POLITECNICO
MILANO 1863

Master of Science in Computer Science and Engineering
Department of Electronics, Information, and Bioengineering

AIRLab - Artificial Intelligence and Robotics Laboratory

Adaptive difficulty selection for skill matching in Robogames

Supervisor: Prof. Andrea Bonarini
Co-supervisor: Ing. Ewerton Lopes

Author:
Boriero Stefano, matricola 876106

Academic Year 2017-2018

To my family

Abstract

The more and more pervasive presence of robots in every day life pushes the urge to study and improve the interaction that occurs between humans and robots. We focused our attention on a competitive scenario, which has been designed following the Robogame guidelines. The aim of the thesis is to design a decision system to adapt the behavior of the robot in order to match the ability of the player, offering an even game. We used an approach based on the TrueSkill framework, tested on a simulation environment and deployed it on the robot. We hosted people to play the game, with both a fixed and an adaptive behavior. We compared answers to a questionnaire between the two versions, and found an improvement on the overall pleasure of the interaction. We also succeeded in making the game even, with an almost 50% share of wins between humans and robot.

Sommario

La presenza sempre più massiccia dei robot nella vita di tutti i giorni spinge il bisogno di studiare e migliorare la qualità dell'interazione tra umani e robot. Abbiamo concentrato la nostra attenzione verso uno scenario di competizione tra uomo e robot, il quale è stato definito seguendo le linee guida dei Robogame. Lo scopo della tesi è quello di definire un approccio decisionale per adattare l'abilità esibita dal robot all'abilità del giocatore umano, nell'intento di offrire un gioco bilanciato. Abbiamo utilizzato un approccio basato sul framework TrueSkill, l'abbiamo testato in un ambiente simulato e successivamente implementato sulla piattaforma robotica. Per valutare l'impatto del nostro lavoro abbiamo invitato gli studenti a giocare, dividendoli in due gruppi: il primo ha affrontato una versione base del gioco, senza segni di adattamento, mentre il secondo ha affrontato un robot con capacità di adattamento. Abbiamo sottoposto un questionario ai due gruppi alla fine dell'esperimento e comparato le loro risposte, trovando segni statisticamente rilevanti di una maggior piacevolezza dell'interazione per quanto riguarda la versione adattiva. Siamo inoltre riusciti a ottenere una ripartizione equa del numero di vittorie tra robot e umani, avvicinandoci all'ideale 50%.

Acknowledgements

I would like to thank all the people that partecipated to the experiment to help me finishing the work: Andrea, Annalisa, Benedetta, Cesare, Davide C., Davide I., Fabio, Francesca, Giacomo, Giulia, Laura, Lorenzo, Mariella, Martina, Matteo, Pietro Ma., Pietro Me., Sabrina, Sara, Simone Bi., Simone Bo. e Stefano. I thank all my colleagues of the AIRLab Claudio, Emiliano, Michele e Riccardo, that helped me by making the time spent in the lab more enjoyable. I thank Laura and Ewerton that worked with me on the project, for their help and their company. I thank professor Bonarini for the time and resources he invested in the project. Lastly, I thank my family for giving me the opportunity to take this path and for the support they always offered me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The latest achievements in the research fields related to robotics and the sprout of the consumer electronics market led to a pervasive presence of robots in every day life. The increased degree of autonomy in the behavior of robots changed the way people used to interact with them: we're moving from an interaction predominantly driven by user initiative, towards an interaction where robots can take autonomous actions. The aim of this work is to devise a decision system able to tailor the decision process on the user and to study whether it can improve the quality of the interaction or not. The study involved a conflicting game between a robot and a human: we tried to modify the ability of the robot in order to match the ability of the player and to offer an even game. In order to evaluate the effect of adaptation on the interaction, we hosted people to play the game with both a fixed behavior and the adaptive one, and collected their opinions on the game. We analyzed their answers to search for a statistically significant difference between the two versions, and evaluated the magnitude of the impact of our work on the engagement. Our analysis suggested that we can say our work had an impact on the engagement, even tough the magnitude of the difference is limited: we believe one of the cause for this is the intrinsic appeal of the game itself even in its most basic version. This work underlines and supports the need for behavior adaptation even in a competitive scenario: future works may try to bring the adaptation to a deeper level, using a richer description of the user and its playing style other than its ability level.
This study can be placed inside the Human Robot Interaction field: more specifically, it focuses on the benefits of adapting the behavior to the final user, with the purpose of improving the overall interaction. Previous researches, like [18, 15, 2], have investigated these questions in cooperative scenarios, where usually the purpose of the robot was to help the human

in carrying out some tasks. These studies confirmed the general assumption that an adaptation improved the quality of the intercourse between human and robot and the likeability of the latter one. Our work wants to analyze similar aspects of the interaction in competitive scenarios: in particular, our scenario can be listed among what are known as Robogames [12], a game scenario where a human user has to play against a robotic opponent. Since this is a completely novel area of robotics, few studies had been carried out. In [9] it has studied the impact of autonomous behavior and emotional feature in a Robogame, but nothing had been questioned about behavioral adaptation. This is where aroused the need for this study. We devised a procedure to select, among a set of parameters describing different characterization of the robotic agent, the parameters that would maximize the probability of drawing the game between the opponents, thus providing an even game. The parameters we considered only constrained the motion capabilities of the platform, assuming that a less constrained robot would exhibit higher ability in the game. We tested this procedure on a simulated environment, where two agents competed against each other. We trained several agents to play the game autonomously with different levels of skills, like the humans in our scenario, and we introduced the possibility to characterize the behavior of the opponent with a set of parameter, as our robot. After reaching a stable and reliable procedure, we deployed it on the robot and conducted experiments with students playing the game, collecting their opinions. We compared the answer of people playing with and without the adaptation to search for statistical proof of improvements in the interaction quality. A deeper adaptation of the robot behavior, especially on the action decision system, should be devised and its implications and impact should be studied. Also, the user classification and the playing style identification remain an open problem, which could lead to better and more user specific adaptation.
The work is structured in the following way:

- **Chapter 2** provides the reader the needed background to understand the problems addressed and the solutions presented. It starts presenting concepts of autonomous robots with focus on holonomic platforms. This part is followed by a presentation of Human Robot Interaction in general and a presentation of Robogames in detail. To complete the background of the tools used in this work, we briefly discuss ranking systems and their use in match making contexts

- **Chapter 3** presents the game scenario and the robotic platform used. We present as well some problems related to autonomous robot we encountered and how we solved them, in order to have a correctly working game

- **Chapter 4** illustrates the game decision system, the architecture used to decided which actions to take during the game. We described the different type of actions allowed and their implementation. In the last section we show how to insert and create new actions in the decision system

- **Chapter 5** starts with a description of the TrueSkill ranking system and proceeds explaining how we used and adapted it in our use case, then presents the simulation environment, starting from how we built it and proceeding to present the results of subsequent refinement of our parameter selection procedure. In the last section, we discuss the implementation on the robotic platform for our scenario and the tweaks it required

- **Chapter 6** presents the experimental environment we used to perform the experiment and the statistical analysis we used to support our claims. We first discuss informally the data collected trough visual representations, then perform the statistical analysis on them and comment the results

- **Chapter 7** is the final chapter, where we recap, discuss the results and suggest the guidelines for future works on the subject

- **Appendix A** contains the questionnaire the participants filled in after the game

Chapter 2

Background

This section will give the reader the needed background to understand the concepts treated and used in the thesis: it will introduce the technical concepts related to robotics, human robot interaction and PIRG. To complete the needed background to cover the thematic areas involved in this work, a brief introduction of ranking systems is presented.

