AUTOMATED MODELING IN A ROBOT-OBJECT DOMAIN

Sunil Sah, Ivan Bratko, Dorian Šuc

University of Ljubljana, Faculty of computer and information science,
1000 Ljubljana, Tržaška 25, Slovenia

sunil.sah3@gmail.com (Sunil Sah)

Abstract

Obtaining a useful model in a given domain depends on the complexity of the model, the learning algorithm, and on the input data. While a specific task limits the choice for the model and the algorithm, the required quantity and distribution of the data could be less well defined, and may in turn be dependent on yet to be determined model parameters. To explore the weak points of an unreliable model, we introduce a data collecting agent, guided by the model. The agent must be able to cope with little data and unreliable model, to collect new data required to improve the model. This paper presents an algorithm, inspired by adaptive robot controllers, that uses local models and strategy search based on measured progress to efficiently collect data required to improve the model of the domain. The algorithm is tested in a simple robot-object domain used in the European project XPERO, which is about autonomous discovery through robot's experiments in its environment.

Keywords: Modeling robot environment, Robot-object domain, Shifting setpoint algorithm, Locally weighted learning, Action planning.

Presenting Author's biography

Sunil Sah. Sunil Sah is a student at the Faculty of computer and information science. He is interested in AI and machine learning, and has been doing research in the Faculty's AI Lab within the European project XPERO.

1 Introduction

Automated modeling of a robot's environment poses various challenges related to generality and efficiency of the learning methods, the use of prior and learned domain knowledge and the control of the robot in a way to use and improve the current model of the domain. Efficient and robust control facilitating the use and improvements of the current model, as addressed in this paper, can be seen as a prerequisite for autonomous learning and discovery through robot's interaction with the environment. In this respect this work is a part of the Sixth Framework European project XPERO[1] with the scientific goal to investigate mechanisms of autonomous discovery through experiments in a robot's environment, as for example in systems [2,3,4].

The fundamental goal of the XPERO project is to identify a small set of basic principles that enable such discovery without substantial amount of prior knowledge. Initial experiments with machine learning from experimental data collected by a mobile robot presented in [5] implicate the need for a corresponding model-guided data-collection algorithm. The algorithm described in this paper will be used in XPERO to control the robot towards the regions of the state space where the reliability of the current model is low, or where the machine learning methods induced some interesting, but unexpected hypotheses. The collected data would be repetitively used to improve the control of the robot and to update the induced model of the environment to support robot learning and autonomous discoveries.

In the next section we define the problem, terminology and the robot-environment experimental domain. In section 3 we briefly describe an existing method and developed enhancements to control the robot in such a way to approach a desired region in the observation space. Note that this is a difficult problem since the relation between robot's actions and their effects is not known in advance, and has to be learned. Section 4 describes some initial experiments in the robot-object domain. At the end we discuss results, relation to XPERO project and future work.

2 Problem definition

The task of automated modeling includes the environment we wish to model and the agent to perform the assignment. The quantities of the environment the agent can directly observe will be referred to as the agent's observation vector, or *observations*. The interface by which it can influence the environment will be called the *action*. The problem is to obtain a model of the environment which, given the agent's current observations and goal state, enables to determine an agent's action that will result in agent reaching the goal.

Even for domains with a simple environment and simple agent, such as a moving robot, the direct relationship between the action variables and observation vector tends to be complex. Such relations may be expressed by complex equations, that are difficult to simplify without guessing suitable intermediate variables. On the other hand, part of the environment independent of the robot, which will be referred to as the *narrow environment*, can be modeled by simpler equations. Simpler models corresponding to the narrow environment are usually easier to learn. In the case when an explicit global model of the environment is not required, the problem can therefore be split into two parts. The first is the problem of determining the data points in the narrow environment required to make the model, while the second problem is to choose the actions that will guide the robot to the required points. This paper presents how a robot controller can be employed for the latter task.

