecting experience with hierarchical hybrid automata, in: Annual Conference on Artificial Intelligence, Springer, 2007, pp. 473–476.

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

[19] R. Hornung, H. Urbanek, J. Klodmann, C. Osendorfer, P. Van Der Smagt, Model-free robot anomaly detection, in: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2014, pp. 3676–3683.

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

[21] P. Crook, G. Hayes, et al., A robot implementation of a biologically inspired method for novelty detection, in: Proceedings of the Towards Intelligent Mobile Robots Conference, 2001.

[22] P. A. Crook, S. Marsland, G. Hayes, U. Nehmzow, A tale of two filters-on-line novelty detection, in: Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292), Vol. 4, IEEE, 2002, pp. 3894–3899.

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

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

[25] T. Krajnik, J. P. Fentanes, G. Cielniak, C. Dondrup, T. Duckett, Spectral analysis for long-term robotic mapping, in: 2014 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2014, pp. 3706–3711.

[26] S. Jiménez, T. De la Rosa, S. Fernández, F. Fernández, D. Borrajo, A review of machine learning for automated planning, The Knowledge Engineering Review 27 (4) (2012) 433–467.
[27] W.-M. Shen, Discovery as autonomous learning from the environment, Machine Learning 12 (1) (1993) 143–165.
[28] P. Doherty, J. Kvarnström, F. Heintz, A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems, Autonomous Agents and Multi-Agent Systems 19 (3) (2009) 332–377. doi:10.1007/s10458-009-9079-8.
[29] M. Beetz, L. Mösenlechner, M. Tenorth, Cram—a cognitive robot abstract machine for everyday manipulation in human environments, in: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2010, pp. 1012–1017.
[30] B. G. Weber, M. Mateas, A. Jhala, Learning from demonstration for goal-driven autonomy., in: AAAI, 2012.
[31] X. Wang, Learning planning operators by observation and practice, Ph.D. thesis, Carnegie Mellon University (1996).
[32] S. Jiménez, F. Fernández, D. Borrajo, Integrating planning, execution, and learning to improve plan execution, Computational Intelligence 29 (1) (2013) 1–36.
[33] S. Rockel, B. Neumann, J. Zhang, K. S. R. Dubba, A. G. Cohn, Š. Konečnỳ, M. Mansouri, F. Pecora, A. Saffiotti, M. Günther, et al., An ontology-based multi-level robot architecture for learning from experiences, in: Designing Intelligent robots: Reintegrating AI II. AAAI Spring Symposium-Technical Report, AAAI Press, 2013, pp. 52–57.
[34] S. Karapinar, D. Altan, S. Sariel-Talay, A robust planning framework for cognitive robots, in: Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence, 2012.
[35] A. Lindsay, S. Franco, R. Reba, T. L. McCluskey, Refining process descriptions from execution data in hybrid planning domain models, in: Proceedings of the International Conference on Automated Planning and Scheduling, Vol. 30, 2020, pp. 469–477.
[36] J. L. Ambite, C. A. Knoblock, Planning by rewriting: Efficiently generating high-quality plans., Tech. rep., UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFORMATION SCIENCES INST (1997).
[37] N. Hawes, C. Burbridge, F. Jovan, L. Kunze, B. Lacerda, L. Mudrova, J. Young, J. Wyatt, D. Hebesberger, T. Kortner, et al., The strands project: Long-term autonomy in everyday environments, IEEE Robotics & Automation Magazine 24 (3) (2017) 146–156.
[38] V. Verma, T. Estlin, A. Jónsson, C. Pasareanu, R. Simmons, K. Tso, Plan execution interchange language (plexil) for executable plans and command sequences, in: International symposium on artificial intelligence, robotics and automation in space (iSAIRAS), Citeseer, 2005.
[39] V. A. Ziparo, L. Iocchi, P. U. Lima, D. Nardi, P. F. Palamara, Petri net plans, Autonomous Agents and Multi-Agent Systems 23 (3) (2011) 344–383.
[40] M. Leonetti, L. Iocchi, Learnpnp: A tool for learning agent behaviors, in: Robot Soccer World Cup, Springer, 2010, pp. 418–429.
[41] S.-O. Bezrucav, B. Corves, Improved ai planning for cooperating teams of humans and robots, in: Proceedings of the Planning and Robotics (PlanRob) Workshop—ICAPS, 2020.
[42] L. P. Kaelbling, T. Lozano-Pérez, Hierarchical task and motion planning in the now. in 2011 ieee icra (2011).
[43] S. Cambon, R. Alami, F. Gravot, A hybrid approach to intricate motion, manipulation and task planning, The International Journal of Robotics Research 28 (1) (2009) 104–126.
[44] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, P. Abbeel, Combined task and motion planning through an extensible planner-independent interface layer, in: 2014 IEEE international conference on robotics and automation (ICRA), IEEE, 2014, pp. 639–646.
[45] E. Fernandez-Gonzalez, E. Karpas, B. Williams, Mixed discrete-continuous planning with convex optimization, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 31, 2017.
[46] M. Cashmore, M. Fox, D. Long, D. Magazzeni, A compilation of the full pddl+ language into smt, in: Workshops at the Thirtieth AAAI Conference on Artificial Intelligence, 2016.
[47] H. Kautz, B. Selman, Pushing the envelope: Planning, propositional logic, and stochastic search, in: Proceedings of the national conference on artificial intelligence, 1996, pp. 1194–1201.
[48] J. Rintanen, Madagascar: Efficient planning with sat, The 2011 International Planning Competition 61 (2011).
[49] J. Hoffmann, The metric-ff planning system: Translating"ignoring delete lists"to numeric state variables, Journal of artificial intelligence research 20 (2003) 291–341.
[50] A. Coles, A. Coles, M. Fox, D. Long, Forward-chaining partial-order planning, in: Proceedings of the International Conference on Automated Planning and Scheduling, Vol. 20, 2010, pp. 42–49.
[51] K. P. Murphy, Machine learning: a probabilistic perspective, MIT press, 2012.
[52] Enrico Scala, The enhsp planning system, https://sites.google.com/view/enhsp/, accessed: 2022-05-06 (n.d.).
[53] kCL planning, Rosplan overview, *ROSPlan*(online), accessed: 2021-01-03 (n.d.).
URL http://kcl-planning.github.io/ROSPlan/documentation/
[54] L. Zhang, H. Zhang, H. Yang, G.-B. Bian, W. Wu, Multi-target detection and grasping control for humanoid robot nao, International Journal of Adaptive Control and Signal Processing 33 (7) (2019) 1225–1237.