Reasoning by Anomaly Detection to Improve Planning Robustness for Autonomous Robots in Changing Environments

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

Abstract

The requirement for autonomous robots to exhibit higher-level cognitive skills by planning and adapting in an ever changing environment and situation is indeed a great challenge for the AI community. In robotics task planning, the typical use of automated planners entails using fixed action descriptions that neglect the subtle differences that appear when environment change occurs or where behavioural capabilities are not adequately captured. Failure of action at execution time can be a result of a mismatch between the actions in the abstract plan and the low-level commands, because knowledge in the abstract level is incorrect or not sufficiently accurate. The choice of parameters in the low-level commands corresponding to the abstract actions, and their related information can strongly affect the success of execution. Incorrect choices may cause plan execution to fail. For a more robust robot capable of adapting to the changing environment, the goal of this project is to bridge the gap between abstract plans and robot action execution. Our approach is to extend a high-level planning platform (ROSPlan) to help create more robust planning systems by using theory refinement (TR) techniques to develop intelligent robot behaviour based on experience. Refinement will involve reasoning over action execution failure using Anomaly Detection (AD) techniques. The general aims of our research are to make it possible for robot task planning to adapt to changing situations towards long-term autonomy, but still retaining some abstract theory of the environment which can be used as an explanation for behaviour. This paper reports on a crucial step in that theory refinement process: determination of the cause of failure in order to drive the changes.

1 Introduction

Autonomous robots in largely unknown environments (e.g., exploration robotics) require systems to act deliberately (Ingrand and Ghallab 2017). In a planning system for a dynamic world, adapting to changing situations while knowledge gathering is a major demand for robust autonomous behaviour. One of the biggest drawbacks of robotic control that embeds a task planner is that the knowledge that the task planner relies on to solve problems changes, making it difficult to imbue this kind of autonomous system with persistent autonomy (Cashmore et al. 2015b).

The plan execution strategy must account for the action/plan failure, which results from ignorance or change (Cashmore et al. 2015a). The success of plan execution can only be achieved through a combination of human-specified knowledge and robot learned knowledge in robot controllers (Stulp and Beetz 2008), with human designers needing to assign action abstractions offline. It is unfeasible to pre-program such robotic applications by predicting, at the design stage, all possible courses of actions on demand (Ingrand and Ghallab 2017).

Parameterisable actions have been proposed and proven as an efficient approach to bridge 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 (Stulp and Beetz 2008; Arkin 1998). For example, it is more flexible to program the action GoTo(Pos) than GoTo-Centre, where "Pos" is a parameter that accepts different values, including the centre (Stulp and Beetz 2008).

However, planners in planning systems treat actions at a level of abstraction that neglects their subtle differences. This neglect leads to the producing sub-optimal behaviour as the planning system is unable to tailor the actions to the execution contexts (Stulp and Beetz 2008). The reason why the planning system ignores action details is to manage action selection and to make planning tractable (Müller and Beetz 2006). By ignoring action parameters, task planning produces an abstract plan that leaves out many of the details of action execution. This often produces insufficient behaviour which can lead to the plan execution failing (Stulp and Beetz 2008).

Our research aims at moving towards the goal of long-term autonomy (Kunze et al. 2018). Long-term autonomy can be achieved by making the system more robust through experiential learning: failures and successes drive the improvement of the pre-programming multi-level representation. The research is based on a multi-level abstraction platform, outlined in Figure 1. On top is the most abstract level being the *Planning Domain Definition Language* (PDDL). Plans are processed and transformed to low-level commands

