TAINN 2003

A Learning Based and Vision Guided Robotic Agent Replanning Framework

Şule Yıldırım, Norwegian University of Science and Technology, Norway
Turhan Tunalı, International Computer Institute, Ege University Bornova İzmir 35100
yildirim@idi.ntnu.no, tunali@ube.ege.edu.tr

Abstract
In this paper, a learning based robotic replanning framework that can handle unexpected events in dynamic worlds is presented. Existing robotic architectures use artificial intelligence planning methods, reactive approaches or hybrid approaches to increase robustness and unexpected event handling capabilities of robotic agents in dynamic environments. This study presents a new hybrid task planning architecture that equips robotic agents with higher level of intelligence with an original replanning method. The replanning method uses an alternative based action selection mechanism to select the most efficient action path among possible alternative action paths. The method stores the costs of actions paths from previous executions as an experience and uses it for improving its future action selection decisions. The fact that a continous vision feedback is supplied to the symbolic planning level (highest level of abstraction) is also a contribution of this study since this way, the architecture detects the presence of unexpected events, generates an updated model of the environment and discards or modifies the existing obsolete action path in case of an unexpected event(s).

Keywords
Robotic Learning, Vision Feedback, Replanning, Unexpected Events.

1. Introduction

The main purpose of Artificial Intelligence (AI) is to form control algorithms that allow an agent to realize its goals by synthesizing a sequence of actions. The subfield of AI called robotics works on both robotic planning and other aspects of robots.

In order to synthesize a sequence of actions for the formulation of planning problems, many methods have been proposed. Compiling planning problems into propositional formulae using systematic and stochastic SAT algorithms have attracted much attention [1]. These methods have a high performance. For example, Blackbox planner [2] can form a 105-action logistics plan for 10^{16} states in 6 minutes. The STRIPS representation method [3], defines each action to be a transfer function from a world to another world that can be defined as a set of precondition and effect conjuncts. Some recent studies use dynamic CSP (Constraint Satisfaction Problems) for the solution of planning problems considering the similarity between CSP and planning problems [4].

However, whatever the method for generating a plan is, it is not realistic to expect a robotic agent to realize a task only by executing a generated plan and without considering the unexpected events and uncertainty in a dynamic world. Hence, the robotic agents should have the capability to discard obsolete plans in case of unexpected happenings and replan, that is, generate new plans applicable in current situation obtained after the unexpected event or modify an existing plan in a way to make it applicable in the new situation.

Robotic plans constitute of actions that the robots execute in a world and the state, position and orientation of the objects in a world changes as a result of the execution of these actions. However, in order for a robot to execute an action on an object, the object should be in the state, position and orientation that a plan action specifies it to be before the execution. This is not always possible since other agents than the robots might change the indicated attributes of objects in a robotic world. Hence it's highly possible that an object is not with the expected attributes before an action is executed. If robots can be capable enough to realize the situation and synthesize new actions or action paths for the current situation, it will be possible to bring the main task that the whole original plan was generated for to completion instead of having robotic agents not knowing what to reason in the new situation to achieve the completion of a defined task.

There have been research on handling unexpected events and uncertainty in dynamic environments with the indicated scope however, none of these has the capability of synthesizing alternative action paths to the goal and selecting the one with the least cost for execution. Rather, the existing approaches use the "*the one that works*" approach and are satisfied to have found an action path that leads to completion without considering whether there's a lower cost action path to the goal or not. However, the alternatives which are to be considered to make a selection among are as follows:

1. Is there an already stored plan which is known to be a minimal cost from the current situation to the goal?
2. Should the situation after the unexpected happening be transformed back to the state before the unexpected event happened by eliminating the effects of the unexpected happening?
3. Should the robotic agent just synthesize a new action path from the current situation to the goal with discarding the original plan and hence computing a new plan?

Existing approaches have the tendency to always use the same alternative in case of all unexpected happenings instead of making a selection among them for the least cost one. The architecture to use the selection based replanning method should have supportive features to make selection mechanism possible to implement. The most important feature of the architecture is the continous vision feedback to the planning level which helps to detect the occurence of unexpected happening and form a new world model to synthesize a plan for in place of the obsolete plan. The other important feature is the learning capability which improves the selection based decision in time and allows to take better decisions. Better decision is the one that *guesses* (selects) the alternative which encapsulates the *least cost path* correctly.

