ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Semester project
Autumn 2008

Path Planning with the humanoid robot iCub

Author:
Panteleimon Zotos

Professor:
Auke Jan Ijspeert

Supervisor:
Sarah Degallier

BIOLOGICALLY INSPIRED
ROBOTICS GROUP (BIRG)

January 10, 2009

Contents

Chapter 1

Path Planning Algorithms

1.1 Introduction

A path planning algorithm's goal is to find an efficient, collision-free path for a robot from a start position to a target position in an environment with objects, whenever one exists, or to determine that no solution exists if so. The path planning problem is known to be PSPACE hard [1], which means that its complexity grows exponentially with the dimension of the configuration space.

The configuration space C is the space of all possible configurations, the complete specifications of the position of every point [2]. The configuration space usually has n dimensions if the robot has n degrees of freedom [3], but in our case the configuration is defined by the position and the orientation of the main body of the robot and by the angle that the body is steered. The configuration-free space (C_{free} or Q_{free}) is defined as the set of collision-free configurations and the obstacle space (C_{obs} or Q_{obs}) as the set of configurations, for which the robot collides with an obstacle [3].

We have chosen to use the Ant Colony Optimization method to solve the path planning problem. Ant Colony Optimization (ACO), which was introduced by Marco Dorigo [23] in his PhD. thesis, is a class of algorithms, whose main idea is inspired by the behavior of real ants. ACO was initially applied to the travelling salesman problem, and to the quadratic assignment problem. Since then, a lot of extended versions have been developed, which have been applied to different combination optimization problems [24]. In section 1.4 ACO will be analyzed in detail. Some other algorithms, which have been used for solving the path planning problem , will be also presented.

Here is an attempt of classification of the path planning algorithms based on S.LaValle [3].

- Grid-based algorithms: A grid resolution is required for these algorithms, the configuration space is decompressed into squares forming a grid and every grid point represents a configuration. The robot moves from a point to an adjacent point, if the two squares are in the C_{free} space. Search algorithms (e.g. ACO) are used to find the desired path.

- Potential-fields algorithms. According to this approach, originally proposed by Khatib[4], the robot is treated as a point robot in the configuration space under the influence of an artificial potential field U. Path planning needs little computation, but it is not always efficient and these algorithms are easily trapped in local minima. Potential-fields algorithms performance and hard cases are more analyzed in the section 1.2.

- Sampling-based algorithms: Sampling-based algorithms avoid the explicit construction of C_{obs} space, they perform a probing of C-space and they represent it with a sampling schema. In order this probing to be constructed, a collision detection feature is necessary. Such sampling-based algorithms are the Probabilistic Roadmap Methods, proposed by Kavraki [8], the Rapidly Exploring Random Tree, proposed by LaValle [20], and the Ariadne's clew algorithm, proposed by Mazer [22].

- Bio-inspired algorithms: Apart from the ACO algorithm, to which we referred at the beginning of this section, genetic algorithms have been also used to solve the path planning problem[31, 32, 33]. Genetic Algorithms (GAs) were invented by John Holland in 1975 [30] and they are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology[1] such as inheritance, mutation, selection, and crossover. A genetic algorithm is started with a set of solutions, called population. Solutions from one population are selected with a probability according to a fitness function and used to form a new population. This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied. More details about the genetic algorithms will be given in section 1.5.

[1]Evolutionary biology is a sub-field of biology concerned with the origin of species from a common descent.

1.2 Potential-fields algorithms

Description of the algorithm

As mentioned in the previous section an artificial potential field U is used in order the robot to be attracted to the final configuration q_f. In fact, the potential field U is consisted of one attractive component that attracts the robot to q_f and a repulsive component that repels the robot from C_{obs}. Thus, the artificial potential field can be expressed by the following formulation:

$$U(q) = U_{att}(q) + U_{rep}(q),$$

where $U_{att}(q)$ is the attractive component of the potential field and $U_{rep}(q)$ is the repulsive component.

The objective of the algorithm now is to find the global minimum in U(q) starting from the initial configuration q_s. The artificial force, F, defined as the negative gradient of U(q), is computed at the current configuration.

$$\overrightarrow{F}(q) = -\nabla U(q)$$

A small step to the direction of this force is made and the whole process is repeated until the goal configuration is reached or the new configuration is sufficiently near to the final configuration. The step size should be small enough, so that to avoid collisions with obstacles and overshooting of the goal.A schema for this algorithm follows:

1. $q(0) = q_1$; i = 0;
2. while $\nabla U(q(i)) \neq 0$ do
3. $\quad q(i+1) = q(i) + \nabla U(q(i))$
4. $\quad$ i = i+1

It is very difficult to generate a potential field directly on the configuration space. Some simple attractive fields according to Spong [2] are the following.

$$U_{att,i}(q) = ||o_i(q) - o_i(q_f)|| \text{ conic well potential}$$

$$U_{att,i}(q) = \frac{1}{2}_i||o_i(q) - o_i(q_f)||^2 \text{ parabolic well potential}$$

Repulsive potential field's objective is to prevent the robot from colliding with the obstacles, but its influence to the robot's motion should minimize as the distance between the robot and an obstacle grows. One suitable potential function for the repulsive field ([2]) is the following:

$$U_{rep,i}(q) = \begin{cases} \frac{1}{2}\eta_i \left(\frac{1}{\rho(o_i(q))} - \frac{1}{\rho_0}\right)^2 & ;\rho\left(o_i\left(q\right)\right) \leq \rho_0 \\ 0 & ;\rho\left(o_i\left(q\right)\right) > \rho_0 \end{cases}$$

Hard cases

Potential fields algorithms's biggest problem, according to Koren and Borenstein [5] , is that they may get trapped in local minima. Randomized methods [3] have been developed to cope with this problem. One simple such method combines the gradient descent with randomization. A heuristic is used to determine when the robot is stuck in a local minimum and then a random walk is used in order to escape. Other search methods apart from the gradient descent that can deal with local minima are Best-First Search and Numerical Navigation Functions[6].

In Best-First Search, a grid is placed over C space and a heuristic is used to search the grid. A tree T of configurations in grid configuration space (GC) is constructed iteratively, in which q_s is the root. At each iteration, the neighbors of leaf with the lowest potential value are examined and the most "promising" (lowest potential) unvisited neighbors are added as children of the leaf to the tree. The iteration stops when q_f is reached or there no more unexplored nodes. A global navigation function is a potential function that has one local minimum at q_f, but unfortunately according to Koditschek [7] it is not possible to design a function like that. A numerical navigation function, which is defined at point on a grid is an alternate design technique [6].

There are also some other problems that PFMs face [5]. A mobile robot may not manage to pass through a narrow passage between two obstacles. Another characteristic of PFMs is that they tend to cause oscillations in the presence of obstacles and in narrow passages. One last problem of PFMs is the GRON problem [10], Goals Non-Reachable with Obstacles Nearby, in which a robot is not able to reach its goal because an obstacle is nearby to the goal and the repulsive field will repel it. New repulsive potential functions [10] have been proposed to cope with the GRON problem.

1.3 Sampling Based Algorithms

1.3.1 Probabilistic roadmap methods

Description of the algorithm

Path planning with the use of roadmap methods consists of three parts, construction of the roadmap, connection of the initial and the goal configurations of the robot to the roadmap and find of a path from the initial to the goal configuration.The main idea of sampling based algorithms is to avoid constructing explicitly the C_{obs} space by sampling C_{space}. In the Probabilistic Roadmap Methods (PRM), random samples from the configuration space of the robot are collected, checked if they exist in the C_{free}, and these configurations that exist in C_{free} are tried to be connected to other nearby configurations.A roadmap (graph) is then constructed that connects two configurations P and Q if the line segment PQ is completely in C_{free}. To explore a path from the start configuration to the end configuration, the existence of the relative path in the roadmap is checked. If this exists, then the planner succeeds and returns the path. Otherwise, there is no way to determine if the path does not exist or if the planner did not take enough samples. The probabilistic roadmap methods (PRM) have been applied to a variety of problems in path planning [11].

The construction of the roadmap is performed by finding candidate neighbors of the already found random configurations and connecting them to the roadmap by simple paths if these paths lie completely in the C_{free}. Candidate neighbors are found by using a distance function, such as the 2-norm, in the configuration space. This process continues until the time is up or the roadmap is good enough. The construction of the roadmap is performed by the following algorithm:
Initially the roadmap (graph) R(N,E) is empty, where N are the nodes and E the edges.

1. N←0, E←0
2. **loop**
3. c← a randomly chosen free configuration
4. $N_c \leftarrow$ a set of candidate neighbors of c chosen from N
5. $N \leftarrow N \cup \{c\}$

6. **for all** $n \in N_c$, in order of increasing D(c,n) **do**

7. **if** $\neg$ *same_connected_component*(c, n) $\wedge \Delta(c, n)$ **then**

8. $E \leftarrow E \cup \{(c, n)\}$

9. update R's connected components

where Δ is a function, which returns whether the local planner can compute a path between the two configurations given as arguments.

In case that C_{free} consists of several big regions connected by narrow passages, the produced roadmap will consist of multiple components, that may be not connected each other or may be sparsely connected. There is an enhancement step, which goal is to connect as many of these components as possible. A simple way to achieve this is by choosing a more sophisticated planner for the construction of the PRM. There are some other approaches also, like identifying the largest connected component and trying to connect smaller components to it. A path smoothing process may also be added after the construction of PRM. [2][8]

Performance

Basic PRM works well if C-space is not cluttered and long, narrow passages are hard to be represented at the roadmap. Heuristics usage help for 'enhancing' roadmap in cluttered areas but to a limited extend.[13] PRM has proven very successful in practice, but it is admittedly very hard to characterize its average performance. The dependence of the algorithm on the number of nodes of the probabilistic roadmap is exponential.[12]

PRM Variants

There are a lot of PRM variants proposed:

- Biased-Sampling Methods (OBPRM[14], MAPRM[15], Bridge Test PRM[16])
- Methods for Highly Constrained Problems
- Lazy Evaluation Methods (Lazy PRM[17], Fuzzy PRM[18], C-PRM[19])
- Cooperative User/Planner Systems

1.3.2 Rapidly - Exploring Random Trees

Rapidly-exploring Random Trees (RRT) method is ideal for path planning problems that involve obstacles and differential constraints (nonholonomic[2] or kinodynamic[3]). As it was mentioned in section 1.1 a rapidly - exploring random tree (RRT) is a data structure designed for path planning in high dimensional spaces[20]. The main idea of this method is to construct incrementally a tree of samples of the configurations space by adding the closest sample to the existing nodes of the tree. The primary advantage of this method is that RRTs can be directly applied to nonholonomic and kinodynamic planning. As roadmaps, they are designed with as few heuristics and arbitrary parameters as possible. What makes RRTs ideal for nonholonomic and kinodynamic problems is the fact that they don't need any connections to be made between pairs of configurations, since one structure is constructed incrementally and at each iteration a new node is added to the tree. RRTs might also be more efficient than standard probabilistic roadmap methods for holonomic path planning [20].

Description of the algorithm

An RRT, T, with K vertices can be constructed by the following algorithm:

1. T.init(x_{init});
2. for k=1 to K do
3. $x_{rand} \leftarrow RANDOM\_STATE()$;
4. $x_{near} \leftarrow NEAREST\_NEIGHBOR(x_{rand}, T)$;
5. u$\leftarrow SELECT_INPUT(x_{rand}, x_{near})$;
6. $x_{new} \leftarrow NEW\_STATE(x_{near}, u, \Delta t)$;
7. $T.add\_vertex(x_{new})$;
8. $T.add\_edge(x_{near}, n_{new}, u)$;
9. Return T;

[2]A nonholonomic system is a system in which a return to the original internal configuration does not guarantee return to the original system position.

[3]Donald et al. [9] defined kinodynamic planning as a class of problems for which velocity and acceleration bounds must be satisfied

In the previous algorithm, X is a state space and is used to express in greater generality what in path planning problems is considered as configuration space. For standard problems, X = C, where C is the configuration space of a rigid body or system of bodies in a 2D or 3D environment. In kinodynamic problems, X = T(C), where T is a state that combines both configuration and velocity [9].

The construction begins from an initial state, x_{init}, and in each iteration a random state, x_{rand}, is selected. The closest neighbor to x_{rand} is found and a u , that minimazes the distance from x_{near} to x_{rand}, is selected. Collision detection is performed to ensure that x_{near} stays in X_{free} and then a new state, x_{near}, is constructed by applying u and an edge is added to the tree.

RRT's properties

RRTs have some very interesting properties, that can turn into strong advantages for particular problems.

- RRTs expand mainly toward unexplored portions of the state space
- RRTs' vertices distribution approaches the sampling distribution
- RRTs are probabilistically complete
- the RRT algorithm is simple
- an RRT always remains connected
- RRTs can be a solution for path planning problems without configuring of the system between two prescribed states being required.