2.1 Mobile Robotics

In recent years the research field dealing with mobile robots, especially autonomous ones, has been facing a huge sprout. In order to work with such robots, it is required to blend in knowledge from many different engineering and science disciplines, ranging from mechanics to computer science, passing by electronics.
To define what a mobile robot is, it is helpful to put it in contrast with manipulators: while a manipulator operates in a constrained environment, has an absolute measurement of position and does not necessarily need to sense the external world, a mobile robot:

- Can operate in unconstrained environments
- Needs internal and external sensing to determine its position
- Needs external sensing to avoid obstacles

This requirements pose many different challenges and issues to be solved in order to work with a mobile robot. One challenge is that of *localization*: the robot has to be able to determine its current position in the environment in order to correctly work. This can be done relying only on internal sensing, usually trough the use of encoders: this technique is called *dead reckoning.*

While it provides an easy implementation, it is prone to errors as any error is just added onto the previous one leading to an erroneous internal belief that does not match reality. To address this problem it is possible to use multiple sensors, both internal and external, and combine them to reduce overall error: this technique is called *sensor fusion* and provides better results for each additional information source, with a properly modeled error.
To exploit the capabilities of a mobile platform, it's required for it to be able to move inside the environment it is placed: to do so, it needs to have a representation of the environment called *map*. There are different types of maps, each one with its own pros and cons: we can find continuous maps, representing obstacles with lines, or discrete ones, with either fixed or adaptive cell dimension. A Priori maps can be a good starting point to specify where the target point is in the environment and start a global planning of the path, but any variation caused by object motion can confuse the robot while navigating; dynamically generated maps can be used in synergy with a priori ones to account for moving obstacles and augment local accuracy.
The last challenge posed by mobile robots is the one of *navigation*. Given its position and a point to be reached in the map, the robot has to be able to plan a path to reach the target. The path is planned at two different level: at the global one, the robot finds a suitable path between the two points in its configuration space; at the local one, it detects obstacles and updates the map according to sensor measurements.
An example of the issues to face when designing an autonomous mobile robot is presented in [8]: this work tackled the problem of mapping and localization by using an a priori map built with a SLAM algorithm and a real-time localization procedure based on LIDAR, IMU and odometry data. They underlined that while this was a procedure easy to implement, it was prone to errors, especially in the odometry estimation: they tried to reduce this effect by using an extended Kalman Filter to perform data fusion. The application domain for these type of robots is almost unlimited: in [3], an autonomous robot has been developed to perform inspection of metal-based bridge to check its maintenance status. Tasks like this one, where there's an high risk for the human, represent a domain where highly autonomous robots would apply best.

2.2 Holonomic Wheeled Robots

Holonomic wheeled robots represent a particular class of mobile robots. In this area, the term holonomic refers specifically to the kinematic constraints of the robot chassis. A *holonomic* robot is a robot that has zero nonholonomic kinematic constraints. Another definition of *holonomic* robot is the following: a robot is *holonomic* if and only if its number of differentiable degrees of freedom (DDOF) are equal its degrees of freedom (DOF). In other words, a *holonomic* robot is able to maneuver around obstacles without affecting its orientation and to follow any arbitrary path. The most frequent type of *holonomic* robot are *omnidirectional robots*, where

$$DDOF = DOF = 3$$

This type of motion requires to be supported by a proper mechanical configuration: in particular, it needs holonomic wheels. Holonomic wheels are wheels with two degrees of freedom; they are also known as omni-directional drive wheels or omni-wheels for short, sometimes written as omniwheel. These wheels have small discs around the circumference which are perpendicular to the turning direction. The effect is that the wheel can be driven with full force, but will also slide laterally with great ease. A representation of an holonomic wheel can be found in figure 2.1.
Full omnidirectionality and the concept of holonomic wheels have been first introduced in [14]. This type of robots offer much richer movement capabilities: the study [11] focused on how to plan and perform simultaneous rotation and translation, a motion that would not be feasible for a non-holonomic platform. More than that, these platforms can follow any given path without steering constraints: [10] shows how to perform optimal path planning with these motion capabilities.

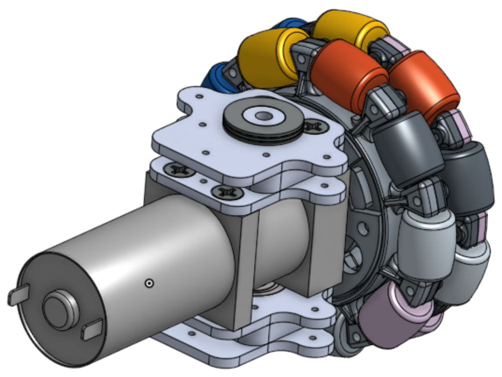


Figure 2.1: Holonomic wheel model

2.3 Human-Robot Interaction

Human-robot interaction (HRI) is the study of interactions between humans and robots. The goal of HRI research is to define models of humans' expectations regarding robot interaction to guide robot design and algorithmic development that would allow more natural and effective interaction between humans and robots. The continuous evolution of the ways robot participate in our everyday life pushes the need of research and studies towards new and better guidelines to drive the interaction. When robots first appeared, their usage was confined to industrial environment only and were not able to take any initiative due to the immaturity of the involved technologies: nowadays the studies and improvements in related fields, such as natural language processing or image processing, make feasible a much richer interaction enabling the robots to take also active initiative.
One of the overarching goals of robotics research is that robots ultimately coexist with people in human societies as an integral part of them. In order to achieve this goal, robots need to be accepted by people as natural partners within the society. It is therefore essential for robots to have adaptive learning mechanisms that can intelligently update a human model for effective human-robot interaction. Several studies have investigated the question whether an adaptation of the interaction to the currently involved human would result in a more pleasant intercourse. In [18] a group of 25 people was asked to rate the interaction with a robot designed for elderly assistance. Each participant collaborated twice with it: on one occasion the robot showed a standard, non adaptive behavior, while on the other one it exploited an adaptation strategy. The study found that the adaptive interaction differs from non-adaptive interaction at a statistically significant level, suggesting a preference towards the former one. A more extensive study [15] examined the impact of an autonomous mobile social service robot prototype in everyday life for elderly people. The prototype was deployed in 18 households for a total of 371 days and provided entertainment, phone service, reminders and control of a manipulator. Findings showed that while the robot met user's expectation, it did not increase safety sensation and was rather perceived as a toy instead of a support for independent living. The main cause was identified in the lack of adaptive autonomy from the robot's side. A follow up study [2] from the same group suggests a framework to enhance the robot with the ability to display different levels of autonomy, which will be used to validate the assumption that it will increase the perceived intelligence of the robot.
An experiment involving children [19] tried to answer whether a robot capa-

ble of adaptive emotion expression could support engagement. A Nao robot was enhanced with the aforementioned ability, and children played a quiz with both an affective robot using the model for adaptive emotion expression and a non-affective robot without this model. The behaviour and opinions of the children, were measured through video analysis and questionnaires. The results show that children react more expressively and more positively to a robot that adaptively expresses itself than to a robot that does not. The feedback of the children in the questionnaires further suggests that showing emotion through movement is considered a very positive trait for a robot. Another promising analysis [1] proposed an architecture to support adaptation in a learning scenario to support engagement. They designed as well an exploratory study with children from 7 to 10 years old, which is still to be carried out.

