TECHNICAL UNIVERSITY OF CRETE, GREECE

DEPARTMENT OF ELECTRONIC AND COMPUTER ENGINEERING

Path planning for NAO Robots using an egocentric polar occupancy map

Iris A. Kyranou

Thesis Committee

Assistant Professor Michail G. Lagoudakis (ECE)

Assistant Professor Vasilios Samoladas (ECE)

Assistant Professor Aggelos Mpletsas (ECE)

Chania, April 2012

Πολυτεχνείο Κρήτης

Τμήμα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών

Σχεδιασμός διαδρομής για Ρομπότ NAO με χρήση εγωκεντρικού πολικού χάρτη εμποδίων

Ίρις Α. Κυράνου

Εξεταστική Επιτροπή

Επίκουρος Καθηγητής Μιχαήλ Γ. Λαγουδάκης (ΗΜΜΥ)

Επίκουρος Καθηγητής Βασίλειος Σαμολαδάς (ΗΜΜΥ)

Επίκουρος Καθηγητής Άγγελος Μπλέτσας (ΗΜΜΥ)

Χανιά, Απρίλιος 2012

Abstract

RoboCup is an international robotic soccer competition aiming at advancing research in autonomous robotics and artificial intelligence. Path planning decisions of a robot are important in avoiding collisions in the dynamic environment of a populated soccer field, since such collisions cause hardware damages, loss of valuable time, and penalties according to the game rules. This thesis describes a path planning method for autonomous robots in dynamic environments based on the construction of a local egocentric occupancy polar map for representing obstacles and the A* heuristic search algorithm for deriving collision-free paths to desired goals. The proposed approach creates and maintains a probabilistic occupancy grid map which discretizes the physical area close and around the robot using variable resolution and holds the robot's belief about the existence or not of obstacles. The occupancy values in the map are updated using real-time sensory information from ultrasonic or any other range sensors. Translation and rotation transformations of the map are implemented, in order to maintain the correct placement of obstacles relative to the robot consistently with its movements. Given a desired target position and orientation, an A* heuristic search algorithm in the three-dimensional space (coordinates on the field and orientation) derives an optimal path taking into account the omni-directional locomotion abilities of the robot (forward, side, rotating steps and combinations). The derived path provides the motion controller with the waypoints needed to set appropriately the velocities of the robot to move along the path. The proposed method has been implemented on Aldebaran Nao humanoid robots and can be used for planning and replanning paths in real-time. The method is deployed by the Technical University of Crete RoboCup team "Kouretes" in the Standard Platform League of the RoboCup competition to move the robots safely around the field without collisions towards any desired position and orientation.

Περίληψη

Το RoboCup είναι ένας διεθνής ρομποτικός διαγωνισμός, που στοχεύει στην προώθηση της έρευνας σε θέματα τεχνητής νοημοσύνης και ρομποτικής. Οι αποφάσεις ενός ρομπότ που σχετίζονται με το σχεδιασμό διαδρομών είναι σημαντικές για την αποφυγή συγκρούσεων στο δυναμικό περιβάλλον ενός πολυπληθούς γηπέδου, καθώς τέτοιες συγκρούσεις προκαλούν βλάβες του υλικού, απώλεια πολύτιμου χρόνου, και ποινές σύμφωνα με τους κανόνες του παιχνιδιού. Στην παρούσα εργασία, περιγράφεται μια μέθοδος σχεδιασμού διαδρομής σε δυναμικά περιβάλλοντα για αυτόνομα ρομπότ, η οποία βασίζεται σε έναν τοπικό εγωκεντρικό πολικό χάρτη πληρότητας για την αναπαράσταση εμποδίων και στον αλγόριθμο ευριστικής αναζήτησης A* για την εύρεση διαδρομών χωρίς εμπόδια προς επιθυμητούς στόχους. Η προτεινόμενη προσέγγιση δημιουργεί και συντηρεί ένα πιθανοτικό πλέγμα χάρτη πληρότητας που διακριτοποιεί τον φυσικό χώρο κοντά και γύρω από το ρομπότ με μεταβλητή ανάλυση και αναπαριστά την πεποίθηση του ρομπότ για την ύπαρξη ή όχι εμποδίων. Οι τιμές πληρότητας στο χάρτη ενημερώνονται χρησιμοποιώντας πληροφορίες πραγματικού χρόνου από αισθητήρες υπερήχων ή άλλους αισθητήρες απόστασης. Προκειμένου να διατηρηθεί η σωστή τοποθέτηση των εμποδίων σε σχέση με το ρομπότ και σύμφωνα με τις κινήσεις του, υποστηρίζονται μετασχηματισμοί μετατόπισης και περιστροφής του χάρτη. Δοθέντων των συντεταγμένων και του προσανατολισμού κάποιου στόχου, μια ευριστική αναζήτηση τύπου A* στον τρισδιάστατο χώρο (συντεταγμένες στο γήπεδο και προσανατολισμός) εξάγει μια βέλτιστη διαδρομή, λαμβάνοντας υπόψη τις ικανότητες μετακίνησης του ρομπότ προς κάθε κατεύθυνση (εμπρός, πλαϊνά, περιστροφικά βήματα και συνδυασμοί). Το μονοπάτι που εξάγεται παρέχει στον ελεγκτή κίνησης τα απαραίτητα σημεία για να ρυθμιστούν κατάλληλα οι ταχύτητες του ρομπότ ώστε να κινηθεί κατά μήκος της διαδρομής. Η προτεινόμενη μέθοδος έχει υλοποιηθεί σε ανθρωποειδή ρομπότ Aldebaran Nao και μπορεί να χρησιμοποιηθεί για το σχεδιασμό και ανασχεδιασμό διαδρομών σε πραγματικό χρόνο. Η μέθοδος χρησιμοποιείται από την ομάδα ρομποτικού ποδοσφαίρου «Κουρήτες» του Πολυτεχνείου Κρήτης στο πρωτάθλημα Standard Platform League του RoboCup για την ασφαλή μετακίνηση των ρομπότ στο γήπεδο χωρίς συγκρούσεις προς οποιοδήποτε επιθυμητό στόχο και προσανατολισμό.

Contents

1 Introduction **1**

1.1 Thesis Contribution 2

1.2 Thesis Outline 2

2 Background **5**

2.1 RoboCup Competition 5

2.1.1 RoboCup Leagues 6

2.2 Kouretes Team 10

2.3 Aldebaran's Nao Humanoid Robot 12

2.3.1 Nao Hardware 12

2.3.2 NaoQi 14

2.3.3 General Overview 14

2.4 Occupancy Grid Mapping 15

2.5 Path Planning 16

2.5.1 A* algorithm 16

3 Problem Statement **19**

3.1 Obstacle Avoidance 19

3.1.1 Obstacle Avoidance in Standard Platform League 19

3.2 Path Planning 20

4 Obstacle Avoidance Module **23**

4.1 Occupancy grid 23

4.2 Map Update 24

4.3 Map Transformations 25

4.3.1 Move Robot 26

4.3.2 Grid Translation 26
4.3.3 Grid Rotation 28
4.4 Path Planning 28

5 From Theory To Practice **33**
5.1 Sonars 33
5.1.1 Nao Sonars 34
5.2 The 10x18 polar grid 35
5.3 Map transformations 35
5.4 A* implementation 36
5.5 Debugging 37

6 Results **39**
6.1 Face Backwards 39
6.2 Avoid obstacle and reach a specified goal 39
6.3 Real field simulation 40

7 Related Work and Discussion **45**
7.1 Obstacle Detection Sensors 45
7.2 Grid Maps 45
7.3 Path Planning Algorithms 48
7.4 Why a Local Polar Grid? 49
7.4.1 Polar vs Regular 49
7.4.2 Local vs Global Grid Map 51
7.4.3 Differences with Nimbro approach [1] 51

8 Future Work and Conclusion **53**
8.1 Future Work 53
8.2 Conclusion 53

References **55**

List of Figures

2.1 Simulation league, 3D field 7
2.2 Small size league 7
2.3 middle size league 8
2.4 humanoid league 9
2.5 Manual settup for kick-off. Blue kick-off 10
2.6 Kouretes team 2011 formation. From left to right in the front row are Astero-Dimitra Tzanetatou, Iris Kyranou, Angeliki Topalidou- Kyniazopoulou, and in the back row standing up Emmanouel Orfanoudakis, Eleutherios Chatzi-laris, Nikolaos Spanoudakis, Michael Lagoudakis and Evangelos Vazaios . . 11
2.7 Nao robot, joints and sensors 13
2.8 Sonar beam regions 16
2.9 The A* algorithm (initial call with A*$(root, [\,])$). 18

4.1 Two positions of the robot [2] 27
4.2 Grid translation: Grid before (a) and after (b) move. Robot moves 0.1m in x and 0.1m in y 27
4.3 Grid rotation: Grid before (a) and after (b) rotate. Robot rotates for 40° 28
4.4 The eight possible node orientations 30
4.5 The neighbors expanding during an A* algorithm run for every node besides the first one, and their orientations. The grey node is the parent node. . . . 31
4.6 Picture 4.6(a) presents the neighbors orientation that does not add extra cost in the $g(n)$ function, in contrast with the orientations shown in picture 4.6(a) 31
4.7 A* algorithm examples 32

5.1 Sonars position on robot torso 34

5.2 Four different resolutions of the polar grid. The four obstacles are in distance 0.55m, on angles 45°, 135°, 225°, 270°. 36

6.1 The robot walks 1.2m on its side and turns to face the opposite goal. . 41
6.2 The robot walks 0.5m in front and 0.5m on its side and faces towards an orientation vertical to the one it had in the beginning of the move. 42
6.3 The robot walks towards the ball avoiding the other robots that get in its way. 43

7.1 The regular grid used by BHuman team [3] 46
7.2 Multi-resilution grid implementes by Nimbro team [] 47
7.3 Log-Poalr grid implementes by Nimbro team [4] 47
7.4 The Rapidly-Exploring Random Tree algorithm. 48
7.5 The Rapidly-Exploring Random Tree algorithm used to find paths on field [3] 49
7.6 In both images the black dot represents the robot. The regular grid 7.6(a) covers the half area in front of the robot, in contrast to the polar 7.6(b). In order to cover same distance range, the cell number in each dimension must double. 50

Chapter 1

Introduction

Path planning decisions of a robot are important in avoiding collisions in the dynamic environment of a populated soccer field, since such collisions cause hardware damages, loss of valuable time, and penalties according to the game rules. This thesis describes a path planning method for autonomous robots in dynamic environments, such as a soccer field, based on the construction of a local egocentric occupancy polar map for representing obstacles and the A* heuristic search algorithm for deriving collision-free paths to desired goals.

The proposed approach creates and maintains a probabilistic occupancy grid map which discretizes the physical area close and around the robot using variable resolution and holds the robot's belief about the existence or not of obstacles. The occupancy values in the map are updated using real-time sensory information from ultrasonic or any other range sensors. Translation and rotation transformations of the map are implemented, in order to maintain the correct placement of obstacles relative to the robot consistently with its movements.

Given a desired target position and orientation, an A* heuristic search algorithm in the three-dimensional space (coordinates on the field and orientation) derives an optimal path taking into account the omni-directional locomotion abilities of the robot (forward, side, rotating steps and combinations). The derived path provides the motion controller with the waypoints needed to set appropriately the velocities of the robot to move along the path.

1.1 Thesis Contribution

This thesis presents a method to create a fully parametrizable probabilistic occupancy grid map, that holds information about the existence or not of obstacles around the robot. The methodology used to update and transform the map consistently to the robot's movement, is also explained in details, and is platform independent.

Furthermore, this thesis deals with the implementation of an A* heuristic search algorithm in the three-dimensional space. The classic A* algorithm as applied on a graph to extract an optimal path, is converted, in order to take into account, besides the coordinates, the orientation of every node it traverses. The paths generated by this algorithm, are provided to a motion controller to produce velocity commands that move the robot to along the path to a desired target.

1.2 Thesis Outline

The main contribution of this thesis is the presentation of the obstacle avoidance and path planning module, its implementation and the benefit of the methodology for robot navigation.

In chapter 2 some necessary background information on the RoboCup competition are discussed, focusing on the Standard Platform League. The RoboCup team Kouretes is presented. Additionally, it briefly describes the robot platform Kouretes them uses in SPL, Aldebaran Nao humanoid robot. Furthermore, the occupancy grid and A* algorithm theory used, are presented.

In chapter 3, the issue is the need of an occupancy grid, to hold information about obstacles existence and a path planning algorithm, to provide path steps that guide the robot to the desired destination.