This paper is organized as follows: In Section 2, our planning architecture to handle unexpected events will be presented. Section 3 will present a case study for unexpected event handling. Finally, in Section 4, concluding remarks will be made.

2. The Vision Guided Intelligent Planning and Replanning Architecture

The Vision Guided Intelligent Planner (VGP) is the implementation of a planning architecture developed during our studies on unexpected event handling in dynamic environments (Figure 1). It is based on the 3T architecture. It has a deliberative planner which generates plans and sends them to the control level through sequencer for execution. A vision system monitors the environment for unexpected events during execution. Planning and execution can be interleaved if there is an unexpected happening to achieve r*eplanning*. Replanning uses planner to generate new plans or modify existing plans. Low level control algorithms that are called behaviours can respond to changes or uncertainty in the environment. Behaviours solve the problem with immediate reactions when a mobile robot is about to bump into an obstacle or when a robot arm is about to put an object in an already occupied place for example.

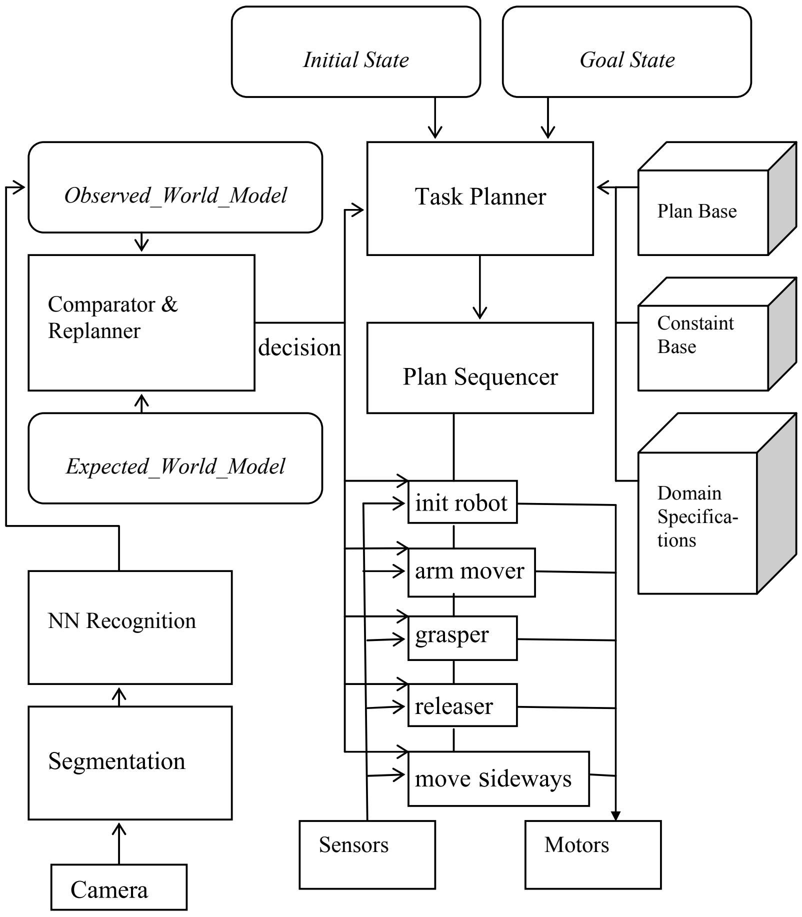


Figure 1. VGP Planning and Replanning Architecture

Although more details can be found elsewhere [5, 6] about the implementation of domain and planning structures, the proposed planning architecture in Figure 1 takes an initial and a goal state of a robot workspace as an input and produces a plan. After execution of each action in a plan, a model of the environment which is called Observed_World_Model is obtained from an image supplied by the vision system. The planner has an internal state called Expected_World_Model which is updated after execution of each action in a plan. These two models should be the same if everything goes as expected during execution. Plan base keeps previously executed plans by indexing them with an initial and goal state pair. Constraints are stored in a *constraint base* while domain specific features (operators, rules etc.) are kept in *domain specifications base*. State inputs, world model generations, planning and execution is scheduled in a task scheduler.

