U N I V E R S I T Y O F T A R T U
DEPARTMENT OF MATHEMATICS AND INFORMATICS
Institute of Computer Science

Juri Gavšin

Using the Concept of Reversibility to Develop Safe Behaviours in Robotics

Master's thesis

Supervisors: Maarja Kruusmaa, Ahto Buldas

TARTU 2007

Table of contents

1 PREFACE

Global trends in robotics research show that robotics is getting more concerned with applications in real-world environments. Robots are moving from industrial environments and research laboratories closer to humans. They are moving to the streets, hospitals, homes, supermarkets and other public places where they closely interact with humans. In these environments safety and reliability of robots as well as robustness of their behaviour is getting more important.

This thesis deals with an aspect of this research trend by investigating an idea of implementing a general law that would increase safety and reliability of robots.

1.1 Objectives

The primary objective of this thesis is to show, analyze and discuss experimental evidence that following some abstract idea can lead to a concrete and safe behaviour. The secondary objective is to build a programming framework for implementing and comparing different learning algorithms using a Khepera-compatible (or similar) robot and to implement easy-to-extend reversibility-based algorithm.

1.2 Contribution and overview

This master thesis is concerned with a particular way of generating reliable behaviours of robots based on abstract ideas. The thesis is arguing about the idea, that the ability of undoing its actions can be useful for generating reliable behaviours for robots. In particular, in this thesis we demonstrate that an abstract rule “Don’t do things that you can’t undo” leads to a concrete safe behaviour – obstacle avoidance, and can also lead to higher level of behaviours.

However, the algorithms and cases described are simpler than in a real life scenarios, e.g. during interaction with moving objects (humans, other robots). This thesis does not intend to present a ready-to-use theory, but to prove a concept and make some suggestions for further development.

My contribution of this thesis is:

1. validation of the idea of reversibility presented by Kruusmaa and Eppendahl
2. extension of the idea of reversibility to action sequences

3. definition of the global and local reversibility
4. development of the programming and test environment to validate the above mentioned ideas and concepts
5. conducting experiments with the developed tools
6. analysis and interpretation of the experimental results

Chapter 2 contains introductory information for this thesis: it shortly describes the research field – developmental robotics, and introduces the concept of reversibility.

Chapter 3 contains theoretical part of the thesis, introducing formal definitions for the notions used further. It also contains a short discussion about weaknesses and strengths of our approach. The first half of this chapter is based on theory developed by Maarja Kruusmaa and Adam Eppendahl (see appendix B), simplified in some areas, but extended and changed in others.

Chapter 4 contains the descriptions of the experiments and algorithms.

Chapter 5 contains test results with a short analysis.

This thesis also contains conclusions, suggestions for further development, as well as the list of references used in this thesis. The summary is in English and Estonian languages.

The CD which is provided along with this document contains the source code, test data, required input files, a copy of this document, as well as some additional sources. Contents of the CD is described and explained in appendix A.

Appendix B is a copy of the article, written by Maarja Kruusma, Juri Gavšin and Adam Eppendahl, which is based on the experiments made by the author of this thesis.

2 INTRODUCTION

2.1 Epigenetic robotics

Epigenetic[1] **Robotics** (a.k.a. **Developmental Robotics**), is a relatively new approach in robotics (and artificial intelligence), which combines developmental psychology, neuroscience and biology with robotics and computer science. The terms **Epigenetic Robotics** and **Developmental Robotics** are essentially equal, with the difference that the latter one is more concerned with integration of developmental psychology and robotics (like formalization of theories in developmental psychology, which are often informal), and the former one has a broader interdisciplinary emphasis.

Epigenetic Robotics studies control systems' development through interaction with the environment, which implies that the system is embodied, the environment situations can be identified and the system is evolving through a prolonged epigenetic developmental process. It also implies that the machines must go through a learning process (supervised or unsupervised) as infants do. The research includes practical goals of:

- enabling robots and other artificial systems to better adapt to their environments, and to better adapt to changes in these environments
- simplifying the problem of programming robots by programming the robots to develop skills for any particular environment instead of programming robots for specific environments

[ER1]

In [ER2], authors do not divide this area into discrete partitions, but distinguish several main regional trends in a continuous research space:

- ***Interaction studies*** – basic social interactions, like low-level imitation, joint visual/shared attention, early language development, etc.
- ***Sensorimotor Development*** – basic sensor-motor joint work coordination, like gaze fixation, hand-eye coordination, navigation, etc.

---

[1] ***Epigenesis*** – a biological „theory holding that development is a gradual process of increasing complexity. (This contrasts with preformationism, which holds that the organism is already present in the gamete(s), merely growing and unfolding during development.) For example, organs are formed de novo in the embryo rather than increasing in size from pre-existing structures." [BC1]

- ***Active Vision*** – real-time, continuously operating vision systems able to adjust their visual parameters to aid task-oriented behaviour
- ***Motivation*** – internal motivational value/reward systems for self-motivation, stability/exploration/exploitation balancing, etc.
- ***Emergence of the self*** – high level cognitive abilities associated with robot self-discovery, like self-identification, distinction between "I" and "You", etc.
- ***Dynamics of development*** – study of the steps in the emergence of perception throughout the learning process.

This thesis addresses the problem of motivation and regulation of sensorimotor interaction, associated with two of the above trends **- Sensorimotor Development** and **Motivation**.

2.2 Reversibility as a basis for safe behaviour

In [ER3], a number of basic visual behaviours (tracking of the moving light, exploration for other light sources) are seen to emerge from abstract motivational principles – stability, predictability and familiarity. The general idea is to identify principles that can be expressed without reference to the ground meaning of sensor-motor values, with the expectation that code based on such principles will function reliably in a broad range of environments and on different robots or on different parts of the same robot.

In [ER4] it is proposed that the principle "don't do what you can't undo" is one of those basic abstract principles that can be used to guide the robot's behaviour. Authors also proposed that obstacle avoidance is a natural consequence of that principle and conducted a 1-dimensional test (the robot moved back and forth between two objects) to back up their hypothesis.

The suggestion is that reversibility, being a necessary condition of controllability, is a fundamental concept when programming robots to behave safely and reliably. The most undesirable actions in the real world (for example, those that damage the robot or the environment) are characterized by irreversibility. Thus, instead of teaching the robot specific routines such as avoiding collisions, avoiding falls, etc., it is better to teach the robot a more general principle of avoiding irreversible actions.

2.3 (Irr)reversibility examples

For example, falling down the stairs is not good because the robot does not know how to climb back. Closing the door is not good because it does not have knowledge of how to open it.

Though, reversibility of the action should not be considered a binary reversible/irreversible choice, but a continuous value, since some of the actions are reversible very easily without any significant effort, but some of them take a lifetime to reverse them. For example, if the stone floor is soiled by spilled coffee then this action is usually easy to reverse by using a wet cloth, but a carpet soiled in similar way is much more difficult to clean back, sometimes impossible, and if the carpet is burned, then this action is most definitely irreversible. By considering this, one could, at least theoretically, calculate what is the cost of spilling coffee and how desirable this action is.

2.4 Reversibility as an extension of stability

The above examples demonstrate that one of the main consequences of reversibility is stability. It can be even argued that they are equal, but, actually, they are different. Since stability is mostly an objective notion, it does mean, that relevant values must be stable, i.e. to stay within some allowed limits. Reversibility doesn't have that limitation and it can be defined so, that the "goodness" of values does not decrease, and, since "goodness" is always a subjective notion, the reversibility itself can be subjective.

Reversibility can be informally defined as a "similarity" of the state before the action and the state after making the action and its counteractions. Thus, if the action has a known counteraction, and the agent (person, robot, etc) knows (from his own or somebody else's experience) that this action and the counteraction in this situation will put the things back the way they were (or acceptably close to that), the action is safe.

One of the problems is that most of the situations are not exactly the same as experienced ones for a robot, especially autonomously developing one. If the state can somehow be identified as non-novel (the best synonym, in my opinion, is "familiar"), i.e. if the state in question is acceptably similar to some of the experienced states with known feedback for the action in question, then information for those states can be somehow interpolated to predict the outcome.

It can also be argued, that not all actions have exact counteractions and if someone is doing something, he/she usually wants to make things better. For example, if the mobile phone does not work properly and this can be fixed by soldering one wire back, the phone after fixing will not be the same, but, definitely better. These cases can be dealt with in the same way, as the ones, where all the actions have exact counteractions, by slightly modifying the definition of reversibility.

For example, if the latter state differs from the original one only positively (in the last example: the phone was virtually the same, but one thing fixed, i.e. better), then the states can be considered similar (reversible) enough. This addition makes it unclear how to decide what is good and what is bad; when solving a specific problem such modified definition is certainly context dependent.

3 REVERSIBILITY MODELS

3.1 Introduction

A **reversibility model** tells the robot which actions are reversible and how to reverse them if they are.

In a fixed, known, exact, deterministic world, modelled by a graph G of states as nodes and actions as links, an action from state s to state s' is reversible if there is a path back from s' to s. Finding reversibilities in G is equivalent to finding loops in G, a standard problem in graph theory. This is all very well for playing games like Sokoban, but real robots face a non-deterministic, inexact, partially known and changing world.

Therefore, we model non-determinism using labelled transition systems, we allow inexactness with (hemi)metrics on the space of states, and we define a reversibility model pragmatically to be a set of expected reversibilities that may grow or shrink as the robot gains experience.

In this thesis we consider one such change in robot's world, addition of sensors to the definition of the world's state, and introduce a notion of refinement that captures the relationship between the robot's world before and after the change. In the learning experiments we describe, a reversibility model for an unrefined world is adapted to a refined world (with side-effects of producing obstacle avoidance and a "stay in a safe area" behaviour).

3.2 Definition of reversibility

Def: A robot's **world** is a labelled transition system[2] $(S, \Lambda, \rightarrow)$, where S is a set of experienced environment states, Λ is a set of labels (a label represents an action or a sequence of actions), and $\rightarrow$ is a set of labelled transitions between states.

---

[2] A labelled transition system is a tuple $(S, \Lambda, \rightarrow)$ where S is a set (of states), Λ is a set (of labels) and $\rightarrow \subseteq S \times \Lambda \times S$ is a ternary relation (of labelled transitions.) If $p, q \in S$ and $\alpha \in \Lambda$, then $(p, \alpha, q) \in \rightarrow$ is written as: $p \xrightarrow{\alpha} q$. This represents the fact that there is a transition from state p to state q with label α α. Labels can represent different things depending on the language of interest… [WI1]

Def: Let **action** be an atomic action or a composite action (sequence of atomic actions).

When the result of an action a in state s is not wholly determined by the robot, multiple transitions from s are labelled with the same action a and it is the world that determines which transition actually happens.