3 Methods and algorithms

As the function of the data-collecting algorithm is limited to providing the means for higher process exploration, the problem can be posed as the task to reach a given position in the observation space and to gather examples there. To this end, an algorithm must already posses some notion of the effect actions have on observations. The general idea is that this notion does not need to be globally correct, and that the effects of the actions are therefore easier to model than the model corresponding to the higher process.

Given the goal position in the observation space where examples are lacking, the algorithm could perform as follows. First, in the vicinity of its current position, examples of various actions and the resulting observations would be gathered. When this data would be deemed sufficient, a model would be created, and this model would then be used to determine an action that moves the robot closer to the goal position. This process would be repeated until the goal position is reached.

There are several properties that an appropriate algorithm for the task should posses. It would have to be able to start with no specific knowledge of the domain, the modeling and moving process would have to be automatic, it should perform with as few examples as possible, and it would have to be able to reach an arbitrary position in the observation space. An attractive method for this problem is locally weighted learning[6], and Shifting setpoint algorithm[7] has many of the needed characteristics.

3.1 Basic algorithm

Shifting setpoint algorithm, or SSA, has been developed as a general robotic controller with the ability of self calibration. Its main characteristic is the execution on two levels. The lower level works as a dynamic regulator, enabling the robot to preserve its position in the environment, while the higher level guides the robot to the given goal. The switch between the lower and the higher level is determined by the reliability of the collected data. If a model constructed from the collected data has a higher reliability score, there is a greater probability that the action based on the model will move the robot closer to the goal state.

The algorithm has been tested in a real application using linear models and with an optimization criterion that minimized both the state error and the size of the action by implementing linear quadratic regulation. With these mechanisms it has successfully learned the task of juggling a stick on a two dimensional plane[7].

3.2 Modifications of the basic algorithm

Our task of continuously guiding the robot in observation space has a few notable differences compared to the task of learning to juggle[7]. In both problems the autonomy of the learning process is desired, with as little of human intervention as possible. However, due to inherent instability of the juggling dynamics, it is unlikely that the problem could be solved without manually reseting the state of the environment at each failed attempt. On the other hand, with data collecting task, the robot itself could be constructed in a way to increase stability, thus reducing the need for environment resets.

Simplifying the problem of determining actions that preserve the robot observation state, more focus can be put into other areas, such as reliably detecting weak dependencies, accounting for the possible dependencies of action variables, and improving the greedy nature of the goal approaching process. Following these newly defined directions, modifications to the basic algorithm include introduction of exploratory behaviour in the lower level of the SSA, using a higher order model in the upper level, and coupling it with a progress based search algorithm.

The following sections describe the choices for the models and training methods, the procedures to determine appropriate actions based on the models, the methods that use these actions to collect relevant data, and the ways to approach the goal state.

3.2.1 Local models

To model the relationship between actions and observations two approaches can be used, and inverse and forward one. An inverse model is supplied the goal observations and computes the actions, while the forward model computes observations from the actions. To be able to independently define the domain of valid actions, we use the forward model.

To reduce the number of required data points to create a reliable model, simpler models are preferred. On the other hand, in situations where there are pairs of dependent action variables, at least a quadratic model is required. Depending on the task we employ a linear or quadratic model, while both are trained using linear regression.

3.2.2 Action selection

Using forward models enables us to choose an action and simulate the resulting change in observations. This enables us to freely constrain the actions, for example by a predefined action cost. The negative effect is that the action is not calculated directly, and requires an optimization algorithm.

If the action variables are limited by thresholds, the model used is linear, and the error is measured by absolute difference, we can use linear programming as an optimization algorithm by introducing new variables and constrains representing error intervals at each observation variable. The optimization criterion is then to minimize the size of these intervals.

Using a quadratic model or a different measure for error makes the search for optimal action less straightforward. A simple solution is to use a general but slow and possibly imprecise grid search algorithm.

3.2.3 Data collection

Data collection is based on the paradigm of lazy learning, where all the past experience is stored. To this effect, following every action issued to the robot, the example is added to the database. The example contains starting observation, action, and the resulting observation, and is indexed by starting observation. Starting observation defines locality of our models, and this organization enables fast retrieving of data for a required model. The model is trained using the actions and changes of observations.