In chapter 4, the core ideas of the approach are represented. Details are provided about the building of the occupancy map and its transformations according to robots moves, in order to always keep the newest and correct information. In addition, the A* algorithm implemented in this thesis is presented.

Chapter 5, provides implementation details pertaining to the Monas architecture and the Nao robot. Moreover, here are presented problems and assumptions made during the implementation.

In Chapter 6, a discussion on the results is taking place by providing several experiments in order to evaluate our work.

In Chapter 7, the different approaches to the subject of obstacle avoidance and path planning implemented by other RoboCup teams are presented and they are compared to this thesis approach.

Future work and proposals on extending and improving our framework are the subject of the chapter 8.

Chapter 2

Background

2.1 RoboCup Competition

RoboCup is an international competition where robotics researchers from around the world come together in a face-to-face game of football. Each game poses a challenge not only in the football skills of the team but mainly in broad artificial intelligence and robotics research areas including real-time sensor fusion, reactive behavior, strategy acquisition, learning, real-time planning, multi-agent collaboration, context recognition, vision, strategic decision-making, motor control, intelligent robot control, design principles of autonomous agents and the ultimate goal to advance the state-of-the-art in the area.

RoboCup's history is not that long; the idea of robots playing soccer game was first mentioned by Professor Alan Mackworth in 1992 but it was not until 1993 that Hiroaki Kitano [5] proposed the formation of the RoboCup Federation.

The proposed goal stated the following

> *"By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup."*
> *(http://www.robocup.org/about-robocup/objective/)*

This quote has inspired a great number of researchers resulting in more than 70 teams competing and collaborating to reach this goal in many different leagues.

2.1.1 RoboCup Leagues

The contest currently has four major competition domains, each with a number of leagues and subleagues. Besides RoboCup Soccer, which we will subsequently analyze more, RoboCup also includes competitions in search-and-rescue missions (RoboCup Rescue), everyday life assisting applications (RoboCup@Home), and age-restricted leagues in soccer, dance, rescue and CoSpace (RoboCup Junior).

RoboCup Soccer League

RoboCup chose to use soccer as a central topic of research, aiming at innovations to be applied for socially significant problems and industries. This choice also serves the goal of developing a team of fully autonomous humanoid robots that can compete against the human world champion team in soccer by 2050 and win. The main focus of the RoboCup competitions is the game of football/soccer, where the research goals concern cooperative multi-robot and multi-agent systems in dynamic adversarial environments. All robots in this league are fully autonomous. The games also serve as a great opportunity to entertain and educate the public in science- and techology-related issues. RoboCup Soccer League separates in five subleagues.

Simulation League

The Simulation League (Figure 2.1) focus on artificial intelligence and team strategy without the necessity to maintain any robot hardware. There are 2 competitions: 2D and 3D, where autonomous agents play soccer in a virtual soccer stadium inside a computer.

Small Size League

The Small Size league or F180 league as it is otherwise known (Figure 2.2), takes place between two teams of five robots each. The robots must fit within an 180mm diameter circle and must be no higher than 15cm unless they use on-board vision. This league focuses on the problem of intelligent multi-robot/agent cooperation and control in a highly dynamic environment with a hybrid centralized/distributed system.

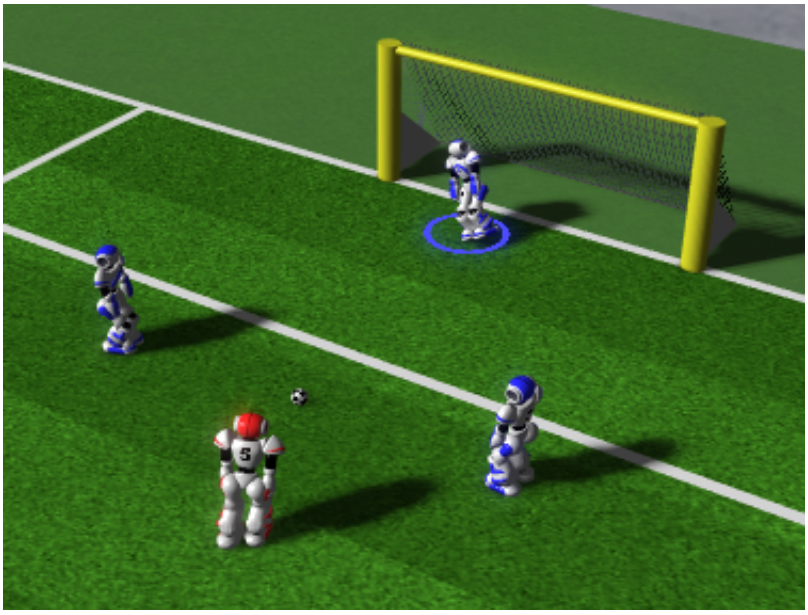


Figure 2.1: Simulation league, 3D field

Figure 2.2: Small size league

Middle Size League

Middle-sized robots of no more than 50 cm in diameter play soccer in teams of up to 6 robots with regular size FIFA soccer ball on a field similar to a scaled human soccer field (Figure 2.3). All sensors are on-board. Robots can use wireless networking to communicate. The research focus is on full autonomy and cooperation at plan and perception levels.

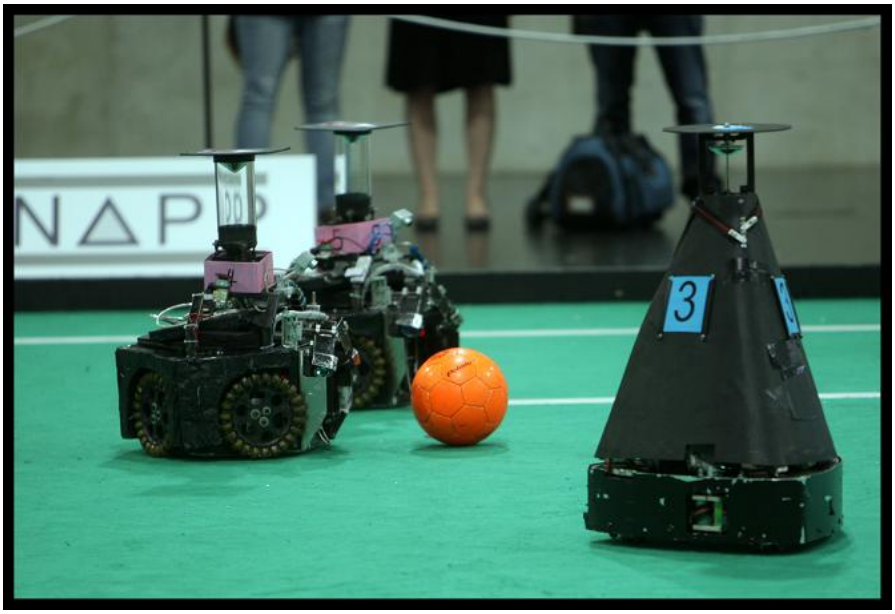


Figure 2.3: middle size league

Humanoid League

In the Humanoid League (Figure 2.4), autonomous robots with a human-like body plan and human-like senses play soccer against each other. Dynamic walking, running, and kicking the ball while maintaining balance, visual perception of the ball, other players, and the field, self-localization and team play are among the many research issues investigated in the league. The league is divided in 3 subleagues, according to robot sizes: Teen Size, Kid Size and Adult Size.(http://www.robocup.org/robocup-soccer/humanoid/)

Figure 2.4: humanoid league

The Standard Platform League

In this league all teams use identical robots, the Aldebaran Nao humanoid robots, and no modifications or additions to the robot hardware are allowed, including off-board sensing systems. Therefore the teams concentrate only on software development, focusing on more efficient algorithms and techniques for visual perception, active localization, omni-directional motion, skill learning and coordination strategies, while still using state-of-the-art robots. The robots operate fully autonomously; there is no external control neither by humans nor by computers.

The SPL robot soccer games, until March 2012, were played between teams of four robots on a 4m x 6m green field, marked with white lines.The official ball is a 65mm in diameter orange street hockey ball, and the two goals have different colors, the one is yellow and the other sky-blue. Robots' teams are specified by waistbands, colored blue and pink. The red team defends the yellow goal, while the blue team defends the sky-blue goal.

A game consists of three parts, two 10-minute halves and a half-time break between halves, during which teams change the colors and the defended goal. The game starts with robots at predefined positions, as shown at Figure 2.4 and progresses much like a

soccer game but with modified rules.

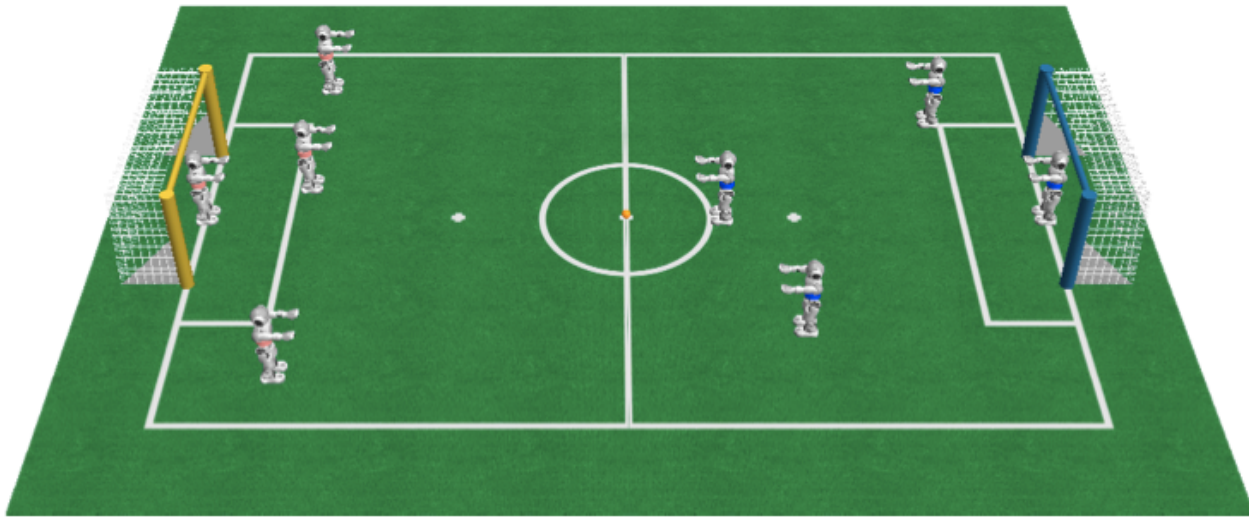


Figure 2.5: Manual settup for kick-off. Blue kick-off

2.2 Kouretes Team

Kouretes is a robotic team based at the Intelligent Systems Laboratory of the Department of Electronic and Computer Engineering at the Technical University of Crete and it was the first Greek team participating in Robocup competitions, specializing in the Standard Platform League and the MSRS Simulation League.

The team was founded in February 2006 by Michail G. Lagoudakis, assistant professor at Department of Electronic and Computer Engineering.The name *Kouretes* refers to the five brothers, Epimedes, Paionaios, Iasios, Idas, and Hercules, that according to Ancient Greek mythology were guarding newborn god Zeus.

Since its foundation the team has participated in every RoboCup competition, at first with the Sony Aibo robots in the Four-Legged league. In 2007, the team began also working with the newly-released Microsoft Robotics Studio (MSRS). By 2008 the team has started developing code for the Aldebaran's Nao humanoid robot, the

new robotic platform on the Standard Platform League and continued competing in the simulations leagues using a combination of C++ and Ruby for the real robot, C# for the simulated robot in MSRS, and Java for the simulated robot in Webots. Since 2009 the team works exclusively on the Nao robots, implementing their own code, a fact that was aided by the completion of the work on the team's software architecture (Monas), by Alexandros Paraschos.

Kouretes have participated in many competitions, exhibitions, and affairs, the most significant being the MSRS Simulation Challenge at RoboCup 2007 in Atlanta in which the team was placed 2nd place worldwide, Robocup 2008, Suzhou, China where the team's efforts were rewarded in the best possible way: 3rd place in Nao league and 1st place in the MSRS simulation, and the most recent in Robocup 2011, Istanbul, Turkey, where the team placed second in Standard Platform League'As Open Challenge Competition

Figure 2.6: Kouretes team 2011 formation. From left to right in the front row are Astero-Dimitra Tzanetatou, Iris Kyranou, Angeliki Topalidou- Kyniazopoulou, and in the back row standing up Emmanouel Orfanoudakis, Eleutherios Chatzilaris, Nikolaos Spanoudakis, Michael Lagoudakis and Evangelos Vazaios .