Def: Action a_1 is a **counter-action** of action a_0 (denoted $a_1 = -a_0$), if a_1 is expected to undo the action a_0, i.e. the sequence $a_0 a_1$ is expected to end in the same state where it started (or acceptably close to it).

Def: A ***reversibility*** for a world W is a pentuple of three experienced states and two actions that initiated transitions between them $(s_0, a_0, s_1, a_1, s_2)$, so that $a_1 = -a_0$, i.e. the composite action $a_0 a_1$ produced a transition from s_0 to s_2 through s_1 in W and, in such transition, state s_2 is expected to be acceptably close to s_0 for any states x and y with $d_{orig}(x, s_0) \leq \varepsilon_{orig}$ and $d_{dest}(y, s_1) \leq \varepsilon_{dest}$, where d_{orig}, d_{dest} are metrics[3] on states and ε_{orig}, ε_{dest} are their thresholds.

Def: The ***reversibility*** $(s_0, a_0, s_1, a_1, s_2)$ ***holds*** in W if $d_{rev}(s_0, s_2) \leq \varepsilon_{rev}$, where d_{rev} is a hemimetric[4] on states and ε_{rev} is a threshold; and **fails** otherwise.

Generally speaking, $d_{rev}(s_0, s_2) \leq \varepsilon_{rev}$ means that the distance from the initial state of the reversibility sequence to the final state is "acceptably close".

3.3 Initial and refined reversibility models

Def: A ***reversibility model*** for a world W is a set of reversibilities that are expected to hold in W.

In practice, a reversibility model could be given in advance, communicated to the robot, learned empirically, deduced from knowledge about the world, or obtained

---

[3] A **metric space** (S, d) is a set S together with a function $d : S \times S \to \Re$ (a **metric**) which satisfies three following conditions:

1. $\forall x, y \in S : d(x, y) = 0 \Leftrightarrow x = y$
2. $\forall x, y \in S : d(x, y) = d(y, x)$
3. $\forall x, y, z \in S : d(x, z) \leq d(x, y) + d(y, z)$

[4] A **hemimetric space** (S, d) is a set S together with a function $d : S \times S \to \Re$ (a **hemimetric**) which satisfies two following conditions:

1. $\forall x \in S : d(x, x) = 0$
2. $\forall x, y, z \in S : d(x, z) \leq d(x, y) + d(y, z)$

in some other way. In the experiments described in the thesis, the robot is given a model for world where all actions in all states are reversible and uses this to learn a model for a refined world.

Def: A ***refinement (of states)*** from a world W to a world W' is a surjective function p from the states of W' to the states of W.

In other words, every state in W is the image of one or more states in W', which "refine" the state in W.

Def: For any reversibility model R for a world W and for any refinement from W to W', with state function p, there is an ***initial refined set of reversibilities*** R' in W' defined as $R' = \{(s_0, a_0, s_1, a_1, s_2) : (p(s_0), a_0, p(s_1), a_1, p(s_2)) \in R\}$

To obtain a reversibility model for the new world W' we may form R' and then remove reversibilities that fail in the refined world. An important aspect of this procedure is that "it gives the robot something to do", though, making its judgements too optimistic: the original model R provides a list of actions together with the circumstances in which they should be tried.

A refined reversibility model in a refined world should be used in conjunction with new (hemi)metrics and thresholds, since old ones are, generally, void or trivial in the new world.

3.3.1 Example of refinement

The kind of refinement we have in mind is produced by extending a robot's sensor vector. Suppose we have a trivial world with an empty sensor vector and the single state S' and actions (labels) given by pairs of integer wheel displacement commands (m_L, m_R). This world is deterministic, all actions are reversible and a good non-trivial reversibility model R can be given by taking $a_1 = (-m_L, -m_R)$ when $a_0 = (m_L, m_R)$.

Now suppose we include one proximity value (say, the front sensor) in the state vector (d_F). Assuming the new sensor does not affect the robot's environment, we obtain a refinement of the original world. The state function *p* is the projection

$$p((d_F)) = (\ ) = S'.$$

When the simple model *R* described above is refined according to this new world, some of the refined reversibilities hold and some do not. In our experiments, the robot tests these refined reversibilities to discover which hold and which fail.

The interesting point here is that the ones that fail generally correspond to collisions of some sort. Consider the following four cases (in which wheel commands' values and proximity values are given, without loss of generality, in comparable units – moving n wheel command units forward decreases the distance to the wall by n proximity sensor units).

(1) The robot does not touch anything, we obtain the successful reversibility:

$$(S(15), \mathrm{A}(10,10), S(5), \mathrm{A}(-10,-10), S(15)),$$

where the robot approaches and retreats from an object without touching it.

(2) The robot touches an object and the object slides, we obtain a failed reversibility:

$$(S(8), \mathrm{A}(10,10), S(0), \mathrm{A}(-10,-10), S(10)),$$

where the robot runs into an object, pushing it 2 units forward, then retreats, and then finds that its proximity sensor now reads 10 instead of the original 8.

(3) The robot touches an object and its wheels slide: from the robots point of view, this is identical to case 2.

(4) The robot touches an object and jams, if motor commands time-out and report success, adjusting the wheel encoder counts as necessary, then this case is again similar to case 2 (and may be thought of as a kind of internal sliding) – we obtain a failed reversibility:

$$(S(8), \mathrm{A}(10,10), S(0), \mathrm{A}(-10,-10), S(0)).$$

Not only does the robot discover that it is ”bad” to push things — without ever knowing what pushing is — but the refined state allows the robot to distinguish those cases in which 'bad things happen' from those in which they do not (by using the additional sensor(s) to distinguish different states in a refined world).

Once the robot learns a valid reversibility model, it may use the model to censor its actions. Note that it is our method of creating a ”pushing is bad” model out of initially refined R' (by pruning it).

3.4 Local and global reversibility of composite actions

It is sometimes beneficial to explicitly be aware that an action is a sequence of some more detailed actions. For example, action “make 100 steps ahead” consists of 100 single-step actions and each single-step action actually consists of several even smaller actions like “strain muscle x”, “relax muscle y”, etc.

Actually, the same definitions of reversibility can be used, by assuming that each action is a sequence of sub-actions – $a_0 = (a_{0,0}, a_{0,1}, a_{0,2}, ..., a_{0,n})$ and every $a_{0,i}$ has a *reverse-action*), and discarding any knowledge about intermediate states while considering the reversibility of the action (a_0) as a whole.

Def: A ***trivial reverse-action*** of a composite action $a_0 = (a_{0,0}, a_{0,1}, a_{0,2}, ..., a_{0,n})$ is another composite action a_1, consisting of a_0 sub-actions' reverse-actions in an inverse order, i.e. $a_1 = (a_{1,0}, a_{1,1}, a_{1,2}, ..., a_{1,n})$, where $a_{1,i} = -a_{0,n-i}$. This is also denoted by $a_1 = -a_0$.

A reversibility of a single non-composite action or the composite action in a context where the composition does not matter, is called ***local***, a reversibility of an explicitly composite action is called ***global***. The notions of *local* and *global reversibility* are relative and are heavily dependent on the context, in which they are considered, i.e. globally reversible composite actions can be a part of more complex actions, where their own complexity is not important and their reversibility in such context is considered local.

Def: A composite action $a_0 = (a_{0,0}, a_{0,1}, a_{0,2}, ..., a_{0,n})$ from state s is ***back-path globally reversible*** if reversibility (s, a_0, s_1, a_1, s_2) holds and $a_1 = -a_0$.

In other words, it is a special case of local reversibility, where actions are explicitly composite and action a_1 is a trivial counter-action of a_0, but any information about intermediate states is discarded. Generally, for a local reversibility to hold a_1 is not required to be a trivial counter-action of a_0.

This definition allows to construct global reversibilities for action sequences of any complexity and of finite length. The notion of back-path global reversibility also relies on the fact that the sub-actions by themselves can be reversed.

Def: A composite action $a_0 = (a_{0,0}, a_{0,1}, a_{0,2}, ..., a_{0,n})$ from state s is ***recursively reversible***, if $n = 0$ or the reversibility $(s_{n-1}, a_{0,n}, s_n, a_{1,0}, s_{n+1})$ holds and the composite action $a_0 = (a_{0,0}, a_{0,1}, a_{0,2}, ..., a_{0,n-1})$ from the same state (or a state acceptably close to it) is also *recursively reversible*, where s_{n-1} is the state after making action $(a_{0_0}, a_{0_1}, a_{0_2}, ..., a_{0_{n-1}})$ from s.

The composite action $a_0 = (a_{0.0})$ with length 1 is *recursively reversible* if the single action $a_{0.0}$ is reversible, since the length of the remaining action sequence is zero and, therefore, is *recursively reversible* by definition.

3.5 (Hemi)metrics and thresholds

Since **metrics** d_{orig}, d_{dest} and hemimetric d_{rev} on states and their **thresholds** (ε_{orig}, ε_{dest}, ε_{rev}) are important to understand reversibility definitions, an explanation can be useful. Let us consider a reversibility $(x, a, y, -a, z)$.

Metric $d_{orig}(x, s_0)$ calculates how far is initial state x from initial state s_0 of some already experienced reversibility $(s_0, a, s_1, -a, s_2)$, i.e. how novel is the state x; if the distance between states x and s_0 is greater than ε_{orig}, then selected reversibility cannot be used to predict how reversible is the action a from state x.

Metric $d_{dest}(y, s_1)$ calculates how far is intermediate state y from intermediate state s_1 of some already experienced reversibility $(s_0, a, s_1, -a, s_2)$ where $d_{orig}(x, s_0) \le \varepsilon_{orig}$; if the distance between states y and s_1 is greater than ε_{dest}, then selected reversibility cannot be used to predict how reversible is the action a from state x.

Hemimetric $d_{rev}(x, z)$ calculates whether the reversibility holds, i.e. how reversible is the action a from the state x by action $-a$; if $d_{rev}(x, z) \le \varepsilon_{rev}$ then reversibility holds, and fails otherwise. The reason to use hemimetric instead of a metric is to allow non-stable, but safe state transitions.

The implementation can calculate novelty/reversibility discretely as yes/no or continuously as, for example, interval [–1, 1] using the thresholds involved to calculate how far the value is from 0.

Thresholds can be adjusted according to the implementation: increased or decreased, or set to 0 or infinity.

If the value $\varepsilon_{orig} \ge \max_{s,s' \in W}(d_{orig}(s, s'))$ (for example, infinity), then it essentially means that none of the states are novel, if at least one suitable reversibility (i.e. forward action of that reversibility and the action in question are the same) has been

experienced before. If $\varepsilon_{orig} = 0$, then it means that the state is not novel only if that particular state has already been experienced.