3.2.3.1 Data quality

The usefulness of the induced model depends on its reliability and on the portion of the observation space it covers. The reliability is directly linked to the number of gathered examples and their distribution. Given a distribution of examples in the observation space, a model with a larger neighbourhood will be more reliable, yet also more averaged and as such possibly too ambiguous to be useful. A model built on a smaller neighbourhood will have fewer examples and be less reliable, but this can be remedied by accumulating more examples.

A simple way to gather examples is by using a random action chosen uniformly from a predefined domain. Since the size of an action generally defines the size of the impact on the observations, if actions are large, the examples will be dispersed in the observation space and the resulting model overaveraged. If the actions are small, the changes in observations might be too limited to include the knowledge that is necessary to efficiently move in the observation space, or may prevent the discovery of weak dependencies.

In an attempt to accumulate all the necessary information, we try to define the appropriate action domain using the effect actions have in the observation space.

3.2.3.2 Model based action domain

One way to achieve an even distribution in a small observation neighbourhood is to use actions we expect to create small random changes in the observations. This procedure requires some prior knowledge of the action–observation dependencies.

To obtain such knowledge, we construct a simple model for this purpose. If such model is linear, only a few examples will need to be collected until the actions are appropriately chosen to produce examples in the neighbourhood. Even when the simple model is unreliable, the chosen actions most likely will not be worse than random ones. When there is enough examples, a reliable model of a higher order can be constructed with the data.

The drawback of this approach is that not much is known about the action domain the simple model defines, which may be very different from the one defined by for instance a cost an action has for the robot. Also, it may still limit the discovery of dependencies that require large actions.

Our proposed solution to this problem is the algorithm that uses both random actions and actions that target a specific observation state.

3.2.3.3 Alternating data collection algorithm

Alternating algorithm combines the uniform collection of examples in action space and regulation to stay in the narrow observation neighbourhood of interest. Since the tasks are mutually exclusive, the algorithm switches between them depending on the current position in the observation space. If the robot is currently in the neighbourhood of interest, it performs a random action. If the robot is outside of it, it uses the simple model to determine an action that will return it into the neighbourhood.

The regulating part of the algorithm effectively tries to return the robot to the original position in observation space to be able to perform another experiment under the same conditions. Although the observations do not contain all the information of the environment, it plays a similar role as a manual environment reset.

3.2.4 Strategy search

Allowing large actions can help the algorithm in the upper level to accomplish an arbitrary goal. However, the action selection is limited by the use of a simple local model. To additionally help the algorithm, we define a process in which the goal state is replaced with the goal space, whose dimensionality is reduced each time the algorithm reaches it. If it is unsuccessful, the goal space is expanded and a different order of reduction is used. This enables the algorithm to approach the goal state in different directions, hopefully avoiding local minima.

4 Experimental results

The purpose of the tests is to examine the principal limitations of the basic and extended algorithms. Considering the difficulty of obtaining a general algorithm for the task, we selected a simple and easy to understand environment, and performed the tests using a noiseless computer simulation.

The goal of the tests is to assess the ability to detect small dependencies between actions and observations, and additionally the ability to reach the designated positions in the observation space.

4.1 Test conditions

Our environment represents a simple robot–object domain. Object properties are position in the xy plane (object_x, object_y), while robot properties are position (robot_x, robot_y) and orientation (robot_or). The robot observations are distance to the object (object_dist) and angle to the object from the current orientation (object_angle). The robot actions are the distance to travel with the next move (move_dist), and the angle to turn during the move (move_angle).

The data available to the algorithm comprises the observation vector: object_dist from the interval [0, 32] and object_angle from [$-\pi$, π], and the allowed actions that have move_dist from the interval [-1, 1] and move_angle from [-.2, .2]. The reason for such action domain is that the largest action normally produces around 3% change in the observation domain, which may be lower or higher depending on the robot state.

4.2 Detection of weak action-observation dependencies

The goal of this test is to determine which methods of example collection are appropriate or necessary to detect a relatively small influence an action variable may have on an observation variable.