More information and news of the team but also of its members can be found at

http://www.kouretes.gr.

2.3 Aldebaran's Nao Humanoid Robot

Nao (Figure 2.7) is an autonomous, programmable, medium-sized humanoid robot developed by French company Aldebaran. It was first presented in 2008, and Aldebaran Robotics has announced its commercial release of Nao robot v4 in 2012 promoting it as an educational robotic platform and a family entertainment robot affordable to most budgets.

It was first introduced to the RoboCup world in 2008, in its initial limited edition (RoboCup edition v2) and it dramatically changed the Standard platform league scenery by completely replacing the Sony's four-legged robot Aibo.

2.3.1 Nao Hardware

Nao , in its current version, is a 57-cm tall, biped robot, that weights 5,2 kg and its RoboCup edition has 21 degrees of freedom, two on its head, four on each arm, one on the pelvis and five on each leg. The Nao robot v3 that was used in this thesis carries a full computer on board with an AMD GEODE processor at 500 MHz, 256 MB SDRAM, and 2 GB flash memory running an Embedded Linux distribution. It is powered by a Lithium-Ion battery which provides about 60 to 90 minutes. NAO robot communicates with remote computers via an IEEE 802.11g wireless or a wired ethernet link.

The Nao robot is equipped with a variety of sensors and actuators.Two 920p cameras are located on its head which provide video capture at maximum 640x480 pixels resolution at 30 frames per second. The two cameras, the first located on Nao's forehead, and the second one at mouth level, cannot operate simultaneously, so each time the robot choses which one provides better cover for the specified task. Two microphones are also mounted on the robot's ears, allowing stereo audio perception, and can be used for sound communication.

On the main body of the robot are located two pairs of ultrasound emitters and receivers, used to provide information about obstacles close to the robot (see section 5.1).

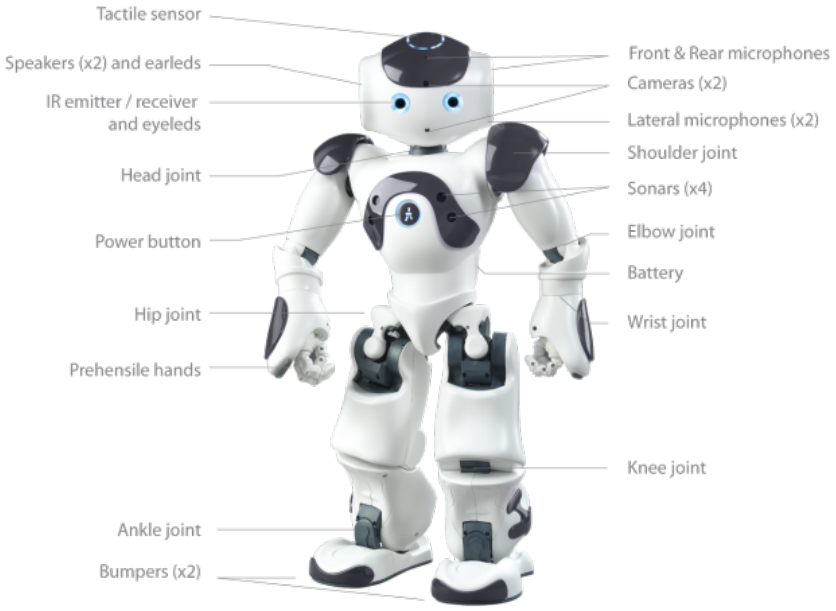


Figure 2.7: Nao robot, joints and sensors

The inertial unit is also located in the robot's torso with its own processor. The central unit is made by two 1-axis gyrometers and one 3-axis accelerometer, that provide real-time information about the torso speed and attitude (Yaw, Pitch, Roll).

Regarding contact sensors, there is a chest button, which is used to open and control the robot, head and hand tactile sensors. A pair of bumpers is located at the tip of each foot, providing information on collisions of the feet with obstacles, and an array of force sensitive resistors is placed on each foot, providing feedback on the forces applied to the feet. Finally the robot has a number of LEDs on its body, usually used as indications for their current state and two speakers, that can be used for robot-human or robot-robot communication.

For its movement, the robot has a variety of motor types with different speed reduction ratios, that each time are used to manipulate the head, arm and leg joints, to provide the appropriate posture and movement to the body.

2.3.2 NaoQi

Nao can be programmed and controlled in many high-level languages, such as C, C++, and Python, using Linux, Mac or Windows. Aldebaran provides a programming framework, NaoQi, that serves as a software middleware. It is based on a client-server architecture, where NaoQi itself acts as a server. This framework allows homogeneous communication between different modules (motion, audio, video), homogeneous programming and homogeneous information sharing.

The NaoQi executable which runs on the robot is a broker. The autoload.ini preference file contains the libraries that are going to be loaded in startup, and each library instantiates one or more module classes that use the broker to advertise their methods. Loading modules forms a tree of methods attached to modules, and modules attached to a broker, with the latter providing lookup services so that any module in the tree or across the network can find any method that has been advertised.

This way NaoQi allows developers use the predefined Aldebaran modules, but also create new modules with specific functionalities. These modules can run either remotely, as executables outside the robot or locally, as libraries uploaded on the robot.

2.3.3 General Overview

In overall, Aldebaran Robotics designed and assembled a low-cost robot, which can be a great, not only scientific, but also entertainment platform, easily programmable, focusing on the wide audience of robotics' researchers and fans. The development of humanoid robots is a tough procedure that only few universities and companies have undergone, and even fewer are located in Europe.

To be fair, one has to admit that the first version of Nao was not functional to the level Robocup teams would like it to be. Nevertheless, most teams were able to present some basic soccer behaviors, confirming that even under these limitations and the minimum available time for development, people involved in Robocup gave their best shot.

Nowadays, in February 2012, code development in Nao V3+, has become a less tedious work, and we are all looking forward to take the best out of this platform, in order to make our contribution in the RoboCup community.

2.4 Occupancy Grid Mapping

The need of a map originates in the robot navigation problem. For any mobile device, the ability to navigate in its environment is the primary goal in order to achieve any task. A successful navigation consists on avoiding collisions, staying within safe and predefined margins and reaching the goal in a reasonable amount of time. In order to succeed in the navigation task, the agent must deal with problems like mapping, localization, path planning and motion control.

Occupancy grid maps (Moravec kai Elfes) address the problem of generating consistent maps from noisy and uncertain measurement data, under the assumption that the robot pose is known [6].

The basic idea of the occupancy grids is the partition of the world space into a finite number of random variables. Each variable corresponds in a specific area, which is called 'cell'. Each cell holds a probabilistic value that corresponds to the occupancy of the location it covers. The value each cell holds, varies from 0 for obstacle-free cells, to 1.0 for fully occupied cells. The unobserved cells hold the value of 0.5.

This structure is useful for combining different sensor scans, and even different sensor modalities (sonar, laser, IR, bumper). The basic model of a single sonar beam has a field of view specified by β, the half angle representing the width of the cone, and R, the maximum range it can detect. This field of view is projected on an occupancy map grid. As shown in figure 2.8, the field of view can be divided into four regions [7]:

Region I , where the affected cells are probably occupied

Region II , where the affected cells are probably empty

Region III , where the condition of the affected cells is unknown

Region IV , where the cells are outside the sonar field of view

The sensor returns a measurement distance r of the object detected. All cells falling within this distance and inside the sonar beam area, are considered belonging at Region I. The area with distance $d < r$ is an empty area (Region II), or else there would be a sonar measurement with $r < d$. So for a measurement r, the cells starting from the one closer to the robot, to the furthest before the sonar measurement, are considered

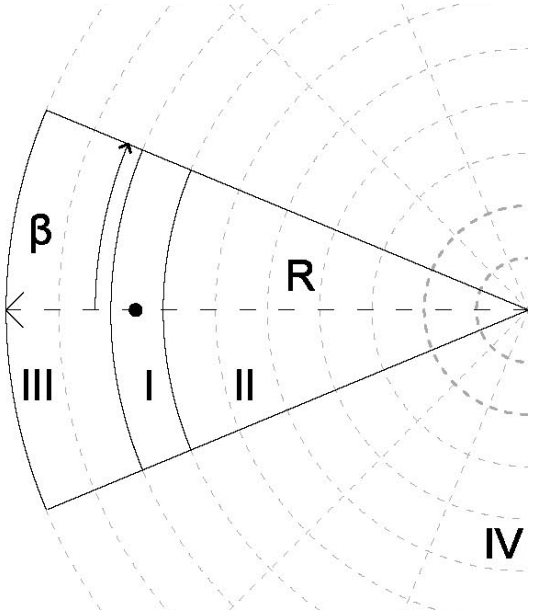


Figure 2.8: Sonar beam regions

empty in the whole sonar beam area. For the cells falling in Region III, no information is obtained by this measurement.

2.5 Path Planning

Besides localization, a map is used to plan obstacle-free paths. *Path planning* deals with the computation of collision-free paths in a robot's environment. Given an initial coordinate location and a target coordinate location, path planning algorithms compute a set of waypoints that represent the best path from where the robot is currently located to where it needs to be. These waypoints are translated to motion commands that move the robot accordingly. In this thesis an A* algorithm 2.5.1 is implemented for this purpose.

2.5.1 A* algorithm

A* Algorithm 2.9 is a computer search algorithm, usually used in pathfinding for holonomic robots (robots that can instantaneously move in any direction), due to its flexi-

bility (it can be used in a wide range of contexts and handle well varying terrain costs) and its performance, that appears to be better than most graph searching algorithms.

The A* search algorithm assumes a metric map and handles the occupancy map as a graph, where the location of each node is known in absolute coordinates, and the graph edges represent whether it is possible to traverse between those nodes. The algorithm produces an optimal path by starting from the initial node and then working through the graph to the goal node adding up the cost of traversing each cell. When looking at each node in the graph, A* uses a heuristic, which is the estimated cost of getting from node n to a goal with the least cost $h(n)$, and it stores the actual cost of reaching node n from the initial state $g(n)$. A* algorithm then combines these two values to estimate $f(n)$:

$$f(n) = g(n) + h(n)$$

As the algorithm runs, the f(n) value of a node tells us how expensive we think it will be to reach our goal passing from that node. Thus, since the goal is to minimize cost, the A* search strategy suggests firstly to try the node with the lowest value of $f(n)$. Provided that the heuristic function $h(n)$ is admissible, A* search is both complete and optimal [8].

A*. Informed search using a heuristic function and memory to avoid re-expansion.

Input: node $node$, list of nodes $fringe$
Output: $success$ or $failure$

if $node.isGoal = true$ **then**
  return $success(node)$
end if
$fringe \leftarrow insertAll(expand(node), fringe)$
if $fringe.empty = true$ **then**
  return $failure$
end if
$best \leftarrow$ the node in $fringe$ with the lowest f-value
$fringe \leftarrow delete(best, fringe)$
return A*$(best, fringe)$

Figure 2.9: The A* algorithm (initial call with A*$(root, [\,])$).

Chapter 3

Problem Statement

3.1 Obstacle Avoidance

For an autonomous agent in a real field, the need to avoid collision with other objects is constant. An obstacle is everything that can be found in the close environment of the robot, and prevents the robot from achieving its goal. These objects can be anything, from known and expected objects, like other robots or goal posts in a soccer field, to unknown such as humans or walls. Moreover, these obstacles will not always be static, but they will also be moving in the same field, and sometimes even coming towards the robot. An impact with an obstacle can have unwanted results, as is the fall of the agent, which makes it more vulnerable to hardware damage and loss of its localization. Eventually, it ends in loss of time, which is crucial when competing in a soccer field, while the robot tries to find again its position and goal.

3.1.1 Obstacle Avoidance in Standard Platform League

In the SPL, eight robots are competing in a 4 x 6 m field. With the exception of the goalies, the aim for the rest of the robots is the fast approach of the ball and the attempt to score a goal. Even if each team is playing with a defender, who is not aggressively approaching the ball, there are still four robots remaining to fight for the ball. Eventually, two or more robots will be found in a very close distance from each other, which ends in collisions. The SPL pushing rules [9] are similar to a real soccer game foul rules and every time a robot pushes another one hardly enough to knock it

off-balance the offending robot gets penalized by staying out of game for 30 seconds. Pushing rules also apply on robots of the same team, and after the count of four pushes, the last robot is removed from the field for the rest of the half. After that, a robot is removed in every two pushes. These rules make it clear that an obstacle avoidance algorithm is essential for a team, in order to succeed in RoboCup games.