[20]

1.3.3 Ariadne's Clew Algorithm

In Ariadne's Clew algorithm, a search tree is constructed in order to explore as much new territory as possible in each iteration. It is the combination of two routines, SEARCH and EXPLORE. The SEARCH routine's goal is to find a configuration that minimizes the distance of the current position from the final position. The EXPLORE routine is trying to create an approximation of the C_{free} space by setting landmark points, to which the path has been already explored. This approach, spends some of the time exploring the space, as opposed to focusing on finding the solution. This algorithm,

which was introduced by Mazer [22], is designed for path planning in high-dimensional continuous spaces and for robots with many degrees of freedom in static or dynamic environments.

Description of the algorithm

The main idea of the algorithm is that while searching for a path from initial state to target state the algorithm collects information about the free space and about the set of possible paths in the free space.

SEARCH algorithm assumes that the trajectory of the robot with k DOF of length l is parameterized as a sequence of n=k*l successive movements. The EXPLORE algorithm works as follows. Its objective is to compute an approximation of the space accessible from the initial state. It places landmarks in the searched space, so that a path from the initial state to the landmark is known. The landmarks are placed as far as possible from one to another by maximizing the distances between them. The searched space is the set of all paths starting from one of the previously placed landmarks.

The Ariadne's algorithm combines the SEARCH and the EXPLORE algorithms:

1. Use the SEARCH algorithm to find if a path exists from initial to goal state.

2. If a path from step 1 does not exist, continue until a path is found.

    - Use EXPLORE to generate a new landmark point.
    - Use SEARCH to look for a path to the goal from the landmark point.

Performance

According to LaValle [3] one disadvantage of Ariadne's Clew algorithm is that it is very difficult to find an optional solution if a new vertex is placed in the explore mode. Experiments by Mazer [22] showed that the algorithm performs well in realistic dynamic environments.

1.4 Bio-inspired Path Planning Algorithms

1.4.1 Ant Colony Optimization Algorithms

Recently, many researchers [25, 26, 27, 28] have implemented bio-inspired methods and particularly the Ant Colony Optimization method to solve the path planning problem. Ant Colony Optimization (ACO)[23, 24] algorithms are part of swarm intelligence, which is a relatively new approach to problem solving based on the collective behavior of decentralized, self-organized systems and takes inspiration from the social behaviors of insects and of other animals. Ant Colony Optimization takes inspiration from the behavior or real ant colonies and is used to find approximate solutions to difficult optimization problems. ACO methods were initially applied to the travelling salesman problem (TSP), but they are adaptable to any environment and taking advantage of these several bio-inspired algorithms can be used to optimize the path planning problem [25, 26, 27, 28].

ACO's main idea and algorithm

ACO is a metaheuristic[4], in which a set of intelligent agents (artificial ants) search for good solution to an optimization problem, that is the problem of finding the optimal path on a weighted graph in the case of path planning. The biological inspiration for ACO comes from the fact that many ant species deposit on the ground a substance, called pheromone, when they are walking to and from food source. Other ants tend to follow the path, where the concentration of pheromone is higher, the most effective path. While several ants are able to select among different routes from their nest to the food source, the first ants that will return from the food source will be these that followed the shortest path, so that path will have the highest concentration in pheromone and the ants will choose this path at their next traverses to the food source. Considering the ant colony optimization for the traveling salesman problem, the artificial ants can move on a graph, whose vertices represent the cities and the edges the connection between cities, and read or modify a variable (pheromone) for each edge. At each iteration of the ACO algorithm , each ant builds a solution by walking on the edges of the graph and visiting each vertex only once (travelling salesman problem's constraints). The next vertex for each ant is selected stochastically depending on the current vertex. When an iteration has finished, the pheromones values

[4] a set of algorithmic concepts for solving a very general class of computational problems

are updated, so that the best solutions are promoted and the future solutions are similar to these.

In order to describe in more detail the ACO algorithms we should refer to an adequate model $P = (S, \Omega, f)$ where S is a search space of a finite set of discrete decision variables X_i, Ω is a set of constraints and f is a function f:S$\rightarrow$ $\Re_0^+$, which has to be minimized. A decision variable X_i, a solution component c_{ij} and a pheromone trail value τ_{ij} for each c_{ij} are also defined. The set of the solution components, C, can either be associated with the vertices V or with the edges E of a graph. The ants deposit an amount $\Delta\tau$ of pheromone on the components depending on the quality of the solution found. ACO metaheuristic follows :

Set parameters, initialize pheromone trails
while termination condition not met **do**
ConstructAntSolutions
ApplyLocalSearch(optional)
UpdatePheromones
end while

Main ACO Algorithms

A variety of ACO algorithms have been introduced. A table, taken by a tutorial by Dorigo[24], with the most important follows:

SUCCESSFUL ANT COLONY OPTIMIZATION ALGORITHMS		
Algorithm	Authors	Year
Ant System(AS)	Dorigo et al.	1991
Elitist AS	Dorigo et. al.	1992
Ant-Q	Gambarella & Dorigo	1995
Ant Colony System	Dorigo & Gambarella	1996
MAX-MIN AS	Stützle & Hoos	1996
Rank-based AS	Bullnheimer et. al.	1997
ANTS	Maniezzo	1999
BWAS	Cordon et al.	2000
Hyper-cube AS	Blum et al.	2001

Model of ACO for robot path planning

A model for robot path planning based on ACO can be introduced. We assume that our working space is a 2-D grid of points, that can be free or occupied by an obstacle. We define the following set of parameters:

- m:number of ants
- taboo_k:cities that the k^{th} ant visited at the current route.
- n_{ij}:visibility of grid(i,j)
- τ_{ij}:pheromone intensity of grid(i,j)
- $\Delta\tau_{ij}$:pheromone increment of k^{th} ant at grid(i,j)
- f_k:objective function
- P_{ij}^k:transform probability of the k^{th} ant

The amount of pheromone is updated with the following formula: $\tau_{ij}^{new} = (1 - \varrho) \cdot \tau_{ij}^{old} + \Sigma\Delta\tau_{ij}^k$ where ϱ is called evaporation factor, expresses the evaporation of pheromone and takes values between [0,1].

1. loopcounter = 0, initialize τ_{ij},$\Delta\tau_{ij}$, m ants at start point.
2. add start node at taboo_k, for each ant k move to next node j according to P_{ij}^k. Put j in taboo_k.
3. compute f_k, record best answer.
4. update τ_{ij}
5. for each grid(i,j) $\Delta\tau_{ij} = 0$
6. loopcounter++. If loopcounter¡max goto 2

An elitist strategy can be used, in which an additional amount of pheromone is given to the nodes(grid points) of the best tour.[26]

Performance of ACO algorithms

It is obvious that if we deploy an elitist strategy to the ACO, pheromone will mostly be distributed on the best path of the map while the other regions on the map receive little or nothing. ACO will find a solution in any case (any number and distribution of obstacles), provided that a solution exists, as it takes a decision in every grid point. Gutjahr, in 2000, [29] has given a theoretical proof of the convergence of the ACO algorithm to the optimal solution with probability arbitrarily close to 1. The convergence probability can be increased by either increasing the number of ants or by decreasing the evaporation factor. However, trying to achieve better and better convergence to the optimal solution will lead to extreme increase of the computation time. Generally, the performance of the algorithm is better with a larger set of ants, but it stabilizes as the amount of ants gets too large.[34]

1.5 Genetic Algorithms

Description of the algorithm

Genetic algorithms(GA) [30] are used for optimization and search problems. GA are stochastic search techniques analogous to natural evolution based on the principle of survival the fittest. GA's model consist of a population of potential solutions of the problem, called chromosomes. In each generation (iteration), every individual of the population is evaluated by a fitness function. The fitter ones are stochastically selected to be reproduced by means of genetic transformations such as crossover or mutation. This process is repeated until a maximum number of generations or a satisfactory solution has been reached. The population converges to high quality solutions and the fittest individual is more likely to be the optimal solution.Traditionally, solutions are represented in binary as strings of 0s and 1s.

Pseudo-code algorithm

1. Choose initial population
2. Evaluate the fitness of each individual in the population
3. Repeat until termination
    (a) Select best-ranking individuals to reproduce

(b) Breed new generation through crossover and/or mutation and give birth to offspring

(c) Evaluate the individual fitnesses of the offspring

(d) Replace worst ranked part of population with offspring

The classical GAs, that use binary string and two basic genetic operators (crossover, mutation) are like "blind" and perform well with little prior knowledge. In path planning, knowledge is renewed and requires incorporation and GAs do not have to perform "blind search", so graphs are used to represent the environment. The solutions can be encoded in a more effective way that the binary strings, such as variable-length chromosomes. More specialized genetic operators may also be used in order to make the path planning more efficient. GAs are effective in both static and dynamic environments. [32]

Model of GA for robot path planning

At this point we can give a rough model of GA for robot path planning, proposed by Wang [33] and Han [34].
Chromosome: a possible solution in our problem is given by a chromosome. That means, that a chromosome contains the set of the grid points of the trajectory from the start point to the end point.
Via points: via points are the genes of a chromosome or in other words the intermidiate grid points from the start point to the end point.
Fitness: this is the value of the evaluation function for each chromosome. In path planning the fitness function may be the distance between the start point and the end point.
Selection: survival of the best fit chromosome in this generation(iteration). This is called also elitism.
Crossover: a genetic operator that express the exchange of bits(genes) between parents.
Mutation: a genetic operator that express the random change of one or more bits(genes) in the chromosome.
Obstacle Avoidance: a line that is created by two via points(grid points) is checked if it is passing through an obstacle. The chromosomes that pass through obstacles are eliminated.

Performance of GA

It is true that as the number and the distribution of the obstacles increase, more generations(iterations) as well as more bits(genes) in each chromosome are needed in order the best solution to be found. Another factor that influence GA's performance is the number of the via points. The more the via points of each chromosome the more optimal(shortest path) solution will be found but also the more generations the algorithm needs to converge. [34]

1.6 Comparison in performance

One of the problems of the Potential-fields algorithms, that would affect in a major degree our implementation is the GRON problem [10], as mentioned in the section 1.2. A robot is not able to reach its goal because of the existence of an obstacle nearby to the goal. In this case, the repulsive field will prevent the robot to approach the goal position. As mentioned in the section 1.3.1 the probabilistic roadmap methods face difficulties when there are cluttered, long, narrow passages in the C-space, because these are hard to be represented at the roadmap. A robot may need to pass through narrow passages in order to reach its goal, so PRM will not be a proper solution to our problem. According to LaValle [20] the Ariadnes Clew algorithm will find rarely an optional solution in dynamic environments and it would be a great disadvantage of our implementation if we chose this algorithm.

There are some references in the literature [34][35], in which comparison of the performance between the ACO algorithm and the genetic algorithms in the robot path planning problem was accomplished. According to Han [34], GAs are usually better in performance in terms of the distance of the path that they will find in comparison with ACO algorithms. However, when the number of the obstacles is increasing, the GAs may face difficulties to find a solution or even they may not find one. That depends mostly on the number of via points that they will use (the more via points are used the more time consuming is the algorithm). In the case of ACO algorithms, the algorithms will find a solution in any case, even if there are many obstacles or if there are narrow passages in the world. Although the GAs seem to find a shortest path than ACO algorithms, GAs may need more computation time[34]. There are some methods proposed at the literature, that combine ACO algorithms and GAs. For example, a genetic algorithm can be used to optimize the search of the shortest path in an ACO algorithm.[27]

Considering the above points and our desire to use a bio-insired algorithm, we have chosen to implement the Ant Colony Optimization algorithm.

Bibliography

[1] Reif, J.H., *Complexity of the movers problem and generalizations*, Proceedings of the IEEE Transactions on Robotics and Automation, 421-427, 1979

[2] Mark W. Spong, Seth Hutchinson and M. Vidyasar, *Robot Modelling And Control*, John Wiley & Sons (2006).

[3] S. M. LaValle, *Planning Algorithms*, Cambridge University Press, 2006

[4] Oussama Khatib, *Real-Time Obstacle Avoidance for Manipulators and Mobile Robots*, The International Journal of Robotics Research, Vol. 5, No. 1, 90-98 (1986)

[5] Koren, Y. Borenstein, J. , *Potential field methods and their inherent limitations for mobilerobot navigation*, , 1991 IEEE International Conference on Robotics and Automation, 1991.

[6] Barraquand J.,Langlois B.,Latombe, J.-C *Numerical potential field techniques for robot path planning*, IEEE Transactions on Systems, Man and Cybernetics, 1992

[7] Koditschek D., *Exact robot navigation by means of potential functions: Some topological considerations*, IEEE International Conference on Robotics and Automation. Proceedings. 1987

[8] Lydia E. Kavraki, Petr Svetska, Jean-Claude Latombe and Mark H. Overmars, *Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces*, IEEE Transactions on Robotics and Automation, VOL.12, NO.4, AUGUST 1996

[9] Donald, B.; Xavier, P.; Canny, J.; Reif, J. , *Kinodynamic motion planning*, Journal of the ACM (JACM) 40 (5): 10481066, (1993)

[10] S. S. Ge and Y. J. Cui, *Path Planning for Mobile Robots Using New Potential Functions*, Proceedings of the 3rd Asian Control Conference, 4-7-2000

[11] Roland Geraerts, Mark H. Overmars *A Comparative Study of Probabilistic Roadmap Planners*, Workshop on the Algorithmic Foundations of Robotics (WAFR'02), pp. 43-57, 2002.