Four methods of example collection are examined. *RN method* uses random actions, *WN method* uses a simple model and a wide observation neighbourhood, *NN* uses narrow neighbourhood, and *AN method* uses the alternating algorithm. Fig. 1 to Fig. 4 show typical distributions of examples using these methods.

The neighbourhood in observation space where the examples are collected to train a model are referred to as the *area* of the model, and its size is expressed in the percentage of the observation domain. The area is *static* when its centre is set to the robot position at the start of the example collection, and *dynamic* when the centre moves together with the robot as it is acquiring new examples.

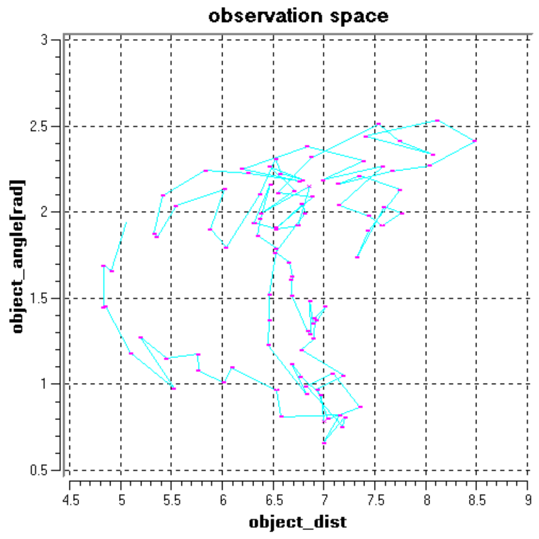

Fig. 1 Example distribution using RN

Fig. 1 shows the distribution of examples using the RN method. Examples are collected using random actions, until there is enough of them to form the final, quadratic model. The final model has a dynamic 5% area.

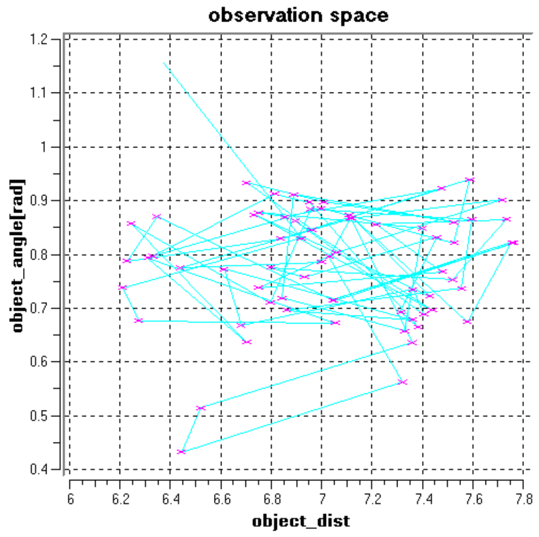

Fig. 2 Example distribution using WN.

Fig. 2 shows the distribution using the WN method. The actions are determined using a linear model with dynamic 5% area that targets random points in the area of the final quadratic model. The final model has a 5% static area.

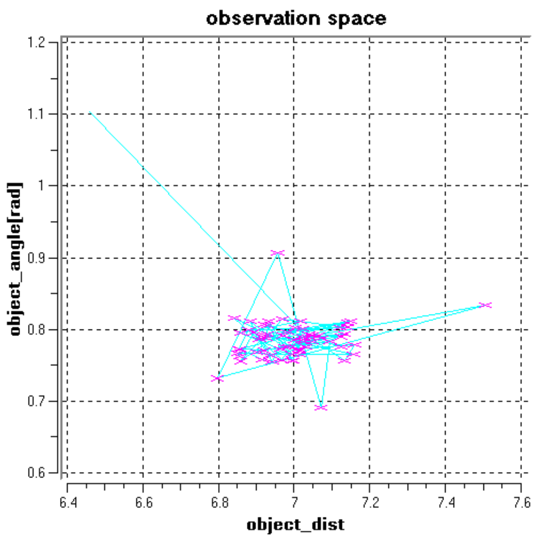

Fig. 3 Example distribution using NN.