3.2 Path Planning

A simple obstacle avoidance algorithm would only take in account the direct front obstacles each time, and simply move right or left in order to avoid whatever is in front of the robot. This reactive approach is fast, but does not take under consideration the rest of the map or even the goal. Thus, the robot tries to walk past the obstacle, usually walking sideways. The problem is that the opponent robot, most commonly executes a similar walk in order to avoid the first robot, ending in both robots walking sideways face to face, and losing the ball.

The desired approach would be the one that not only avoids the direct obstacles but also calculates a path that will guide the robot to the ball fast and without collisions. That is succeeded with a path planning algorithm, that uses a map of the robots' surrounding objects and decides an optimal collision-free path. Moreover, although the robot's walk is omnidirectional (meaning it can walk in every possible way) the front walk is the fastest one. Therefore a path that takes advantage of the free spaces in a way that the robot walks straight front is preferable. If the robot knows from the beginning that an obstacle is in a specific position, it can decide a path, that exploits the faster straight walk earlier in the path, therefore approaching the ball faster that the opponent.

Additionally, approaching the ball any possible way is not always the best choice. If the robot knows that it is facing its own goal and the ball is in a distance front of the robot, then just going straight front finally brings the robot in a position that is not useful to kick front, since that would mean a point in favor of the opponent. In this situation, the preferable walk is to approach the ball with a longer path, but guide the robot to face the opponent goal. This way, the robot has a better position for a front kick, and simultaneously is defending its goal, by standing between the ball and the opponent's winning kick.

This is succeeded by implementing a path planning algorithm that also takes under consideration the desired robot's orientation in reference to the opponent's goal, apart from the goal's coordinates.

Chapter 4

Obstacle Avoidance Module

The methodology suggested first builds a local obstacle occupancy map, that is updated constantly with real-time sonar information. The robot is considered to be constantly in the center of the occupancy map, and facing front. Therefore, in order to always keep updated and correct information about the position of the obstacles, translation and rotation methods are implemented to change the map accordingly to the robot's movement. Finally, after obtaining the current, updated and transformed map, an A* algorithm decides the best path towards the ball. Information about this methodology is presented subsequently.

4.1 Occupancy grid

For collision avoidance, the first step is to detect the obstacles and keep information about their position. In order to accomplish this, each robot builds and maintains a local obstacle map of its surrounding.

The robot's world is discretized and stored in a n x k polar grid covering the 360° area around the robot. The robot is always located in the center of the grid. The areas that are defined by the concentric circles are referred in the thesis as *rings* and the slices in which the diameters cut the circle are referred as *sectors*

This polar grid is updated using distance readings from the ultrasonic sensors, which are not a reliable source of information about obstacles. The possibility of false positives is high, the sensor's coverage increases, especially closer to the beam range,

and the low update rate can end up in a very inaccurate and uncertain model about the robot's world. Thus, the map update cannot depend only on instant readings, forgetting everything that was previously sensed. On the other hand, holding every information that was read is not a good idea, considering that sensor readings can be so noisy, and the environment is dynamic.

The solution chosen was to fuse new readings with the old ones, and apply an aging algorithm to bring the unobserved cells' occupancy values closer to the one suggesting little knowledge about the obstacles or free spaces existing in the cells' cover area (value of no knowledge 0.5, see 2.4). The way this is implemented is explained in the following section.

4.2 Map Update

Every cell in the occupancy grid, is initialized in the unknown state and holds the value 0.5. The first step in updating the map is to determine the area the sonar reading covers. The sonars on the robot's torso are placed on a specific and known angle θ.

Given a reading r from a sonar, and knowing the sonar's position, the four regions shown in figure 2.8 can be determined. Any reading $r > R$, where R is the sonar beam distance range, is not taken under consideration.

The polar values of the reading (r, θ) are mapped in a specific cell on the polar grid. This cell falls in **Region I**. All the neighbor cells with same distance (r) from the robot, and in an angular distance that falls into the effective cone of the sonar beam, are also considered to be in **Region I**.

Now, given the reading r we know that the area with distance $d < r$ is an empty area (Region II), or else there would be a sonar measurement with $r < d$. Therefore for a measurement r, the cells starting from the one closer to the robot, until the last before the sonar measurement, are considered empty in the whole sonar beam area.

The map is actually updated by increasing (by a factor $IncrFactor$, currently 1.6) the present occupancy value each cell holds every time the cell falls in Region I and decreasing (by a factor $DecrFactor$, currently 0.8) the occupancy value when in **Region II**, as shown in the following equations

$$Grid[ring][sector] = Grid[ring][sector] * IncrFactor$$

$$Grid[ring][sector] = Grid[ring][sector] * DecrFactor$$

The maximum value a cell can hold is 1.0 and it means the cell is considered occupied by an object at the corresponding position in the field, and the minimum value is 0.08, which responds to the confidence that the cell is empty.

For the cells after the last reading, no knowledge is obtained and they fall in **Region III**.

One problem arising is that the robot is not a one-dimensional object, but has some volume that should be taken under consideration. If not, the calculated paths will pass through areas that a robot does not actually fit, ending to collisions. The problem becomes more obvious considering the total area covered by three cells close to the robot, as opposed to three cell in distance R.

In order to solve this problem, each obstacle measurement is expanded, so as the final obstacle area will have a radius equal to the robot's body radius. Thus ensuring, that the areas marked as empty, and so free to walk, are safe for the robot to pass through.

Over time, obstacle cells might leave the sensors' scope and fall in **Region IV**. Moreover, odometry accumulates a significant error, ending in inconsistent information. To avoid keeping any old and inconsistent information, all cells are updated accordingly to an aging model, that in regular time intervals (currently 2 s), brings a cell's value closer to the unknown state (value 0.5), by a factor $AgeFactor$ (currently, 0.8).

$$Grid[ring][sector] = (Grid[ring][sector] - 0.5) * AgeFactor + 0.5$$

4.3 Map Transformations

The grid is local and moves accordingly with the robot. Locomotion of the robot triggers appropriate transformations on the polar grid. In order to always maintain the correct position of the mapped obstacles, according to the robot's position and orientation each time, the map supports translation and rotation algorithms.

4.3.1 Move Robot

The robot's coordinates and orientation are given by a global system and are updated by the robot's odometry measurements. Robot's x, y and $angle$ positions are initialized with the first odometry values read, where x is the distance covered on x-axis, y is the distance covered on y-axis, and $angle$ is the angular distance in robot's orientation. After initialization, every time the ObstacleAvoidance module is called, the new odometry measures are read, and the difference between them, and the old ones is calculated. These measurements are then translated into a local coordinates system, used to decide the actual front and side distance, and angle covered by the robot.

Figure 4.1 shows two different positions for the robot. The distance covered on x-axis is $diffX = x_1 - x_0$, on y-axis is $diffY = y_1 - y_0$ and the angular distance is $diffA = \theta_1 - \theta_0$. These differences correspond in distance covered in the global coordinates system. Thus, a transformation to the robot's local coordinates system is needed. First the distance covered by the robot and its rotation are calculated:

$$diffD = sqrt(diffX^2 + diffY^2)$$

$$diffT = atan(diffY, diffY)$$

And then the distance actually covered by the robot is:

$$diffX = diffD * cos(diffT - \theta_0)$$

$$diffY = diffD * sin(diffT - \theta_0)$$

Now these values ($diffX$, $diffY$) with the angle difference (diffA) are used to transform the grid and move the obstacles in the correct cells.

4.3.2 Grid Translation

The translation method uses the polar coordinates of each cell to calculate the cartesian coordinates. The cartesian coordinates of each cell are pre-calculated in order to minimize the computational complexity. Then it simply adds to this cell's coordinates the distance covered on x, and y-axis, and transforms the new cartesian coordinates back to polar. These are now the coordinates of the new cell that is going to be replaced by the old one. If more than one translated cells fall into the new cell's area, then the

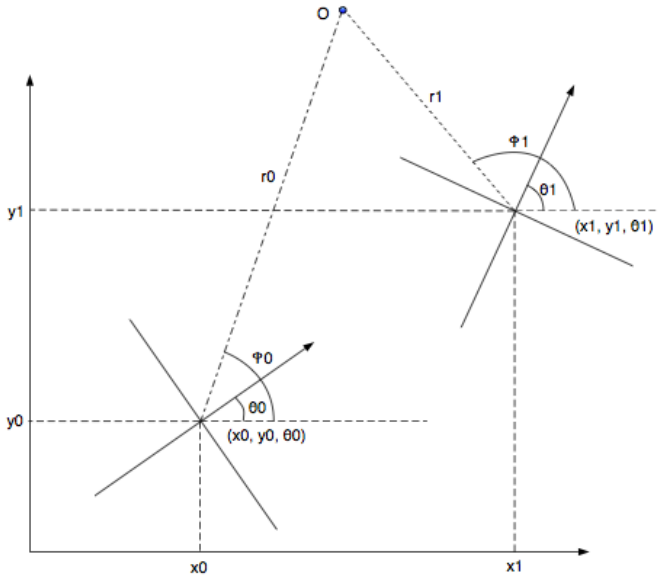
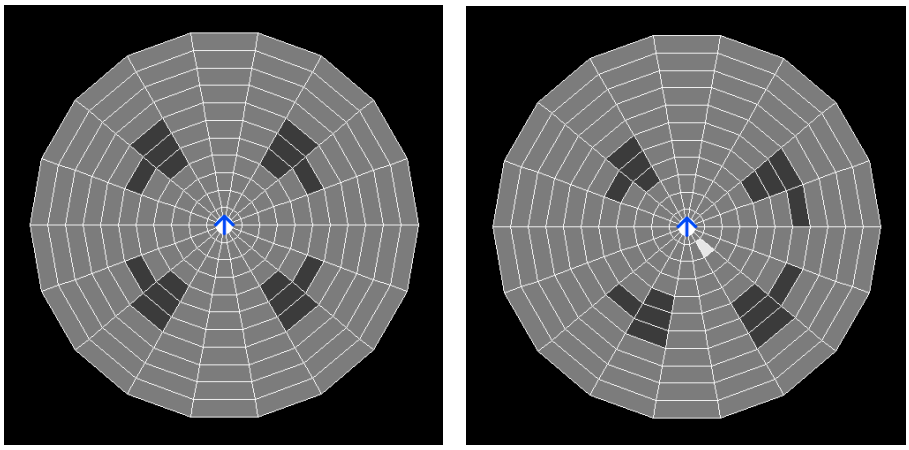


Figure 4.1: Two positions of the robot [2]

average occupancy value is stored in the new cell. An example of a grid moving 0.1m in x-axis and 0.1m in y-axis, is shown in figure 4.2

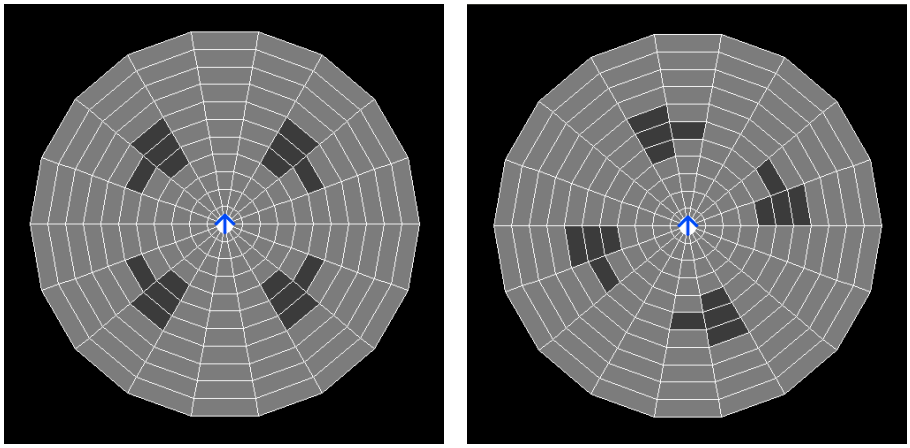


(a) Grid with four obstacles before move (b) Grid after move

Figure 4.2: Grid translation: Grid before (a) and after (b) move. Robot moves 0.1m in x and 0.1m in y

4.3.3 Grid Rotation

The rotation is less computationally complex than the translation of the grid. The polar map offers a straight forward approach to the rotation calculations. Each cell s rotated for n sectors, is simply replaced by the $s + n$ cell, wrapped around N, where N is the total number of grid cells. An example is shown in figure 4.3



(a) Grid with four obstacles (b) Grid after rotate

Figure 4.3: Grid rotation: Grid before (a) and after (b) rotate. Robot rotates for 40°

4.4 Path Planning

The map holds each moment the current perception of the robot's world. For the calculation of the obstacle-free path that will guide the robot to a desired destination, an A* algorithm is used (more information about A*algorithm at 2.5.1).