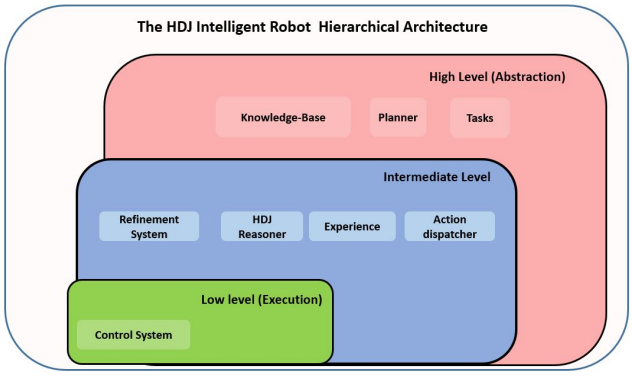

Figure 1: HDJ Intelligent Robot Hierarchical Architecture

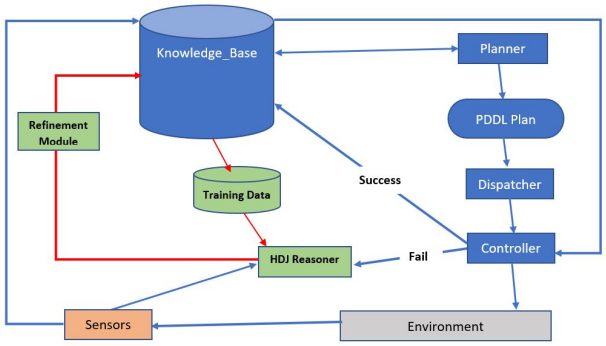

Figure 2: Overview of the HDJ framework. A combination of ROSPLan components (Blue boxes) and HDJ components (Green boxes)

by the intermediate level and are finally executed by the controller (last level). In this work, we propose the HDJ robotics planning framework. It is to extend the standard robotic framework, ROSPlan, by HDJ components, represented in Figure 2.

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

Examining the differences between the information related to a failed action and previous experience could lead to finding out the cause of that failure. The outcomes will be used to design a more robust planning execution, by enriching the knowledge-base that the task planner relies on to produce plans, thereby producing robots adaptable to changing environments.

The novel aspect of our contribution is the addition of an autonomous reasoning capability performed by our proposed *Discrimination Process Algorithm* (DPA, section 3.2). It employs Anomaly Detection (AD) techniques to recognise items of data that differ from the previous successful executions. AD techniques have been employed in different fields, such as in exposing bank fraud or medical problems (Hodge and Austin 2004) and usually, anomalous items indicate or reveal the relevant issues, problems, faults or errors (Hodge and Austin 2004).

Our idea 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 feedforward and feedback components (Schmidt 1975; Glover 2004). For example, picking a pan up does not need knowing its exact weight in advance; humans can determine this easily by picking it up and slightly increasing the exerted force until the pan leaves the surface. For a similar scenario, the HDJ extending components refine the original domain model knowledge that is relevant to the differences extracted by DPA. It employs a learning process to learn new data and these learned values will be used to update the knowledge that the planner relies on to produce plans.

The work reported here is carried out as part of the first author's PhD thesis and has resulted in the HDJ framework. Although our method embedded in HDJ is generic, this research is carried out using a real physical NAO robot. NAO is a small humanoid robot about 60 CM tall, with features including with 25 degrees of freedom (DOF). Packed with sensors and it can walk, speak, and recognise faces and objects. We embedded the NAO with a task planner and software which translates generated plans into actions that can be executed and can generate real states using sensor information.

In this paper, we describe how to extend the robotics planning framework with the HDJ software layer. The current version implements an algorithm to store planning experiences and uses that to identify failures in operation that require changes to the planning knowledge in order to make the system more robust (section 3). Currently, we are extending this to create a full theory refinement algorithm to automatically adjust the representation of that knowledge to fit the robot's experiences.

The following sections provide further details. The next section is the HDJ system overview and hierarchical architecture, while section 3 covers the system mechanism and algorithms. The evaluation in section 4 demonstrates and evaluates the method as embedded in the NAO robot's architecture. The last section is the Conclusion.

2 System Overview

The HDJ Intelligent Robot Hierarchical Architecture is inspired by previous work on robotics architectures. For example, the layering used in the functional architecture within a UAV (Doherty, Kvarnström, and Heintz 2009). It 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. The top layer is the most abstract level and traditionally con-

tains the aspects of the explicit Knowledge-Base (KB) of the robot architecture, along with high-level reasoning functions such as planning and high-level tasks. The Intermediate layer is a reactive layer consisting of the action dispatcher and the HDJ's components. Action corresponding parameters are processed here. The last layer is the control system where actions are transformed into the most low-level motor commands and skills to be executed, and feedback of the effects is returned.

The framework we suggest, represented in Figure 2, extends ROSPlan (Cashmore et al. 2015a) by HDJ autonomic components. ROSPlan is a framework for carrying out task planning in ROS, linking existing ROS components to planning tools and integrating planners with the ROS system. ROSPlan is maintained by KCL-Planning (Planning at King's College London) and employs the Planning Domain Definition Language PDDL 2.1 and POPF, a temporal metric automated planning engine, commonly used in AI planning. The HDJ combines bottom-up processes from sensors to meaningful data, with top-down mechanisms such as the focus of attention, reasoning with sensor models and planning with sensing actions (Ingrand and Ghallab 2017).