[12] Lydia E. Kavraki, Mihail N. Kolountzakis, and Jean-Claude Latombe, *Analysis of Probabilistic Roadmaps for Path Planning*, IEEE Transactions on Robotics and Automation, VOL. 14, NO. 1, FEBRUARY 1998

[13] Nancy Amato *Randomized Motion Planning: Techniques and Applications* , notes on the course Randomized Motion Planning: Techniques and Applications, University of Padova (Fall 2004)

[14] N. Amato, O. B. Bayazit, L. Dale, C. Jones, and D. Vallejo. *OBPRM: An obstacle-based PRM for 3D workspaces.*, In Robotics: The Algorithmic Perspective. AK Peters, 1998.

[15] Steven A. Wilmarth, Nancy M. Amato, Peter F. Stiller, *MAPRM: A Probabilistic Roadmap Planner with Sampling on the Medial Axis of the Free Space*, Proceedings IEEE International Conference Robotics and Automation,1999

[16] David Hsu, Tingting Jiang, John Reif, Zheng Sun, *The Bridge Test for Sampling Narrow Passages with Probabilistic Roadmap Planners*, IEEE International Conference on Robotics & Automation, 2003

[17] Bohlin R, Kavraki L.E, *Path planning using lazy PRM*, IEEE International Conference on Robotics and Automation, (ICRA 2000), Proceedings. 2000.

[18] Nielsen C.L., Kavraki L.E., *A two level fuzzy PRM for manipulation planning*, IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS 2000). Proceedings. 2000

[19] Guang Song, Shawna Miller, Nancy M. Amato, *Customizing PRM Roadmaps at Query Time*, Proceedings of the 2001 IEEE International Conference on Robotics & Automation, 2001

[20] Steven M. *Rapidly-exploring random trees: A new tool for path planning*,techreport,1998

[21] Anthony Stentz *Optimal and Efficient Path Planning for Partially-Known Environments*, Proceedings IEEE International Conference on Robotics and Automation, May 1994

[22] Emmanuel Mazer, Juan Manuel Ahuactzin, *The Ariadnes clew algorithm* ,Journal Artificial Intelligence Research,1998

[23] Marco Dorigo *Optimization, Learning and Natural Algorithms*, PhD thesis, Politecnico di Milano, Italy, 1992

[24] Marco Dorigo, Mauro Birattari, and Thomas Stützle, *Ant Colony Optimization, Artificial Ants as a Computational Intelligence Technique*,IRIDIA Technical Report Series,September 2006

[25] Xiaoping Fan, Xiong Luo, Sheng Yi, Shengyue Yang, Heng Zhang, *Optimal Path Planning for Mobile Robots Based on Intensified Ant Colony Optimization Algorithm*, Proceedings of the 2003 IEEE International Conference on Robotics,Intelligent Systems and Signal Processing, Changsha, China - October 2003

[26] Gengqian Liu Tiejun Li Yuqing Peng Xiangdan Hou, *The Ant Algorithm for Solving Robot Path Planning Problem*, Proceedings of the Third International Conference on Information Technology and Applications (ICITA05), 2005

[27] Beatriz A. Garro, Humberto Sossa and Roberto A. Vzquez, *Evolving ant colony system for optimizing path planning in mobile robots*,Fourth Congress of Electronics, Robotics and Automotive Mechanics,2007

[28] Hao Mei, Yantao Tian, Linan Zu, *A Hybrid Ant Colony Optimization Algorithm or Path Planning of Robot in Dynamic Environment*, International Journal of Information Technology, Vol.12, No.3, 2006

[29] W.J. Gutjahr, *A graph-based Ant System and its convergence*, Future Generation Computing Systems, 16:873888, June 2000.

[30] Holland, John H , *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975

[31] Roger Toogood, Hong Hao, and Chi Wong *Robot Path Planning Using Genetic Algorithms*, IEEE International Conference on Intelligent Systems for the 21st Century, 1995

[32] Yanrong Hu Yang, S.X., *A knowledge based genetic algorithm for path planning of a mobile robot*, Proceedings IEEE International Conference on Robotics and Automation, 2004

[33] Chunmiao Wang, Y. C. Soh, Han Wang, Hui Wang *A Hierarchical Genetic Algorithm for Path Planning in a Static Environment with Obstacles*, Proceedings of the 2002 IEEE Canadian Conference on Electrical & Computer Engineering, 2002

[34] Kyung min Han, *Collision free path planning algorithms for robot navigation problem*, Master Thesis, AUGUST 2007

[35] Cezar A. Sierakowski and Leandro dos S. Coelho, *Study of two swarm intelligence techniques for path planning of mobile robots*, Proceedings of 16th IFAC World Congress, 2005

Chapter 2

ACO implementation

In this chapter the details of the implementation of the ACO algorithm will be given. Placing the path planning problem at a grid of squares, the selection criteria of a path will be analyzed in the first section. In sections 2.2 and 2.3 the heuristic function and the pheromone update policy will be presented. When implementing the ACO algorithm several design parameters need to be defined (section 2.4). Several simulation were performed in different environments in order to explore the characteristics of the algorithm and to take some decisions over the selection of the parameters. The description of these simulations and their results will be given at the final section (2.5).

2.1 Path nodes generation

As we mentioned before, the model of the world, which we had considered for the path planning problem, is a grid of squares. There is a square of the grid considered as start position and another one considered as end position. The agents must find a path from the start to the end position. Each agent has a current position in the grid, can move by one square and has to make a decision about its new position. This decision is based on the amount of the pheromone in the grid square and on the value of a heuristic function. The next grid square is selected according to a pseudo-random distribution based on the following probability function.

$$P(S_i) = \begin{cases} \frac{\tau_i^\alpha \eta_i^\beta}{\Sigma_i \tau_i^\alpha \eta_i^\beta} & if q \geq q_0 \\ 1 & if q < q_0 \quad and \quad i = argmax(\tau_i^\alpha \eta_i^\beta) \\ 0 & if q < q_0 \quad and \quad i \neq argmax(\tau_i^\alpha \eta_i^\beta) \end{cases} \tag{2.1}$$

where α is the pheromone factor, β the heuristic factor, τ_i is the amount of the pheromone in the grid square i and η_i is the value of the heuristic function

computed for the grid square i. The pheromone factor, α, and the heuristic factor, β, indicate in which degree the amount of pheromone or the heuristic function value, respectively, influences the selection of the next position.

If a uniformly random value , $0 \leq q \leq 1$, is less than the selection parameter q_0 (see which value is chosen in section 2.4) the next position of the agent will be the position that appears to be the best one according to the heuristic function and the existing amount of pheromone of this position. Otherwise the next position will be randomly selected based on the weighted probability distribution $\mathrm{P}(S_i)$, where S_i is a grid square. When the next position is selected, it will be appended at the end of the current trajectory of the given agent.

The set of the possible next positions of an agent are the adjacent grid squares of its current position considering that it can move diagonally, so there are 7 possible new positions. Each agent must check if any of its possible next positions are occupied by an obstacle and exclude them from the set of the possible next positions. Moreover, each agent can select a grid square as its new position only if this square does not exist already to its trajectory.

2.2 Heuristic

ACO algorithm is able to take advantage of the specific characteristics of a problem by using a well defined heuristic function. As we notice from (2.1), the probability distribution of the next position of an agent takes into account both pheromone, τ, and heuristic, η, information. Thus, a good heuristic function can lead to promising solutions. The heuristic function used in this implementation of the ACO algorithm is the Euclidean distance of the end position to the current position, as it has been proposed by many researchers that tried to implement ACO algorithm on the path planning problem [2, 3, 4].

$h_i = \sqrt{(x_{end} - x_{curr})^2 + (y_{end} - y_{curr})^2}$ (2.2) The heuristic value at each grid square is computed by: $\eta_i = \dfrac{1}{h_i}$.

2.3 Pheromone update policy

Pheromone is lied at the end of each generation at paths found by agents and which can lead to the end position. Pheromone is the way that agents can communicate and can learn solutions that have already found by other agents. The pheromone update policy consists of two steps, pheromone deposit and pheromone evaporation.

The goal of the pheromone deposit step is to make information known from one generation to the next ones. At the end of each generation, pheromone is thrown to the best paths that have been found by the agents. The method of pheromone deposit used in this project is the elitist approach, proposed by Dorigo [1], according to which only the generation-best solution and the overall-best solution contribute to the pheromone update at the end of each generation. The amount of pheromone that will be deposited to a path is proportional to $1/\mathrm{J}(x_i)$, where $\mathrm{J}(x_i)$ is an objective function that is desired to be minimized. The objective function used is the length of a path, measured as the number of the grid squares members of the path.

After the end of each generation, an amount of the existing pheromone at the grid is evaporated. By this way, the agents are allowed to "forget" previous solutions and so, stagnation can be prevented. The approach used at our implementation of ACO algorithm is that pheromone, τ, is reduced in every grid square by a constant factor ϱ, $\tau_{i+1} = \tau_i(1 - \varrho)$.

2.4 Parameters

ACO algorithm is a very flexible and configurable algorithm, as there are a lot of design parameters need to be selected at the implementation stage and which are affecting the performance of the algorithm.

- number of ants (agents)
- number of generations (iterations)
- α, the pheromone factor
- β, the heuristic factor
- τ_0, the initial pheromone value
- q_0, the selection parameter

- ϱ, evaporation factor

The effect of the first four factors, number of ants, number of generations, α and β, to the algorithm's performance will be analyzed in the simulation section 2.6.

The initial pheromone value τ_0 is the amount of pheromone that initially is thrown to each square in the grid and this was set to 10. The selection parameter q_0 was selected based on the results of Dorigo [1] and was set to 0.5. Large values of q_0 mean that the algorithm will mostly focus on already known solutions while small values mean that it will try to explore in bigger depth the search space. Dorigo's results were taken into account again to set the evaporation factor, ϱ, to 0.1.

2.5 Termination conditions

A proof of the convergence of the ACO algorithm to the optimal solution is given by Gutjahr [6] and Dorigo [1], but it is not possible to make an estimation on the time of convergence. Thus, a termination condition needs to be defined.

One approach would be to set a time limit for the run of the algorithm, such that it gives the best solution found after a specific time range. One possible problem of the ACO algorithm is stagnation, in which the algorithm finds a local minimum and throws pheromone to it for continuous generations making it increasingly impossible to move to a better solution. A restart of the algorithm in the case of stagnation detection or running the algorithm for instance 5 times and choosing the best execution may solve this problem.

In the simulations of the next section, we simply run the algorithm for a given number of generations in order to detect the effect of the different parameters to the performance of the algorithm.

2.6 Simulation model

Simulations were performed for 3 different environments (18x18 square grid) and for different values of the number of ants, number of generations, α and β in order to detect the behavior of the algorithm with different parameters. The start position is the top left square and the end position is

the down right square. In Figure 2.1, Figure 2.7 and Figure 2.12 we see a visual representation of the 3 environments. The black squares are occupied by obstacles, whereas the white ones are free. The environment 1 has the fewest obstacles and the environment 3 the most. For each set of parameters values we performed 10 different runs of the algorithm and we extracted the average length of the solutions. The aim of these simulations is to find the most appropriate values from the above mentioned parameters, such that the algorithm converges to a satisfying solution for all three tested environments. Another concern that must be addressed is the variance of the solutions found by the different executions of the algorithm for the same set of parameters. The variance of the solutions needs to be minimized, because it is not acceptable to have totally different solutions for the same values of the parameters for different executions.

2.6.1 Environment 1

The first environment (Figure 2.1) is a rather simple environment, in which the shortest path from the start position to the end position is 21 steps. Using 50 generations and setting α and β equal to 1, we examined the effect of the number of ants at the performance of the ACO algorithm. As we see in Figure 2.2 the algorithm has a rapid improvement, but after some iterations it fails to make significant improvement possibly due to stagnation and it nearly stabilizes to 28 steps. In Figure 2.3 we see that there is a very small improvement (it stabilizes at almost 30 steps) when we increase the number of generations and while we have used 20 ants and set α and β equal to 1. Another possible reason why the algorithm fails to present bigger improvement with the increase of the number of ants and generations is that the first environment is a very simple one but with a great variety of good solutions, so the algorithm easily falls into a local minimum.

However it is interesting to identify the variance of the solutions that ACO finds in different executions for the same parameters. As mentioned above, we have made 10 executions for every set of parameters, so it is easy to check the variance of the solutions of the algorithm by examining the standard deviation of the 10 executions for each set of parameters. Figure 2.4 shows the distribution of the standard deviation for the simulation of Figure 2.2, where we had some fixed values for the number of generations, α, β and increasing values for the ants. The variance of the found solutions decreases as the number of ants increase. This figure indicates that a good value for the number of ants would be above 100, as it seems that the algorithm's behavior stabilizes for large number of ants.