The occupancy grid is easily conceived as a graph, considering each cell as a graph node and the connectivity between neighbor cells as the graph edge. In the polar grid described, each node has eight neighbors, except the first one, that simulates the robot itself and has N neighbors (N, being the number of sectors of the polar grid), and those that fall on the first and the last ring, that only have five neighbors.

The A* algorithm implemented serves as a level between behavior and motion control. It takes as input the desired destination and orientation, and the output is the

waypoints translated to actual walk commands. Each one of the cells, represents the possible positions in which the robot can end up after the move and is translated in walk commands by the motion controller. Assuming that the robot faces the way the blue arrow points (see figure 5.2 the sector following the arrow's direction is the front move. The contradictory sector is the back move, and the ones lying on the x-axis are the left and right side moves.

In the simple A* algorithm (see section 2.5.1), all the node's neighbors are expanded. That would mean that the first node would expand N neighbors, including those being on the back of the robot. The motion controller handles these cells as backwards movements. The problem lies on the fact that, although the robot's walk is omnidirectional (meaning it can walk in every possible way) the front walk is the fastest one. Therefore, the calculated path should take advantage of that fact, choosing paths and take advantage of the free spaces in a way that the robot walks straight front.

That is succeeded by not allowing the A* algorithm expand node's neighbors that end in backward movements. The algorithm expands the three front neighbors and the two side ones, because they are translated in front and side movements, and ignores the three neighbors that translate in back movements (back, right back, left back), as unwanted move situations in a soccer field.

Another problem, mentioned in section 3, is that approaching the ball any possible way is not always the best choice for the robot. If the robot knows that it is facing its own goal and the ball is in a distance in front of the robot, then just going straight front finally brings the robot in a position that is not useful to kick front, since that would mean a point in favor of the opponent. In this situation, the preferable walk is to approach the ball with a longer path, but guide the robot to face the opponent goal. This way, the robot has a better position for a front kick, and simultaneously is defending its goal, by standing between the ball and the opponent's winning kick.

This suggests that besides the coordinates, the goal should also be described using its orientation. The goal node (r, s), where r is the ring and s is the sector with front orientation is different than the same node with back orientation. The possible orientations of a cell are discretized into eight, differing by 45°. These are as shown in figure 4.4.

Arriving to a front neighbor looking backwards is not a realistic position for a robot, because the small distance between the cells (currently 10cm) is not enough

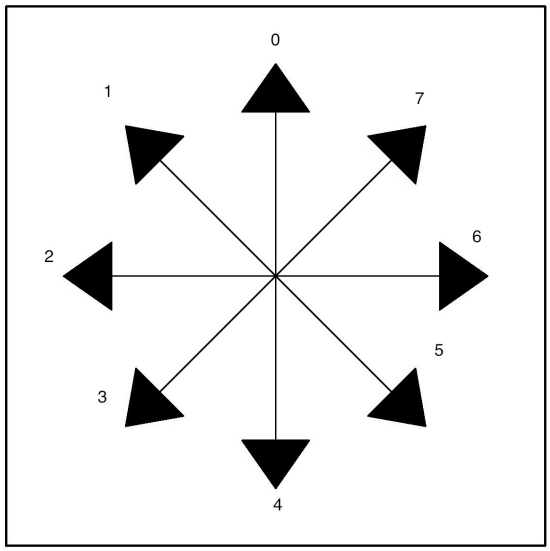


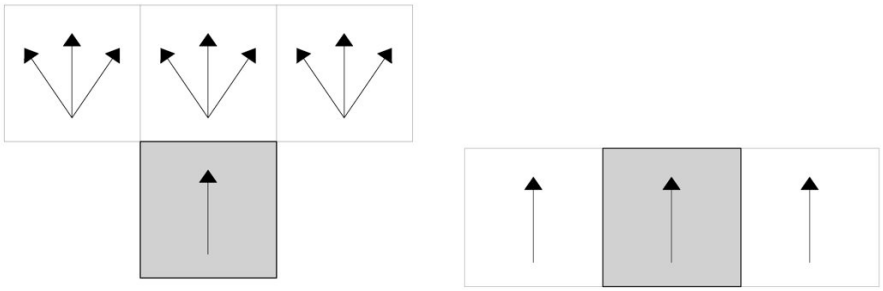
Figure 4.4: The eight possible node orientations

for the robot to actually carry out the move. Taking this under consideration only the neighbors that end up in a position the robot have actual time to arrive in such a small movement are kept. Finally, the nodes chosen were the three front neighbors with orientations responding to direct front, right front and left front orientations in accordance to the actual robot's orientation, and two side neighbors (left and right) with the same orientation as the robot (see figure 4.5).

The only exception is the first node, representing the robot. The first node's neighbors are the three front neighbors, that are actually translated by the motion controller to front move (the middle neighbor), and rotational in place moves to the right and left, and the two nodes that lie on the x-axis, and are translated to the side steps.

The cost moving from a node to its neighbor, $g(n)$, is calculated differently, according to the particularities of each move. Besides the euclidean distance between the current node and its neighbor, other metrics to take into account are the angular distance between the two nodes, the side step cost, and the cell's occupancy value.

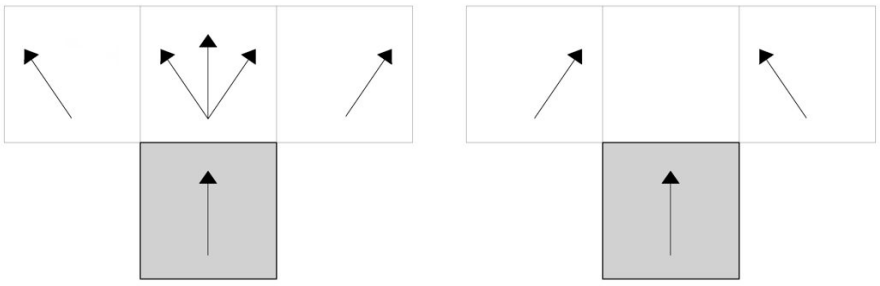
For the three front neighbors $g(n)$ is the sum of the euclidean distance to reach the new nodes, plus the angle difference between the current and the neighbor node. Moreover, special attention is given to the orientation of the current node and the neighbor's



(a) The three front neighbors. (b) The side neighbors

Figure 4.5: The neighbors expanding during an A* algorithm run for every node besides the first one, and their orientations. The grey node is the parent node.

orientation. The figure that follows 4.6(a) shows the neighbors that the robot can reach with a straight and rotational move only. This move takes advantage of the fast front walk of the robot in contrast to the two positions shown in figure 4.6(b). These are translated by the motion controller as semi side steps with simultaneous front movement, and are actually slower in execution that the first steps, adding extra cost in the $g(n)$ function.

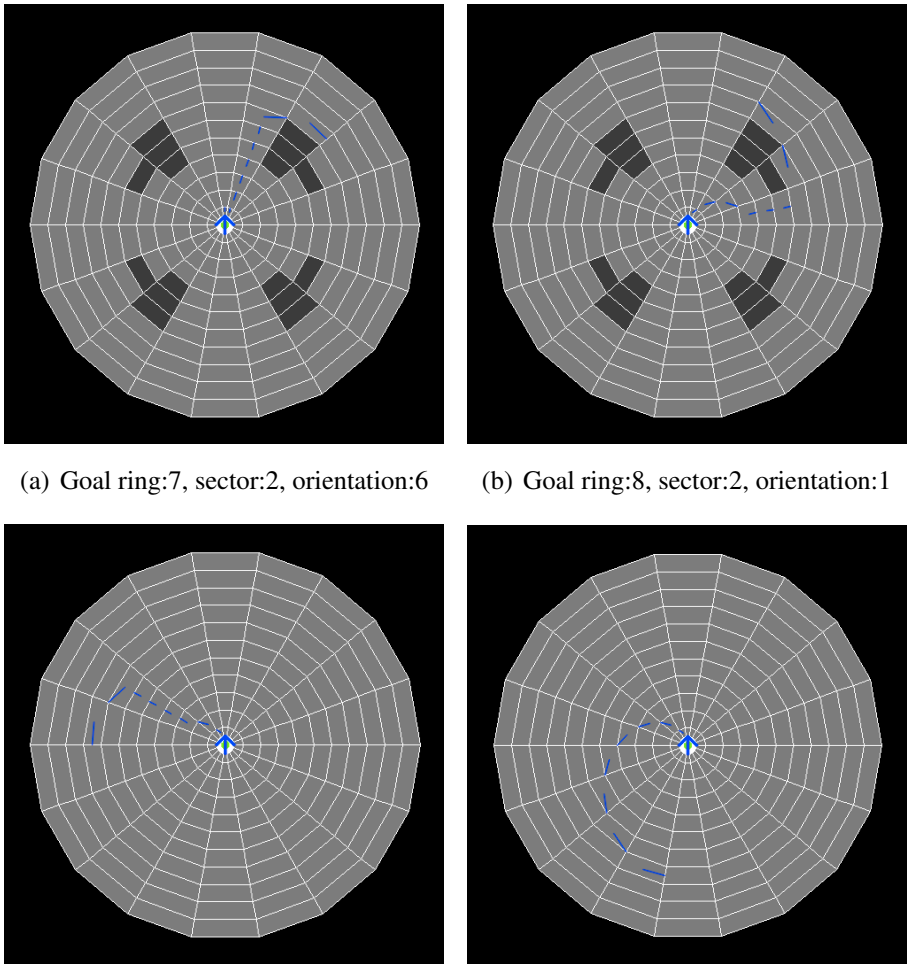


(a) Movement with no extra side cost. (b) Movement with side cost.

Figure 4.6: Picture 4.6(a) presents the neighbors orientation that does not add extra cost in the $g(n)$ function, in contrast with the orientations shown in picture 4.6(a)

Euclidean distance is also used as the A* heuristic, which is admissible (since it does not overestimate the distance to the goal), ensuring an optimal solution.

Some examples of the A* algorithm results, are presented in the following image 4.7



(a) Goal ring:7, sector:2, orientation:6

(b) Goal ring:8, sector:2, orientation:1

(c) Goal ring:8, sector:8, orientation:2

(d) Goal ring:8, sector:12, orientation:2

Figure 4.7: A* algorithm examples

Chapter 5

From Theory To Practice

This section describes all the optimizations , implementation decisions and details of Obstacle Avoidance Module. The primary goal is this module to be used on a robot for the collision avoidance and path panning, specifically for the Standard Platform League competition.

5.1 Sonars

The word “sonar” is originally an abbreviation for “SOund, NAvigation and Ranging”, and refers to the sound detection technology used primarily for tracking enemy submarines during World War II. There are two types of technology that share the name ”sonar”: passive sonar, that is only listening for the sound made by vessels, and active sonar, which creates a pulse of sound and listens for reflections (echos) of the pulse. The acoustic frequencies used in sonar systems vary from very low (infrasonic) to extremely high (ultrasonic).

In the simplest terms, an electrical impulse, converted into a sound wave is emitted into the air by the transmitter. When this wave strikes an object, it rebounds, and this echo returns to the receiver. Since the speed of sound in the air is known, the object’s distance can be calculated by simply measuring the time lapse between the transmitted signal and the received echo.

The transducer concentrates the sound into a beam. When a pulse of sound is transmitted from the transducer, it covers a wider area the farther it travels, creating a

cone-shaped beam. The sound is stronger along the center line or axis of the cone and gradually diminishes as it moves away from the center.

5.1.1 Nao Sonars

The Nao robot is equipped with two ultrasonic sensors (active sonars), located on its torso (as shown at image 5.1), which allow it to estimate the distance to obstacles in its close environment. The sonars are actually two transmitters (emitting pulses of sounds) and two receivers (listening for the echoes of the emitted sounds), on the front of the torso, that generate sound waves at 40Khz frequency. The receiver's sensitivity is -86 dB, the resolution 1 cm, and the effective cone 60 °.