If the value $\varepsilon_{dest} \geq \max_{s,s' \in W}(d_{dest}(s,s'))$ (for example, infinity), then familiarity of the intermediate state plays no role in familiarity of the whole reversibility and only the familiarity of the initial state matters. There is usually no point in making ε_{dist} too small, if the number of states is large, since this will heavily reduce the number of suitable states available for comparison; but, if the number of states is small, then it might be useful.

If the value $\varepsilon_{rev} \geq \max_{s,s' \in W}(d_{rev}(s,s'))$ (for example, infinity), then all reversibilities hold. If $\varepsilon_{rev} = 0$ then it doesn't yet mean that the action a is reversible from state x if and only if action-counteraction sequence ends in the same state, i.e. $x = z$; hemimetric definition allows two non-equal states to have zero distance and it is not symmetric. Informally speaking, if state s' is "better" than s then $d_{rev}(s,s')$ may be 0, but it does not imply that $d_{rev}(s',s)$ is also 0, i.e. it is safe to go from the worse state to the better, but not vice versa.

Extreme ε_{orig} and ε_{rev} values are often quite meaningless because of their triviality, thus, in general, some intermediate values are strongly recommended.

3.6 Limits / discussion

The generality of the concept of reversibility is the source of both the strength and the weakness of our approach. Although, it is not completely general, it allows a wide range of implementations to use the notion of reversibility without modifying the theory. For example, since the definitions are based on (hemi)metrics, both discrete and continuous spaces can be used. Raw sensor data and internal non-physical "sensors" are usable as inputs for interstate distance computations.

At the same moment, our approach definitely should not be considered as some sort of a panacea. As any other algorithm/concept it has its own limitations and there definitely is a long way ahead to develop the theory and implementations further.

The disadvantage is that the quality of the algorithm strongly depends on the implementation, especially on the choice of right (hemi)metrics (d_{orig}, d_{dest}, d_{rev}) and thresholds (ε_{orig}, ε_{dest}, ε_{rev}). The problem is, as with most algorithms implemented on

real robots, noise and dynamic nature of real-life data. Thus, the (hemi)metrics must somehow deal with the dynamics of the environment and the choice of threshold values must also take noise into account.

Linearity of the sensors' data is also important. If it is very non-linear then the same physical distance between states would result in considerably different internal distance depending on the sensors' values, thus, making it almost impossible to choose the right general thresholds.

There is also a major problem in our approach to distinguish between successfully reversed actions and the sequence of actions, where both the action and the counteraction do not succeed. Since in the latter case the state is, generally, stable and the sequence ends in the same state after it started, the action is considered reversible. This can, actually be fixed by introducing another (hemi)metric to measure a distance between the start and the end points of actions, for example, $d_{move}(s, s')$ *with* a threshold ε_{move}.

A new notion of reversibility holding "strongly" can be introduced so that: reversibility $(s_0, a_0, s_1, a_1, s_2)$ **holds strongly** if it holds and $d_{move}(s_0, s_1) > \varepsilon_{move}$ or $d_{move}(s_1, s_2) > \varepsilon_{move}$. If, for example, $d_{move} = d_{rev}$ and $\varepsilon_{move} = \varepsilon_{rev}$ then this would efficiently solve the problem of distinguishing jamming from successful reversible actions.

4 EXPERIMENTS

4.1 Introduction

The main purpose of the thesis is to collect and analyze experimental data to back up the suggestion that abstract principles, specifically “Don’t Do Things You Can’t Undo”, can generate concrete and safe behaviours. It implies that the algorithm, based on such abstract principle must perform comparably to other well-known algorithms in some non-trivial task. To have the results that can be trusted, the comparison must be as fair as possible, which makes it harder to choose the task to be used in comparison and the algorithm (or algorithms) to compare with.

4.2 The Task and Algorithms

After considering different tasks, the task of obstacle avoidance was chosen. This task is simple enough to understand and to describe. At the same time, it is a very natural and safe feature of an autonomous object (robot) to avoid obstacles. Though, even this simple task can be understood differently. I will describe in detail my version of this task to compare different algorithms as “fairly” as possible.

The task details are simple: the robot makes pseudo-random (using C/C++ `random()` function) moves in non-dynamic real-life environment and the algorithms predict if the action will succeed or not. Robot software receives some input from the environment (through sensors) and can make some actions (through motors commands). The randomness is used to automatically generate test runs. No algorithm affects the robot behaviour (thus, none of them is preferred), which makes it possible to seamlessly simulate the same test runs with new versions of the algorithms. In this task algorithms can be easily compared by the percentage of correctly predicted action successes.

The very obvious algorithm for comparison is the random prediction that returns the possibility of success of the action based only on some internal pseudo-random value. Because of its simplicity, this algorithm can be considered a feasibility check, i.e. if the efficiency of some algorithm (with real return values) is below “random”, there is no point to use it and it is a good reason to redesign the algorithm or try another one.

After choosing a feasibility check algorithm, the "reference" algorithm must also be chosen, but this is much more difficult to do. The problem is that it should perform well in real-life environment and it would be convenient if it is also simple enough to be easily understood.

Also, for the comparison to be "fair", this "reference" algorithm must be of self-learning/trial-and-error type, i.e. it should learn using previously obtained experience without supervision. A very simple reinforcement learning algorithm, described further, is chosen as such an algorithm.

4.2.1 Reinforcement learning algorithm (RL)

The reader can find a very good and through introduction to reinforcement learning in general in [RL1].

Shortly, reinforcement learning is concerned with problems in which autonomous agent searches for the best strategy to act in the surrounding environment by trial–and–error process. The only feedback for the agent is a numeric "reward" for every transition from one state to another and the primary objective of agent's mission is to maximise the long-term reward.

Reinforcement learning algorithms attempt to generate an optimal policy that consists of the best action choice for each state to give the highest sum of rewards in the future. The most popular approach to create such policies is a "value function" approach, by which only a set of estimates of expected returns for the policy is maintained and the policy is modified by actual returns. This approach contrasts with "direct approach", that suggests sampling returns for each possible policy by following it, and then selecting the one with the largest expected return).

Value function approach has two variations: state value function $V(s)$ and state-action pair value function $Q(s,a)$; the former one estimates the expected return starting from state s and following the policy thereafter, and the latter - expected return when taking action a in state s and following the policy thereafter.

My simple algorithm, denoted further as "RL" is a "state-action pair value function" type, though it is different from classical RL algorithms. First, the algorithm does not have a terminal state, so collision avoidance is considered to be a continuous task of getting as much cumulative non-negative reward as possible. Second, it is concerned only with immediate rewards without considering in what order the states and actions are sequenced. It is made this way because of the fact that policies are

observers and predictors in the experiments none of them is allowed to influence the flow of the experiment.

The algorithm itself is the following:

RL algorithm

Initially $Q(S,A)=0$ for every state-action pair.
$0<\alpha\leq 1$ is the constant learning rate value.

---

1. Get the current state S_i and the intended action A_i.
2. If the current value of the action value function $Q(S_i,A_i)>0$, predict no collision. If $Q(S_i,A_i)=0$ then make a random prediction (or predict nothing). Predict a collision otherwise.
3. After executing A_i get the reward signal r for that action.
4. Update the action value function: $Q(S_i,A_i) \leftarrow \alpha * r + Q(S_i,A_i)$.
5. Go to step 1.

The states in this algorithm are discrete – the sensor space is divided to regions of the same size. The reward signal for an action is defined by checking if the motor command was successfully implemented, i.e. if wheel counters' values are acceptably close to the projected ones:

$$r = \begin{cases} (w_L + w_R + e_L + e_R)/100, \text{if there is no collision} \\ (e_L + e_R)/100, \text{if there is a collision} \end{cases},$$

where w_L and w_R are modules of accordingly left and right wheel commands in the action and e_L and e_R are modules of positioning errors of accordingly left and right wheel movements. Thus, a successful action is rewarded more if it moves the robot for greater distance and an unsuccessful action is penalized depending on the size of the error.

4.2.2 Reversibility-based algorithm (IRR)

Reversibility-based algorithm, denoted further as IRR is also quite simple. Acronym IRR stands for IRReversibility, since it is more correct to say, that it is based on irreversibility – prediction are based on how irreversible the similar experienced actions were.

The algorithm itself is the following:

IRR algorithm

A set of experienced reversibilities is always available. Initially the set is empty, used (hemi)metrics are defined and used thresholds are set.

1. Get the current state S_i and the intended action A_i.
2. Search through experienced reversibilities to find reversibilities where forward action is the same (a_0=A_i) and $d_{orig}(S_i,s_0)\leq\varepsilon_{orig}$, where s_0 is an initial state of an experienced reversibility and a_0 is its forward action. Predict the outcome basing on the reversibilities found (several methods can be used, see further text for details).
3. Wait for the action to finish. Get the current state S_{i+1} after making action A_i and a new intended value A_{i+1}. If A_{i+1} is not a reverse-action for A_i then go to step 1.
4. Execute step 2 with A_{i+1} and S_{i+1} instead of A_i and S_i.
5. Wait for the action to finish. Calculate $d_{rev}(S_i,S_{i+2})$ and add obtained reversibility to the set of experienced ones.
6. Go to step 1.

Steps 2 and 4 are doing the same thing – predicting the outcome of the next action, the only difference is that step 3 does it for a forward action and step 5 – for a reverse-action; this is done by analyzing the reversibilities found. The general way is to compute some value $v_{prediction}$, representing basic joint d_{rev} return value of the reversibilities found. This value can be used to predict the outcome: no collision if $v_{prediction} \leq \varepsilon_{orig}$, a collision otherwise. If there are no reversibilities found, then a random prediction can be made, or no prediction at all.

$v_{prediction}$ value for a set of reversibilities found can be calculated in many ways. For example, a method might select reversibilities from available ones:

- the one(s), having the shortest d_{rev} distance
- the ones, having distance d_{rev} less than some predefined value
- up to some predefined number of them
- the ones, forming the largest identifiable cluster

and return

- some percentile value (for example, 0 – minimal value, 100 – maximal value)
- the mean value
- the median value

of the set of selected values.

There are many permutations and combinations of them and new ways of limiting the set and calculating $v_{prediction}$, but the general algorithm remains the same.

One of the algorithms also considered is a hybrid of IRR an RL algorithm denoted further as REW. It uses RL logic (i.e. reward signal) for its predictions, but IRR logic for getting the prediction value, i.e. it is RL with a continuous state space:

REW algorithm

A set of experienced state-action-reward triplets is always available. Initially the set is empty, used (hemi)metrics are defined and used thresholds are set.