2.1 HDJ Intelligent Robot Hierarchical Architecture

High Level: planner, Knowledge-Base and task Planning is the core deliberation functions in autonomous systems (Ingrand and Ghallab 2017). Plans are produced as a high-level instruction. The planning system synthesises a plan and keeps it simple to be traceable by the planner by abstracting it away from the high-level action parameters (Müller and Beetz 2006).

```
; Plan found with metric 0.001
; States evaluated so far: 3
; Time 0.00
0.000: (goto nao wp0 wp1)  [0.001]
0.001: (grip nao redcup wp1 grp)  [0.001]
```

Figure 3: Planner output (a simple task plan). The actions: 'goto' and 'grip' are causally linked and represented while neglecting their features and attributes.

Figure 3 shows an example of the planner output. It is a simple task representation that includes only two actions ('goto' and 'grip'). The actions are listed and causally linked while neglecting their features and parameters. The planner produces plans using a domain model (DM) and problem instance that are stored in the Knowledge-Base (Cashmore et al. 2015a). It needs to have up-to-date knowledge of action schemes that represents the robot's knowledge of the effects of its actions as well as knowledge of objects, their attributes and the immediate environment where the robot is operating.

The Knowledge-Base stores a PDDL domain file, object instances, facts and data that are not part of the PDDL model. The KB is designed to collate the up-to-date model of the world and is continuously updated by sensory data. (Cashmore et al. 2015a). It stores PDDL objects such as 'redcup' in (see section 3.2), facts, functions, a model of the world and goals. In addition, it includes facts and data that are not part of the PDDL model(Cashmore et al. 2015a). The planner in ROSPlan is permitted to query the Knowledge-Base for information related to a planning problem (initial state and goals) or to query the Knowledge-Base for environment changes (updated model world) that are relevant to the current task (Ingrand and Ghallab 2017; EmaroLab 2017). For example, in the kitchen domain (section 4) in the operator 'goto' , the objects of types 'waypoint' and 'robot' 0f the predicate (at-robby ?r ?waypoint). If any instance(of these objects) is updated or removed, the Knowledge-Base will notify the planner to produce a new plan.

```
(:action GoTo
    :parameters  (?r - robot ?from ?to - table)
    :precondition (and  (at-robby ?r ?from))
    :effect (and  (at-robby ?r ?to)
                  (not (at-robby ?r ?from))))

(:action grip
    :parameters (?r - robot ?obj - object ?table - table ?g - gripper)
    :precondition  (and  (at ?obj ?table) (at-robby ?r ?table) (free ?r ?g))
    :effect (and (carry ?r ?obj ?g)
                 (not (at ?obj ?table))
                 (not (free ?r ?g))))
```

Figure 4: A segment of DM shows the pre-condition (at-robby ?r ?table) that implement the plan rule (27cm>dis>15cm)

Tasks are given to the robot in the form of a PDDL Problem file to be executed. Operators libraries (DM) consist of a group of operators that are usually applied within a particular domain. Figure 4 shows a segment of the kitchen domain. For complex task execution, operators are joined and concatenated by planners. For example, the kitchen domain (Müller and Beetz 2006) contains a set of operators that are frequently used in kitchens. The task in Figure 3 is to grip and move a cup from a table to another one. A navigation ('goto') and grasping ('grip' or 'drop') operators are employed in this scenario. For 'grip the cup' task, the planner has to combine two operators ('goto' and 'grip'). Figure (5) represents the task scenario.

Intermediate Level: Action Dispatcher, Experience, HDJ Reasoner, and Refinement System Adequate integration between planning and acting is critical for successful plan execution (Ingrand and Ghallab 2017). The intermediate level includes the action dispatcher and action interface, and the extension components of HDJ (the green boxes Figure 2). The action dispatcher in charge of the transition of the operators into action models, feed-forward process. The HDJ components process the feedback that comes from the controller (the lowest level) for learning action models and modifying the state instances in the Knowledge-Base.