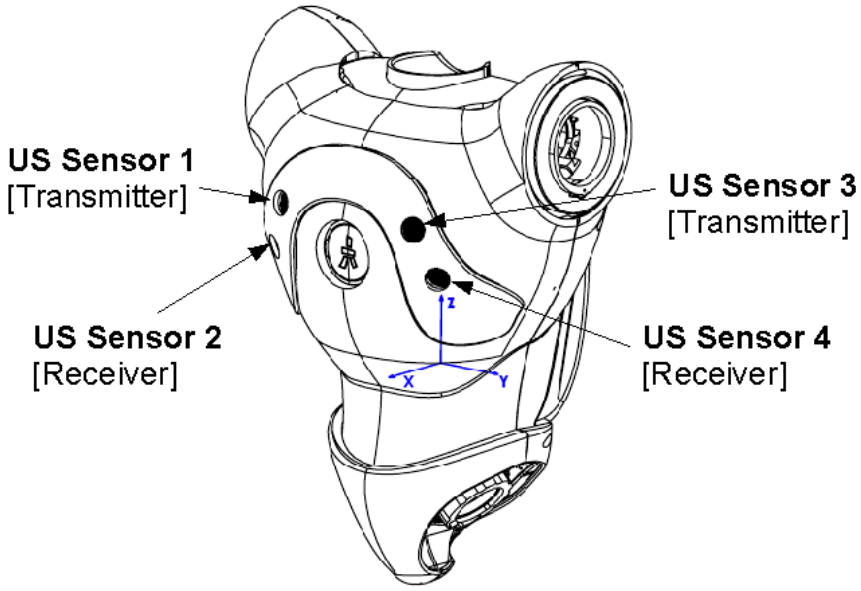


Figure 5.1: Sonars position on robot torso

Theoretically the sonars' detection range goes from 0.25m to 2.55m, but in reality no measurement less than 0.29m has been noted. The robot has no distance information about obstacles that exist closer than 0.29m. Each time the sonars give a value less than 0.29m we know the existence of an obstacle close to the robot, but cannot be sure about its distance from the robot.

The actual position of the sonars relative to the torso frame appear on the following table:

Sonar Name	X(m)	Y(m)	Z(m)	WX(rd) [deg*]	WY(rd) [deg*]	WZ(rd) [deg*]
US Sensor 1	0.0537	-0.0341	-0.0698	0.0	-0.1745 [-10.0]	0.3490 [-20.0]
US Sensor 1	0.0477	-0.0416	-0.0509	0.0	0.2618 [15.0]	-0.4363 [-25.0]

US Sensor 3 and US Sensor 4 are symmetrical, to US Sensor 1 and US Sensor 2 accordingly, with respect to sagittal plane of the robot.

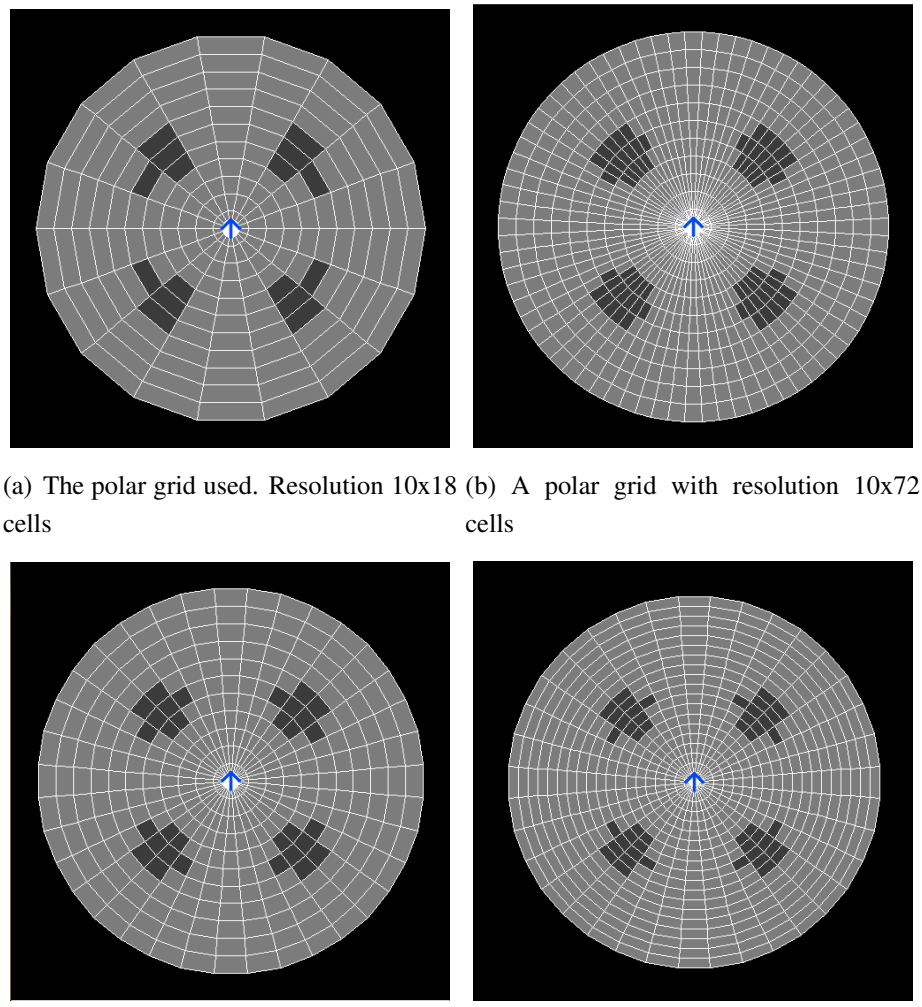
5.2 The 10x18 polar grid

Although the sonar's documentation states that it receives maximum distance measures of 2.55m, in reality it has good performance only in measurements less then 1m. Thus, the robot's world is discretized and stored in a n x k polar grid covering the 360° area of radius 1m around the robot. The map is fully parametrizable, and currently 10 rings and 18 sectors in each ring are used, which corresponds in 10cm resolution in distance from center, and 20° resolution in angle respectively. The robot is always located in the center of the grid. Figure 5.2 shows the described polar grid in the different resolutions.

Since the detected objects in the field most commonly are robots, each measurement from the sonars is expanded to the robot's radius.

5.3 Map transformations

If the distance covered by the robot is less than the grid's ring distance, the movement cannot be shown on the map. Similarly, if the angle difference is less than the sector angle distance, a rotation cannot be shown on the map. Therefore a translation in the grid will happen after the robot covers 10cm distance movement, and a rotation will occur after a 20° angle rotation movement of the robot. For smaller movement, no change appears on the grid.



(a) The polar grid used. Resolution 10x18 cells

(b) A polar grid with resolution 10x72 cells

(c) A polar grid with resolution 10x32 cells

(d) A polar grid with resolution 18x32 cells

Figure 5.2: Four different resolutions of the polar grid. The four obstacles are in distance 0.55m, on angles 45°, 135°, 225°, 270°.

5.4 A* implementation

For the A* algorithm described, a structure is needed to hold the nodes that where generated and wait to be visited. This was implemented with a list that inserts the nodes sorted by their $f(n)$ values. After the first node, with the smallest $f(n)$ value is removed from this list it is added to another list, holding the nodes that were visited.

Front walk (m)	Side walk (m)	Rotation (deg)
0.151	0.0826	53.3

Table 5.1: Average distances covered in front, side and rotation walks for one minute run.

In order to use less memory resources, this was not implemented as a list, but an array 'whatList', with size equal to the total size of the cells. It was used, to hold information about the list each cell is currently in. The cell can be in openList, closedList or in any one of those two, because it was not yet reached. Each node generates its neighbor nodes, the way described in section 4.4. The cost to reach each neighbor node, $g(n)$, is a combination of euclidean distance, with rotation distance between the current node's orientation and its neighbor's orientation. The algorithm returns a path, only when reaches in the desired cell with the desired orientation.

As mentioned before, the robot walks faster in front movement. The side steps are limited by the connected hip, and the same distance with the front move is covered in more time. Bellow is a table containing the average distances for the front, side and rotational move for a minute run.

In this table it is obvious that the side steps cover a little less than the half distance in the same time as the front move. This information is used in the $g(n)$ cost calculation of the A* algorithm.

The path returned is used by a simple motion controller to decide the next robot's move. It actually uses the first path waypoint returned, to calculate the velocity commands. In order to implement a smoother move towards the goal, the motion controller examines some steps after the first and the estimated distance to the goal, to adjust the speed. The farther the robot is from the goal the faster the move will be. As the robot reaches the destination, the walk commands slow down to avoid great overshoot.

5.5 Debugging

For debugging purposes, an offline code was implemented. While running on robot, the ObstacleAvoidance module kept a log, about the information the offline code could not have access, such as sensory information about obstacles and odometry, messages

received from another module, stating the desired goal, and the time on regular intervals. This information was then used as input in the offline code, that used the same functions with the Obstacle Avoidance module running on the robot. The offline code, used only the personal computer's resources in order to draw the grid, as perceived by the robot every time the ObstacleAvoidance module run (which is currently, every 250ms). The offline code can run step by step a whole game a robot played, and it also prints useful information on the console, used to monitor the changes in the grid.

Chapter 6

Results

A number of test were run in order to evaluate the methodology described in this thesis. In this section three different experiments are described and presented.

6.1 Face Backwards

In this experiment the robot is positioned in the middle of the field and its goal is to walk 1.2m on the side, and face the goal opposite to the one it was facing when starting. A sequence of pictures follows 6.1 to show the robot's actual move and the polar grid representing the robot's knowledge about its environment in every instance. The calculated path is shown with red lines that also indicate the robot's orientation.

6.2 Avoid obstacle and reach a specified goal

In this experiment the robot in positioned in the middle of the field and its goal is to walk 0.5m on the front and 0.5m on the side. The desired orientation of the robot when it reaches the goal is vertical to its orientation in the starting position. The path the robot is following is represented with the red lines on the grid shown on the upper left corner of the figures 6.2.

6.3 Real field simulation

In this section the robot runs all the modules used in a real game. The ball is positioned in some distance, but in a place the robot can see it. Between the robot and the ball two robots are positioned, close, simulating objects that get into the main robot's way to the ball. The goal is for the robot to avoid these two obstacles while approaching the ball. Following there is a sequence of pictures 6.3 presenting the robot's movement to approach the ball. Simultaneously, the current polar grid is shown where the darkest areas represent the obstacles and the lighter colored areas represent free space. As it was previously noted the red lines show the calculated path.

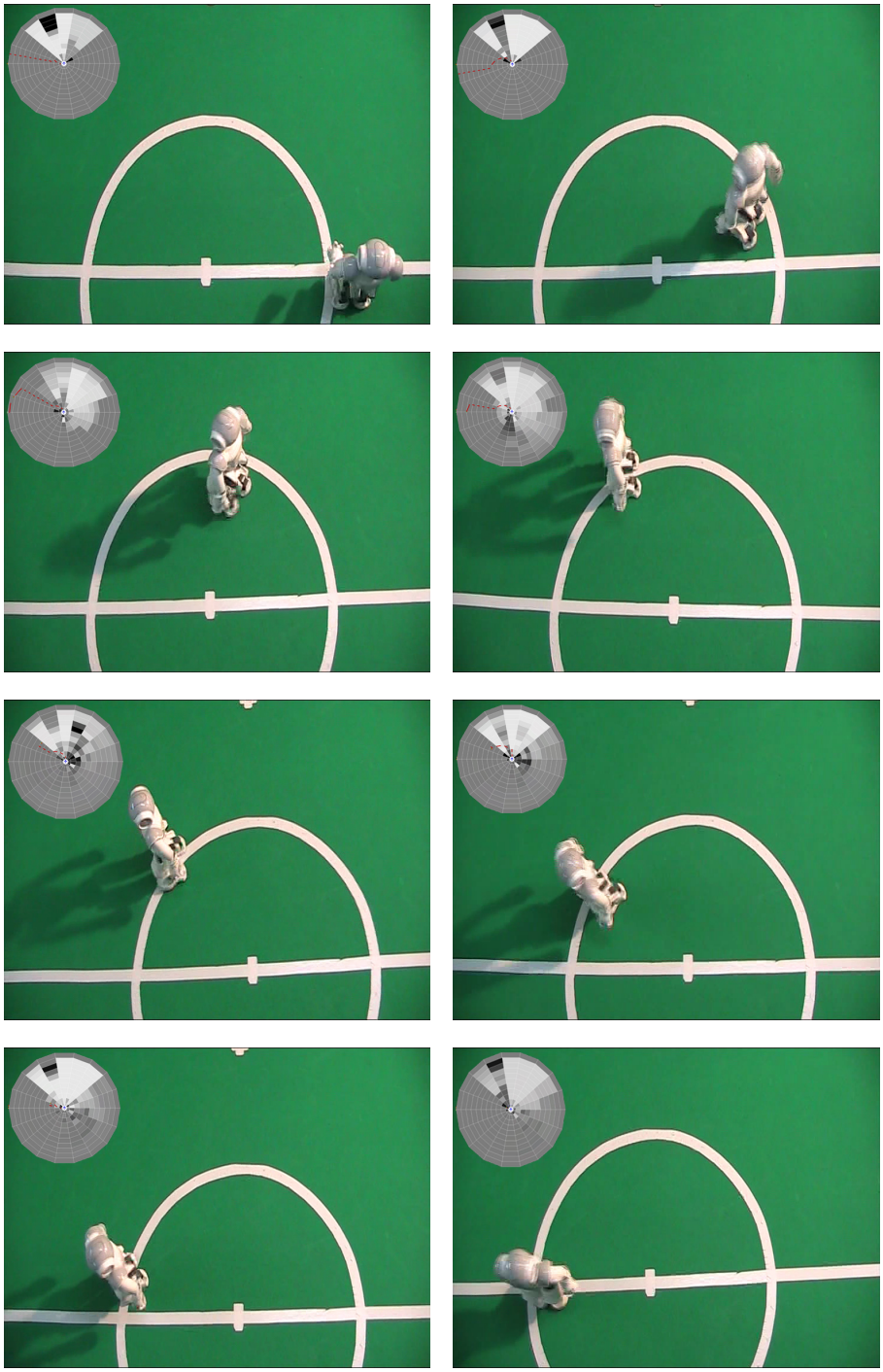


Figure 6.1: The robot walks 1.2m on its side and turns to face the opposite goal.