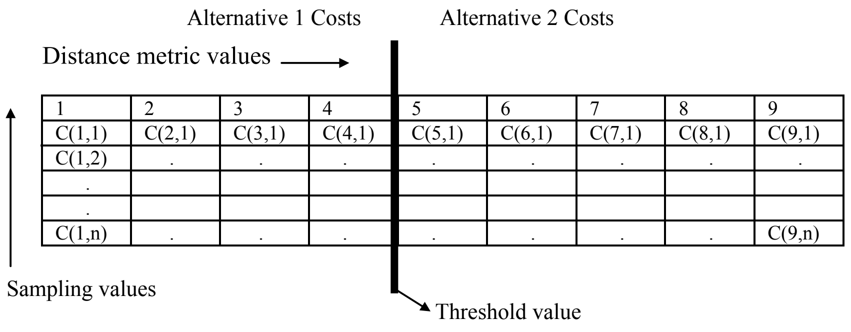
In case of unexpected events, there might be a previously stored optimal cost plan in the *plan database* from the unexpected state to the goal state. In such a case, the optimal plan is retrieved and executed. In case that there's not a stored optimal plan, a new plan to backtrack to the state before the unexpected event can be generated and then the rest of original plan can be executed. Some existing planners use that approach [7]. Another approach is to generate a new plan directly from the unexpected state to the goal state. This approach is efficient when the unexpected state is so different from the state before the unexpected event occured that it's less costly to generate and execute a new plan from the unexpected state to the goal state.

Each plan step is compiled into one or more reactive behaviours, i.e. init robot, arm mover, grasper, releaser or move sideways by a plan sequencer at execution time. Each behaviour receives input from sensors. An input can suspend an active behaviour in case of unexpected happenings such as obstacles.

Camera views can capture the information that cannot be detected by sensors. A camera view also helps to form a current model of the world (*Observed_World_Model*). *Observed_World_Model* might represent the unexpected state after an unexpected happening. Otherwise, it will represent the state the world got into after executing the most recent action. It is compared with *Expected_World_Model* to figure out whether the current world model is as expected after the execution of an action or not. A "dm" variable symbolizes the difference between two world models. If "dm" value is different than "0" then the current world model is not as expected and replanner should be called since the existing plan will not be valid any more.

Whether "dm" value is small or not is decided by comparing the "dm" value with a threshold "thr". If an unexpected event occured and there's an optimal plan stored from the resulting unexpected state to the goal state, then that plan will be retrieved from the database and executed. If not, depending on whether the "dm" value is small or not, another alternative will be chosen. If the "dm" value is smaller than "thr" or equal to it then the unexpected state is backtracked to the state before the unexpected happening occured and the rest of original plan is executed (alternative 1). In such a case, *new_plan* is a plan that transforms the unexpected state to the state before the unexpected happening occured. Function *alter1_cost()* calculates a cost for using alternative 1, considering the planning time and the total distance to be travelled to reach the goal state. Otherwise a plan for alternative 2 which is to transform the unexpected state to the goal state is generated and a cost for it is calculated. Whichever alternative is used, the related cost is stored in a cost table to improve future decisions for alternative selection.

At the beginning, the alternative selection mechanism of replanning might not guess the least cost alternative correctly since the threshold value is assigned randomly. This value needs to be adjusted as more costs are calculated and stored for occuring unexpected events in time. The costs are kept in a cost table (Figure 2) with dimensions P (> 0) and Q (> 0). The threshold value divides the table into two regions: the left region displays alternative 1 cost values whereas as the right region displays alternative 2 cost values.



1	2	3	4	5	6	7	8	9
C(1,1)	C(2,1)	C(3,1)	C(4,1)	C(5,1)	C(6,1)	C(7,1)	C(8,1)	C(9,1)
C(1,2)	.	.	.	.	.	.	.	.
.								
.								
C(1,n)	.	.	.	.	.	.	.	C(9,n)

Figure 2. A Cost Table

The distance metric values in the table show the possible values for the difference between the unexpected state and the state before the unexpected happening. For instance, the values of 1, 2 etc.in the first row of the table symbolize the situation when the difference is 1, 2 etc. respectively. This difference will either be equal to, smaller, or bigger than the threshold value. Each cost C(dm, n) in a column defines the cost for a selected alternative after an unexpected happening with the indicated dm occurs. If this is the first unexpected happening to occur for that "dm" value then cost will be placed at col = dm, and row =1, that is at (dm, 1). On the other hand, if it's the nth unexpected happening to occur for the same "dm" value, then it will be placed at col = dm, and row = n, that is, at (dm, n).