2.3.1 Physically Interactive Robogames

Robots toys involved in a game scenario are often limited when it comes to the way the user can interact with them. They usually take little to no initiative, making the only interaction possible a stimulus-response reaction. Nowadays, the introduction of low cost sensors and computing power paved the way to a richer interaction and games specifically designed for mobile robots. PIRGs (Physically Interactive Robogame) have been introduced in [12]: in a PIRG physical, autonomous (often mobile) agents are actively engaged in a game that creates some sort of interaction, either competitive or cooperative, between humans and robots, moving in the physical world. Robogames will be one of the next robotic products for the mass technological market, thus demanding a large exploration of new methodologies and applications, especially for what concerns methods for enabling high autonomy, intelligence, and adaptive behavior, in order to respond to demands in engagement support like the ones expressed in [22, 20, 21]. Being able to evaluate how people play is crucial for an adaptive game. In the virtual game industry, several studies have been published, most of them using artificial intelligence and machine learning algorithms. For instance, [4] reports about emergent self-organizing maps for grouping types of players.
A recent study [9] tried to evaluate the impact of autonomous behaviors and emotional features in a drone-based PIRG. Emotional features were encoded and conveyed through a mixture of motion patterns and sounds. The participant had to play three times: once against a robot showing autonomy and emotions, once against a robot showing autonomy but not emotions, and once against a robot remotely controlled with emotions. Data were col-

lected via questionnaire and the results show that autonomous behavior did not influence user experience in a noticeable way, while emotional features noticeably increased players' engagement and enjoyment of the game.

2.4 Ranking Systems

The main focus of a ranking system is to find an ordering between a set of players based on their ability. Some well known examples of ranking systems usage can be found in many different professional sports: the ATP maintains a global ranking for tennis players, the FIFA a world ranking for national teams and so on. The feature that makes ranking systems adaptable to be used in in many different contexts is that they focus only on finding an ordering between players: while defining a metric for the skill is a hard and often domain dependent task, finding an ordering based on game outcomes is a simpler and more generic approach to the problem. One of the most famous rating system is the Elo system [5], named after its inventor Arpad Elo, which was adopted first by the United States Chess Federation (USCF) in 1960 and then by the World Chess Federation (FIDE) in 1970. The basic idea behind it is that each player in each game can be represented as a normally distributed random variable that is updated after every game based upon the game outcome and the relative rankings of the opponents. Taking inspiration from this system, many others have been generated, including Glicko [6] and TrueSkill [7].
An attempt to use an adaptation of the Elo rating system for matchmaking has been carried out in [13]: in this scenario the ranking system has been used to select suitable questions for students given their estimated rank. The devised on-line procedure has been tested against an off-line procedure, showing comparable results. The main benefits highlighted by the proposed procedure are the ease of computation, compared to maximum-likelihood methods, and the flexibility to adapt to both new questions and students. In [16], an adaptation of the TrueSkill system has been devised to support preference elicitation in a movie recommendation context. They included in the process information about the items to recommend encoded in a vector of features: the study showed promising results in the preference prediction while tackling the cold start problem as well.
We adapted Trueskill to our type of games to be able to evaluate the performance of the player in short type so to adapt the performance of the robot to obtain even games.

Chapter 3

Robotic Platform

This chapter presents the robotic platform we adopted and developed: we will start from the physical properties of the robot and the game, then proceed to describe the hardware used, including sensors and actuators. Then we will proceed to discuss the implemented software solutions to tackle general issues related to autonomous mobile robots as player tracking, obstacle recognition and localization.

3.1 Triskar

The robotic platform is an omnidirectional platform: it can move and rotate in any direction along a fixed plane in space. It has an equilateral triangle base with sides of length 0.6m, where at each corner there's an omnidirectional wheel. It has a protecting plastic band 0.2m tall around the base to absorb impact and a central metallic body which height is 1.1m. On the central body is placed the on board computer hosting the computational core. Figure 3.1 shows a picture of the robot, while sketches about its geometrical properties can be found in figure 3.2.
The platform hosts a computer to support execution in the middle, and two laser sensor on the side to have a 360° visibility.

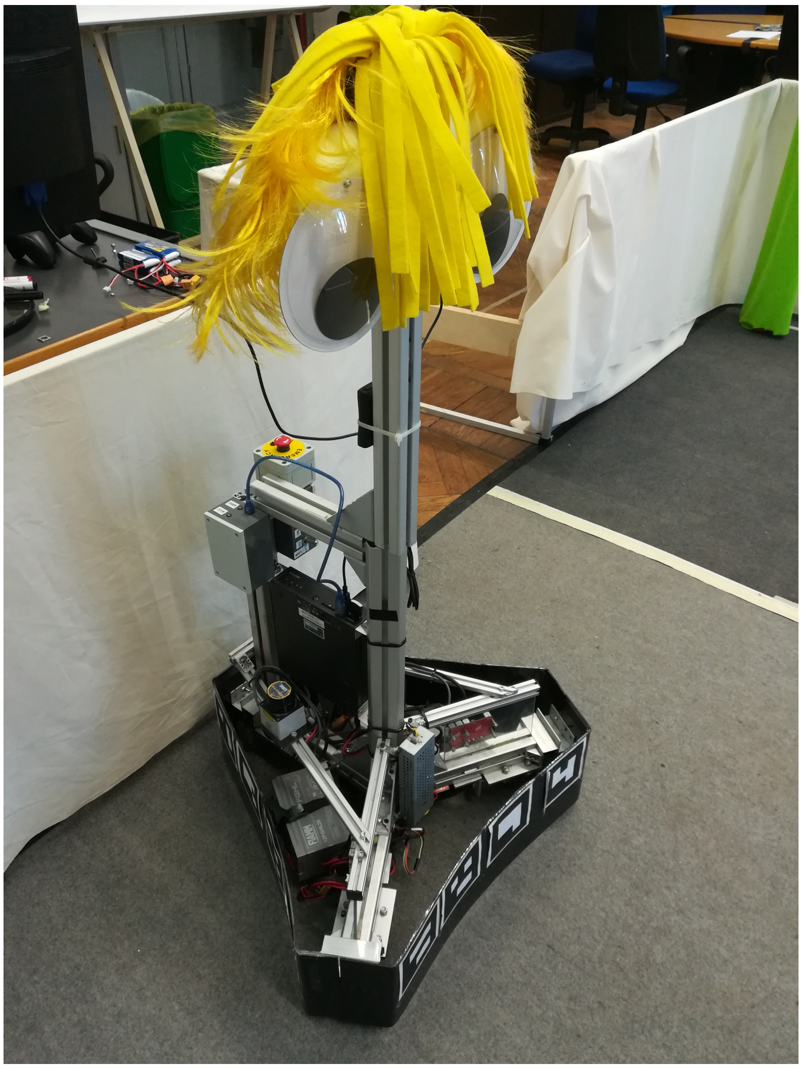
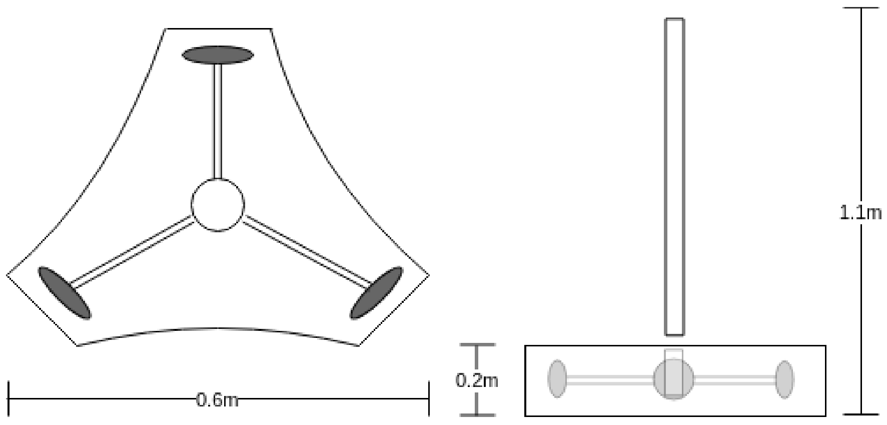