---

1. Get the current state S_i and the intended action A_i.
2. Search through experienced state-action-reward triplets to find the ones where the action is the same and $d_{orig}(S_i, s_0) \leq \varepsilon_{orig}$, where s_0 is an initial state of the triplet. Predict the outcome basing on the triplets found (several methods can be used, similar to the IRR algorithm, but using reward instead of irreversibility).
3. Wait for the action to finish. Add obtained triplet to the set of experienced ones.
4. Go to step 1.

4.3 Comparability of algorithms

As one might have noticed already, IRR, REW and RL algorithms are not very different and can be safely compared to each other and to REW algorithm.

Actually, RL and IRR are quite similar, the main difference is that the former one is based on an artificial reinforcement signal and the latter one is based on the action reversibility. To be more specific, the reversibility-based algorithm can be considered as RL algorithm with reversibility as a reward signal.

It also means that they can be easily compared, since the comparison will not be between different algorithms, but between two types of reinforcement signal – external (artificially generated by the environment, based on how close real wheels' counter values are to the desired ones after executing an action) and internal (generated by algorithm itself, based on action reversibility in situations similar enough without any prior knowledge about sensors or motors). The REW algorithm is considered to indicate where the state distinction is more important than the feedback and vice-versa.

The comparison also implies that the wheel slippage is not an issue in the experiment, because in such case the reinforcement signal for RL an REW will give a false positive reward for physically incomplete actions. I would also like to draw attention to the fact that wheel slippage is not a problem for IRR. Though, wheel jamming for both forward and reverse actions will be registered as almost perfectly reversible action by reversibility-based algorithm, but will be processed absolutely correctly by external reward-based RL and REW algorithms.

4.4 Software architecture

One of the secondary objectives of this thesis is to create a framework to allow a simple comparison of different algorithms for the task and to implement the reversibility-based algorithm so that it would be easy to extend. The architecture of the program is as simple as possible and extendable; it is shown in figure 1.

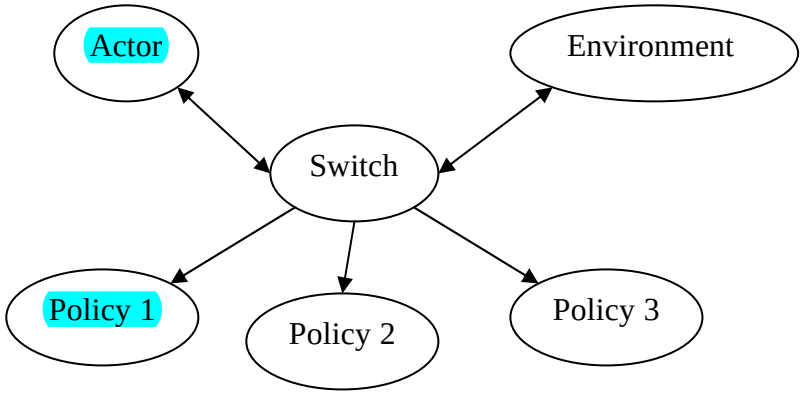


Figure 1: Interconnections between the different modules of the program.

The central building block is a "Switch", through which all other modules communicate with each other. The "Switch" itself doesn't initiate any actions; it works as a hub able to record and multiplex messages from one module to another. The "Actor" decides which action to perform next and can request sensor data from "Switch". The commands of action execution and sensor data retrieval are actually

routed to “Environment” module, where they are processed and communicated from/to the physical robot.

Policy modules are proxies to represent different algorithms. Each of them receives a complete communication between “Actor” and “Environment” modules, which allows them to analyze the data and predict whether the proposed action will succeed and even advise which action to make.

4.5 Physical experimental setup

All our real-world experiments are conducted with a small standard robot “Khepera II” that has 8 infrared sensors and two wheels independently driven by a step-motor. It is connected to the computer through a serial interface and power cables, i.e. sensor readings and motor commands are communicated directly from/to the computer. During the experiments the robot is placed into a standard carton box with all the walls of the same texture and colour. A partition of the same material (and length of shortest box wall) was also used to make the available working area smaller, if needed.

Figure 2 presents two views of the Khepera robot. The physical experimental setup can be seen in figure 3: it is one box in another, the purpose of having the outer box is to protect robot sensors from light interference and thus reduce sensor noise and uncertainty.

In figure 4 photos A, B and C show different configurations for setting up the available working area for local reversibility 1D/2D experiments. Photo D in figure 4 explains box setup for the global reversibility experiment; the robot is at “HOME”, determined by a small construction that permits precise positioning.

In figure 5 photos A and B show the solution to the problem of overheating power elements of power/control adapter for Khepera robot. It was the reason of constant robot reboots with a wrong feedback as a consequence. Though, the problem of robot reboots was not finally solved, and it still rebooted approximately once in 100-2000 steps, depending on how many obstacles it ran into.

A

B

Figure 2: Khepera robot

A

B

Figure 3: Experimental setup

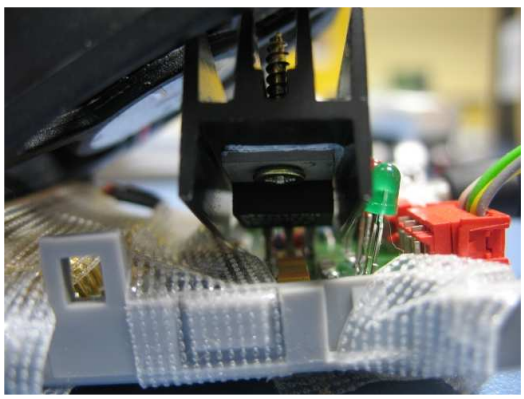
A

B

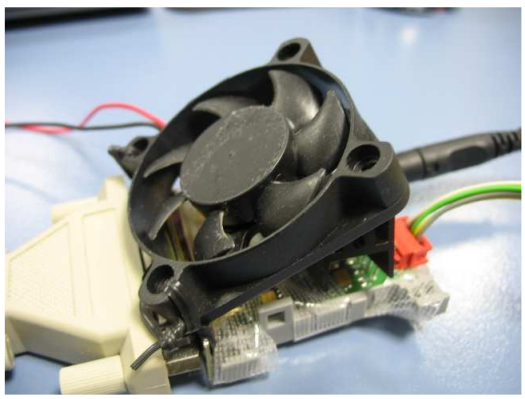
C

D

Figure 4: Box with different available working areas



A



B

Figure 5: Adapter with extra cooling

4.6 Implementation details

The program operates with a list of discrete actions, which are pairs of robot wheel movements in a form of $(left\_wheel\_movement, right\_wheel\_movement)$ They can be easily replaced with sequences of actions, because algorithms don't really care how complex the action is and operate with indices of available actions. The actions used are:

$a_1 = (300,300)$ – *long step forward*,

$a_2 = (100,100)$ – *short step forward*,

$a_3 = (-100,-100)$ – *short step backward*,

$a_4 = (-300,-300)$ – *long step backward*,

$a_5 = (200,200)$ – *medium step forward*,

$a_6 = (-200,200)$ – *rotate counter clockwise*,

$a_7 = (200,-200)$ – *rotate clockwise*, and

$a_8 = (-200,200)$ – *medium step backward*.

The numbers are internal robot wheel counter units of length ~0.08mm. Positive numbers mean wheel rotation that moves the robot forward.

Sensor input data for algorithms is also reduced: the maximum number of logical sensors is 4 to make the learning processes faster. Two parallel front sensors and two parallel rear sensors form two logical sensors and left/right sensors form another two logical sensors, one for each side. Two sensors next to the front sensors on both sides are left unused, since front and side sensors are already used and it would just increase the number of input channels without adding significant value to the experiments.

The experiments can be divided into two separate groups: local 1D/2D experiments – to compare the reversibility-based algorithms with others, and a global reversibility-based experiment without a comparison. The purpose of the latter experiment is to show that the higher level of behaviours can emerge basing on the same principle, though there is no comparison made, since global reversibility experiments are more difficult to benchmark: there is no "standard" and it is hard to invent a straightforward one.

4.6.1 Local reversibility 1D/2D experiments

In 1D experiments only front and back sensors are used and the only actions available are moving forward and backward, i.e. $A_{1D} = \{a_1, a_2, a_3, a_4,\}$. The 2D experiments use all available sensors and another set of actions: $A_{2D} = \{a_5, a_6, a_7, a_8,\}$. It can clearly be seen that $(a_1, a_4), (a_2, a_3), (a_5, a_8), (a_6, a_7)$ are pairs of actions that are expected to reverse each other.

In the beginning of an experiment, the robot is provided with initial worlds W_{1D} and W_{2D} with reversibility models R_{1D} and R_{2D} accordingly (i.e. sets of reversibilities that hold initially). In these initial worlds there is a single state S', thus:

$$R_{1D} = \{(S', a_1, S', a_4, S'), (S', a_4, S', a_1, S'), (S', a_2, S', a_3, S'), (S', a_3, S', a_2, S')\}$$

$$R_{2D} = \{(S', a_5, S', a_8, S'), (S', a_8, S', a_5, S'), (S', a_6, S', a_7, S'), (S', a_7, S', a_6, S')\}$$

In these initial worlds W (hemi) metrics and thresholds do not matter, since there is only one state and distance calculated by any (hemi)metric is 0. This initial world can be imagined as an environment where robot does not have any feedback from the environment – no sensors are used.

In these experiments, refined worlds W_{1D}' and W_{2D}' with refined reversibility models accordingly R_{1D}' and R_{2D}' are constructed. In these worlds the state vector is extended with 4 proximity sensor values:

$$S(d_F, d_B, d_L, d_R),$$

where d_F is a front sensor, d_B – rear sensor, d_L – left sensor and d_R – right sensor. 1D metrics calculations did not involve those side sensors, since they were not important. A refinement state-function *p* is a projection returning the single state of the initial world:

$$p(S(d_F, d_B, d_L, d_R)) = S'.$$

Thus, since $\forall S \in W' : p(S) = S'$, new initial reversibility models are:

$$R_{1D}' = \{\forall S_0, S_1, S_2 \in W_{1D}' : (S_0, a_1, S_1, a_4, S_2), (S_0, a_4, S_1, a_1, S_2), (S_0, a_2, S_1, a_3, S_2), (S_0, a_3, S_1, a_2, S_2)\}$$

$$R_{2D}' = \{\forall S_0, S_1, S_2 \in W_{2D}' : (S_0, a_5, S_1, a_8, S_2), (S_0, a_8, S_1, a_5, S_2), (S_0, a_6, S_1, a_7, S_2), (S_0, a_7, S_1, a_6, S_2)\}$$

During the experiments, failed reversibilities are marked as invalid ones to prevent the robot to make similar actions that can also be irreversible.