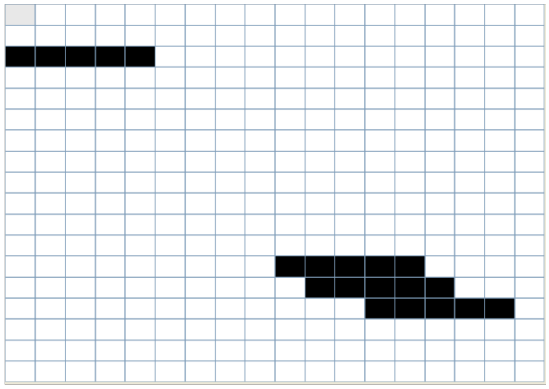


Figure 2.1: Environment 1

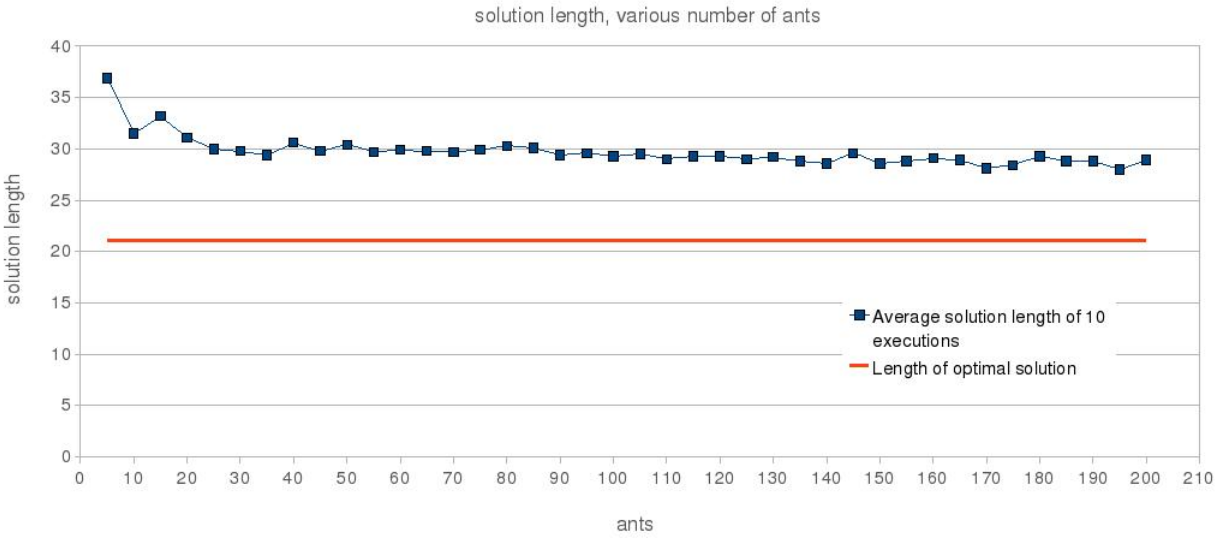


Figure 2.2: Increasing ants at the first environment.

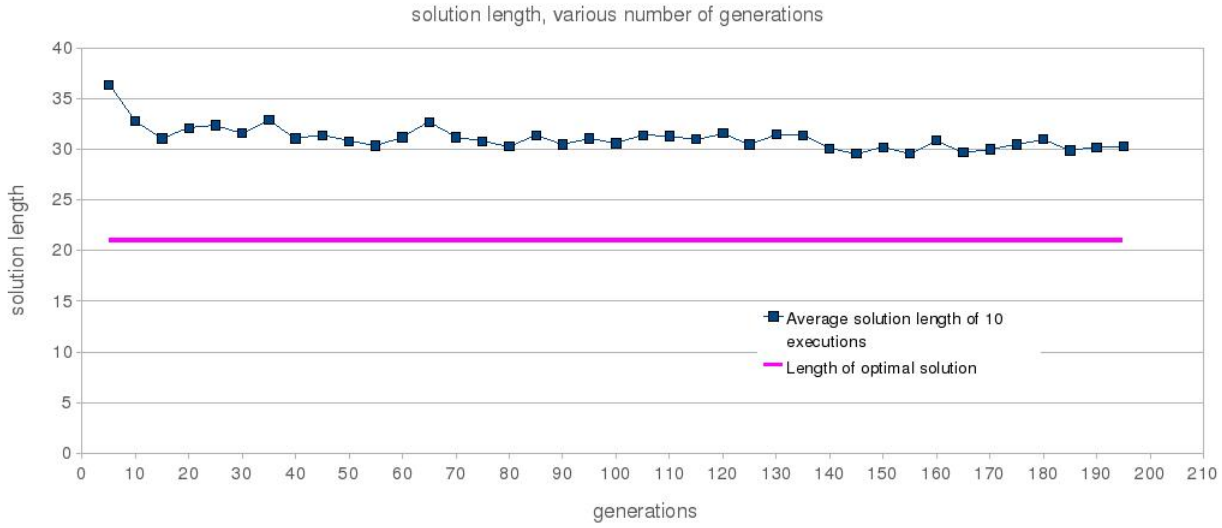


Figure 2.3: Increasing generations at the first environment.

In Figure 2.5 and Figure 2.6 the influence of the pheromone factor, α, and the heuristic factor, β, is demonstrated for the simple environment 1. For large values of α (bigger than 1), the algorithm is expected to stagnate to the first good solution found, whereas large β (bigger than 1) would lead the agents to choose among solutions with high heuristic value. Since environment 1 is a simple environment and the Euclidean distance, used as heuristic function, is a good heuristic for such a simple problem it is logical that the algorithm performs better with a bigger β than α. A high value for α leads the algorithm to stagnate to the first good solution found. Both tests have been done for fixed values of ants and generations(with 20 ants and for 100 generations).

2.6.2 Environment 2

The second environment is a more complicated environment (Figure 2.7) with optimal solution of 32 steps, where we expect the algorithm will need more generations to converge to a good solution. Indeed, in Figure 2.8 we see that as the generations increase the results of the algorithm improve, but the improvement is getting smaller and smaller after approximately 80 generations have occurred and it stabilizes at about 47 steps. This test was made with 20 ants, but if 100 ants are used, as proposed in the previous paragraph, the algorithm converges in less generations to a better solution of almost 41 steps (Figure 2.9). In both tests, with 20 and 100 ants, α and

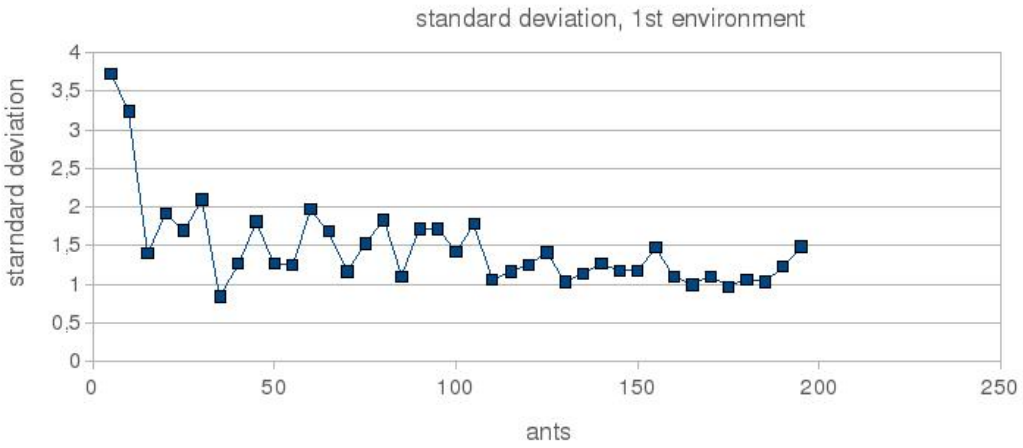


Figure 2.4: Variance of the solution found for increasing ants at the first environment.

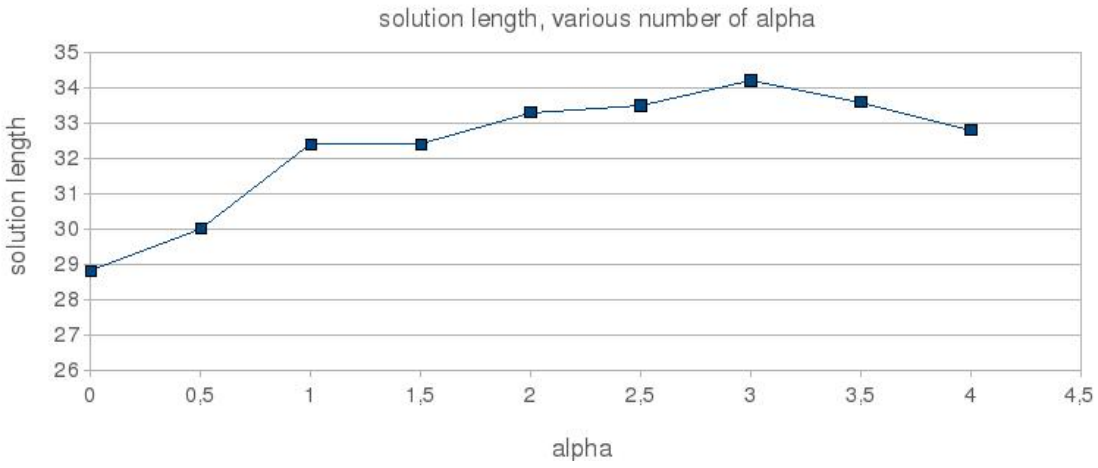


Figure 2.5: Solutions found for various values of alpha at the first environment. beta = 1

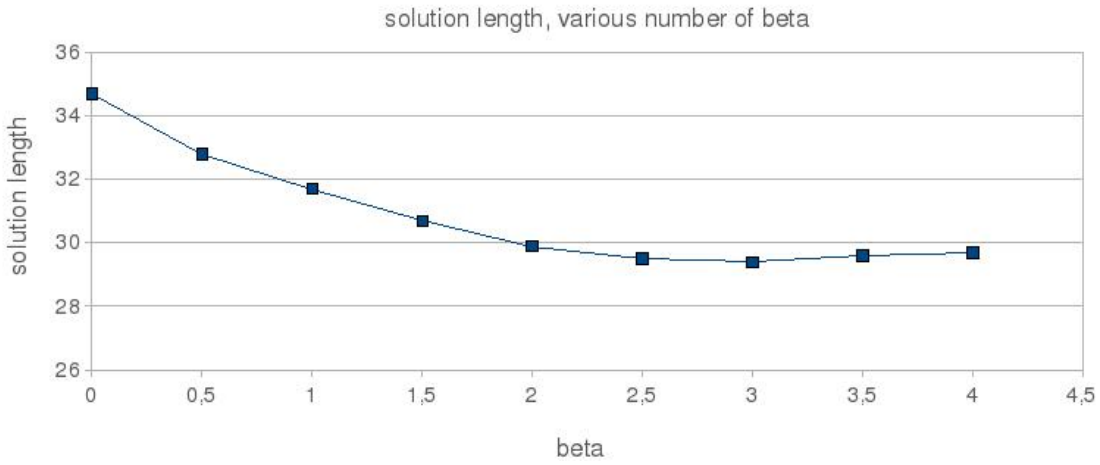


Figure 2.6: Solutions found for various values of beta at the first environment. alpha = 1

β were set to 1.