At this level, action parameters corresponding to each action are retrieved and processed by the action interface as part of dispatching. In ROSPLan the dispatcher selects actions of the current plan (task) and dispatches them to the low-level action space (in the next layer) to be executed by

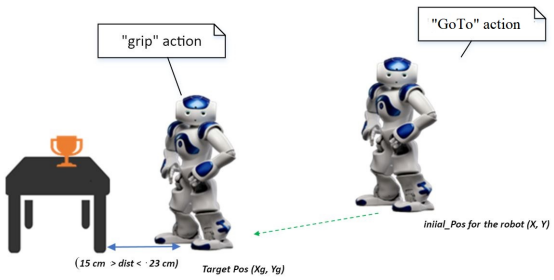

Figure 5: Kitchen Scenario Task: “GoTo” and “grip” actions are combined.

the control system (see Figure 1). Actions are situation dependent. In the control systems of autonomous robots, actions include parameters that accept changed values which enables the implementation of the same operator in different situations (Stulp and Beetz 2008). In ROSPLan each PDDL operator is linked to a ROS action message to be dispatched to the lower-level controllers. The action dispatcher extracts real values from the Knowledge-Base before the action is sent to the lower controller.

The HDJ’s components implement the refinement process. The HDJ reasoner implements DPA (explained in section 3) and is in charge of defining the learned values (Vs). The refinement model uses the learned values to update the knowledge-Base. Datastores/Experience stores the training data (TD) which is the history of successful executions and which typically, this maps actions model parameters and contains the necessary information and all relevant information that is required by the implementation of DPA.

The training data, which is considered as the normal data, is used by the anomaly detection methodology. If the feedback coming from the control system returns ”execution failed”, the DHJ reasoner combines the information of the failed execution with the training data and then calls DPA. The reasoner considers the anomalies as being suspicious of causing the failure and uses the output of the DPA to learn new state values and pre-conditions and then passes these values to the refinement model, which will update the KB. This avoid future failed execution for the same situation context.

Low level: Control System and Sensorimotor Acting often has a central role in deliberation for autonomous systems (Ingrand and Ghallab 2017). Planning systems translate plans into low-level motor commands to be executed. Actions are passed to the control system as low-level commands with their function parameters(Stulp and Beetz 2008). The lower controller uses functioning models to transform the planning operators into skills (‘procedures’) that are further refined into commands. It executes actions and responds re-actively to immediate changes and provides feedback (Cashmore et al. 2015a).

Perception is also a critical components in robotics. Systems that reason why execution fails need access to enough information about the goal and what the robot is doing (sensory data, actions, and their effects). Then it will be possible to detect and assess why execution failed (Schmill et al. 2007). Moreover, it is critical for identifying objects, states and chains of events related to the robot’s activity. The values of parameters for any action are of two types: (a) the current state values of variables and (b) their goal values. The current state values are called ‘observable state variables’. Observing has to be tightly integrated with planning, acting and monitoring and combines bottom-up processes from sensors to create meaningful data.

3 HDJ Mechanism and Algorithm

This section presents the Outlier Detection approaches, data collecting process and a description of our novel Discrimination Process Algorithm. DPA adapts our hypotheses in reasoning by searching for differences in the failed execution. It searches anomalies in the combination of the information of the failed execution and the training data (normal data).

3.1 Outlier Detection Approache

Anomaly Detection refers to the problem of searching patterns in data that differs and raises suspicions about a specific problem or issue (Zimek and Schubert 2017). We use the same technique to compare the values of the failed action execution with the training data by DPA. The purpose is to discover the suspicious values in the fail action execution and we suppose that the outlier value is responsible for failing the action execution.

Several anomaly detection techniques exist, such as Nearest-Neighbour, Cluster-Based, Classification-Based and ML Techniques. Selecting the proper technique is induced by many factors. The main factors are the nature of the data, availability of labelled data and type of anomalies to be detected. The nature of the data is based on specific data categories: point anomalies, contextual anomalies, and collective anomalies. For example, a point anomaly occurs at the level of a single feature whereas a contextual anomaly occurs in a specific context, such as for models that consider time series. The collective anomaly is a grouping based anomaly and occurs when a group of points appears far from the other groups; Each point by itself might not be an anomaly, but their whole combination as a group seems isolated (Chandola, Banerjee, and Kumar 2009).

The main assumption in the Nearest-Neighbour technique is that anomalies are take place far from the closest neighbour of normal data instances (Tan, Steinbach, and Kumar 2016). The Cluster-Based technique is based on grouping similar data instances into clusters. In the Classification-Based techniques, a classifier model is learned from the labelled data the anomalies are detected if not matching the feature space of the learned model. The ML technique can be useful for non labelled data models(Chandola, Banerjee, and Kumar 2009).