Figure 3.1: Triskar

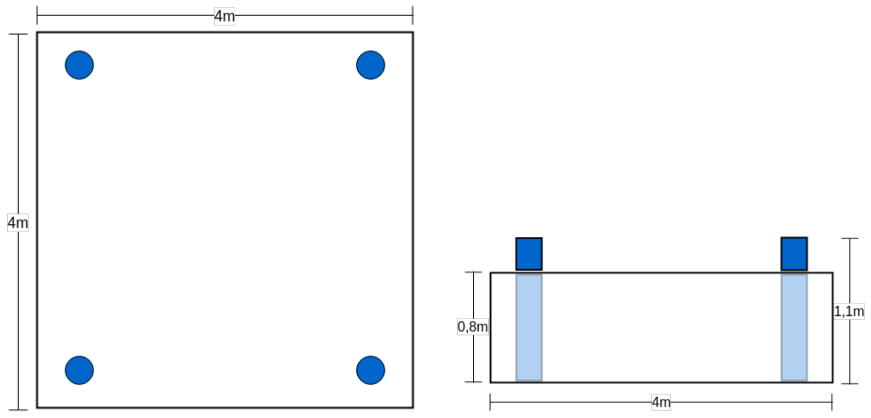


(a) Top view *(b) Side view*

Figure 3.2: Robotic platform views

3.2 Robotower

The game is played inside a 4m x 4m playground, surrounded by an 0.8m tall tarp. There are four towers placed in the corners of the playground. Each tower is a cylinder 1.1m tall and with a diameter of 0.15m. On top of the tower there are four leds and one button, as shown in figure 3.4.



(a) Playground top view *(b) Playground side view*

Figure 3.3: Playground geometrical properties

The goal of the robot is to physically take down one of the towers by colliding with them. The goal of the player is to defend the towers by posing himself on the robot's path: at the same time, the player should conquer each tower. In order to conquer one tower, the player has to light up all the four leds of the tower: to light up one led, he has to keep the button

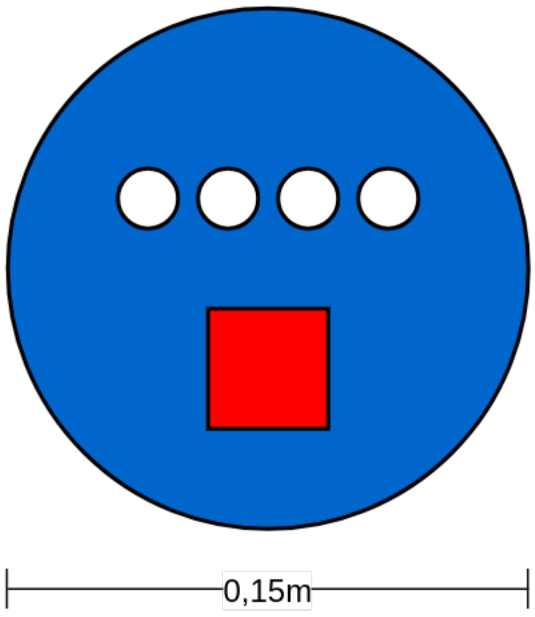


Figure 3.4: Tower top view

on the top of the tower pressed for two consecutive seconds. Once a tower is conquered, the robot can no longer try to take it down. The game is a conflicting one: while the player is keeping the button on one tower pressed, he has to monitor robot's actions to prevent it to take down another tower. The player wins when conquers all towers, while the robot wins when it takes down at least one tower.

3.3 Hardware

3.3.1 Motors

Each one of the tree wheels is powered by a MAXON 118798 DC motor RE36 GB 70W KL 2WE. Its characteristics are reported in figure 3.5 and table 3.1.

3.3.2 Encoders

To collect and monitor the speed of the motors a 110513 tacho ENCODER HEDS 5540 500IMP 3K is mounted on each one. The schematics of the encoder can be found in figure 3.4. Encoders contain a single Light Emitting Diode (LED) as its light source. The light is collimated into a parallel beam by means of a single lens located directly over the light source. Opposite the

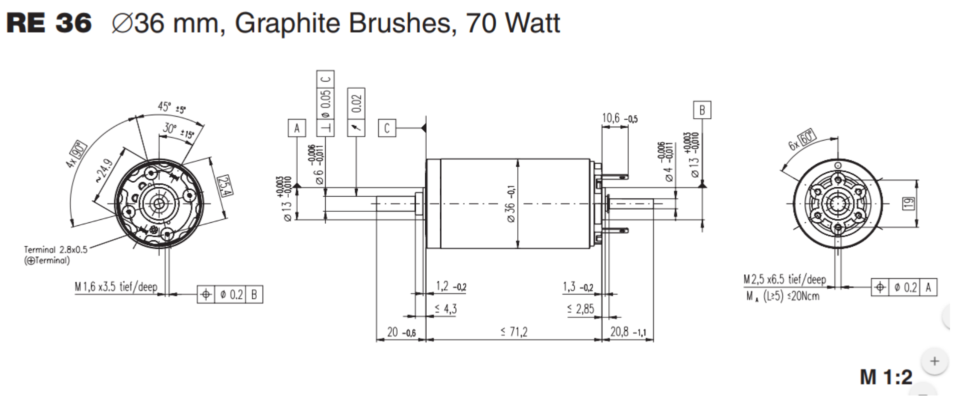


Figure 3.5: Motor schematic

Assigned power rating	70 W
Nominal voltage	24 V
No load speed	70 x 60
Stall torque	783 mNm
No load current	105mA
Terminal resistance	1.11 ohm
Max. permissible speed	8200 rpm
Max. efficiency	85%
Torque constant	36.4 Nm/A
Speed constant	263 rpm/V
Mechanical time constant	6ms
Rotor inertia	67.7 gcm^2
Terminal inductance	0.2 mH
Reduction ratio	[14:1] (166158 planetary gear GP32A 2.25NM)

Table 3.1: Motor datasheet

emitter is the integrated detector circuit. This IC consists of multiple sets of photodetectors and the signal processing circuitry necessary to produce the digital waveforms. The code-wheel rotates between the emitter and detector, causing the light beam to be interrupted by the pattern of spaces and bars on the code-wheel.

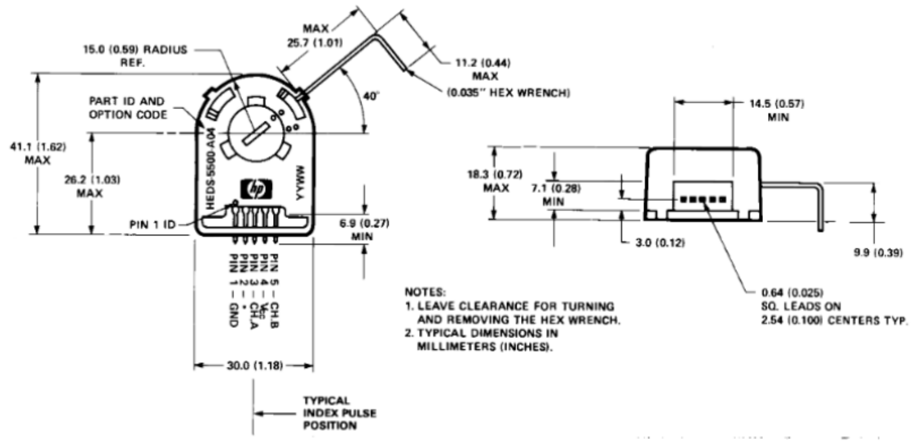


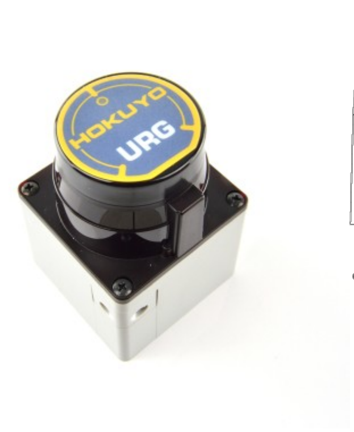
Figure 3.6: Encoder schematic

These interruptions are detected by the photo diodes that send signals to the circuitry in order to be processed. This circuitry produces the final output as an index pulse P O which is generated once for each full rotation of the code-wheel. circuitry that produces the final outputs that is an index pulse P O which is generated once for each full rotation of the code-wheel.