The threshold value is not adjusted until the cost table is completely full. Each cost table entry symbolizes an execution for a different initial and goal state pair. When the table is full, the average of the values in each column is taken. An abnormal *difference* between the column with the threshold value and the previous column indicates that a threshold adjust operation is necessary. The idea behind the threshold adjust operation is that if the averages of the costs on the left region of the threshold are too different from the averages of costs on the right region then the threshold is not stabilized yet and needs to be adjusted. The difference is expected to have a reflection on the averages of the threshold column and the column before it. The curve with the averages of each column plotted is given in Figure 3. In the figure, there is a difference between the average of the costs in left region and the averages of the costs on the right region. For that reason, the band width of the columns up to the threshold value is different from the band width of the columns beyond the threshold value.

Adjustment of threshold value is done by increasing or decreasing the threshold value by a certain amount to stabilize it at the best value after several adjustments. A heuristic is used to decide on an increase or a decrease. The heuristic is that if there's an increase from the previous column to the threshold column, then alternative 1 region column averages have a lower band width than alternative 2 region column averages. For that reason, the averages in region 1 will be increased whereas the averages in region 2 will be decreased by excluding the column previous to the threshold column from region 1 to region 2. This is accomplished by shifting the value of the threshold one column to the right. However, if there's a decrease from the previous column to the threshold column, then the threshold value is decreased; that is, it is shifted to the left one column on the cost table.

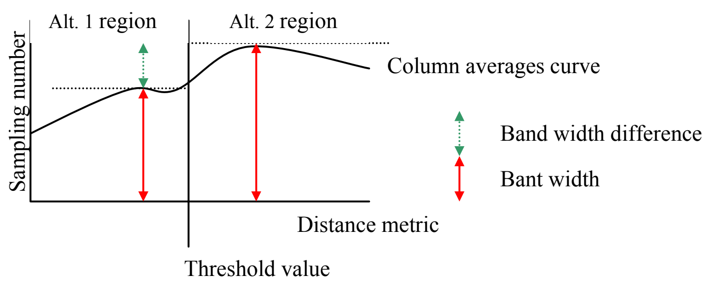


Figure 3. Column Averages Curve

A cost function to calculate the entries of a cost table is defined as follows:

$C_p = c_1 T_p + c_2 E_p$ where c_1 and c_2 are constants (1)

T_P: the time to generate a new plan for a given path p.
E_p: the amount of energy that a robot arm will spend to accomplish a plan for a given path p.

Thus, costs for Alternative 1 and Alternative2 are defined as follows:
$C_1 = c_1 T_1 + c_2 E_1, C_2 = c_1 T_2 + c_2 E_2$

T_1, T_2 are the planning times to generate a new plan respectively for Alternative 1 and Alternative 2. E_1, E_2 are the energy amounts spent by the robot arm to execute plans for Alternative 1 and Alternative 2.

The threshold adjust operation has a “learning” feature. The system gains more information on execution as time passes and more unexpected events occur. This leads the system to give more correct decisions in time. This method has been developed during this study and it is novel. This method, as in other methods in literature, increases the performance P (the probability of giving the correct decision in case of an unexpected event) of the system making use of the experience E (accumlated cost values in time) for a given class T (blocks world problems) of defined tasks. Some learning mechanisms improve a learning function whereas with the one discussed in this study the threshold value is learnt by improving it in time.

3. A Case Study for Implementing VGP Architecture

The Vision Guided Planner (VGP), has been implemented on two blocks world problems. This case study depends on first of these problems which is mixed pieces problem. Details about other problem and possible application domains can be found in [8]. A definition for the mixed pieces problem is as follows: Given are the initial and goal states in a two dimensional grid like space. Blocks are placed in the cells of the grid. The problem is to find the sequence of actions with minimum cost from the initial state to the goal state. The key point in this problem is that there can be several blocks of a block type and each block of a block type can be placed in any of the destination places for the given block type. The following are other important issues of the problem:

1) A destination position of a block can be occupied by another block.