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

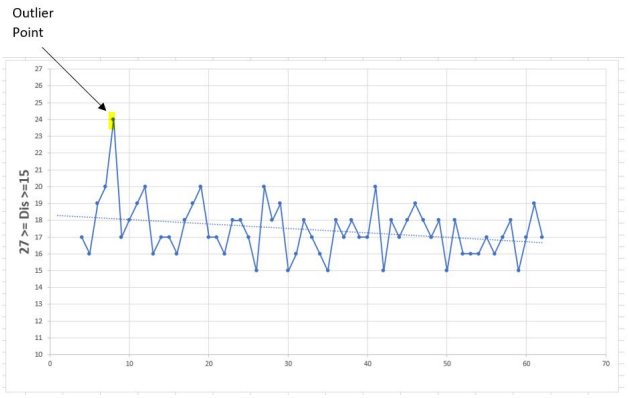

Figure 6: Failed execution with anomaly detection. The outlier 'distance' value appears far a way from its nearest neighbour in TD

normal data. The record of the failed execution is compared and tested against the normal data by extracting the anomalous values of the parameters. Figure 6 shows the draw profile of the history of the distance values where the outlier point represents the value of the distance instance of the failed execution.

3.2 Data Collection

This section covers two types of data collection: First: Learning object specification which is part of the KB information and second, the training-data collection, stored: The datastores, and which is necessary for processing DPA (section 3.2).

Learning Goal Specification In planning architecture, the KB includes DM and all information related to the domain. Learning the domain objects is critical for plan execution as the robot needs to be able to attach meaning by coupling symbols to objects. This sensor(actuator)-to-symbol interpretation is called as Anchoring Problem (Escudero-Rodrigo and Alquézar 2016). For example, in the kitchen scenario (section 4) NAO needs to learn the object 'redcup'. We used Choregraphe 2.1.4 to process sensory visual information on the NAO robot (Aldebaran b). The 'ALVisionRecognition' vision module allows the NAO to identify and recognise objects that are previously learned and whose information is already stored in the robot database. The NAO robot has two video cameras, with 1280×960 resolution, located in its forehead and mouth, and these cameras can capture up to 30 frames/images per second (Aldebaran d).

The learning method which allows NAO to recognise objects is explained in the Choregraphe documentation. The video monitor panel, provided in Choregraphe, enables the robot to learn objects to be recognised where the vision recognition information is stored in the robot database (Aldebaran e) (see Figure 7). NAOqi Trackers API (Aldebaran a) enables the NAO robot to track different objects and targets. Figure 8 shows the python code for setting Trackers API and defining the targeted object.

The 'ALTracker' module uses different means such as

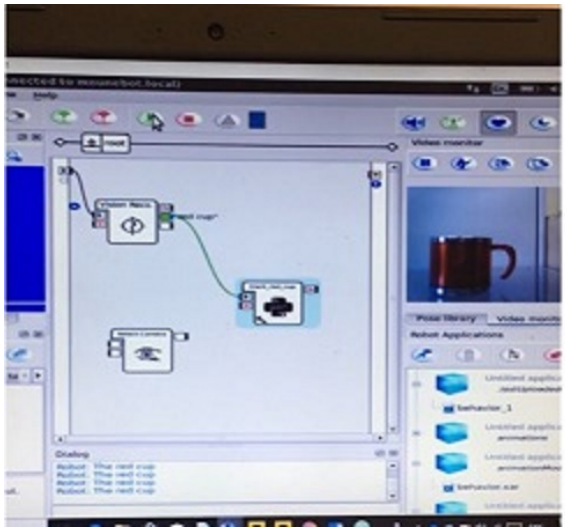

Figure 7: The Choregraphe video monitor panel: The NAO robot learns the object 'Red cup'. The window in the right shows the camera vision of NAO robot detecting the cup

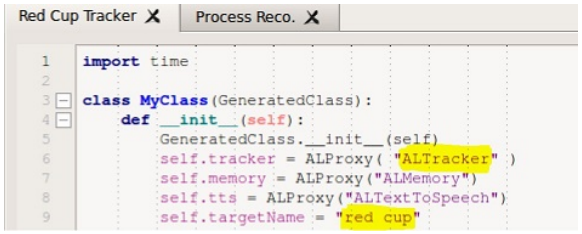

Figure 8: Python code: using Trackers API

head and whole body (Aldebaran a) and can identify the position of the object recognised by the robot's vision and give that position based on the desired reference frame. Reference frames are in three types: robot_frame, torso_frame and world_frame (Aldebaran a) (see Figure 9).