ACO algorithm's performance greatly depends on the quality of the heuristic[24]. If the used heuristic function exploits well the specifications of the problem, then the solutions found by ACO will be of high quality. At the case of environment 2, the Euclidean distance cannot describe with high faithfulness the characteristics of the problem, since it can lead to areas blocked by obstacles. In Figure 2.10 and in Figure 2.11 we see the effect of α and β in the quality of the found solutions for fixed values of ants (20) and generations (50) at the second environment. Increasing the influence of the heuristic (bigger values

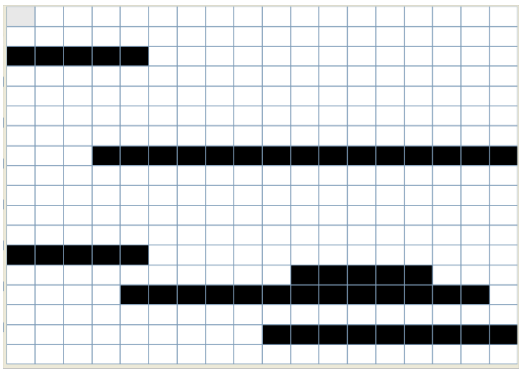


Figure 2.7: Environment 2

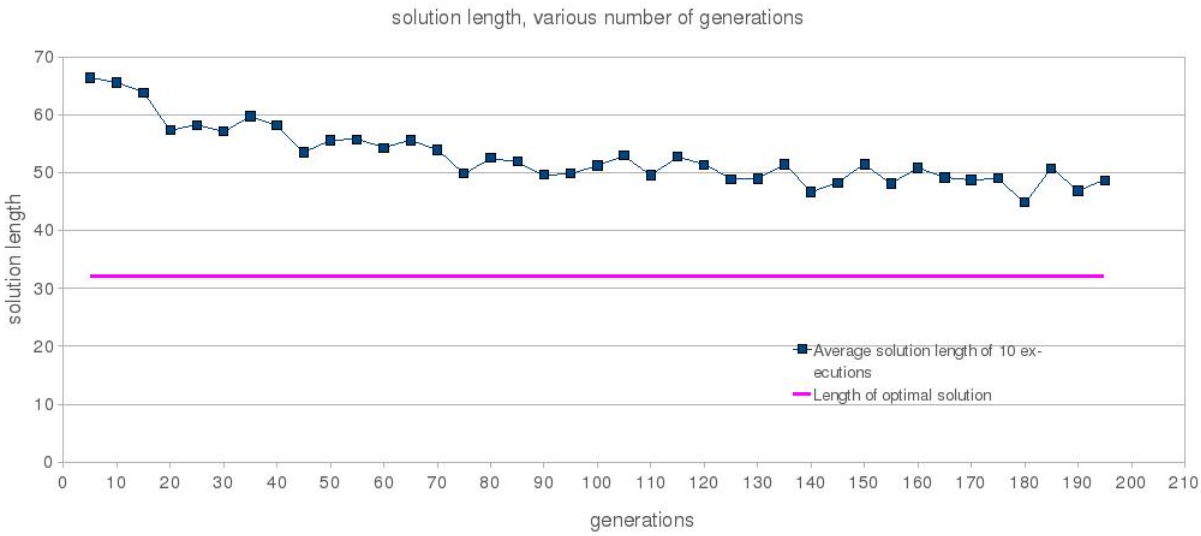


Figure 2.8: Length of solutions for increasing number of generations. 20 ants. Environment 2

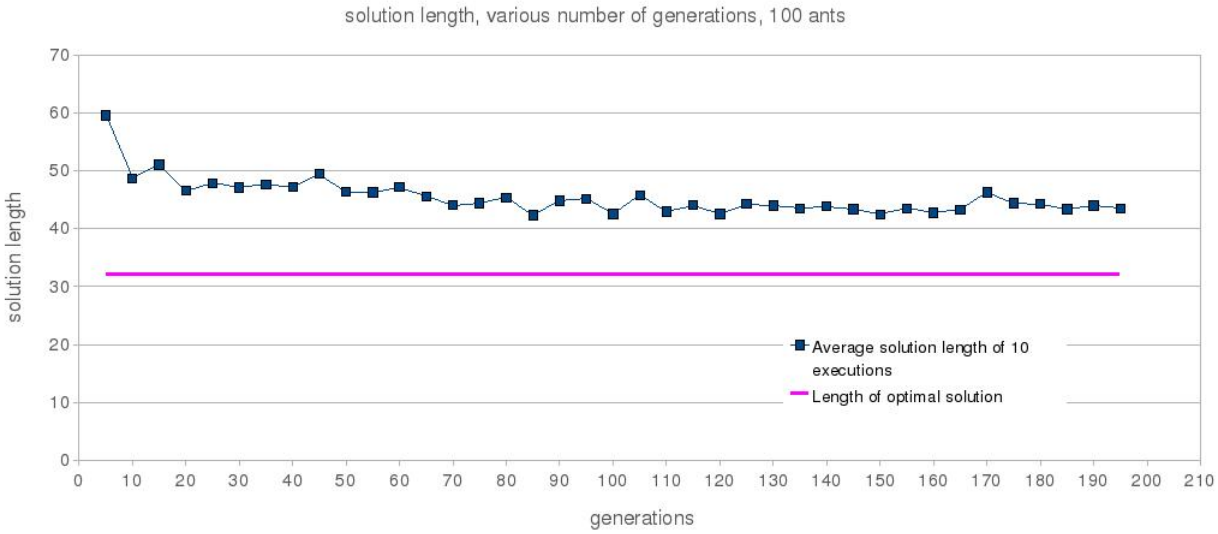


Figure 2.9: Length of solutions for increasing number of generations. 100 ants. Environment 2

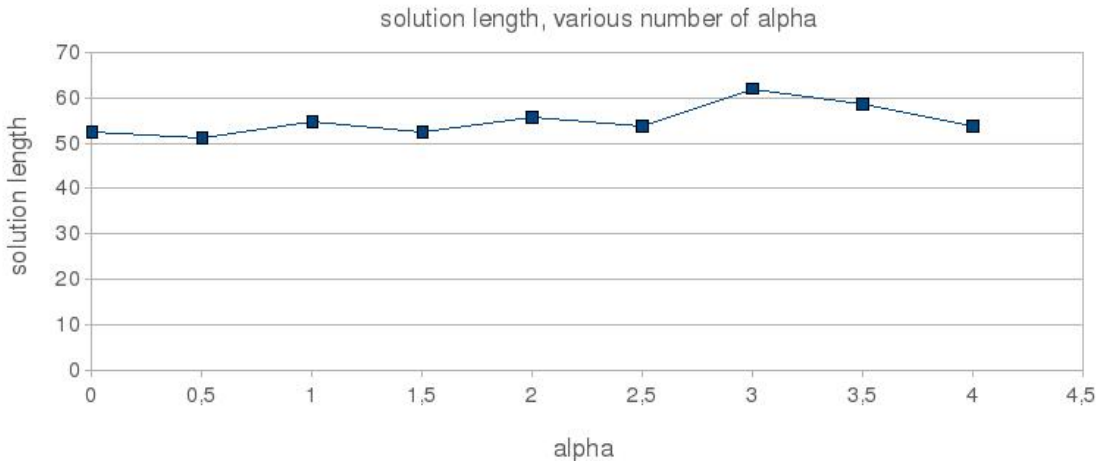


Figure 2.10: Length of solutions for different values of alpha. Environment 2

of β) leads to worse solutions, which informs us that we may need a better heuristic function for this kind of environments. Increasing the influence of α leads again to stagnation to the first good solution found. Some researchers have proposed heuristics that take into account the distribution of the obstacles in the environment, as for example Wen Ye et. al [5] that have used a heuristic,which computes the distance between the current position and the nearest obstacle besides to the distance from the goal position.

2.6.3 Environment 3

The third environment is a slightly more complicated environment than the second one. There are a couple of points at the grid, where the agents have to decide which direction to follow in order to find a good solution. The optimal solutions here consists of 34 steps. We used 100 ants for the test of increasing generations (Figure 2.13) at this environment, since the problem is more complicated and this number of ants was proposed from the previous simulations. In Figure 2.13, we see that the algorithm converges rather fast, after about 60 generations, to a relatively good solution (around 40 steps), since the shortest path for this environment is 34 steps.

We used 4 different plots (Figure 2.14) to explore the influence of the pheromone (α) and the heuristic (β) factors at the third environment. The combinations of α and β, $1 \leq \alpha, \beta \leq 4$, were examined for increasing number of generations and with 20 ants. The Figure 2.14 shows the average solution length found of 10 executions from the above combinations of α and β. We

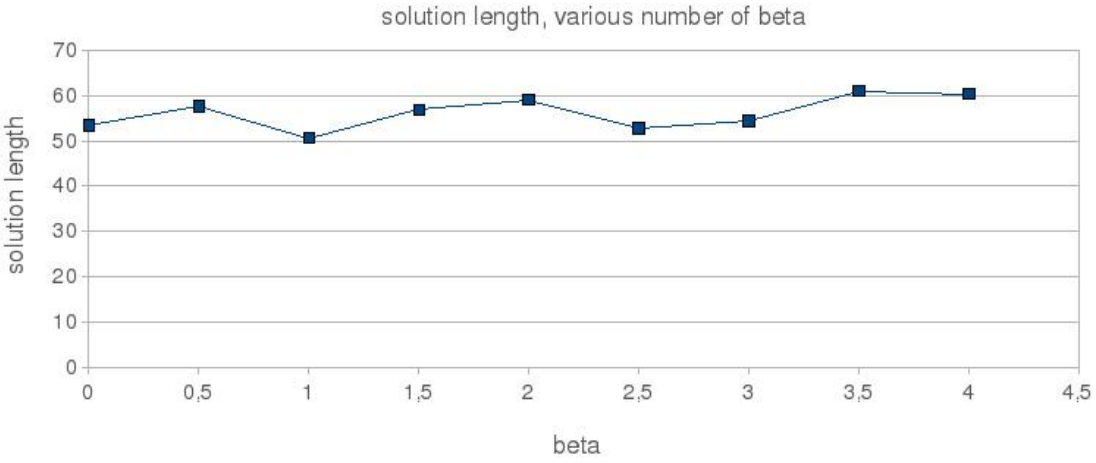


Figure 2.11: Length of solutions for different values of beta. Environment 2

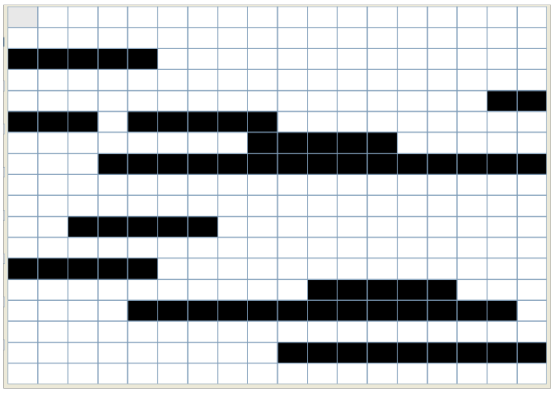


Figure 2.12: Environment 3

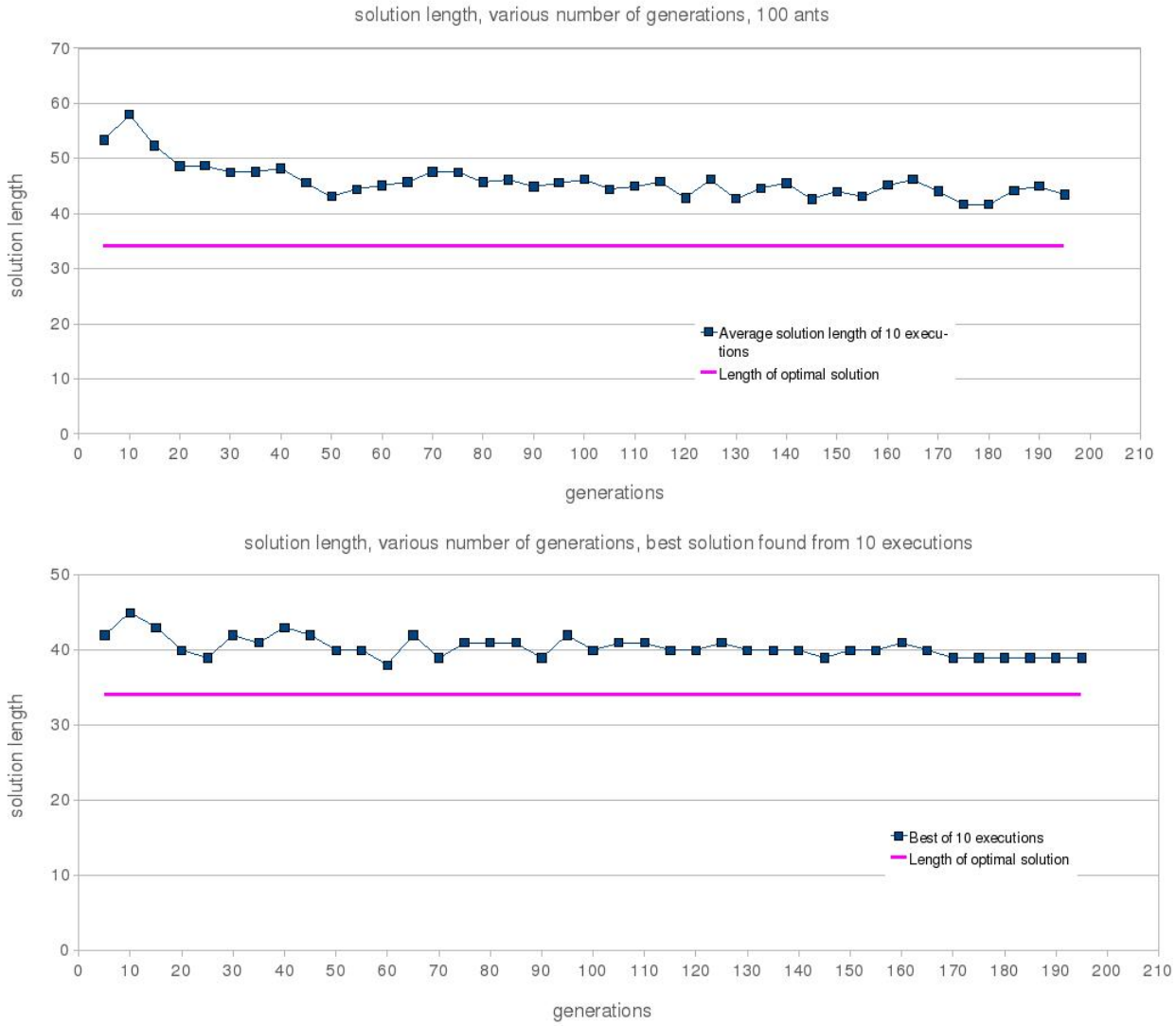


Figure 2.13: Length of solutions for increasing number of generations. 100 ants. Average solution length and best solution found of 10 executions. Environment 3

conclude that all the combinations converge eventually at solutions of approximately the same quality, but the number of generations needed for the convergence is different. For a rather complicated environment like the third one, it is necessary that the influence of the pheromone factor is bigger than the influence of the heuristic factor. The pair of $\alpha = 2, \beta = 1$, for example, has one of the best behaviors, since it converges rather fast in about 20 generations. We should note that the quality of the solutions would be far more better if we have used 100 ants instead of 20 for these tests, as indicated by Figure 2.13. The results may be different if we used a more appropriate heuristic, that would take advantage degree the characteristics of the environment. In this case, possibly it would be better to set a bigger value for β. However, in a partially-known and dynamically changed environment, we do not want a heuristic that depends on the environment to have a big influence at the search of the path.