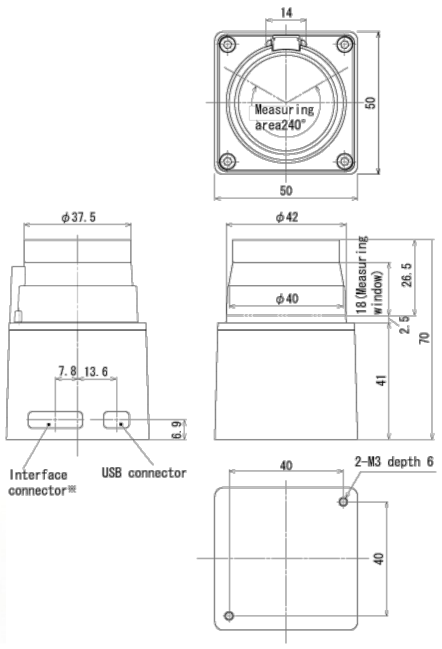
3.3.3 Laser Scanner

In order to perceive the environment in the most precise way, the robot is equipped with two laser scanners Hokuyo URG-04LX, shown in figure 3.7. These laser sensors perceive the range of obstacles on a plan with a field of view of 240° and a resolution of 0.36° , the maximum detectable distance is 5.6m and they can be connected to the computer by means of a USB interface, the required voltage is 5V.
The Hokuyo URG-04LX consists of a compact stacked structure with a spindle motor and the actual scanner on top of it. The motor rotates a small transmission mirror that deflects the vertical laser beam coming from the top of the sensor into horizontal direction. This allows the laser beam to scan a planar area around the sensor with an opening angle of 240° . A second mirror below, the reception mirror, deviates the horizontal laser beam

captured by a lens into vertical direction again.
A full scan is performed every 100ms. The two laser scanners are mounted on each side of the lower chassis. In order to maximize the scanning angular range, the two sensors are placed at about 120° with respect to the longitudinal axis of the robot. In this way a 360° coverage around the robot is achieved as shown in figure 3.8, adding robustness to the detection of obstacles.



(a) Laser sensor



(b) Laser schematic

Figure 3.7: Laser sensor

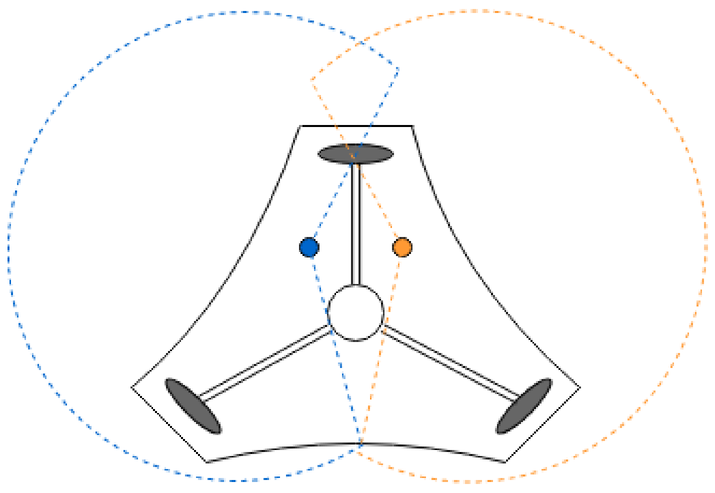


Figure 3.8: 360° *coverage ensured by laser positioning. The dashed blue line represents the field o view of the left scanner while the orange represents the right one*

3.3.4 Shuttle

The on board computation is performed by a Shuttle XPC Slim DH270. The device has a 190x165x43mm steel chassis for 1.3kg of weight presenting numerous threaded holes (M3) at both sides of the chassis to allow an easy fixture to the mobile robotic platform. The operating system is Ubuntu 16.04.3 LTS (64bit).

Processor	Intel Core i7 Kaby Lake
RAM memory	16 GB DDR4

Table 3.2: Computational power

3.3.5 Tower equipment

In order to control the leds and to enable communication between the towers and the robot, we equipped each tower with a ESP32 Core Board V2, an Arduino compatible board with built in WiFi functionality. It includes USB to serial programming interface, that also provides power supply for the board, and has pushbuttons to reset the board and put it in upload mode. Using this board we deployed the control logic of LEDs directly on the board, which communicates the status of the tower each second to keep the connection alive.

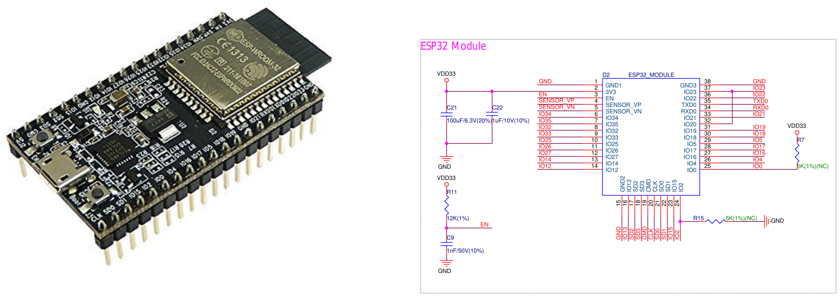


Figure 3.9: The ESP32 Core Board V2 and its schematic

3.4 Software

Enabling the robot to move autonomously required the usage and development of several side components that we're going to present in the next sections.

3.4.1 Robot Operating System - ROS

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. It embraces a distributed software philosophy, swaying developers towards designing highly independent modules, called nodes, resulting in more general purpose solutions. These nodes can communicate with each other trough the usage of messages: they can publish and subscribe to topics of interest in order to orchestrate the execution flow.
The large community of users made possible to find ready made solutions for common problems relative to robotics, such as static mapping and localization, facilitating our work during the development process.

3.4.2 Mapping and Localization

To build the a priori map of the environment, we used a SLAM algorithm before playing the game, producing a result like the one in figure 3.10.

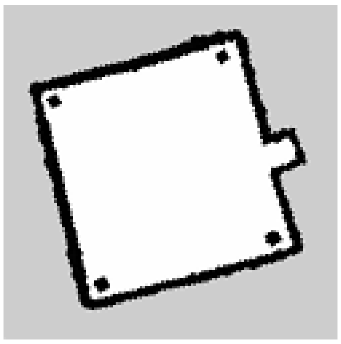


Figure 3.10: Map resulting from SLAM process

During the game, to localize the robot inside the map we opted for an AMCL approach and relied on the ROS node provided in the navigation stack. To feed the node, we passed information sensed by the laser scans, and from the odometry based on encoders.
The result of this simple localization procedure presented some flaws: when the robot performs a crisp and sudden movement, it causes the laser scan and the map to be misaligned, resulting in a non accurate localization: this poses problems in navigation as often the robot failed to reach the towers as the map resulted rotated or translated with respect to robot's notion. An example of this effect can be found in figure 3.11.