Fig. 3 shows the distribution using the NN method, which is the same as WN, except that the final model has a 1% static area.

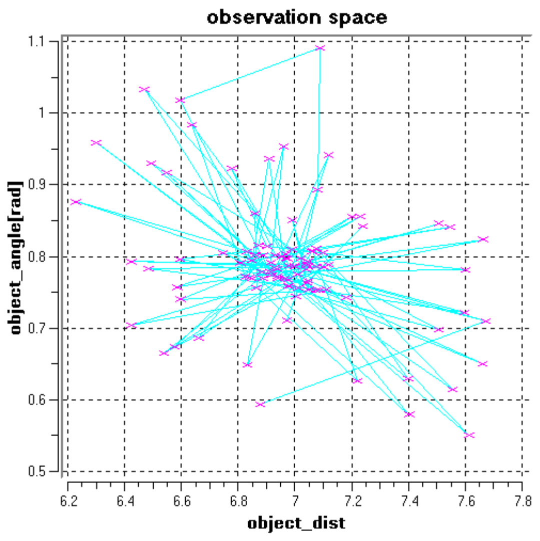

Fig. 4 Example distribution using AN.

Fig. 4 shows the distribution using an AN method. If the current robot position is inside the final model area, examples are collected using random actions. If the current robot position is outside of the final model area, the actions are determined as with the NN method. The final model has a 1% static area.

To asses different collection methods the final model of each method is used to predict an action that will move the robot closer to the object. Both move_dist and move_angle influence object_dist, but the effect of move_angle is much smaller.

The starting robot distance to object is 7 and the orientation is π/4 away from being directed at object. The influence move_angle has on the object_dist depends on move_dist, however, since the maximum turn a robot can perform is only about 10 degrees, this is always significantly lower than the dependency between move_dist and object_dist.

If move_angle is selected with the correct sign, the model is considered successful. The selection of move_dist is trivial and is ignored.

Tab. 1 Success rates for collection methods

N	RN	WN	NN	SN
2	36	59	56	51
3	47	55	62	61
4	42	48	64	71
5	37	47	66	90
6	35	42	67	83
7	39	52	54	94
8	46	49	67	97
9	43	52	66	100
10	44	55	68	100
11	41	65	72	100
30	54	61	70	99

Tab. 1 shows the success rates for all collection methods with N examples collected to train the final model. The success rates are defined as the number of successful models out of 100 trained.

Tab. 2 Efficiency of collection methods

N	RN	WN	NN	SN
2	96	59	21	20
3	72	43	32	26
4	60	49	40	27
5	55	53	46	30
6	50	50	53	31
7	44	59	55	34
8	40	64	58	35
9	34	68	62	36
10	30	63	61	38
11	28	72	65	38
30	8	86	81	44

Tab. 2 shows the efficiency for all collection methods. The efficiency is measured as the percentage of the examples used to train the final model from all gathered examples.

4.3 Approaching goals in observation space

The goal of the following test is to measure the ability of the algorithm to reach an arbitrary position in the observation space. For this purpose, a series of positions are chosen at random from the observation space. The goal of the algorithm is to reach those positions in sequential order.

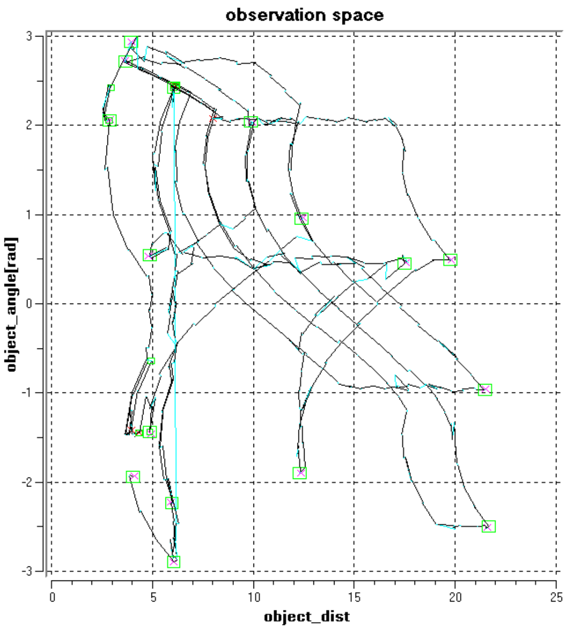

Fig. 5 Robot path in observation space.