In the new world new metrics and constants should also be defined. In all the tests conducted $\varepsilon_{dest} = \infty$, which makes the choice of d_{dest} irrelevant, in other words, intermediate state in reversibility tests is totally discarded. Different d_{orig} and d_{rev} (hemi)metrics, mostly Manhattan and Euclidean distances, with different constants ε_{rev} and ε_{orig} are considered during the experiments, though graphs only for Euclidean distance with constant thresholds are presented in the test results section of this thesis.

The algorithm for the "Actor" module in these experiments is fairly simple:

"Actor" module algorithm

1. Identify current state S_i.
2. Select a random A_i action from available ones.
3. Query algorithms, if the action A_i form state S_i should succeed.
4. Execute the action A_i, compare the answers to the real result.
5. Identify current state (S_{i+1}).
6. Select action A_{i+1}, a reverse-action for A_i. Repeat steps 3 and 4 with S_{i+1} and A_{i+1} instead of S_i and A_i.
7. Identify current state (S_2).
8. Select a random action from available ones and execute it
9. Go to the step 1.

Step 8 is needed for the robot to explore the area around, because without this step and without any obstacles in a one-step distance the robot would move in the same very small area for a very long time without any exploration. The pattern of robot movement is clearly suitable for IRR, since it includes an action and a counter-action in each cycle.

The diagrams in figure 6 are intended to explain how it physically looks like. Dark grey circle with black borders is the robot, which has actually a form of a circle, if viewed from the top, thick black lines are box walls. Dotted lines in the right diagram show possible positions of the additional wall to limit the area (area to the right of that additional wall was accessible to the robot).

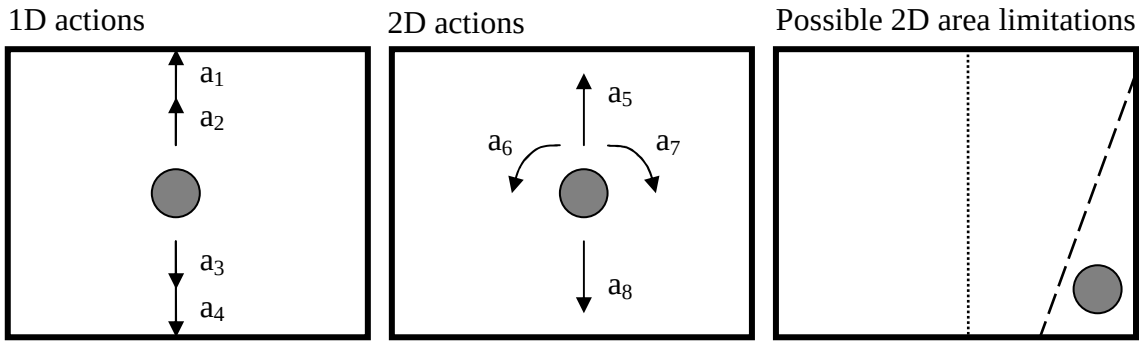


Figure 6: Robot moves in 1D/2D experiments and possible 2D area setups

It was necessary to feed the algorithms with a data set, where there are a lot of failed actions, since the area of the full box was large enough for the robot to make hundreds of steps without touching any walls.

The longest 2D test run made was using the setup with dashed additional border, with a form of a triangle. The problem was that if there were too many successful/unsuccessful actions, then it was hard to tell, if the algorithm worked fine, or it was just too optimistic/pessimistic. This small triangular area allowed varying actions' success rate in a wide range to show that the results are not too correlated to the success rate.

All compared algorithms, except for RND (random algorithm), are implemented by the same class by using different (hemi)metrics implementations and joint (prediction) value calculator implementations. This is done on purpose to minimize possible differences between them and to make the comparison as straight as possible.

4.6.2 Global reversibility experiment

The purpose of this experiment is to study the global reversibility. Ideally, if all the last steps of the sequences, starting from a single action sequence, are reversible, then all those sequences should also be reversible but in practice, imprecise mechanics of a robot or environmental conditions can change that.

For example, if a robot has 5% localization error at each step and the thresholds are high enough to accept such errors for a sequence (action, counter-action) to be marked as reversible, then a ten-action sequence can finish with an error of 100% of the step length, which is quite a big error to consider. The reason for such error is, actually, irrelevant, since it can be mechanical malfunction or some environmental influence, like uneven or incline ground/floor. What is relevant is the ability of a robot to find a safe area for itself, where it can move confidently. Thus, this experiment was

made to back up the suggestion that this safe area can be partly identified by a robot itself, if it travels only along the paths with global reversibility under control, i.e. if the distance between the start and end states of those sequences is less than some predefined constant using some predefined metric.

For the experiment to be of a reasonable duration, all sequences start from the same point called “HOME”. It is set in one of the box’s corners (figure 4 D), where the robot is placed by hand with ~1-2mm precision at the beginning of each sequence run. The same set of actions as in 2D local reversibility experiments is used and states are uniquely identified by a sequence of actions that were made from “HOME” state to that particular state. Sensor data is used only to calculate local reversibility for the last step of a sequence and local reversibility of the whole sequence, but not for the identification of states.

The algorithm tested in this experiment is fairly simple and intuitively deductible from the definition of globally recursive reversibility (see diagram in figure 7).

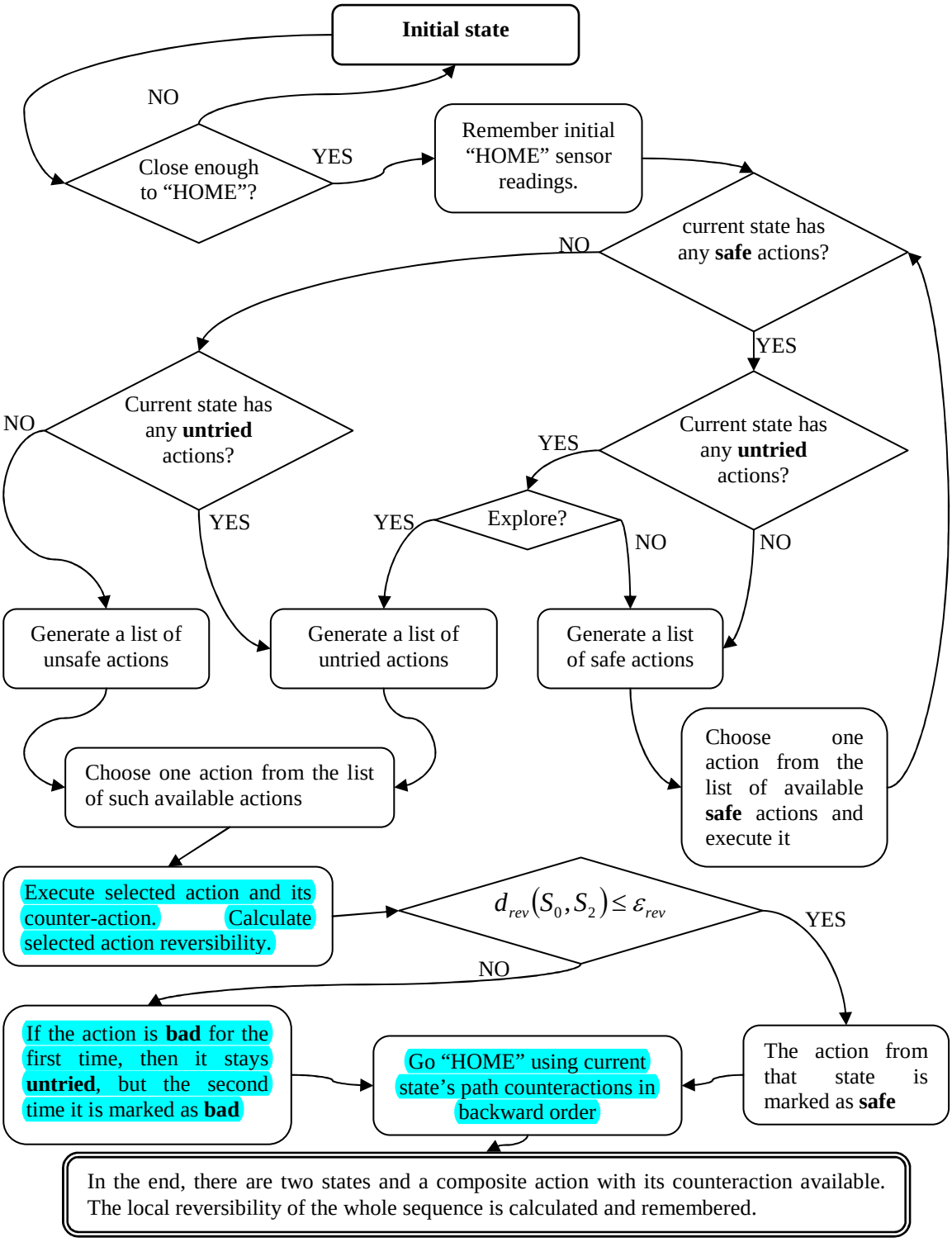


Figure 7: Algorithm of global reversibility experiment

5 TEST RESULTS

5.1 Local reversibility 1D/2D test results

Charts below represent the success rate of prediction for the algorithms during every 100 actions. There are 6 different algorithms in those charts, identified by following acronyms:

- "RL" – simple reinforcement learning algorithm
- "IRR" – reversibility based algorithm
- "REW" – a modification of "IRR", that uses reward signal instead of reversibility for prediction
- "RND" – random prediction algorithm
- "RL2" – a modification of RL, that receives the same feedback as IRR
- "REW2" – a modification of REW, that receives the same feedback as RL

Algorithms RL and REW2 gain experience with every step, but algorithms IRR, REW and RL2 – with every obtained reversibility, i.e. every 2-3 steps, depending on the order of actions.

There were several test runs made, but only two are presented here, one for 1D and one for 2D. Those runs were chosen, because both of them include periods with both wheel jamming and normal performance.

Both local reversibility 1D and 2D experiments used Euclidean distance as (hemi)metric for all distance calculations. $\varepsilon_{dest} = \infty$ also for both 1D and 2D experiments. In 1D experiments $\varepsilon_{rev} = 100$, $\varepsilon_{orig} = 1000$. In 2D experiments $\varepsilon_{rev} = 2200$, $\varepsilon_{orig} = 10000$. For RL algorithms $\alpha = 0.1$.

Figure 8 represents success rate of predictions for IRR and RND algorithms. This is a feasibility check to show that IRR algorithm performs significantly better than RND algorithm. It should also be noted that the algorithm reached approximately 80% success rate quite quickly and continued to perform further this way. The downward peaks at steps ~1300-1600 (values 13-16 on X axis) and ~1700-1800 (values 17 on X axis) show performance of the algorithms during periods, where wheels were jammed.