As we see at the diagrams from the three simulations, the solutions are improved by smaller rate as the number of the generations increases, which means that after a big number of iterations the result of the algorithm stabilizes and there is no need to perform any more iterations. This is the reason why in our simulations, we have stopped the execution of the algorithm after 200 iterations. As mentioned in 1.4.1, Gutjahr [6] have shown that the convergence probability of the ACO algorithm can be increased by increasing the number of ants. In Figure 2.15, the improvement of the solutions found for the third environment as the number of ants increases are shown. We have extracted the best solutions from each set of executions (10 executions) for the same number of ants. There is an improvement at the solutions found, but the rate of improvement is getting smaller and smaller as the number of ants increases.

2.6.4 Simulation results - Parameters selection

During the simulations at the three environments, we tried to decide which values would make ACO algorithm to perform well for the path planning problem. As mentioned before, we have set the number of ants to 100 in order to eliminate the variance of the found solutions. An increased number of ants leads also to higher quality solutions as we concluded from the comparison of Figure 2.8 and Figure 2.9.

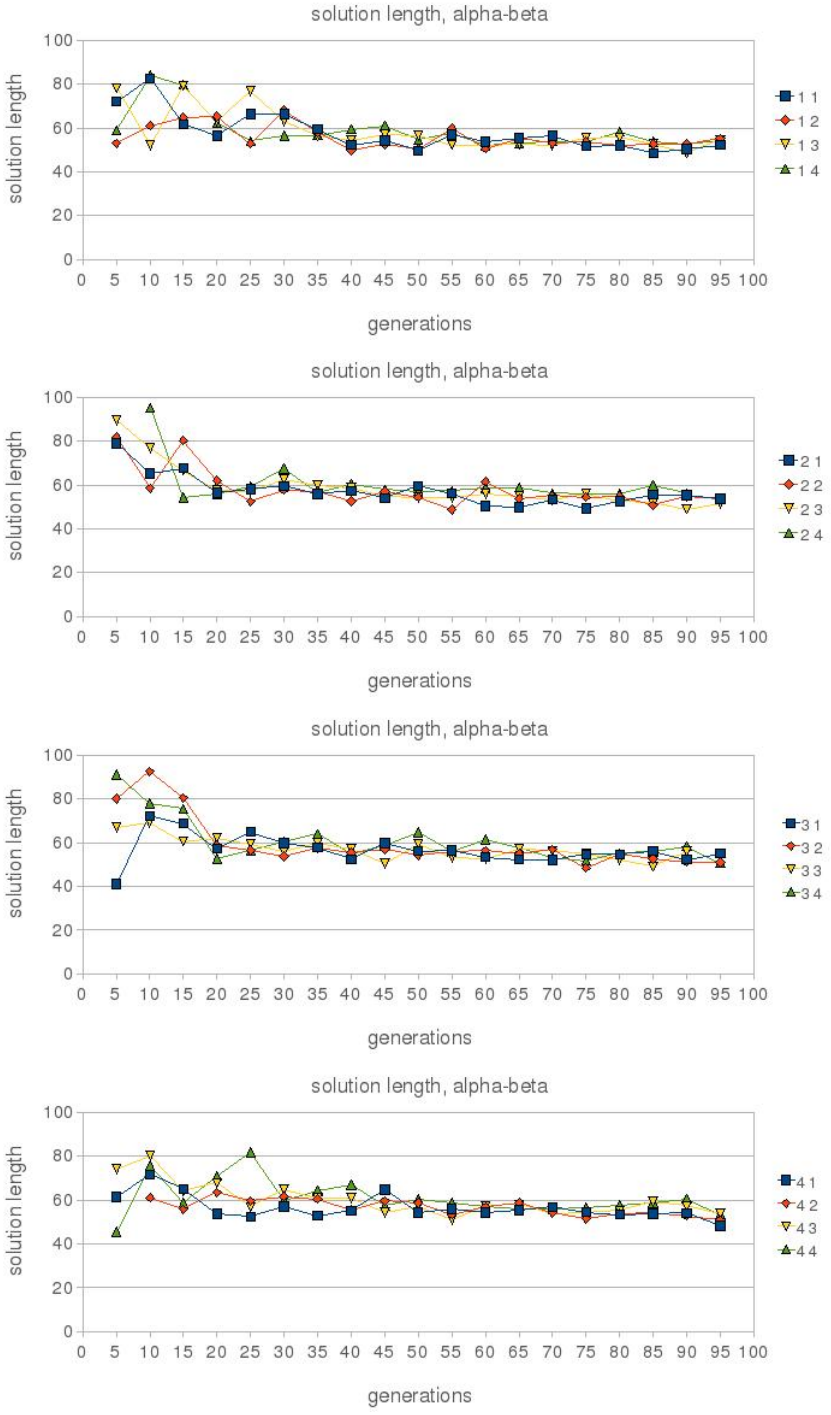


Figure 2.14: Different combinations of α and β. The first column at the legends represent the values of alpha and the second columns represent the values of beta. Environment 3

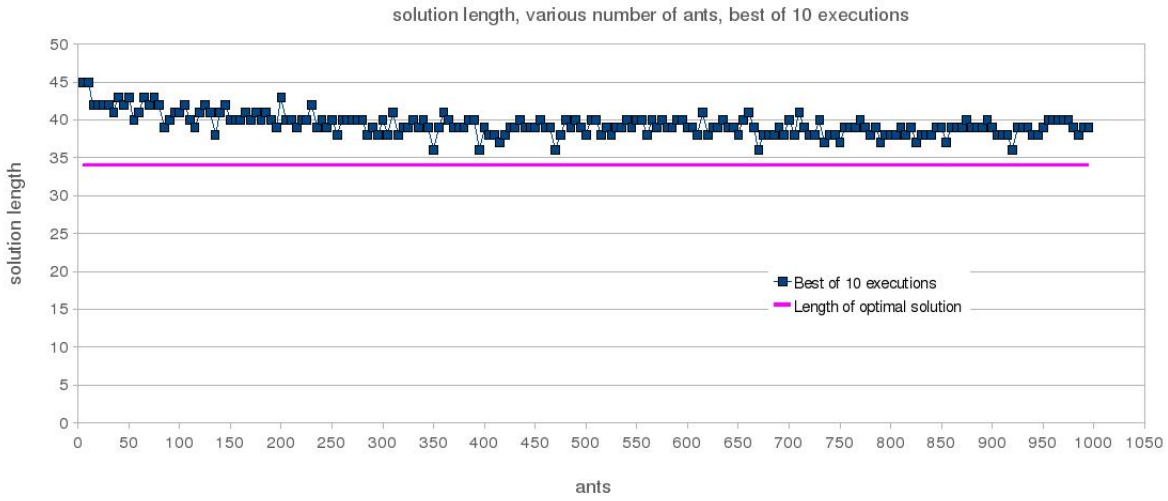


Figure 2.15: Length of solutions for increasing number of ants. Best solutions found from 10 executions for each set of parameters.Environment 3

As the number of generations is concerned, two different methods can be implemented. As it was mentioned in section 2.5, one approach would be the algorithm to do the most generations possible in a specific time range and then select the best solution found. Another approach would be to set a fixed number of generations and stop the execution of the algorithm when this has been reached. In this case, we would propose a relatively big number of generations in order to ensure that a good solution will be found. As we see in Figure 2.13.a value of 100 generations can lead to good solutions.

The selection of the α and β factors depend on the quality of the heuristic function. If a very good heuristic is available we should use bigger values for β in order to take advantage of it. In our case, where the distribution of the obstacles in the environment is rather random, it is difficult to find a very good heuristic for every kind of problem. We would choose a bigger influence of the pheromone factor and set $\alpha = 2, \beta = 1$.

ACO algorithm was originally developed and applied at the Traveling Salesman Problem (TSP) and even in this case a good heuristic was needed to be competitive with existing methods [24].

Bibliography

[1] Marco Dorigo, Thomas Sttzle, *Ant Colony Optimization*, MIT Press, 2004.

[2] Ying-Tung Hsiao, Cheng-Long Chnang, and Cheng-Chih Chien, *Ant Colony Optimization for Best Path Planning*, lnternational Symposium on Communications and Information Technologies (ISCIT 2004), 2004

[3] Chunxue Shi, Yingyong Bu, Ziguang Li, Jun Tan, *Solving Path Planning Problem by an ACO-PSO Hybrid Algorithm*, International Conference on Intelligent Systems and Knowledge Engineering (ISKE 2007), 2007

[4] Gengqian Liu, Tiejun Li, Yuqing Peng, Xiangdan Hou, *The Ant Algorithm for Solving Robot Path Planning Problem*, Proceedings of the Third International Conference on Information Technology and Applications (ICITA05), 2005

[5] Wen Ye, Dengwu Ma, Hongda Fan, *Path Planning for Space Robot Based on The Self-adaptive Ant Colony Algorithm*, Proceedings of the IEEE International Conference on Robotics,Intelligent Systems and Signal Processing, 2003

[6] W.J. Gutjahr, *A graph-based Ant System and its convergence*, Future Generation Computing Systems, 16:873888, June 2000.

Chapter 3

iCub infant-like robot

3.1 iCub crawling architecture

The iCub is an infant-like robot developed as part of the RobotCUB project[1]. It has the size of a 2 years-old child and the ability to crawl. We used a simulation of this robot at the ODE-based software Webots^{TM} in order to perform some tests of the implementation of the ACO algorithm. In this section we will make a brief summarization of the architecture used for the locomotion of this robot.

An architecture of 3 layers (planner, manager, generator) has been developed [2]. The planner represents the mental aspect of the task, the manager's responsibilities are the selection, the timing and the coordination of different behaviors and the generator uses central pattern generators (CPGs) [3],[4] in order to produce trajectories. Generator's implementation allows the generation of discrete (i.e. short-term) and rhythmic movements and the combination of them.

The discrete movements are modeled as solutions of a dynamical system with a globally fixed point :

$$\dot{h_i} = d(p - h_i) \quad (3.1)$$

$$\dot{y_i} = h_i^4 v_i \quad (3.2)$$

$$\dot{v_i} = p^4 \frac{-b^2}{4}(y_i - g_i) - bv_i \quad (3.3)$$

This is a critically damped dynamical system, so that a discrete trajectory is generated. The output y_i converges asymptotically and monotically to

a goal g_i and the speed v_i converges to zero. The speed of convergence is controlled by b. The values of p and d are chosen so to ensure a bell-shaped velocity profile.

The rhythmic movements are modeled as solutions of a limit cycle dynamical system :

$$\dot{x_i} = \alpha(m_i - r_i^2)(x_i - y_i) - \omega_i z_i \quad (3.4)$$
$$\dot{z_i} = \alpha(m_i - r_i^2)z_i + \omega_i(x_i - y_i) + \Sigma k_{ij} z_i + u_i \quad (3.5)$$
$$\dot{\omega_i} = \frac{\omega_{stance}}{e^{-fz_i} + 1} + \frac{\omega_{swing}}{e^{fz_i} + 1} \quad (3.6)$$

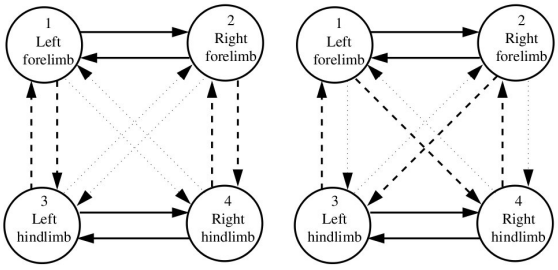
where $r_i = \sqrt{(x_i - yi)^2 + z_i^2}$, ω is the frequency of oscillations in $rad \cdot s^{-1}$, ω_{swing} and ω_{stance} are the frequencies of the swing and stance phases respectively and α is a positive constant that controls the speed of convergence to the limit cycle. When $m_i > 0$ Hopf oscillations are performed with amplitude $\sqrt{m_i}$, frequency ω_i and an offset g_i. These oscillations can be turned off via Hopf bifurcation when $m_i < 0$. The term k_{ij} represents the gain of the coupling between the rhythmic unit generators i and j. Feedback information is added to the system by the term u_i.