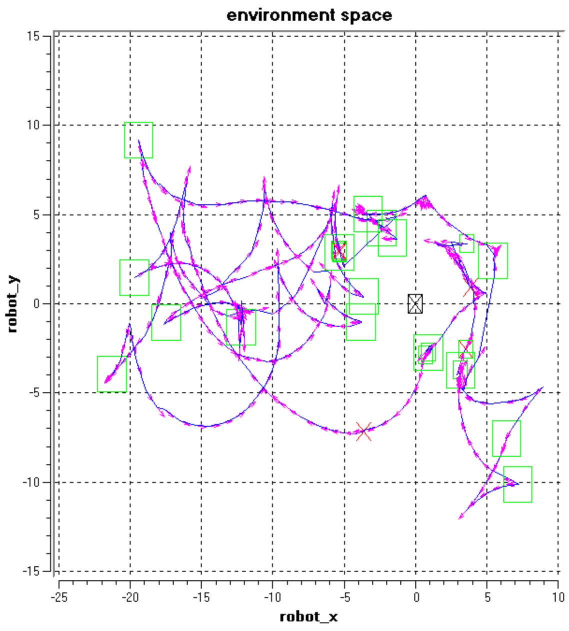

Fig. 6 Robot path in environment space.

An example using 16 positions is illustrated in Figs. 5 and 6. Fig. 5 shows the robot path in the in the observation space, while Fig. 6 shows the robot path in the environment. The position of the object is marked with the crossed square.

Tab. 3 Success rates depending on goal size

Goal size	.5	1.5	5
Success rate	94	98	100

The success rates of the algorithm were measured by the number of successfully reached positions out of 100 specified. Tab. 3 shows the results for various goal sizes, measured in percentage of the observation domain.

5 Discussion and conclusions

5.1 Interpretation of results

In all experiments, the maximum action is purposely limited to produce manageable changes in the observations. Still, collection with random actions produces scattered examples, as seen in Fig. 1. Fig. 2 and Fig. 3 show that introducing a simple model to focus the search alleviates this problem. Using a combination of methods as with AN, the results are naturally more dispersed. However the switch criterion based on the position in the observation space ensures the robot never leaves the vicinity of the starting point, as happens often with random actions. In addition, a large fraction of the examples lies in the close vicinity, as Fig. 4 demonstrates.

Although Tab. 1 indicates the only reliable model for any reasonable number of examples is obtained with the AN collection method, the results of other methods are still included as control data. As the method WN collects the data in the roughly the same area as AN and with the actions of the same magnitude, the difference of success rates can possibly be attributed to either a better distribution of examples in action space with AN, or a WN producing too ambiguous model due to larger area, with the latter being more probable.

On the other hand, the comparison of AN and NN suggests that to obtain a model focused in the narrow area in observation space, there is a possibility that the restriction on the actions would limit the ability to detect weak dependencies, which are only sufficiently exposed with large actions.

Tab. 2 provides additional insights and control information. The efficiency of the random example collection drops rapidly when more examples are needed, as the chance that the algorithm will remain in the given vicinity is smaller. WN and NN share about the same efficiency, which becomes larger with more examples. This is also expected, since the linear model that guides the robot to stay inside the area that is being investigated becomes more reliable as examples are added. This is also true for AN. Since large random actions almost necessarily move the robot outside of the neighbourhood of interest, the maximum efficiency of the AN algorithm would be a half of the WN or NN. The results confirm this expectation.

The efficiency of the WN, NN and AN methods show that the linear model only rarely fails in its attempt to guide the robot inside the defined area. Since this problem is in many ways similar to guiding the robot to an arbitrary position in observation space, one may wonder why the linear approach is convenient for the former task, but not the latter.