5.1.1 IRR vs. RND

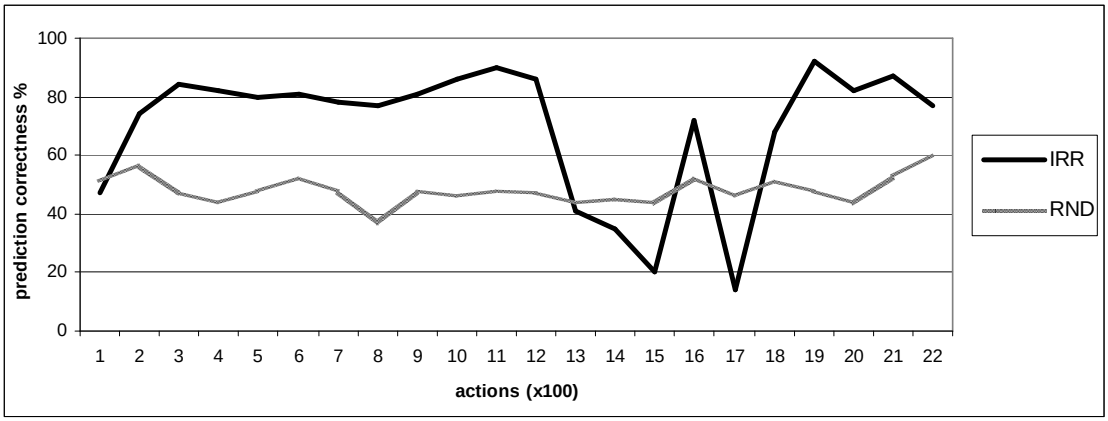


Figure 8: 1D experiment, comparison with random prediction algorithm

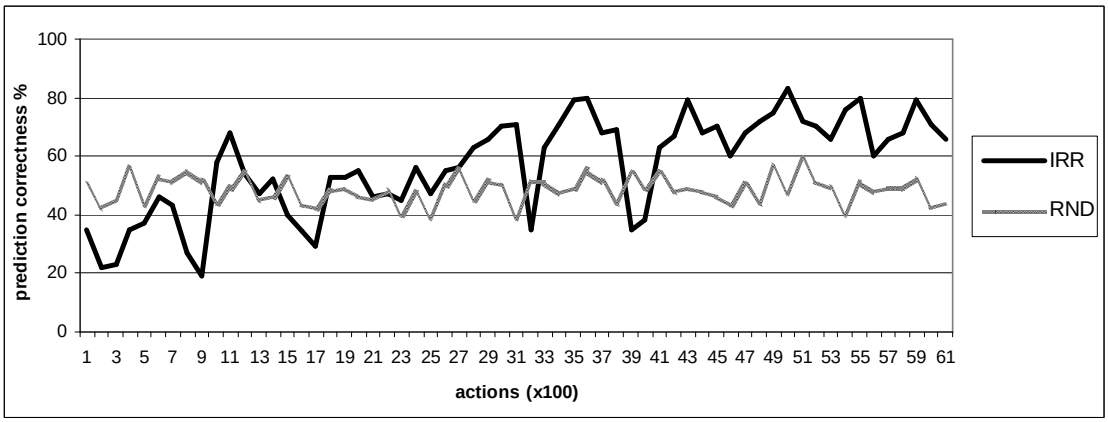


Figure 9: 2D experiment, comparison with random prediction algorithm

The test run for the 2D algorithm (figure 9) also included a period where wheels were jammed at steps ~3800-4000 (values 39-40 on *X* axis). These jamming areas are clearly identified by two downward peaks in the middle of the diagram. The diagram also shows that learning rate increased gradually and reached the steady range of 60-80% by the middle of the diagram, i.e. after ~ 3000 steps.

5.1.2 IRR vs. REW vs. REW2

Figures 10 and 11 show the comparison between irreversibility-based IRR algorithm and its reward-based counterparts – REW and REW2. REW2 is very close to IRR and they differ only in the way the prediction is calculated, i.e. they use exactly the same states for prediction. The first half of the diagram shows how the algorithms work during normal robot operation without wheel jamming. This part shows that IRR algorithm is more successful in its predictions, than REW, but less successful, than REW2. Superiority of REW2 algorithm over IRR and REW is very logical, since REW2 algorithm is fed with 2-3 times more information than IRR, but the superiority

of IRR over REW2 was a surprise, this means that in this test run reversibility values are more relevant than reward signals. REW and REW2 algorithms predict failure while wheels are jammed significantly more successfully, it was not a surprise and the reason for this is already explained in item 4.3.

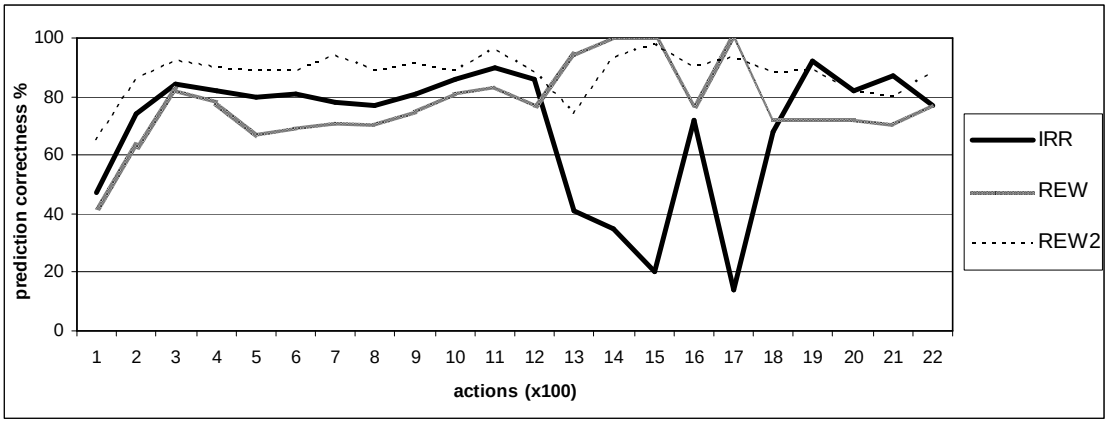


Figure 10: 1D experiment, comparison with hybrid algorithms

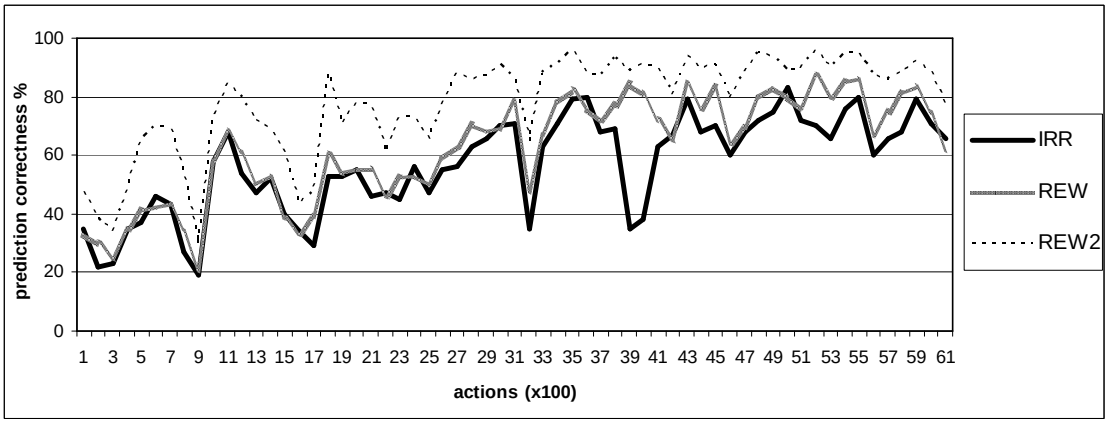


Figure 11: 2D experiment, comparison with hybrid algorithms

There is one downward peak in the 2D diagram (figure 11) at value 32 of *X* axis. Wheels are not actually jammed there, but the situations experienced were new: the IRR algorithm did not predict any outcome at all and that decreased success rate heavily. This can also be confirmed by the behaviour of the graphs – all of the algorithms experienced drops in success rate, which contrasts with the area, where wheels were jammed – REW and REW2 algorithms performed steadily, but IRR experienced a significant drop in the success rate of predictions.

5.1.3 IRR vs. RL vs. RL2

Figures 12 and 13 show the success rate of prediction for IRR, RL and RL2 algorithms. The data of 2D experiments is noisier, which is why the algorithm with less state precision and the same information (RL2) outperforms IRR. This contrasts

with the worse performance of the REW algorithm (which is similar to IRR and RL2) against IRR. The difference between state identification is clearly a reason here, RL and RL2 have discrete states. Their whole space state is divided into 1296 regions, with not more than 1/3 used during the experiment. It is very logical that the RL algorithm outperforms IRR because of 2-3 times more info fed into it.

Figure 12: 2D experiment, comparison with RL algorithms

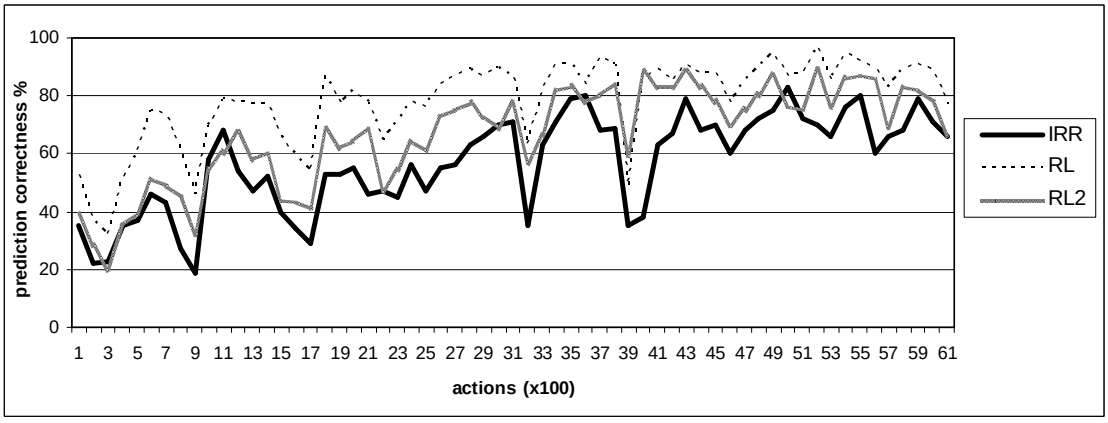


Figure 13: 2D experiment, comparison with RL algorithm and its modification

5.2 Global reversibility test results

This section represents the test results of global reversibility experiments by visualizing robot motion. Every run started at "HOME" – a big black dot in the upper left corner of the picture. All endpoints are marked by small dots, the colour of the dot and the last segment depends on local reversibility of that particular last action of the sequence – red means that the last action was irreversible, blue means the opposite.