2) The blocks in a work space can be mixed by an external agent during execution of an initial to goal state plan. The external agent is interpreted as an unexpected event and the existing plan will become obsolete due to the change in the workspace so replanning will be necessary.

If the blocks in the problem are taken as the pieces of a chessboard and if the work space is taken as the chessboard, it can easily be seen that there can be several blocks of a block type. For example, the initial state can be as in Figure 4 where there are two labelled A, two labelled K, etc. blocks (chess pieces). It's accepted that there can be 6 different types of blocks as in a chess game for this case study. The number of blocks on the board is related to the size of the board as in a chess game. A board with $n \times n$ dimensions has $n \times n / 2$ number of blocks at most ($n > 0$). The number of the types of blocks (A, V, S, K, P, F) and the number of the blocks (pieces) on the board (2 A type blocks, 6 P type blocks) is generated randomly by the planner. Board size is user input.

For example, the block labeled V at coordinates (4, 2) in Figure 4 can either be placed at (2,1) or (5,1) for its destination in Figure 5. Similarly, the block labeled V at coordinates (6, 7) can either be placed at (2,1) or (5,1) for its destination. Which V will be placed at which coordinates depends on minimizing the total distance the robot arm travels. In the proposed solution, robot arm is taken to its initial position. The planning details are given in [9].

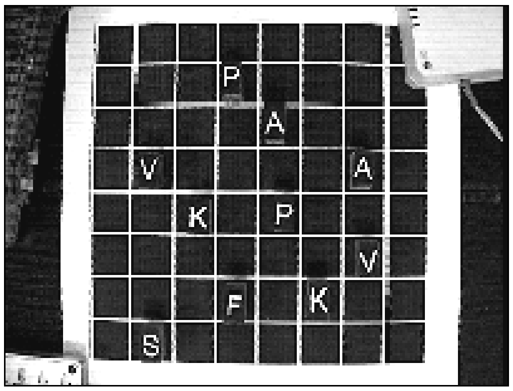


Figure 4. An Initial State for Mixed Pieces Problem

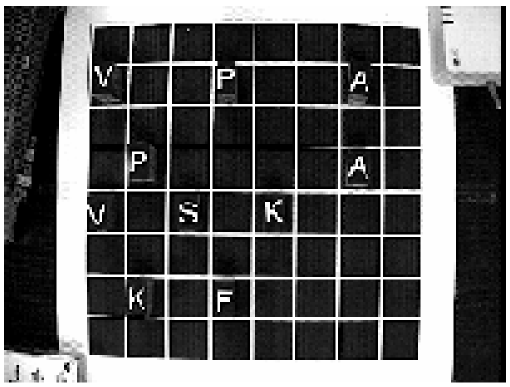


Figure 5. A Goal State for Mixed Pieces Problem

While executing the plan for the mixed pieces problem, unexpected events are allowed to occur. The distance metric which represents the amount of mix up on the board is calculated after the unexpected event occurs. The distance metric value is the number of blocks whose positions have changed after the unexpected event. If there's a previously stored plan from the unexpected state to the goal state then that plan is put into execution. Otherwise, the distance metric is compared by the threshold value to decide on which alternative to persue.

3.1 Threshold Value Adjust for the Case Study

A board with dimensions 16 × 16 is used for the case study. There are 128 blocks on the board. There are two colors of blocks and each color has 64 blocks. All the block types are on the board and the number of blocks for each type is given in Figure 6. Two blocks with same labels but different colors can be considered as two different block types. Nevertheless, a black V in initial state will never be placed in a destination place for a white V.

The cost table has dimensions 80 X 5. The distance metric for this problem is the number of pieces that are not in their expected places after the unexpected event. This number includes the number of blocks that were placed in their destinations but now removed by an external agent as the result of an unexpected happening. The values for the distance metric can be $0 < dm < 80$.