3.4.3 Tower navigation

Given the problem with localization, the system often mislocalized the towers, resulting in an non accurate navigation. In order to address this problem, we exploited the geometrical properties of the playground and of the towers. First of all we analyzed the laser scan in order to find clusters that would be compatible with a cylindrical shape: the set of clusters found would embrace both the towers and the player's legs. Given this information, we try to reconstruct the rectangle formed by the four towers: to have robust result in any game situation, we decided to reconstruct the rectangle using only 3 vertices. This choice has come after the consideration that the presence of the player between the robot and one tower will actually hide the tower from the laser sensors, thus we do not always have information about all 4 towers, but we have information about at least 3 towers with an high confidence. A visualization of the issue can be found in figure 3.12

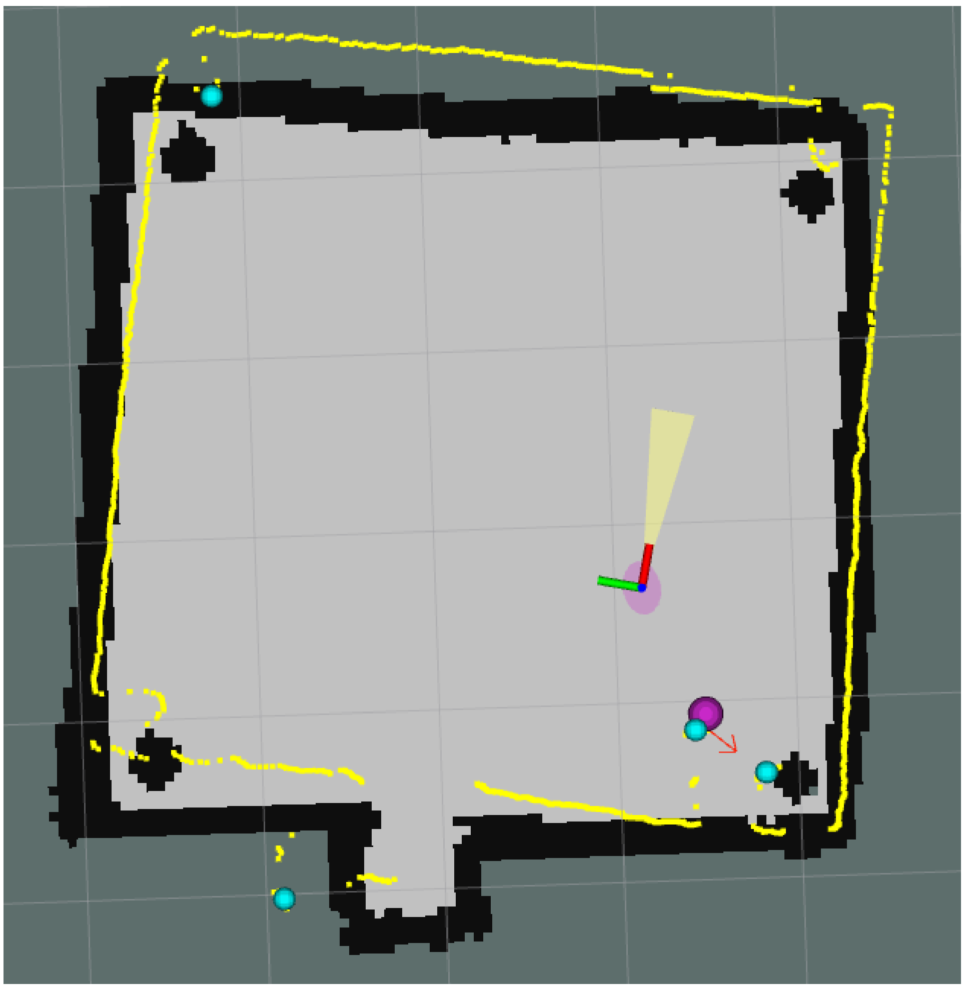
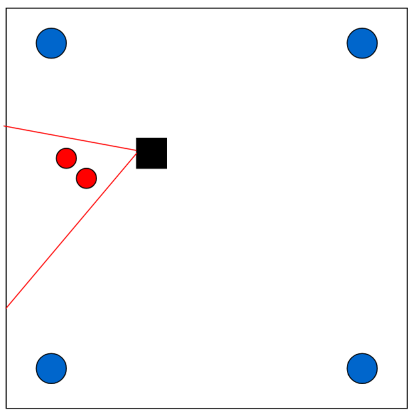


Figure 3.11: Example of misalignment between map and laser scan


(a) All towers are visible to lasers

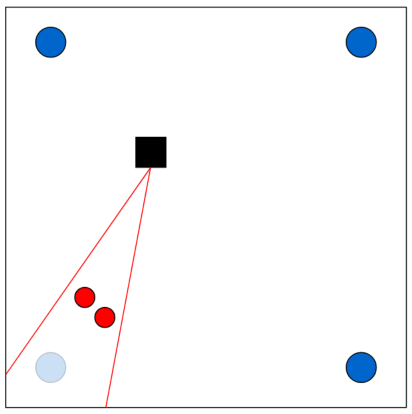
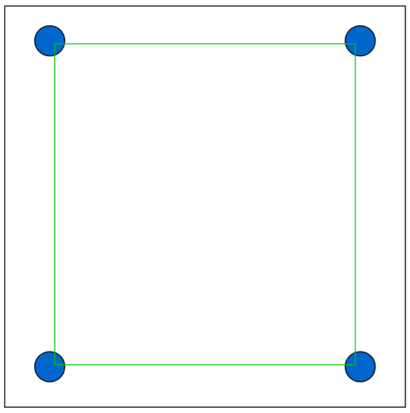
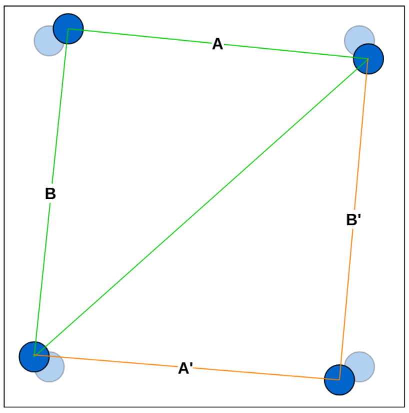

(b) Bottom left tower is hidden by the player

Figure 3.12: Possible configurations of detected cylinders

Given the initial position of the towers with respect to the map, we calculate the area of all the triangles we can obtain by splitting in half the rectangle, as well as the length of each side. When a set of new legs comes in we consider all the triplets from the set and evaluate thee area of the triangle they form: if the area matches one of the initial triangle, we check all the pairs from the triplets to evaluate whether their distance is compatible with one of the sides of the triangle. If this second step is successful, we have obtained the three sides of a triangle formed by three towers. To obtain an estimate of the position of the missing one, we have to identify which sides are the catheti: then, we choose one to be reproduced and one to be kept fixed. We estimate the orientation and length of the one to be reproduced, and we reproduce a copy of it sprouting from the opposite vertex of the fixed cathetus. The result of the process can be visualized in figure 3.13



(a) Initial configuration



(b) Green lines are the matched triangle, orange one the reconstructed sides

Figure 3.13: Results of the reconstruction procedure when the towers are mislocalized

Once we have found the four vertices of the rectangle, we have to sort them in order to obtain a proper rectangle. To do so, we simply project the points over the x axes and sort them in ascending order: we know that the two middle points occupy opposite vertices, like the extreme ones: any sequence of the type (*extreme*, *middle*, *extreme*, *middle*) will be a proper sequence for a rectangle: an example can be found in figure 3.14

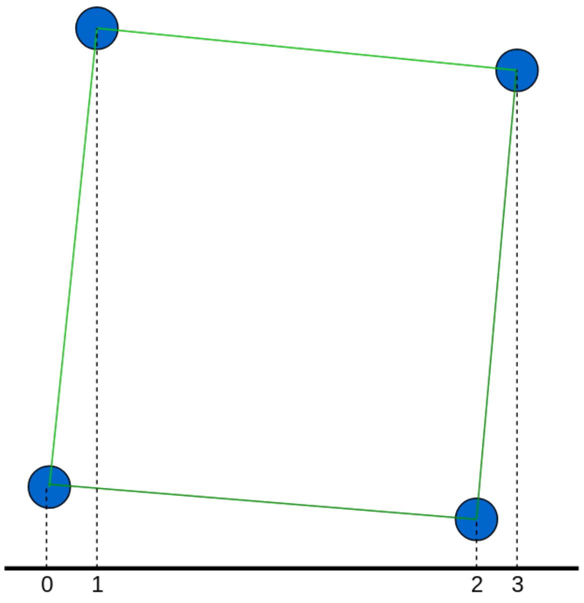


Figure 3.14: Projection over the x-axis. A proper sequence could be (0,1,3,2)