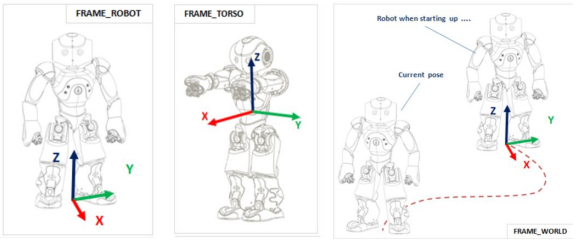

Figure 9: Frames Types
Source: (Aldebaran c)

The control system of the NAO robot needs specific information to detect the target and then to enhance the gripping motion. As a first step, the robot has to learn the targeted object 'red cup' The 'getTargetPosition' function returns the [x, y, z] position of the target in the desired frame. We collected our data by employing the torso_frame (shown in Fig-

ure 9). FRAME_TORSO: attached to the robot's torso reference that makes sense in the orientation of the torso frame. The standard axis orientation: x forward, y left and z up. The distance calculation is done based on the average target size by using 'ALMath' library (Aldebaran c).

Training Data The training-data can be defined as a history record of all previous successful executions. For example, when the robot can reach the cup and grip it, as represented in Figure 10.

Figure 10: Success execution: The NAO robot can reach the cup

Discrimination Process Algorithm This algorithm relies on the feedback coming from the control system after each execution. As described in section 2.1, the control system sends feedback to the Knowledge-Base; The 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). The output is a report of the extracted differences that are considered as the potential cause of failure. It will be used to learn success values and will then refine the PDDL model by updating the knowledge-Base. Any detected anomaly is a value that indicates parameters or pieces of information in the Knowledge Base, such as pre-conditions and states. Future failure can be avoided by updating the knowledge of the task's planner.

The human mind inspires the main idea behind this algorithm. According to Schmidt (Schmidt 1975), human action control is a hybrid model that combines both feed-forward and feedback components. Schmidt argued that humans set a motion schema by specifying the relevant attributes of that motion but leave free parameters to be specified online while collecting environmental information. Schmidt's argument is supported by neuroscience evidence (Glover 2004). This indicates the integration of the off-line action planning with the online sensorimotor specification (see the pseudo-code of the algorithm).

```
1  while plan execution do
2      if not (Feedback.Success) then
3          TD: = Training-Data
4          FD: = Failed-Data
5          Anomaly Detection(FD, TD)
6          if QueryResult.count > 0 then
               // If Outliers exists
7              Return Out
8              Learning-Processing (Out)
               // Learn new success values
9              Return LV
10             Refinement-model(LV);
11         end
12     else
13         SD: = Successful action Data
14         Add-Training-Data (SD);
15     end
16 end
```

Discrimination Process Algorithm (DPA): Detail of differences detection, learning success values and refinement process. If action execution failed (*FD)*, then call AD. *Anomaly Detection (FD, TD)* extracts differences (*Out*) from the failed action. *Learning-Processing (Out)* learns new success values(*LV*) based on the detected anomaly/ies. *Refinement-model(LV)* refines KB by exchanging the *LV* with (*Out*). Otherwise, if the execution was successful(*SD*), *Add-Training-Data* adds *SD* to the Traning-data.

4 Evaluation

The HDJ extension system has been implemented as described, utilising ROS and ROSPlan software, and is used to control the NAO robot. The goal of this section is to evaluate the system by exploring its behaviour within a real kitchen scenario (Müller and Beetz 2006), as represented in Figure 5. Here, the task given to the robot is to grip and move a cup from a table to another one. The distance between the robot to the table is the main factor of our experiment. The environment is not changed for each execution (episode), but the initial position of the robot to the table is randomly changed.

The HDJ software reasons about the cause of the failure and then refines the PDDL model based on the reasoning outcome. In this section we test the efficiency of the HDJ system reasoning function with failed plans to identify the cause of execution failure.

The Artificial Intelligence and Machine learning fields aim to create human-like intelligence and cognitive functions associated with the human mind, such as learning and reasoning (Russell Stuart and Norvig 2009). For this reason, the evaluation of this system is a comparison of the predictions made by the robot against the researcher's predictions ('human predictions'). The predictions are to answer the question why has the execution failed? When execution fails to execute the given task, the robot needs to realise why it has happened. Then, to avoid any future failure, an update to the Knowledge-Base should be made by the refinement model. We evaluate the efficiency of the HDJ software based on its predictions, which are the detected anomalies by the DPA. For each failed execution, we check the output report of the HDJ (see Table 1). If the distance is included as a potential cause of failing the execution, then we consider the HDJ results are correct. Otherwise, we consider that it failed

to recognise the cause of failure.