White Group		**Black Group**	
Block Type	*Number of block*	*Block type*	*Number of block*
V	4	V	4
S	4	S	4
A	8	A	8
K	16	K	16
F	16	F	16
P	16	P	16

Figure 6. The Table That Shows the Number of Pieces for Each Group

The costs for 5 different unexpected events will be recorded in each column of the cost table. There should have occured 400 unexpected events to fill the cost table completely. The abnormal increase or decrease is assumed to be above "1" or below "-1" between the threshold column and the previous column. Threshold value is assigned as "5" at the beginning. An abnormal decrease will shift the threshold value by 10 (step-size) to the left whereas an abnormal increase will shift the threshold value by 10 to the right. Figure 6 presents the changes from the previous column to the threshold column. If an increase is detected, the threshold value is increased to 15, 25, 35 and 45 in order. When the threshold value is 45, the difference between previous column and the threshold column is -3.08 so the threshold value is decreased by 10 to obtain 35 which is the stabilized threshold value. This is the threshold value where the difference is a decrease (-3.08) after several increases (+3.16, +7.68 etc.) so it's taken as the stabilization value (Figure 7).

Threshold Value	Difference
5	+3.16
15	+7.68
25	+7.38
35	+5.37
45	- 3.08

Figure 7. The Difference between the Threshold and the Previous Column

Figure 8, 9, 10, 11, 12 and 13 are the tables for distance metric value costs. The tables show the plots for averages of columns in a cost table where averages below threshold present the values for alternative 1 and averages above threshold value present the values for alternative 2. That is, alternative 1 is chosen for low distance metric values and alternative 2 is chosen for high distance metric values. The stabilization for threshold value at dm=35 is reasonable when the areas of graphs in Figure 12 and Figure 13 are compared. In Figure 13, the graph of the column averages of costs where alternative 2 is used for all distance metric values is given. In Figure 12, the left side of threshold value is the graph plotted using alternative 1 costs and the right side is the graph plotted using alternative 2 costs. The area under the graph of Figure 12 is 1281 where as the area under the graph of Figure 13 is 1336 (the difference is 55). On the other hand, the area under graph in Figure 11 is 1522. However, if alternative 2 values were used for all distance metrics for the case in Figure 11, the area would be 1602 (graph is not plotted). The difference between the two areas would be 80. As a result, the case study supports the idea that alternative based mechanism is a method to decrease costs in total. On the other hand, obtaining a difference of "80" at stabilized threshold value (35) whereas obtaining a lower difference of "55" at an unstabilized threshold value supports the idea that threshold can be improved to give better decisions in time.

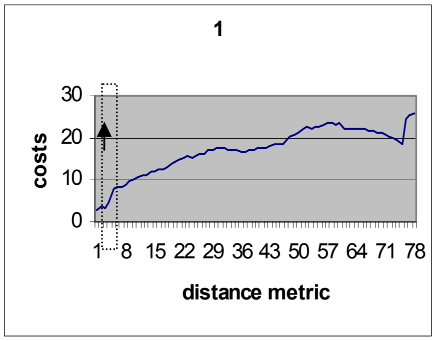


Figure 8 Thr = 5, cost table averages.

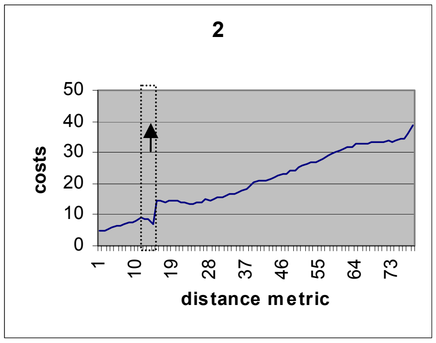


Figure 9 Thr = 15, cost table averages.

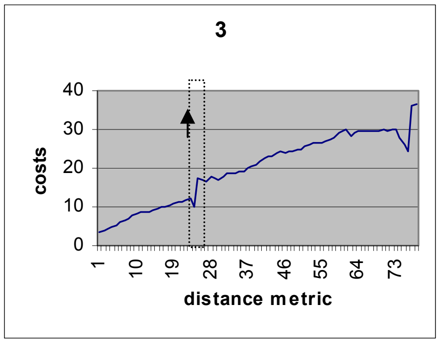


Figure 10 Thr = 25, cost table averages.