The coupling of the different unit generators in one network constructs a central pattern generator (CPG) [5], which guarantees fixed time relationships between different rhythmic outputs. In order to produce complex movements, superimposition of discrete and rhythmic unit generators is performed by injecting the discrete movement into the rhythmic one. A network of coupled oscillators has been developed by Righetti and Ijspeert [6] with the abilities of walk, trot, pace and bound. In Figure 3.1, the couplings of the constructed network for different types of gaits are shown.

In addition, feedback information from load sensors at the hands and the knees of the robot is included in the rhythmic PG in order to modulate the onset of swing and stance phases. According to Righetti and Ijspeert [6], the following formula is used:

$$u_i = \begin{cases} -sign(y_i) \cdot F & fast\ transitions \\ -\omega_i x_i - \Sigma k_{ij} y_j & stop\ transitions \\ 0 & otherwise \end{cases}$$

where F controls the speed of the transition. There are fast transitions during stance when the weight under the foot reduces and during swing when the



(a) Trot Pace Bound Net. (b) Walk Net.

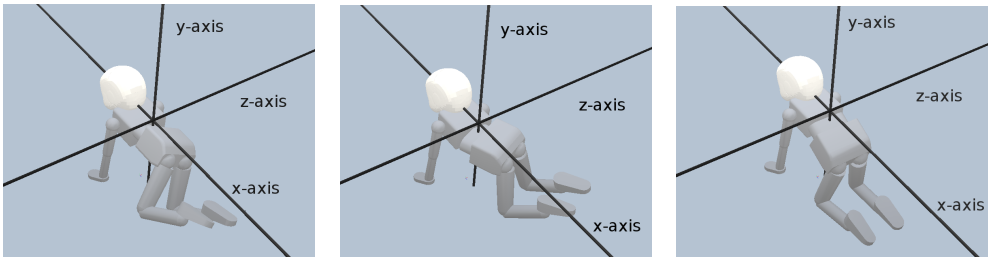
Figure 3.1: 4 cell networks for different types of gait.Same coupling between cells is shown by arrows of the same type. Figure taken by Righetti and Ijspeert [6]

foot touches the ground. Stop transitions conditions exist from swing to stance when the limb is not yet in contact with the ground and from stance to swing when the limb still supports significant body weight.

3.2 Steering

The steering ability of the iCub robot was not previously modeled, but that was essential for this project as long as the robot has to walk among obstacles in order to reach a goal position starting from an initial position. We have built an implementation for the steering of the iCub on the ODE-based software Webots^{TM}.

The iCub has 3 degrees of freedom at the middle of its torso. The second half of the robot's body (this is the part where the legs belong) can rotate against the first part (where the arms and the head belong) around the 3 axises according to the Figure 3.2. In order to steer the robot, we rotate the leg part of iCub around the axis y and the axis x. If the robot for example has to turn right, the upper part of its torso has to turn a bit right, so the leg part of the body has to rotate a bit around the y axis anti-clockwise as we see in Figure 3.3. But in this case, the right leg will not touch the ground as Figure 3.4 indicates, so we have to rotate the leg part around the x-axis, too.



(a) Rotation around the x-axis (b) Rotation around the y-axis (c) Rotation around the z-axis

Figure 3.2: Possible rotations of the second half of the iCub's body against the first half.

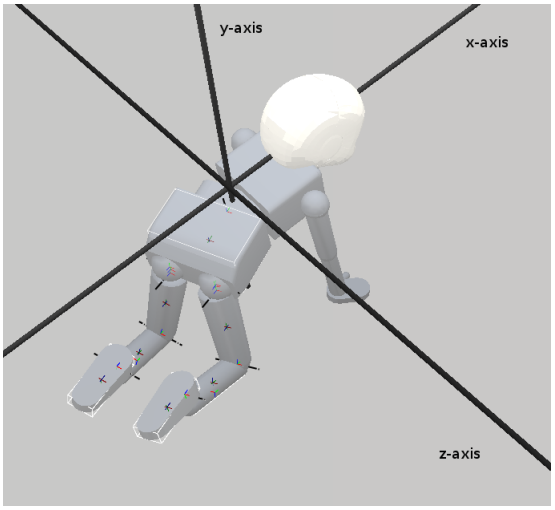
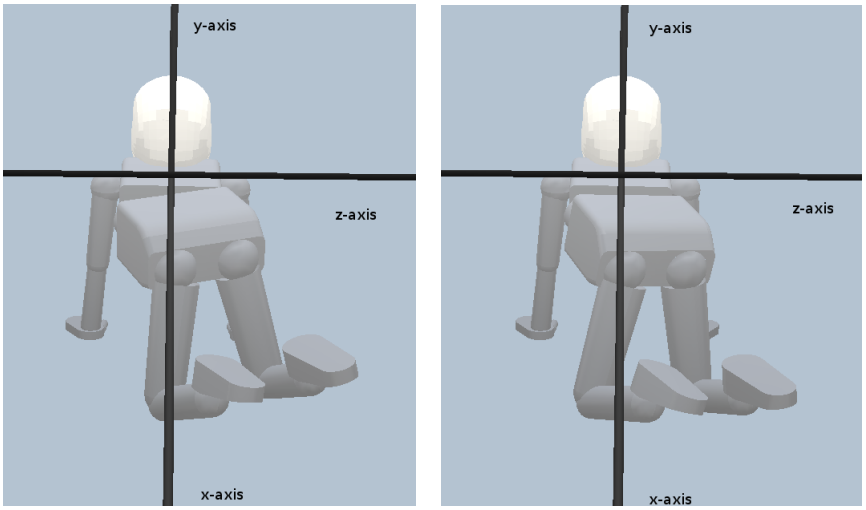


Figure 3.3: iCub turns right by rotating around the y-axis.



(a) Steering without rotating around the x-axis

(b) Steering with rotating around the x-axis

Figure 3.4: We need also to rotate the leg part of the body around the x-axis, otherwise the inner leg does not touch the ground.

After tests with the simulation platform we concluded that the robot would turn easier if the outer limbs were making a bigger step than the inner ones. Taking that into account, when the iCub turns right, the amplitude of the CPGs that control the left arm and the left leg increases a bit, whereas the amplitude of the CPGs for the right limbs decreases a bit. The opposite happens when the robot turns left.

To sum up, all the previous actions, which take part in case of steering, must happen in a very small degree at each step, so that the motion will be as smooth as possible. After tests in Webots^{TM}, we have concluded at the following schema in case of right turn:

if the angle of the y-axis is less than 0.43 do

angle of the y-axis += 0.05

angle of the x-axis += 0.012

leftFactor += 0.025

rightFactor += -0.025

amplitude of left limbs = leftFactor*amplitude of left limbs
amplitude of right limbs = rightFactor*amplitude of right limbs

The angles are measured in radians. The robot can steer up to a certain angle, because if it tries to steer more it loses its stability and falls. In case of left turn an analogous schema is performed, while if the robot has to crawl straight the angles are reset to zero and the leftFactor and rightFactor to one.

Bibliography

[1] N.G. Tsagarakis, G. Metta, G. Sandini, D. Vernon, R. Beira, F. Becchi, L. Righetti, J. Santos-Victor, A.J. Ijspeert, M.C. Carrozza, and D.G. Caldwell, *iCub - The Design and Realization of an Open Humanoid Platform for Cognitive and Neuroscience Research*, Journal of Advanced Robotics, Special Issue on Robotic platforms for Research in Neuroscience, 21(10):11511175, October 2007.

[2] S. Degallier, L. Righetti, L. Natale, F. Nori, G. Metta, and A.J. Ijspeert, *A modular bio-inspired architecture for movement generation for the infant-like robot icub*, In Proceedings of the second IEEE RAS / EMBS International Conference on Biomedical Robotics and Biomechatronics, BioRob, 2008.

[3] S. Grillner, *Biological pattern generation: The cellular and computational logic of networks in motion*, Neuron, 52(5):751766, December 2006.

[4] A. J. Ijspeert, *Central pattern generators for locomotion control in animals and robots: a review*, Neural Networks, 21(4):642653, 2008.

[5] M. Golubitsky, I. Stewart, and A. Torok, *Patterns of synchrony in coupled cell networks with multiple arrows*, SIAM J. Appl. Dynam. Sys., 4(1):78100, 2005.

[6] L. Righetti and A.J. Ijspeert, *Pattern generators with sensory feedback for the control of quadruped locomotion*, In Proceedings of the 2008 IEEE International Conference on Robotics and Automation (ICRA 2008), pages 819824, May 2008.

Chapter 4

Simulation - Results

4.1 Simulation setup

As mentioned in the previous chapter, we have integrated the ACO algorithm in a simulation of the robot iCub using the ODE-based software Webots^{TM}. We have constructed a simulation world represented by a chess-like board of 8x8 squares surrounded by a wall and with 5 square obstacles inside (Figure 4.1). The start position of the robot is the lower left square and the end position is the upper right square. The goal of the simulation is the robot to crawl from the start position to the end position through the result path of the ACO algorithm without colliding with an obstacle or the wall, which is the boundary of the board.

We have made 4 kinds of optimizations on the output of the ACO algorithm. Firstly, considering that the robot is not able to steer in a very big angle, we have modified the ACO algorithm in order not to take into account at the computation of the path the positions 1,7 and 8 of the Figure 4.2 as possible new positions. If the robot is at the center square, it cannot steer enough to reach one of these squares. Secondly, although the robot can move diagonally as we see from the possible movements in Figure 4.1, a diagonal movement must not be valid in the case of Figure 4.3, in which there are two obstacles adjacent to the robot's movement.

The last two improvements on the output path are shown in Figure 4.4. In Figure 4.4 a) the squares with the red spots are part of the path that the algorithm has generated. Considering that the robot gets into the picture by the left side, it has to steer twice consecutively, once right and once left, in order to reach the most right red-spotted square. In this case, we replace

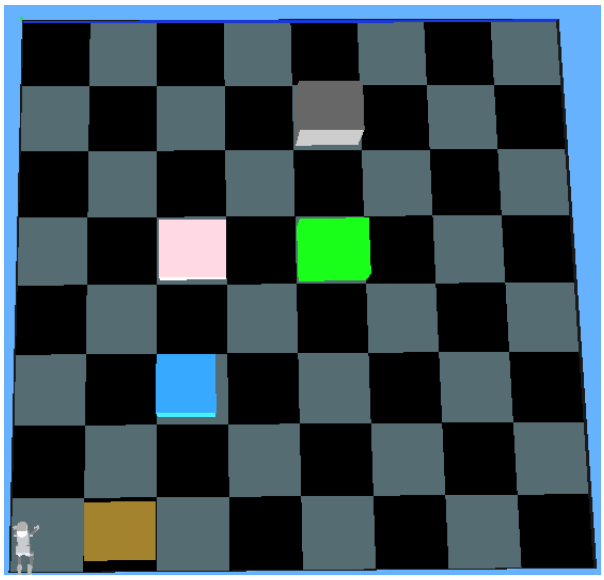


Figure 4.1: The simulation world. 8x8 chess board with 5 square obstacles.

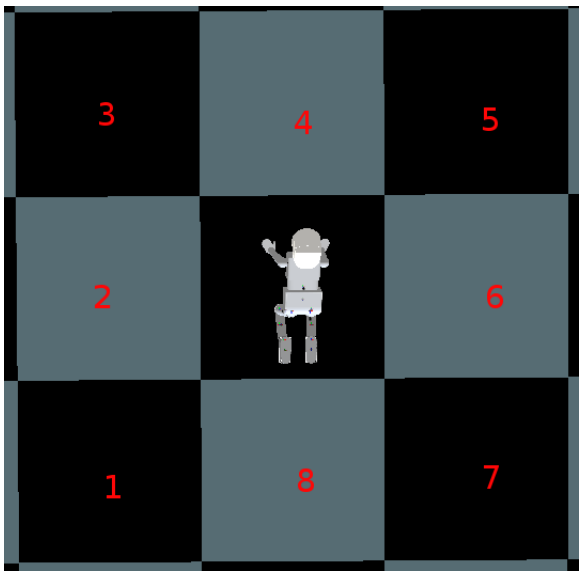


Figure 4.2: Possible next positions of the robot. The iCub cannot steer enough to go to the positions 1,8,7, so they are excluded from the computation of the path.