Action	Cause-Value	Attribute
grip	24	maxdis

Table 1: Report on the outlier detected output of DPA

4.1 Experiment Implementation

In the kitchen domain, the distance between the robot and the table is a free parameter that falls within a range of values. A distance rule 'dis' defines this range (23cm $>$ dis $>$15cm). Figure (4) shows part of our domain model. The predicate '(at robby ?r ?table)' is the pre-condition of the 'grip' action and it is related to the rule 'dis'. It is true, as long as the distance to the robot meets the condition.

The 'distance' feature is the main factor of our experiment. For each episode, we change the initial position of the robot to the table. We set the robot position to the table in a way that leads to failing the execution. We intentionally set a wrong range of values in the 'dis' rule so that instead of (23cm $>$ dis $>$ 15cm), we set it to (27cm$>$ dis $>$ 15cm).

We developed a function that measures and validate the distance It sets the instance of the state (at robby ?r ?table) according to the validation process. When the execution fails, DPA extracts the differences and learns the success values; then it sends the learned values to the refinement model. In this case, we use the Nearest-Neighbour anomaly detection approach. Each attributes' value, of the failed execution, is compared to the distribution of alternative attributes in the training data. It considers rules of form X where X attribute taking on particular value. It seeks the value of X of the failing record that is not observed in the training data.

The detected anomaly is passed to the learning process to learn new values. The learned values *V*, which we called as success value, is selected in a way that bridges the gap between the training data and the failed value. The refinement model is in charge of updating the KB. Rosplan provides services to update the knowledge-Base (Cashmore et al. 2015a). It removes the fail values to be exchanged with the learned values.

```
(:objects
    wp0 wp1 wp2 - table
    kenny - robot
    Gripper - gripper
    RedCup - object
)

(:init
    (at-robby kenny wp0)
    (at RedCup wp1)
    (free kenny Gripper)
)

(:goal (and
       (at-robby kenny wp1)
       (carry kenny RedCup Gripper)
```

Figure 11: Problem File

4.2 The Results

To evaluate the success rate of our learning system, we tested 65 failed executions. The system correctly predicted 63 causes of failure. Table 2 shows the confusion matrix of the performance evaluation of 65 predictions. P= correct predictions while N= false predictions. For the overall evaluation results see Table 3.

	Predictions		
n=65	**P**	**N**	
P	TP=61	FN=2	63
N	FP=2	TN=0	2
	63	2	65

Table 2: The Confusion Matrix to evaluation the Performance of HDJ

This real application brings the complications that action learning systems tend to ignore. One of those is the hardware specifications, for example the accuracy of the robot vision. In this empirical test, the robot vision fails to calculate accurately the distance between the robot and the table, which means the values of the distance in the edges can be miss-counted, as seen in Figure 12.

TP	TF	Precision	Accuracy	FNR	TPR
61	2	96.8%	93.8%	3.17%	96.8%

Table 3: The overall evaluation results

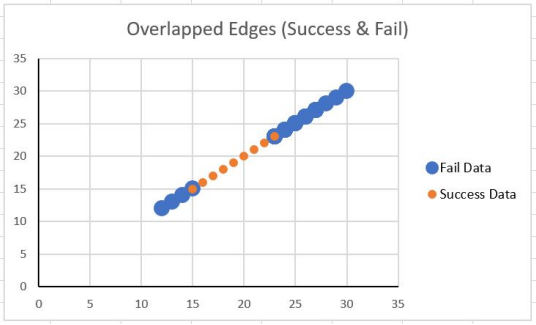

Figure 12: Overlapped Edges: Orange dotes represent the successful range of distance (Training data). The blue dots represent the values of failing distances.

For example: with 13cm distance, the robot can reach the cup, but at 13.5cm it fails to touch it. The robot vision counts the distance as 13cm in both cases, and this leads to 13CM appearing in the training data as successful values. But it is possible that the robot may succeed or fail in cases at the boundaries of operation. So, there will be no differences in the distance to be captured by the DPA. Figure 13 represents the predictions made by human and robot

This issue can be overcome by improving the hardware specifications of the robot vision. (Zhang et al. 2019) also highlighted this issue, which is the low resolution of the

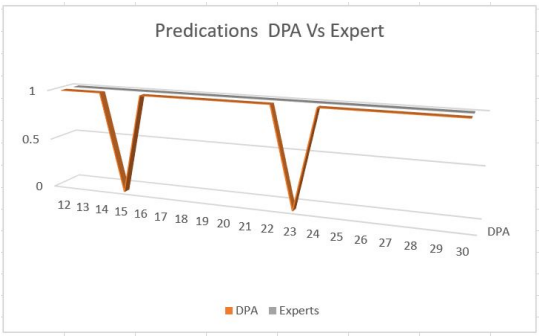

Figure 13: Graphical representation of the right Predictions made by human vs. robot

NAO robot's camera, and they agreed the need to use a high-definition depth camera, such as Kinect, to help the NAO robots to achieve such tasks. Also, (Müller, Frese, and Röfer 2012) used a NAO with a stereo vision head that was explicitly designed for their 'Grab a Mug' experiment.