At every step the robot makes a pseudo-random move. i.e. pseudo-random decision either to follow an already known (tried) path or an unknown path to explore the neighbourhood. Figure 14 visualises irreversible sequences of pseudo-random robot runs. Borders of the box can be clearly identified by just looking at the figure,

red irreversible ends of sequences show where actions were unsafe – when the robot tried to go through a wall. By marking these actions as undesirable, like it was done in the previous experiments would lead to obstacle avoidance behaviour. However, due to the global reversibility introduce here, more complicated behaviour would emerge. Long blue sequences show that recursively globally reversible sequences of actions are not always locally reversible, and thus unsafe.

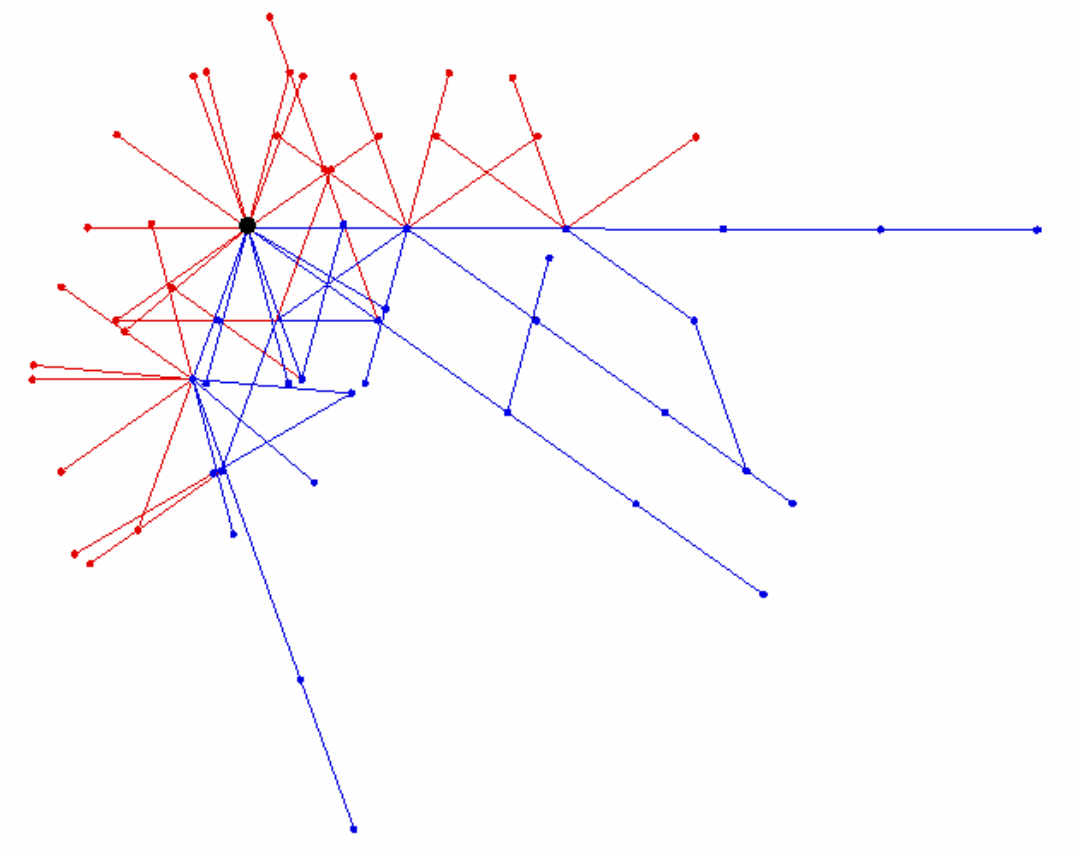


Figure 14: Randomized runs, globally irreversible sequences

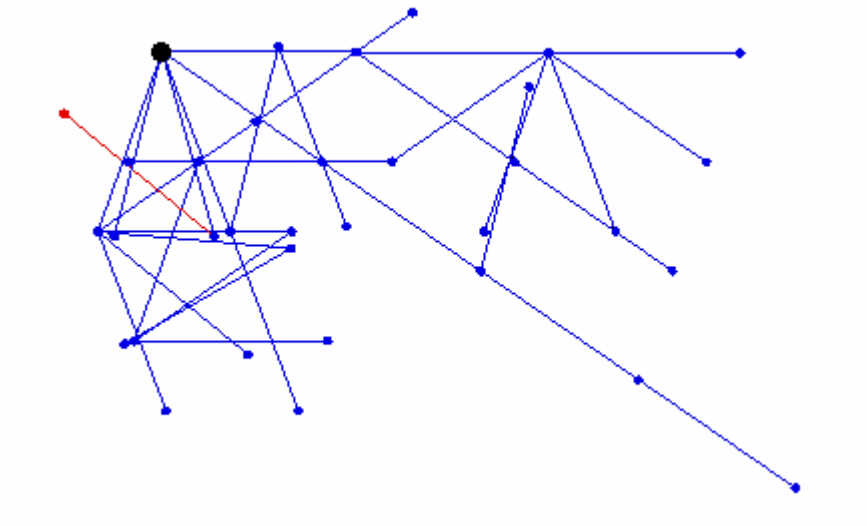


Figure 15: Randomized runs, globally reversible sequences

Figure 15 visualises reversible sequences of the same pseudo-random runs. It contains mostly short sequences not ending in the walls. Though there is one sequence that ended in the wall, but was locally reversible anyway. There can be two explanations: either the last action was successful and that last action is judged as irreversible because of noise, or it is just a coincidence.

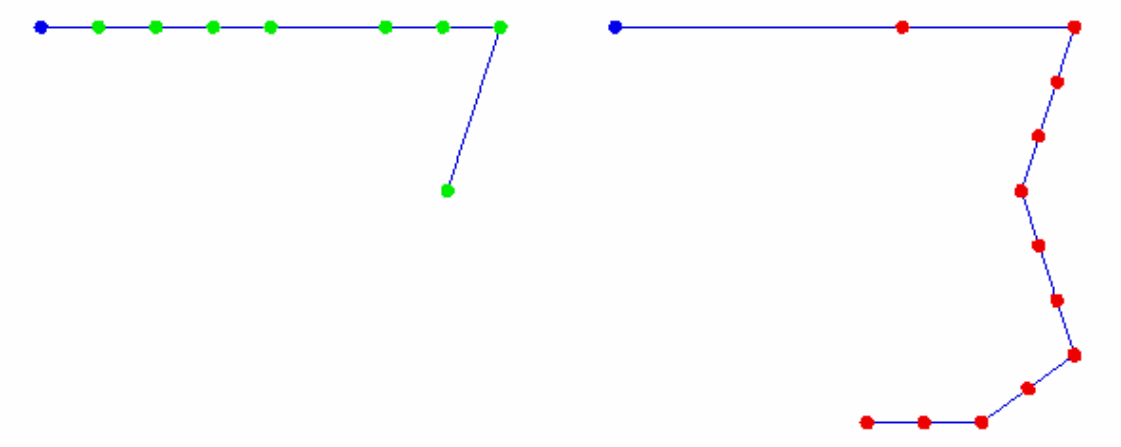


Figure 16: Predefined run #1

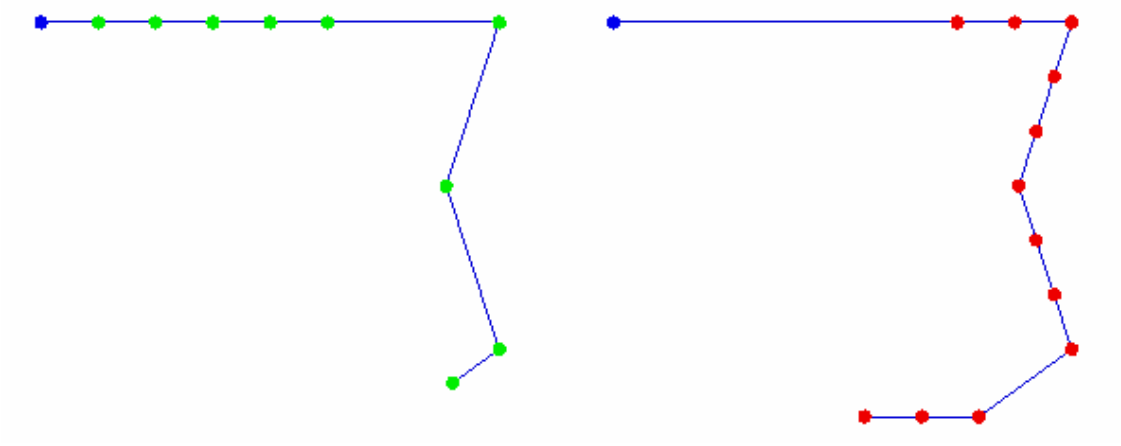


Figure 17: Predefined run #2

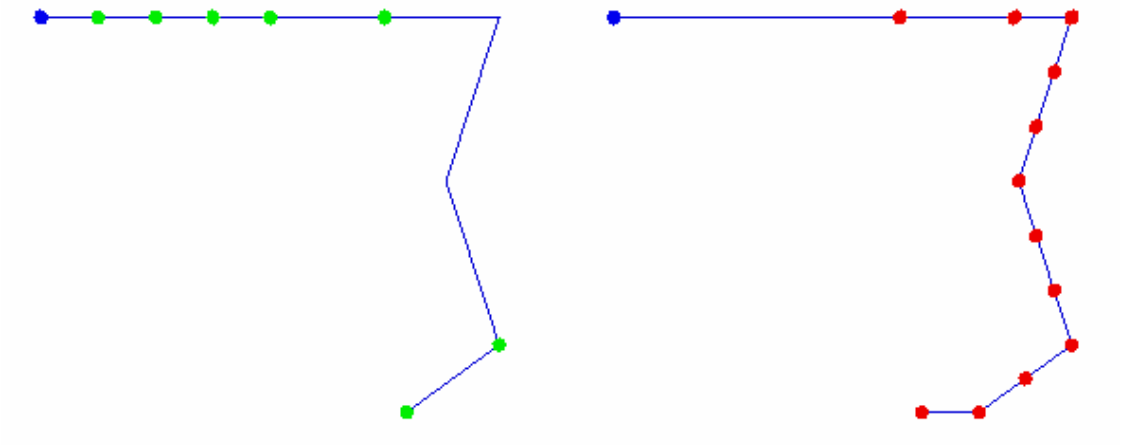


Figure 18: Predefined run #3

Figures 16-18 visualise 3 test runs for the predefined path approximately along the wall of the box without any obstacles in the way, i.e. the whole path is supposed to be safe.