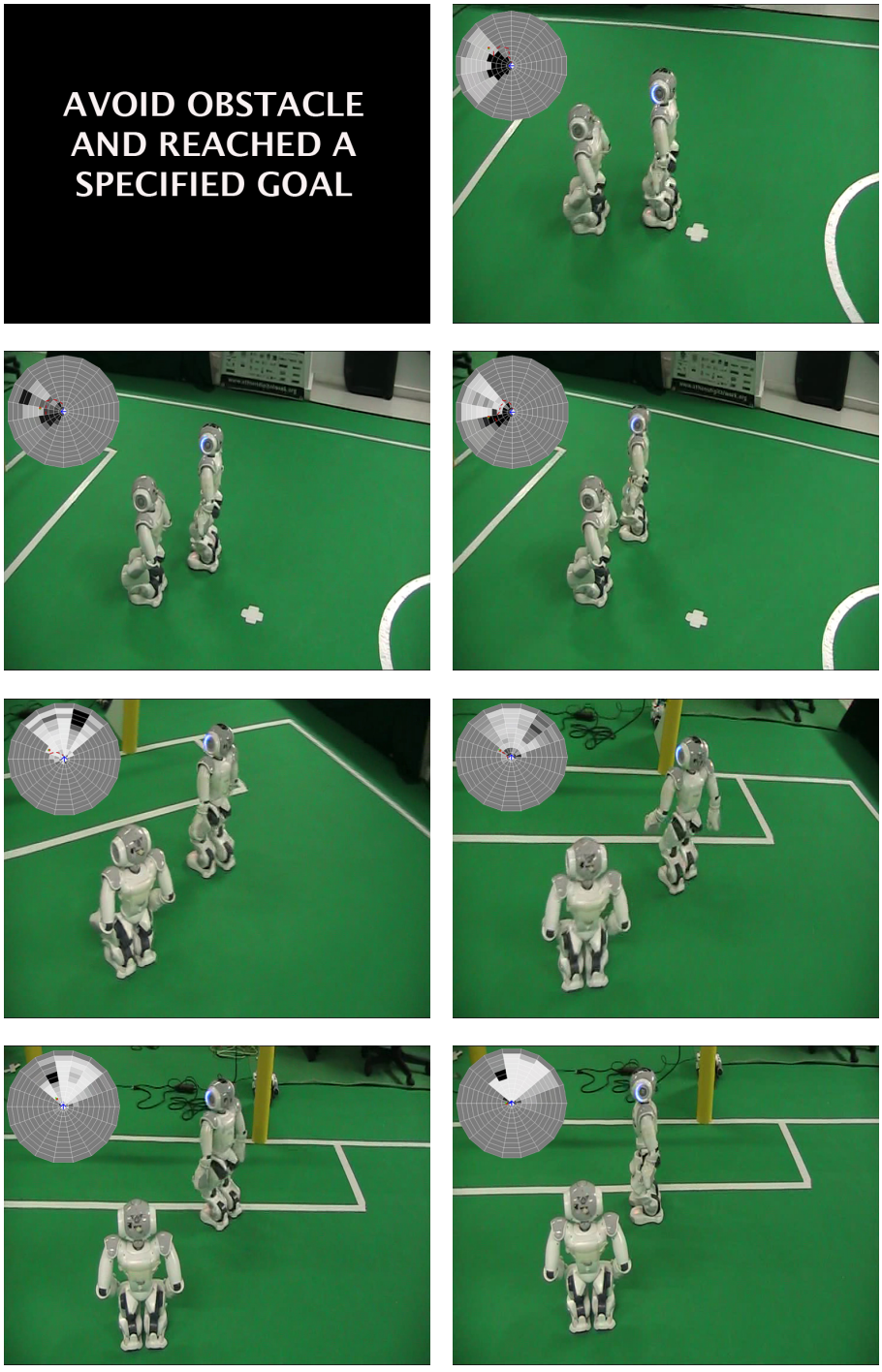


Figure 6.2: The robot walks 0.5m in front and 0.5m on its side and faces towards an orientation vertical to the one it had in the beginning of the move.

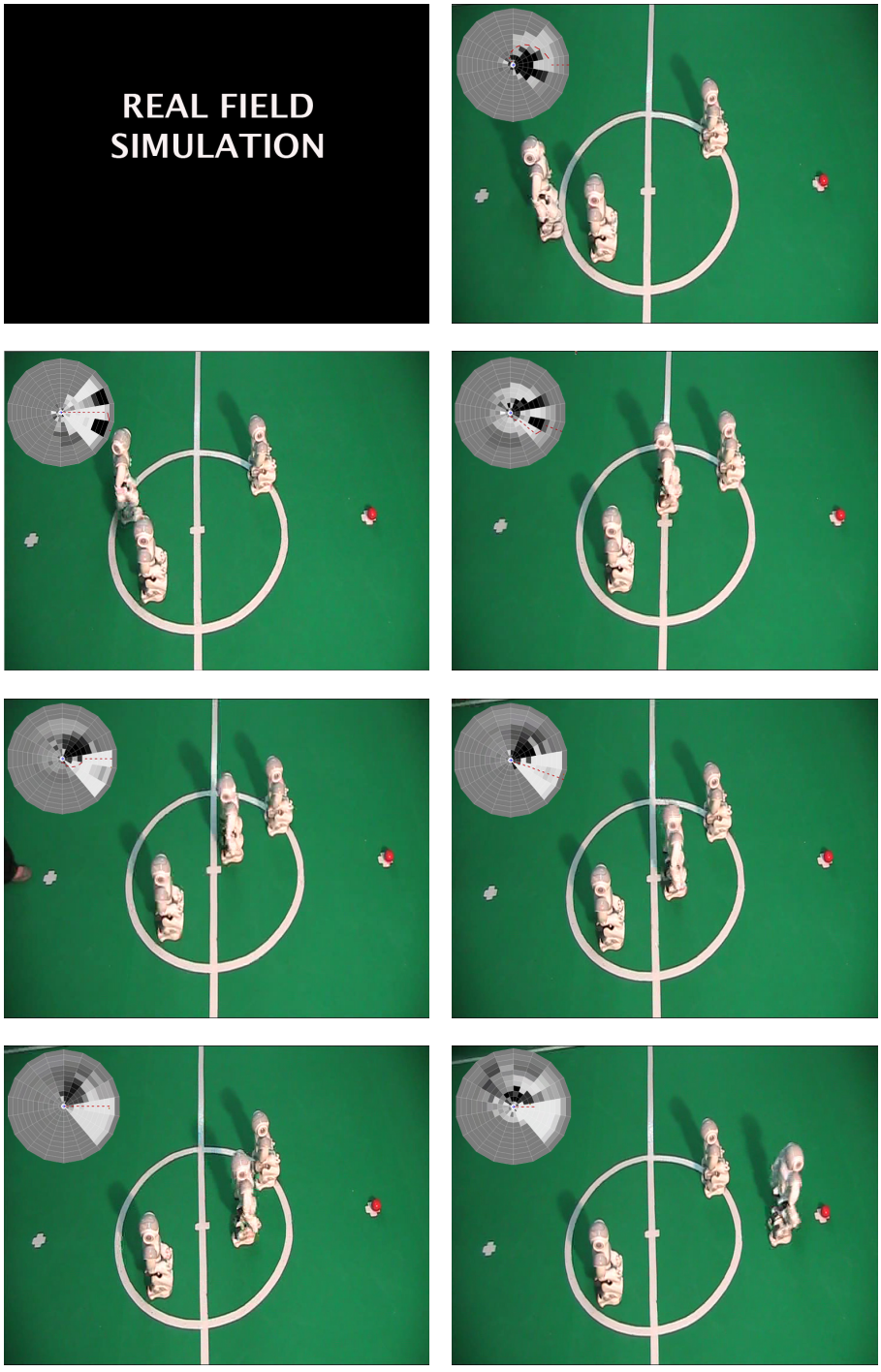


Figure 6.3: The robot walks towards the ball avoiding the other robots that get in its way.

Chapter 7

Related Work and Discussion

In this section, a short overview is presented of what other RoboCup teams have published trying to solve obstacle-avoidance and path-planning related subjects. There are various approaches for obtaining obstacle information, representing the world map and path planning algorithms applied.

7.1 Obstacle Detection Sensors

Besides sonars, vision can provide information about the existence and distance of an obstacle from the robot. B-human team [3] and NTURobotPal are two of the teams that use both techniques. Information about distance and angles to obstacles can also be provided from other robots playing in field, that inform their teammates via communication.

7.2 Grid Maps

The grid maps seen in the RoboCup fields can be regular (Figure 7.1), multiresolutional regular (Figure 7.2), or Log-Polar grids (Figure 7.3). Moreover, some teams choose grids that cover the entire field environment, while others prefer local grids, in which each robot holds and stores information about its own environment.

The advantage of the global map is that, at any possible moment, the robot knows

the distances and angles to all current obstacles. Thus, the robot can run a well informed path planning algorithm, even if the goal is in a blind spot.

The regular grid discretizes the environment into equally-sized cells, as shown in Figure 7.1 and is the most common teams' choice. The Nimbro team [4], has implemented a multi-resolutional regular grid, shown in Figure 7.2. This approach deals with computational complexity issues, related to grid size. Additionally, this approach offers higher resolution closer to the robot, which is more useful in deciding the next step while path planning.

Finally, the most similar approach to this thesis, is the Log-Polar grid, implemented by team Nimbro. The grid, shown in Figure 7.3, has 16 cell rings and covers the angular area around the robot with 16 steps too. Due to the logarithmic calculation of each cell's distance from the robot, the grid is big enough to plan paths for any two points within the SPL field boundaries.