The necessity of having a quadratic model for guiding the robot in the observation space follows from the geometry of the robot movements. The direction in which the robot turns can bring it closer or farther away from the object, depending on whether the robot is moving forward or backward. This dependency between move_angle and move_dist cannot be expressed with just a linear combination, while a quadratic model has a possibility of modeling such behaviour with the product of the two. Additionally, as previously mentioned, to be able to detect minor action-observation dependencies, the examples must also be gathered in a relatively narrow part of the observation space.

The reason why linear model is successful without any of the two properties could lie in the fact that in many cases, move_angle chiefly defines the object_angle, and move_dist chiefly defines the object_dist. To reach a nearby observation position, it is enough to roughly model these two major dependencies, while ignoring the other, minor combinations. This would suggest that in this domain, gathering examples in the vicinity requires less knowledge than reaching an arbitrary position in observation space.

Tab. 3 shows the success of reaching such positions. Judging from the experiments not presented, hard-to-reach positions in this domain are rare among all possible positions, so the results are expected, but not particularly insightful. The conclusions that can be drawn are that there are goal states that are difficult to reach for our algorithm, although it seems that it always manages to get in the vicinity of the goal.

5.2 Future work

One of the main topics that still require further investigation is the relationship between the size of the actions and convergence of the SSA. Since SSA seems to retain many limitations of the greedy algorithms even when used with goal space reduction, the amount of space the robot can observe while deciding on the next move is of critical importance.

While larger actions may help to improve convergence to an arbitrary observation position and even make goal space reduction search redundant, they could also require local models to be more complex. The usefulness of these modified methods will in part be dependent on how easy it is to return to the vicinity of the observation area that is being explored from the area reachable by large actions.

Another important topic of research is the possibility of using multiple local models of varying degrees and possibly other parameters, and then choosing among them by measures such as reliability and accuracy. This would trade efficiency of the algorithm for adaptability, which may be reasonable considering the progress in hardware computing capability. In addition, before the algorithm would be of practical value, the sensitivity to motor and sensor noise would have to be examined closely.

In the context of XPERO project, we plan to use the described algorithm with machine learning methods to close the autonomous discovery experimental loop. In such a scenario, the robot would first randomly collect some data that would be used to induce an initial model of the environment. The robot would then be required to collect the data in the regions of the state space where the reliability of the current model is low, or where the machine learning methods induced some interesting, but unexpected hypotheses. The collected data would be repetitively used to improve the control of the robot and to update the model of the environment.

6 Acknowledgment

This research was partly supported by the European Commission's 6th Framework Project XPERO, and the Slovenian research agency ARRS (research program Artificial Intelligence and Intelligent Systems).

7 References

[1] http://www.xpero.org.

[2] Jaime Carbonell, Oren Etzioni, Yolanda Gil, Robert Joseph, Craig Knoblock, Steven Minton, and Manuela Veloso. Planning and Learning in PRODIGY: Overview of an Integrated Architecture. In Goal-Driven Learning, Aswin Ram and David Leake (Eds.), MIT Press, Boston, MA, 1995.

[3] S. Yamada and R. Iwasaki. Integration of Operator Learning, Planning and Execution in a Heterogeneous Multi-Robot System. *International Symposium on Robotics (ISR-2001)*, Seoul, Korea, 2001.

[4] R. D. King, K. E. Whelan, F. M. Jones, P. G. K. Reiser, C. H. Bryant, S. H. Muggleton, D. B. Kell, and S. G. Oliver. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427:247-252. 2004.

[5] I. Bratko, D. Šuc, I. Awaad, J. Demšar, P. Gemeiner, M. Guid, B. Leon, M. Mestnik, J. Prankl, E. Prassler, M. Vincze, J. Žabkar. Initial experiments in robot discovery in XPERO. *Workshop Concept Learning for Embodied Agents, ICRA 2007*.

[6] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11-73, 1997.

[7] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 11:75-113, 1997.