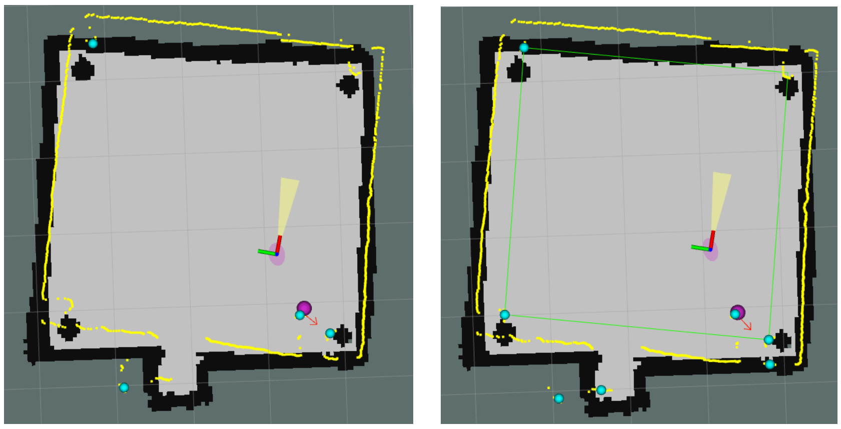

(a) Initial configuration (b) The green box is the reconstructed rectangle

Figure 3.15: Results of the reconstruction procedure when the towers are mislocalized. They should be localized above the black regions

3.4.4 Player Tracking

Localizing the player in the field is a crucial but complex feature of the system. To do so, we relied on the information coming from the laser scan and from the occupancy grid. We tried to match evidences of objects sensed trough the laser with obstacles represented in the occupancy grid from the a priori map. Since the occupancy grid is referred to the map frame, we need to transform the laser scan points from the robot frame to the map frame. We tried to rely on the tf framework provided by ROS but, given the amount of points to be transformed, it often failed: we had to translate them by hand exploiting the information about the robot's pose in the map frame. Once we translated successfully all the points to the map frame, we check them against the occupancy grid: if, given a threshold, the contour of the point in the occupancy grid contained an obstacle, we marked the point as belonging to the a priori map: otherwise, we marked it as a new obstacle and associate it with the presence of the player, under the assumption that the player is the only entity not present in the a priori map.
Again, the flaws of the localization brought noise to this estimate: when the robot is moving too fast, the map may rotate so much that one of the walls is considered to be inside the playground, thus being considered as evidence of the presence of a player. To reduce this effect, we exploited the notion of cylindrical objects inside the field: after matching the scan against the occupancy grid, we refine the result matching the outcome against the cylindrical elements of the field: if what has been marked as a novel object

in the field is not a cylinder, it is discarded. After this second filtering, we are left with an appreciable estimate of what is the laser scan of the player: sometimes it happens that one of the towers is marked as moving obstacle, but does not introduce too much noise.
After all this cleaning procedure, the laser scan associated to the player is fed to a Particle Filter that updates the position of the player. The result of this filtering pipeline can be seen in figure 3.17

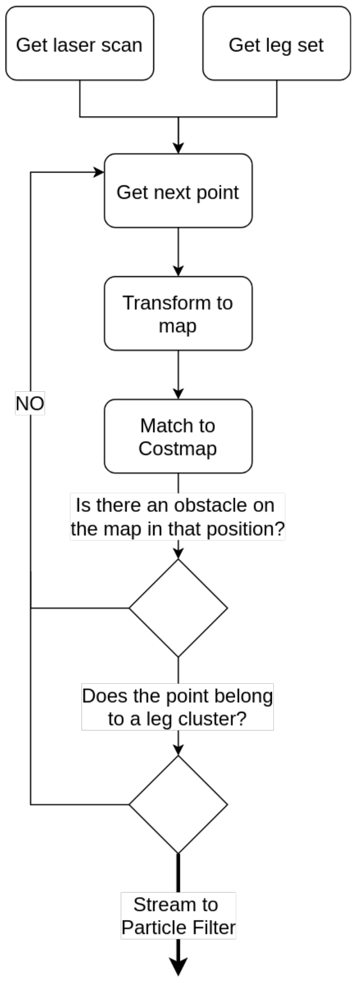
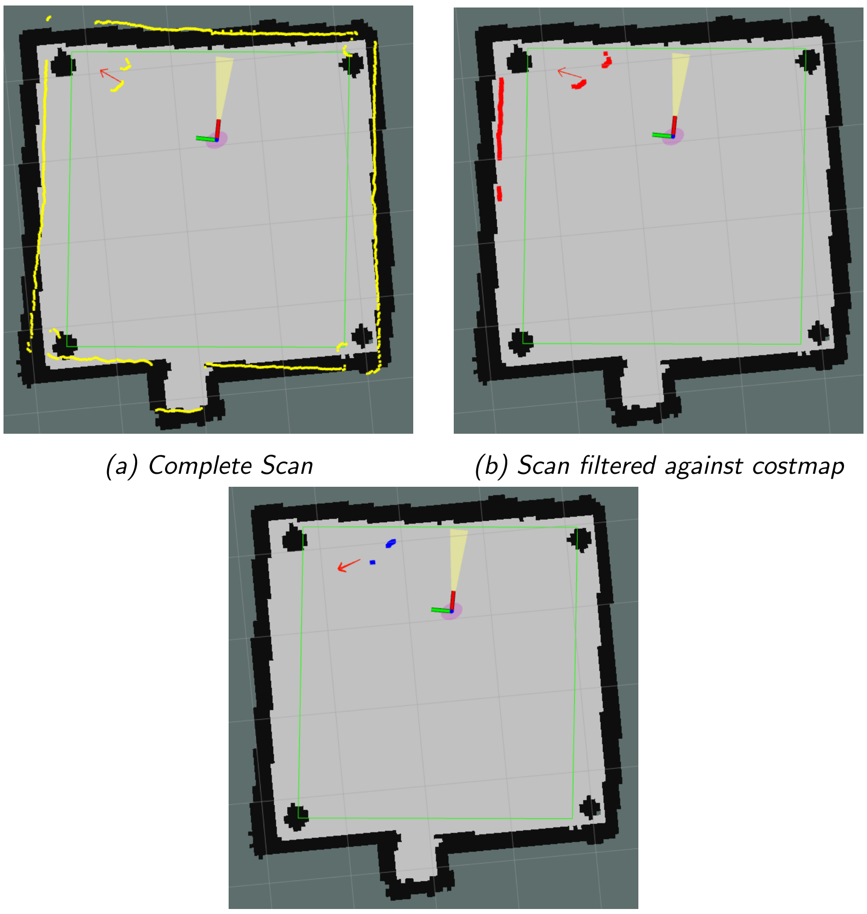

Get laser scan
Get leg set
Get next point
Transform to map
Match to Costmap
NO
Is there an obstacle on the map in that position?
Does the point belong to a leg cluster?
Stream to Particle Filter

Figure 3.16: Control flow to filter laser scan

3.4.5 Obstacle Detection

The nature of the game poses a problem when it comes to obstacle recognition: since we have to collide with the towers, we cannot consider anything coming from the laser scan as an obstacle, but we must avoid only the player.


(a) Complete Scan

(b) Scan filtered against costmap

(c) Scan filtered against legs

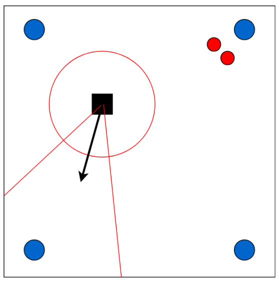
Figure 3.17: Results of subsequent filtering of laser scan to find the player

The work done for the tracking of the player comes in handy in solving this problem. We define a situation as *unsafe* if we think we're going to collide with the player, otherwise we tag the situation as *safe*.
First of all we need the position of the player with respect to the robot frame, so that we can evaluate its orientation with respect to the robot. Then we need the current velocity of the robot, so that we can evaluate the orientation of its motion. If we find that the two orientations are aligned, we perform another check on the situation: we take the segment of scan relative to the motion direction of the robot and see if there's anything that had been sensed too close. If we don't find anything too close in that segment, we can say that the situation is *safe*, otherwise we say that the situation is *unsafe*.