Each figure consists of two subfigures, with locally reversible sequences of actions on the left and irreversible ones on the right. Green and red dots represent endpoints of locally reversible and irreversible sequences of actions respectively. “HOME” is a blue dot in the upper left corner of each subfigure. These three runs show that the robot was able to identify unsafe regions of the sequences along the predefined path – it is clearly visible that longer actions, despite being globally recursively reversible, are not back-path globally reversible and cannot be expected to bring the robot back home.

6 CONCLUSIONS AND FURTHER WORK

This master thesis validates the idea of using reversibility for developing safe behaviours for autonomous robots. It shows that:

1. by applying reversibility models and suppressing irreversible actions the robot develops obstacle avoidance behaviour
2. by comparing the test results of obstacle avoidance with reinforcement leaning algorithms it shows that the robot using reversibility models converges to the obstacles avoidance behaviour as fast as the robot using the reinforcement leaning algorithm especially developed for obstacle avoidance
3. by applying reversibility models to action sequences and suppressing irreversible actions the robot develops safe navigation and obstacle avoidance behaviour
4. more generally, it shows that both elementary and complex behaviours can be developed using reversibility models and suppressing irreversible actions
5. even more generally, it shows that abstract "moral" principles can be used to develop robust behaviours for autonomous robots

During research and source code development some of the issues were not solved and scheduled to to-do list, some ideas for future development were not even tried out. Below is the list of things to do further (in the close or more distant future):

1. To introduce a notion of "strong" holding with d_{move} hemimetric (see item 3.6). Implement it and conduct experiments.
2. To introduce (and incorporate into the theory) a metric (and a threshold) to measure distance between different actions to allow usage of continuous actions. Implement it and conduct experiments.
3. Conduct more simulations with many different metrics and many different algorithms to select reversibilities from a set of available ones to find the best ones.
4. Optimize the source code for faster execution and possible real-time usage.

5. Change the source code to compile on other operating systems (code compiles in Visual Studio 2005, but most of the code is ready or almost ready for Gnu C++ compiler).
6. Conduct research on higher level of behaviours and try to use the concept of reversibility in common real-life tasks and experiments.

Certainly, the list is not complete and many other things can be done in the future.

SUMMARY

This thesis investigates the concept of reversibility as a reason of safe behaviour. It validates the idea that by suppressing irreversible actions the robot will develop safe behaviours. More generally, it investigates the idea if concrete safe behaviours can be developed from abstract principles.

This thesis formalizes the approach by defining reversibility models, developing an algorithm for learning it and defining the notions of global and local reversibilites as well as metrics on the sensor-action spaces.

Further on, it implements the above mentioned algorithms and validates the performance on a real robot. The results show that the robot develops an obstacle avoidance behaviour and territorial behaviour by learning reversibility models and suppressing irreversible actions.

SISUKOKKUVÕTE

See magistritöö käsitleb pööratavuse printsiipi kui turvaliste käitumiste eeldust autonoomsetele robotitele. Töö teema kuulub kognitiivrobootika valdkonda, mis üldiselt tegeleb tehissüsteemide iseõppimise ja arengu küsimustega. See magistritöö testib hüpoteesi, et õppides selgeks pööratavuse mudelid ja vältides pöördumatuid tegevusi, arenevad robotil välja turvalised käitumised.

Selles magistritöös formaliseeritakse probleem, defineeritakse pööratavus, globaalne ja lokaalne pööratavus ning pööratavuse mudel ning esitatakse algoritmid nende mudelite õppimiseks. Seejärel testitakse algoritme reaalsel robotil ning võrreldakse nende tulemust traditsiooniliste algoritmidega.

Tulemused näitavad et jälgides pööratavuse printsiipi kujuneb robotil välja käitumine takistuste vältimiseks, mis oma soorituselt on sarnane traditsiooniliste algoritmidega, mis on spetsiaalselt mõeldud takistuste vältimise õppimiseks. Kui pööratavuse printsiipi rakendada ka tegevuste jadadele kujuneb peale taksituste välimise välja ka territoriaalne käitumine, s.t. keelatakse need tegevused mille tõttu robot ei oska tagasi koju pöörduda.

Katsetulemusi võib vaadelda kui osalist kinnitust et konkreetseid käitumisi saab tuletada üldistest abstraktsetest printsiipidest ja et pööratavuse printsiibi rakendamine aitab tuletada turvalisi käitumisi.

Selle magistritöö tulemused võivad olla kasulikud rakendatuna teenindusrobotitel, kes töötavad inimestega koos või nende läheduses või osaliselt tundmatus ja ebaturvalises keskkonnas.

REFERENCES

[BC1] Epigenesis definition. Available from http://www.biochem.northwestern.edu/holmgren/Glossary/Definitions/Def-E/epigenesis.html (25.02.2007)

[ER1] About Epirob 2006. Available from http://www.csl.sony.fr/epirob2006/aboutEpirob.htm (25.02.2007)

[ER2] F. Kaplan and P.-Y. Oudeyer (2006). Trends in Epigenetic Robotics: Atlas 2006. *In Proceedings of the Sixth International Workshop on Epigenetic Robotics.*

[ER3] F. Kaplan and P.-Y. Oudeyer (2003). Motivatonal principles for visual know-how development. *In Proceedings of the Third International Workshop on Epigenetic Robotics.*

[ER4] A. Eppendahl and M. Kruusmaa (2006). Obstacle Avoidance as a Consequence of Suppressing Irreversible Actions. *In Proceedings of the Sixth International Workshop on Epigenetic Robotics.*

[WI1] State transition system – Wikipedia, the free encyclopedia. Available from http://en.wikipedia.org/wiki/State_transition_system (25.02.2007)

[RL1] Richard R. Sutton and Andrew G. Barto (MIT Press, Cambridge, MA, 1998, A Bradford Book). Reinforcement Learning: An Introduction.

APPENDIX A: Source code and test data explanations

The CD, attached to this thesis contains source code, test data and some additional files, like an electronic copy of this paper in *DOC* and *PDF* formats as well as an electronic copy of the article from appendix B.

Folder structure of the CD with description of the most important files and folders:

File name and place	Description
JuriGavshin.doc	DOC version of this thesis
JuriGavshin.pdf	PDF version of this thesis
\Appendix A	Folder, containing source code and test data files
\Source Code	Folder, containing implementation source code with configuration files
MScDiploma.sln	VisualStudio2005 solution file. This file should be opened to compile and run the code.
\Testing	Folder of the Visual Studio 2005 C++ Project with the source code that transforms test data from the old format to the new one.
\TheApplication	Folder of the Visual Studio 2005 C++ Project with the source code that implements algorithms for experiments and the framework for testing.
\src	Folder, containing source code (.cpp, .h)
\logs	Folder, containing logs and settings
\New Test Data	Folder containing test data in the new format
\Old Test Data	Folder containing test data in the old format, undocumented
\Appendix B	
ICRA07_0181_FI_50.pdf	PDF version of the article from appendix B

Source code explanations

The code is located at **\Appendix A\Source Code\TheApplication\src**.

File **main.cpp** contains 4 functions – *main*(), *run*(), *simulate*() and *explore*(). Function *main*() is the entering point of the application, where one of the other functions is executed. Function *simulate*() simulates 1D/2D local reversibility test runs, function *run*() makes 1D/2D local reversibility test runs and function *explore*() makes or simulates global reversibility test runs.

Class `Communicator`, defined and implemented in files **Communicator.cpp** and **Communicator.h**, is a proxy to communicate with Khepera II robot and implements its most relevant commands available.

Class `OpenGL_Display`, defined and implemented in files **OpenGL_Display.cpp, OpenGL_DisplayCommon.cpp, OpenGL_Display.h and OpenGL_DisplayCommon.h**, is a utility class, allowing to simply display different objects in 3D space using OpenGL API. This class implements the functionality to take care about the entire infrastructure including window creation, lightning and fog setup with mouse and keyboard handling, allowing to browse the 3D space freely. This class can be sub-classed and the subclass should only implement the execution of OpenGL commands to draw what is needed to be visualised.

Class `PathExplorerVisualiser`, defined and implemented in the file **PathExplorer.h**, is a subclass of the `OpenGL_Display` class and visualises global reversibility test runs. The same file also contains class `PathExplorer`, that implements the functionality to make and simulate global reversibility test runs.

File **ShitchBase.h** contains `SwitchBase` class, which implements the most basic “Switch” module functionality – retrieving sensor data from connected environment and sending action commands while registering this communication at connected “experiences” – proxies for algorithms in the local 1D/2D experiments.

File **ShitchBaseStruct.h** contains classes that `SwitchBase` class operates with: experience implementation must sub-class the `EnviromentBase` class, “experience” implementations must subclass `ExperienceBase` class, sensor data class must subclass `SensorDataPackBase<T>` class and the action class must be a subclass of `ActionPackBase<T>` class.

File **ShitchDiscrete.h** contains `SwitchDisrete` class, which implements the functionality to deal with discrete choice of actions, thus, operating with action

indexes. This class also implement additional functionality for automatic logging of inter-module communications, with a possibility to simulate logged history. This file also contains the `KheperaLoggingActor` class, that was used to make test runs for 1D/2D experiments. It produced and still produces a log file in the old format.

File **ShitchDiscreteStruct.h** contains classes that `SwitchDiscrete` class operates with: experience implementation class `ExperienceDiscrete` that operates with discrete actions by their indexes as well as `KheperaEnvironment` class that is a proxy for `Communicator` class mentioned above to communicate with Khepera robot. It also contains `KheperaSimulationEnvironment` class that can be used to simulate environment by using output log in the old format to simulate a test run, though, new log format in conjunction with the simulation feature of `SwitchDiscrete` class is strongly recommended.

File **Irr_Discrete.h** contains two classes: `IrreversibilityPolicy` – the class implementing reversibility ideas, and `Irr_DiscreteExperience` – the proxy class for `IrreversibilityPolicy` to communicate with `SwitchDiscrete` class. Class `IrreversibilityPolicy` does not implement any calculations and does not select reversibilities itself, but uses so called „calculators" to do this (they are implemented in the file **Irr_iscreteStruct.h**). This fact allows to use this same class for all IRR, REW and RL algorithms implementations by using different „calculators" (see function *`simulate`*`()`, where those algorithm implementations are initialized).

Test data explanations

The test data (i.e. log) files are located at **\Appendix A\New Test Data** and **\Appendix A\New Test Data**.

The first folder contains test data in the new format: global reversibility experiments data in the **Global** subfolder and transformed (from the old format) data for 1D/2D local reversibility experiments in the **Local** subfolder.

The second folder contains test data in the old format for 1D/2D local reversibility experiments.

APPENDIX B: “Don’t Do Things You Can’t Undo: Reversibility Models for Generating Safe Behaviours” article for ICRA’2007 by Maarja Kruusmaa, Juri Gavšin and Adam Eppendahl.

Next pages contain the article that is an appendix to this thesis.