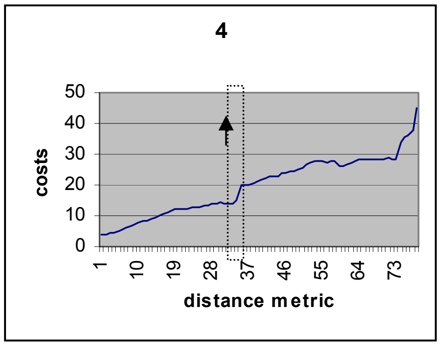


Figure 11 Thr = 35, cost table averages.

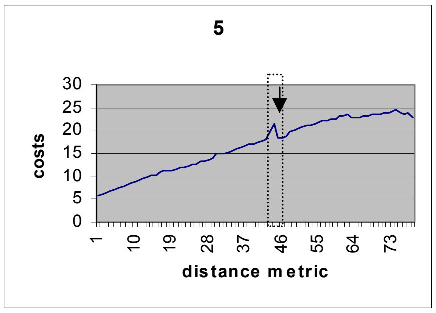


Figure 12 Thr = 45, cost table averages.

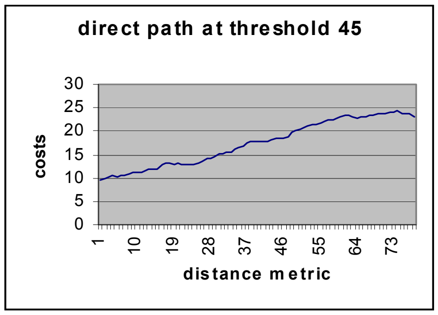


Figure 13 Cost table averages obtained using alternative 2 for every distance metric in **Figure 12**.

4. Conclusion

In this paper, a robotic architecture which handles unexpected events with replanning at the highest level of symbolic representation and which has learning based capabilities is presented. If there is more than one alternative as an action path in case of an unexpected happening, the one with the least cost is considered. The possible alternatives can be to backtrack to a state with the effects of unexpected happenings cleared or to make a new plan from the unexpected state to the goal state. Additionally, the mechanism is enriched by indexing optimal plans for an initial and goal state pair if there is one in a plan base and to retrieve them in case the planning needs to be done for the same initial and goal state pair. Vision support is used for detection of unexpected events in the environment and events that cannot be detected by low level sensors. It also helps to generate a current model of the environment to generate a plan.

References

[1] Weld, D. S., 1999, “Recent Advances in AI planning”, AI Magazine.

[2] Kautz, H. , Selman, B., 1998, “Blackbox: A new approach to the application of theorem proving to problem solving”, In AIPS98 Workshop on Planning as Combinatorial Search, pp. 58-60.

[3] Fikes, R. E., and Nilsson, N. J. STRIPS: a new approach to the application of theorem proving to problem solving, *Artificial Intelligence*, **2**, 3-4, 1971, 189-208.

[4] Falkenhainer, B, Forbus, K., 1998, “Setting up large scale qualitative models”, In Proc. 7th National Conf. AI, pp. 301-306.

[5] Yıldırım, Ş. and Tunalı, T., 1999, “A new methodology for dealing with uncertainty in robotic tasks”, XIV. Int. Symp. on Comp.& Inf.Sci., Kuşadası, TURKİYE.

[6] Yıldırım, Ş. and Tunalı, T., 2002, “A Proposal For a Robotic Planning/Replanning Framework”, TAINN'02, Istanbul.

[7] Haigh, K.Z., 1998, Situation-Dependent Learning for Interleaved Planning and Robot Execution, Ph.D thesis, CMU.

[8] Yıldırım, Ş., A Vision Guided Intelligent Task Planner for a Robot Arm, Doctoral Dissertation, Ege University International Computer Institute, 2002 (in Turkish).

[9] Yıldırım, Ş. and Tunalı, T., "The Implementation of a Robotic Replanning Framework.", Advances in Information Systems, Second International Conference, ADVIS'2002, pp. 195-204, İzmir Turkey, October 2002 Proceedings,Springer-Verlag, LNCS2457.