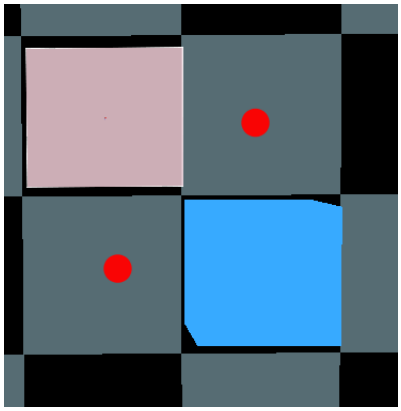


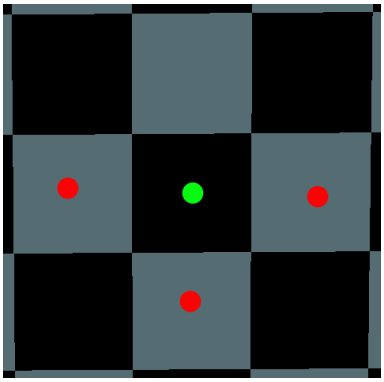
Figure 4.3: Although the robot is able to move diagonally, the squares with the red spot must not be part of the output path of the ACO algorithm, because of the existence of the adjacent obstacles.

the middle red-spotted square with the square with the green spot, provided that there is not any obstacle in this position, and so the robot keeps its straight trajectory. In Figure 4.4 b), we suppose that the robot gets into the image from the red-spotted square at the bottom and goes out from the red-spotted square at the left part. In this case, there is no need for the robot to make a 90^o degrees turn if there is no obstacle that would disturb a diagonal movement. So, in Figure 4.4 b) the middle square with the red spot will be eliminated.

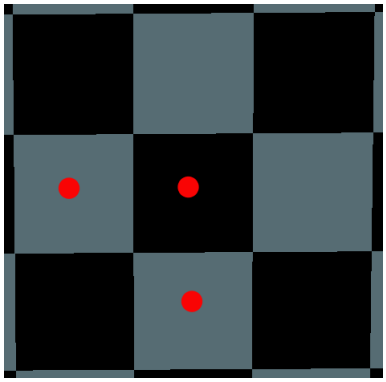
4.2 Simulation results

We have tested the algorithm's performance and the iCub's locomotion for various obstacle distributions (Figure 4.5) and in the majority of the cases the robot had reached its goal. In Figure 4.6 and in Figure 4.7, we can find out that the paths, which the robot had followed in two different cases, were the shortest possible.

Another feature of the simulation is the ability to change the obstacles' positions dynamically. When the robot detects a change in the obstacles' positions, it stops and recomputes a plan with start position its current position and the same end position and it continues its crawling from this position to the end. The robot may be trapped among obstacles as long as they change their positions dynamically and in this case it will not have a possibility to escape (Figure 4.8).

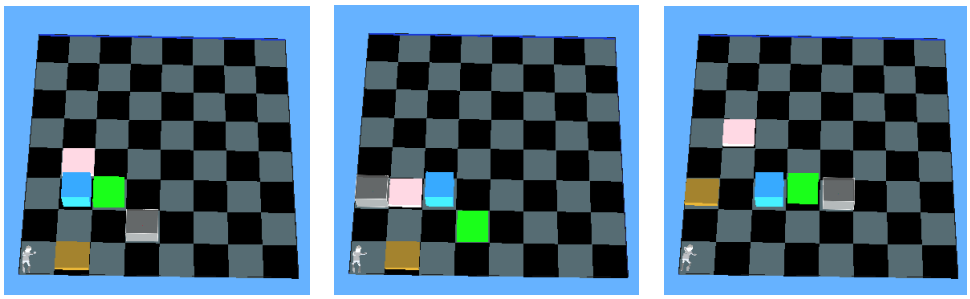


(a) The square with the green spot replaces the middle square with the red spot.



(b) The robot does not have to make 90^o degrees turn, so the middle square with the red spot is eliminated.

Figure 4.4: Optimizations to the result path of the ACO algorithm.



(a) (b) (c)

Figure 4.5: Examples of boards, in which the iCub crawled successfully among the obstacles to the end position.

In the case of the dynamically changed environment, the robot may fail to steer enough in order to follow a change to its path. For example in Figure 4.9 the blue obstacle appeared at a square to which the robot was going to. After the recomputation of the path the iCub had to steer right, but it didn't manage to steer enough to get in the next square and as a result it could not continue its trajectory.

4.3 Future Improvements

The performance of the ACO algorithm can be improved by two easy to implement ways.

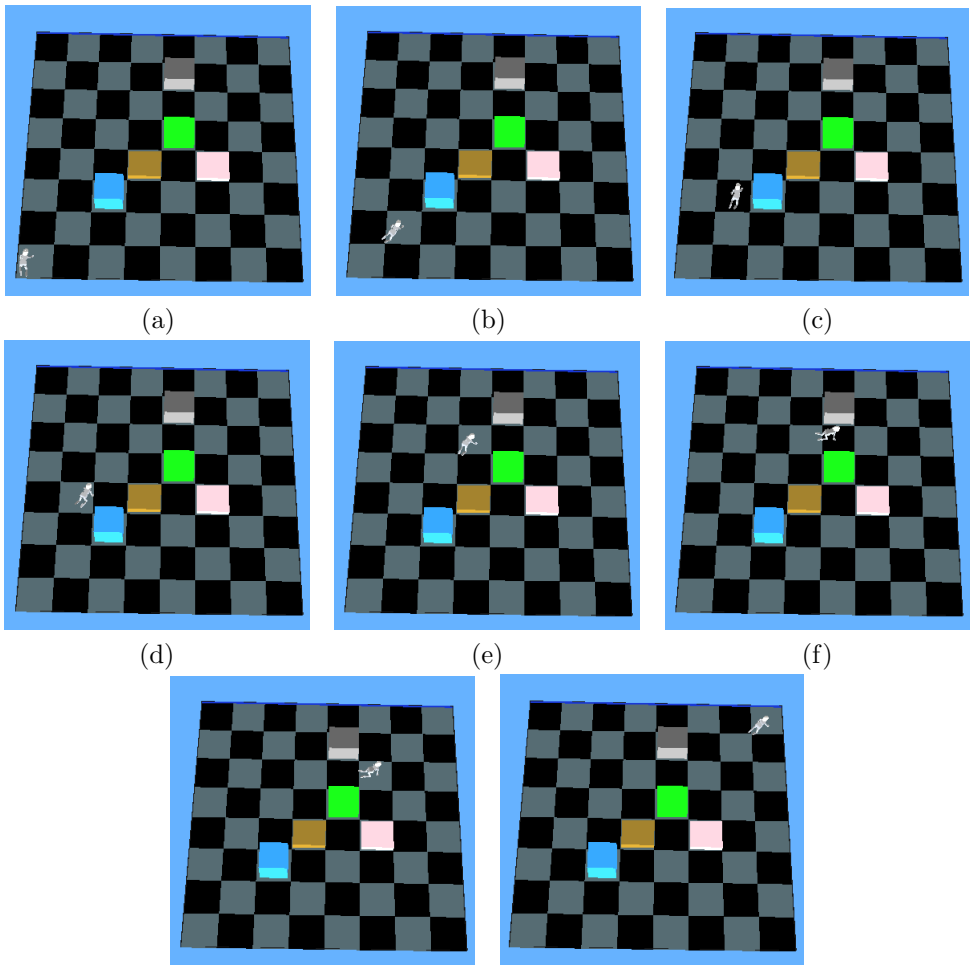


Figure 4.6: In this case the iCub followed the shortest possible path.

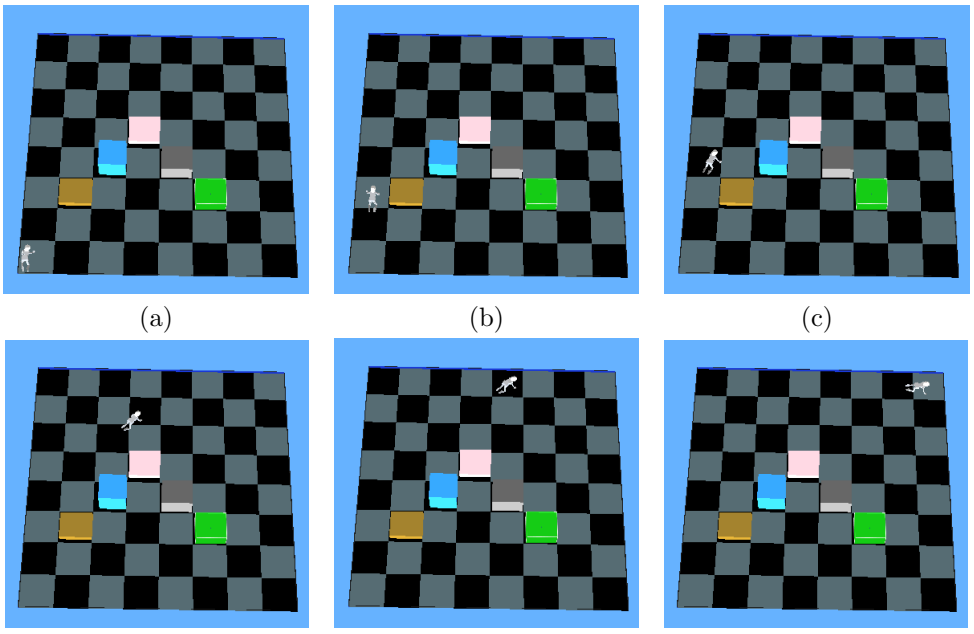


Figure 4.7: In this case the iCub followed the shortest possible path.

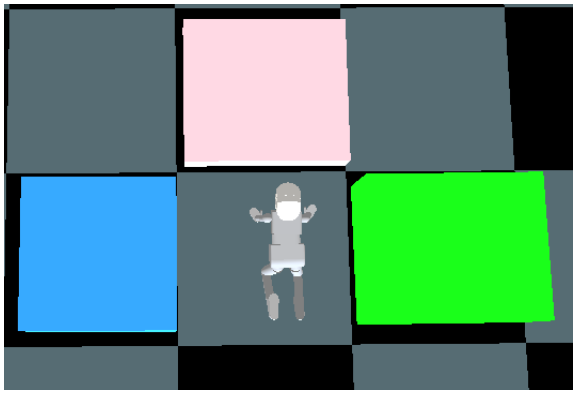
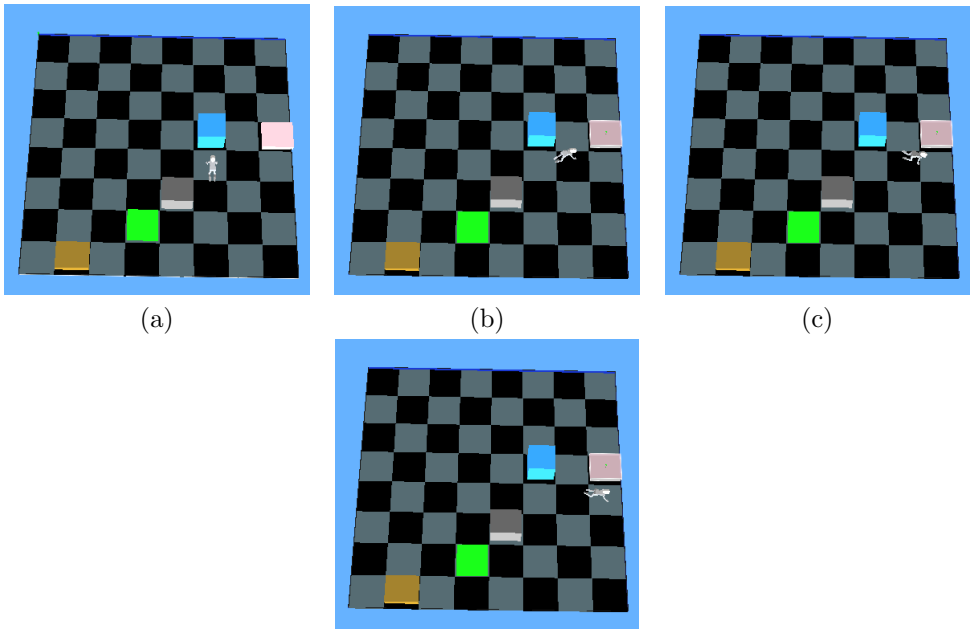


Figure 4.8: As long as the obstacles can change their positions dynamically, the iCub may be trapped.



(a) (b) (c) (d)

Figure 4.9: The iCub could not steer enough and it lost its path.

- Recomputation of the robot's path to the goal position during the simulation, for instance after every 5 steps, assuming as start position the robot's current position.
- Multiple executions of the ACO algorithm (e.g. 10 executions) and selection of the shortest path of all the outputs.

It would be also useful if the robot had the ability to detect if it is following a wrong route, for example in the case of dynamically changed environment that we described above, and try to find again its path. Finally, the integration of the ACO algorithm to the real iCub robot and testing of its performance in real environment conditions would be a challenging work.

4.4 Acknowledgements

I would like to thank my supervisor, Sarah Dégallier for her time and help, as well as my professor, Professor Auke Jan Ijspeert, for his useful advises and guidelines.