5 Conclusions

In this work, we propose a reasoning method to predict the cause of action failure, that may occur during the lifetime of a cognitive robot in a changing environment. The reasoning process is based on using anomaly detection techniques. The framework is for refining the PDDL model by enriching the Knowledge-Base that the task planner relies on to produce the plans.

The real-world application brings complications related to the real environment and sensors issues. This problem can be overcome by improving the hardware specifications. The experiment's results show the accuracy of our system is 93.8%. Accordingly, we can conclude that our proposed system is efficient and can contribute to bridging that gap between actions at a high-level stage (abstracted actions) and actions at a low-level stage (low-level commands). The system's output can contribute to improving planning quality using automated learning and adaptation models based on experience and online sensory data in changing environments.

Selecting the AD technique is critical and vital for the success of our reasoning method. However, it offers a level of flexibility, because it is not limited to a single AD approach and we can implement more than one approaches. The experimental work has proved our concept that examining the differences in the failed execution is for the benefit of automated refinement process towards long-term autonomy. The existence of a variety of AD techniques adds extra benefits to our methodology. It opens the door for a variety of implementations of different action models and problems and it increases the automation aspects of our reasoning methodology. Significantly, most of these techniques are connected to the fast developing approach that is ML.

References

Aldebaran, R. Altracker. http://doc.aldebaran.com/2-1/naoqi/trackers/altracker.html. Accessed: 2019-10-09.

Aldebaran, R. Alvisionrecognition. *http://doc.aldebaran.com/2-1/naoqi/vision/alvisionrecognition.htmlalvisionrecognition*. Accessed: 2019-10-09.

Aldebaran, R. Cartesian control. http://doc.aldebaran.com/2-1/naoqi/motion/control-cartesian.html#motion-cartesian-effectors. Accessed: 2019-10-09.

Aldebaran, R. Nao - video camera. http://doc.aldebaran.com/2-1/family/robots/video_robot.html?highlight=camera. Accessed: 2019-10-09.

Aldebaran, R. Recognizing objects. *http://doc.aldebaran.com/2-1/software/choregraphe/tutos/recognize*$_o$*bjects.html.Accessed* : $2019 - 10 - 09$.

Arkin, R. C. 1998. *Behavior-based robotics*. MIT press.

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

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; De Carolis, V.; and Maurelli, F. 2015b. Dynamically extending planning models using an ontology. In *Proceedings of the ICAPS Workshop on Planning and Robotics*, 79–85. Citeseer.

Chandola, V.; Banerjee, A.; and Kumar, V. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41(3):1–58.

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

EmaroLab. 2017. Emarolab/owl-rosplan. *EmaroLab*(online). https://github.com/EmaroLab/OWL-ROSPlan.

Escudero-Rodrigo, D., and Alquézar, R. 2016. Study of the anchoring problem in generalist robots based on rosplan. In *CCIA*, 45–50.

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

Hodge, V., and Austin, J. 2004. A survey of outlier detection methodologies. *Artificial intelligence review* 22(2):85–126.

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

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

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

Müller, J.; Frese, U.; and Röfer, T. 2012. Grab a mug-object detection and grasp motion planning with the nao

robot. In *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, 349–356. IEEE.

Russell Stuart, J., and Norvig, P. 2009. *Artificial intelligence: a modern approach*. Prentice Hall.

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

Schmill, M.; Josyula, D.; Anderson, M. L.; Wilson, S.; Oates, T.; Perlis, D.; and Fults, S. 2007. Ontologies for reasoning about failures in ai systems. In *Proceedings from the Workshop on Metareasoning in Agent Based Systems at the Sixth International Joint Conference on Autonomous Agents and Multiagent Sytems*.

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

Tan, P.-N.; Steinbach, M.; and Kumar, V. 2016. *Introduction to data mining*. Pearson Education India.

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

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