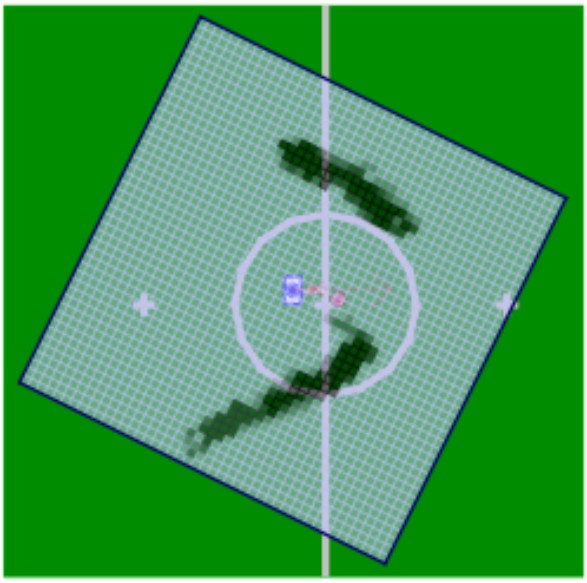


Figure 7.1: The regular grid used by BHuman team [3]

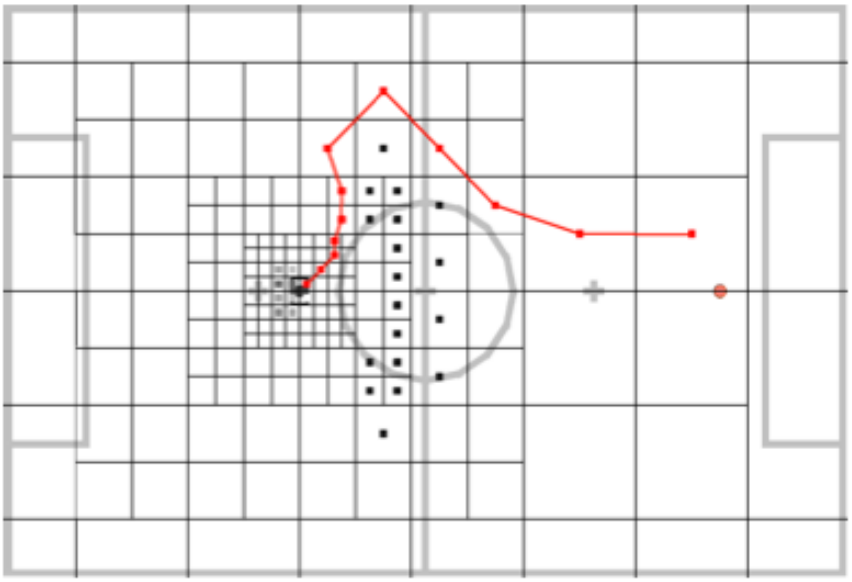


Figure 7.2: Multi-resilution grid implementes by Nimbro team []

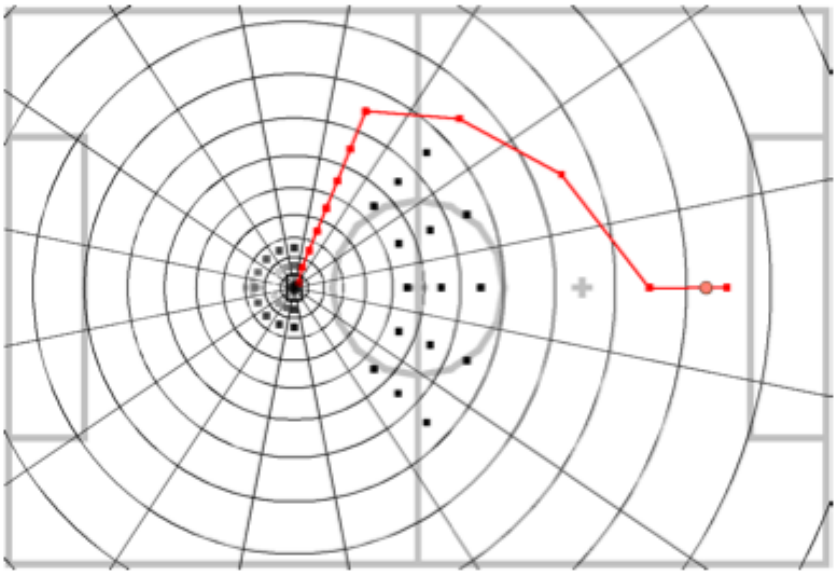


Figure 7.3: Log-Poalr grid implementes by Nimbro team [4]

RRT. The Rapidly-Exploring Random Tree algorithm
$set.addstartingPosition$
while target.notReached **do**
$randomPos \leftarrow$ chose random position within search space
$neighbor \leftarrow$ nearest neighbor in set
$neighbor.explandTowards(randomPos)$
if $collisionBetween(creatededge, obstacle) = false$ **then**
$set.add(randomPos)$
end if
end while

Figure 7.4: The Rapidly-Exploring Random Tree algorithm.

7.3 Path Planning Algorithms

Some of the teams participating in SPL do not implement a path planning algorithm, but rely on a simple avoidance of the direct front obstacles.

The most common algorithm to solve path planning problems, amongst teams that use occupancy grids to hold information, is the A* algorithm 2.5.1. It is an optimal and complete algorithm, with good performance.

B-human is a collegiate project at the Department of Computer Science of the University of Bremen and the DFKI research area Safe and Secure Cognitive Systems. Its path planning implementation uses an extended bidirectional Rapidly- Exploring Random Tree (RRT 7.4). This is a non-optimal algorithm based on the random exploration of the search space and works on continuous values.

This algorithm builds up a tree that quickly expands in few directions of the search space, because in each step of the algorithm the tree is enlarged by one edge in a random direction . For this general algorithm, different variants exist of which the extend and bidirectional1 ones were used here. Using the extend variant restricts the expansion towards the random position to a given distance that is the same for each expansion, which has a direct influence on the expansion of the tree, whereas the used bidirectionally has no influence on the tree itself but it is used to decrease the run time.

This is achieved by creating two separate trees, one beginning from the start point and one from the end point. Another modification is to replace the random position by the target position or a waypoint of the last found path with a given probability. Using this modifications helps to avoid oscillations of the path, for instance, if there is an obstacle on the direct way to the target (see Figure 7.5) [3] .

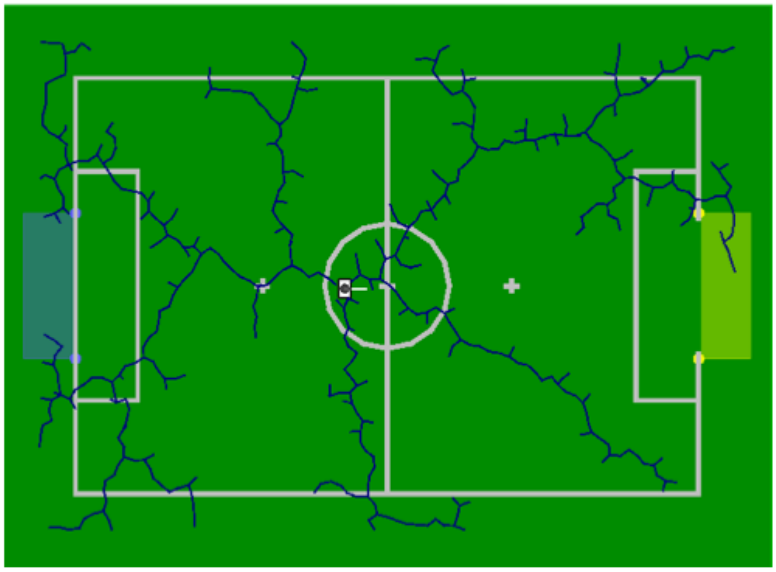


Figure 7.5: The Rapidly-Exploring Random Tree algorithm used to find paths on field [3]

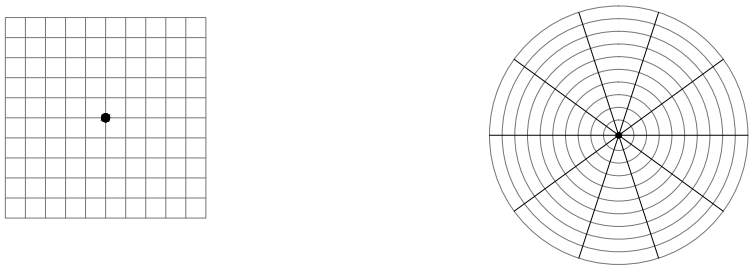
7.4 Why a Local Polar Grid?

The previous sector presented different approaches to the obstacle avoidance and path planning problems. In this section the differences of the current approach are discussed and compared

7.4.1 Polar vs Regular

A regular grid needs more cells to cover the same area as the polar grid. Assuming two egocentric grids, one polar and one regular that have the same resolution (10 cells

x 10 cells), as shown in figure 7.6. If the distance between two neighbor rings is the same, it is obvious that in order to provide the same range as the polar grid (including information about obstacles lying behind the robot), the regular grid has to hold the double number of cells. Higher number of cells to search through a path planning algorithm, increases not only the use of the robot's memory but also the computational complexity. The regular grid method also does not scale well with increasing grid size.



(a) A regular 10x10 cells grid.

(b) A polar grid with 10 rings and 10 sectors.

Figure 7.6: In both images the black dot represents the robot. The regular grid 7.6(a) covers the half area in front of the robot, in contrast to the polar 7.6(b). In order to cover same distance range, the cell number in each dimension must double.

Moreover, these is a great visual similarity between the polar grids sectors and the sonar cone-shaped beam. With the polar grid approach, the update of the map, using information of a certain beam, becomes very simple. Ultrasonic measurements bear distance information, and the position of the sonar is in a known angle, so the polar expression of the obstacle measurements is straight-forward. The polar grid is a circle, so the distance from the center to any edge is the same everywhere and equals to the maximum sonar distance range. In the regular grid the distance from the center to the four corners is 1.4 times the sonar range. These areas are considered in calculating the path, increasing the computational complexity.

The polar grid's angle resolution decreases as the distance from the center increases. This is actually a very good representation of the sonar's loss of accuracy the farthest the recognized object is. Moreover, this representation is a good analogue

to uncertainties occurring in dynamic environments when distance from the robot increases.

7.4.2 Local vs Global Grid Map

The advantage of the global map is that, at any possible moment, the robot knows the distances and angles to all current obstacles. All robots inform a shared map of the field about the obstacles found in their way, and consult the map in order to run a well informed path planning algorithm. The robot can calculate a good approach even if the goal is in a blind spot.

A local map requires less memory and computational resources than the global. The robot holds information only for close obstacles, and calculates a path based on this information. Calculating a path is also computationally more efficient in a local map, due to smaller size than the global. Since the environment changes constantly, the robot must reevaluate the paths calculated very often, taking under consideration new obstacles, or new open spaces on the field. Robots move constantly, and consequently holding information for far objects and planning long paths is not worthwhile.

7.4.3 Differences with Nimbro approach [1]

The approach closer to the obstacle avoidance and path planning problem is, as mentioned, the Nimbro team approach. Nimbro team, is the University of Bonn team, competing in the teen-size humanoid league.Their aproach uses a polar grid and plans the path using an A* algorithm. Nimbro team holds a global map, and rings' distance increases logarithmically. The path planning algorithm used, neglects the orientation and velocities of the robot. The preferred move for the robot is the front move, and a virtual obstacle is positioned behind the robot to force it to move forward.

The above approach is described in detailed in [1], where other grid approaches are discussed as well. The Nimbro team tested three different grids, a regular, a multiresolutional regular (see figure 7.2), and a multiresolutional log-polar grid (see figure 7.3).

Chapter 8

Future Work and Conclusion

8.1 Future Work

The sonars used to update the occupancy grid map, are poor in accuracy. Vision-provided observation about obstacles are more accurate. As soon as the vision observations are ready the enswmatwsh is straight forward. Bumper sensors on robots' feet provide information about direct contact with obstacles, which can also enswmatwjei to the current model.

The polar map information can be projected on the world model, and inform a global map about obstacles appearing in each robots' field of view.

8.2 Conclusion

Our approach is not a breakthrough, but it is an honest effort of solving an existing problem.

Although working with sensors that are so inaccurate, as sonars, experiments have shown that the robot holds a good representation about its surrounding environment. Problems with false positive values, still occur, especially when the robots run in our lab environment that more than 20% of the space is covered with a metallic reflecting material.

References

[1] Ricarda Steffens, M.N., Behnke, S.: Multiresolution path planning in dynamic environments for the standard platform league vi, 51

[2] Lagoudakis, M.G.: Mobile robot local navigation with a polar neural map vii, 27

[3] Thomas Rofer, Tim Laue, J.M.A.F.F.F.K.G.C.G.T.J.d.H.A.H.A.H.D.H.P.K.T.K.C.K.B.M.O.J.L.R.F.W.: B-Human team report and code release 2011 (2011) Only available online: www.b-human.de/download.php?file=coderelease11_doc. viii, 45, 46, 49

[4] : Nimbro teensize 2012 team description. Only available online: http://www.nimbro.net/ viii, 46, 47

[5] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., Matsubara, H.: Robocup: A challenge problem for AI. AI Magazine **18**(1) (1997) 73–85 5

[6] Sebastian Thrun, Wolfram Burgard, D.F.: Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series) 15

[7] Murphy, R.R.: An Introduction to AI Robotics (Intelligent Robotics and Autonomous Agents) 15

[8] Peter E. Hart, Nils J. Nilsson, B.R.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Science and Cybernetics, SSC-4(2):100-107 (1968) 17

[9] RoboCup Technical Committee: RoboCup Standard Platform League (Nao) Rule Book (2011) 19