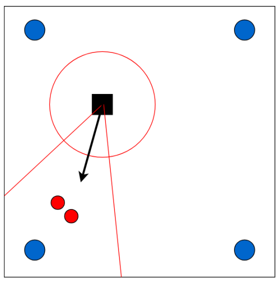
Algorithm 1: Safety Condition Algorithm

```
input  : Player position wrt robot $player$, Robot velocity $vel$
output: Safety condition
player_direction ← atan($player.y$, $player.x$);
motion_direction ← atan($vel.y$, $vel.x$);
if player_direction- motion_direction < $motion\_threshold$ then
    start ← motion_direction − (motion_threshold / 2);
    end ← motion_direction + (motion_threshold / 2);
    motion_scan_segment ← getLaserScanSegment(start, end);
    for $distance$ $in$ motion_scan_segment do
        if $distance < safety\_threshold$ then
            return false;
    return true;
return true;
```

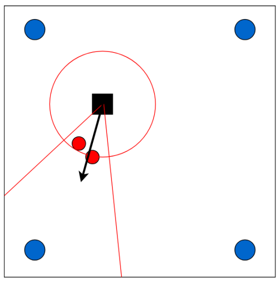
When we find ourselves in an *unsafe* situation, we rollback to saying that the situation is *safe* only when we do not sense anything in the full scan that is closer than the safety threshold.



(a) Safe condition: the player is not in the direction of motion



(b) Safe condition: the player is in the motion direction but distant enough



(c) Unsafe condition: the player is in the motion direction direction and is too close

Figure 3.18: Different possible situation to evaluate safety

Chapter 4

Proposed Adaptive Steering Planning Architecture

Designing a planning system for a PIRG poses some more challenges when compared to a planning system for a non adaptive game. In the latter case the goal of the agent is to win the game: this means it should aim at following an optimal strategy in order to maximize its own outcome. In our scenario the goal of the agent is to offer a pleasant interaction experience to the user: this means that the agent should try to match the user playing style in order to make the game even, rather than play optimally. Given the capabilities of the robotic platform, maximizing the agent's outcome would result in a game too difficult to be matched by a human counterpart. To fulfill these requirements it is necessary to design a system architecture capable of adapting its decision flow and its motion abilities.
In the first part of this chapter we're going to introduce steering behaviors, which are the founding motion patterns used to implement the motion abilities of the platform, then we will present the architecture and the decision hierarchy designed to fulfill the requirements at high level. Lastly we are going to give details about the lower level implementation for our scenario, describing different situations that could occur during the game paired with sample trajectories the decision system would result into.

4.1 Steering Behaviors

Steering behaviors have been proposed first by Reynolds in [17]: they are high level guidelines of motion patterns that give an actor the ability to navigate around their world in a lifelike and improvisational manner. The main idea is to adjust gradually the velocity direction to be parallel to the direction of the target to reach. To do this adjustment forward Euler integration is used: at each simulation step, the steering force to be applied is calculated following the current behavior guideline. Then it is applied to a representation of the vehicle, called vehicle model, which embeds its kinematic properties and its steering capabilities. This application will produce an acceleration vector to be added to the current velocity vector of the vehicle, gradually aligning the velocity direction to the desired target. This is the backbone update procedure that is common in the steering behavior framework:

$$\begin{aligned}
&steering\_force \leftarrow truncate(steering\_direction, max\_force)\\
&acceleration \leftarrow steering\_force/mass\\
&velocity \leftarrow truncate(velocity + acceleration, max\_vel)
\end{aligned}$$

Each steering behavior will then characterize the way the steering direction is calculated, generating a different motion. Here follows a brief presentation of the steering behaviors taken into consideration in our implementation.

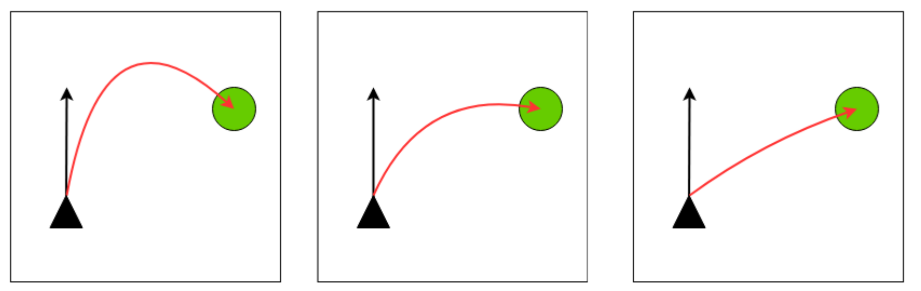
4.1.1 Vehicle model

As previously pointed out, one of the two core components of the steering behavior framework is the vehicle model, that is the way we represent the locomotion platform of our character. Given the fact that our platform is holonomic, we have no steering limitations and we can model it as a point mass. The features describing the vehicle model will then be its mass and its maximum speed. Changes in the vehicle model representation will affect the motion of our robot as well, as displayed in figure 4.1

4.1.2 Seek

Seek (or pursuit of a static target) acts to steer the character towards a specified position in global space. This behavior adjusts the character so that its velocity is radially aligned towards the target. The *desired_velocity* used to calculate the velocity update will be a vector with direction parallel to the straight line passing through the character and the target, and with magnitude equal to *max_speed*.

$$\begin{aligned}
&desired\_velocity \leftarrow normalize(position - target) * max\_speed\\
&steering \leftarrow desired\_velocity - current\_velocity
\end{aligned}$$



(a) Higher mass values will produce long turns and overshooting trajectories *(b) Medium values will produce smoother trajectories* *(c) Lower values will produce crisp and almost straight movements*

Figure 4.1: Effect on the path followed by the character under variations in the mass value for the vehicle model. The black arrow represents the current velocity of the character, the green circle the target to reach and the red line the trajectory followed.

An example of the motion produced by this behavior can be seen in figure 4.2a.

4.1.3 Flee

In opposition to the seek behavior, that enables the character to reach a given target, the flee behavior has been designed: a character driven by this behavior will naturally escape from the target. The *desired_velocity* used to calculate the velocity update will be the same as for seek, but with opposite direction.

$$desired\_velocity \leftarrow normalize(target - position) * max\_speed$$
$$steering \leftarrow desired\_velocity - current\_velocity$$

An example of the motion produced by this behavior can be seen as well in figure 4.2a.

4.1.4 Arrival

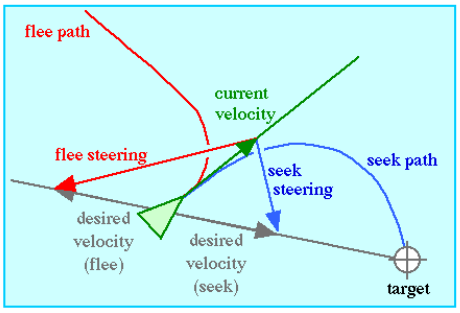
The arrival behavior serves the same purpose of the seek behavior, that is reaching a target, but instead of moving through the target at full speed, this behavior causes the character to slow down as it approaches the target, eventually slowing to a stop coincident with the target. This will prevent the overshoot that often affects the seek behavior. The arrival behavior is parameterized by the *slowing_distance*, the distance from the target at which the character will start slowing down. The *desired_velocity* used to calculate the velocity update will be the same as for seek when the distance from the target is greater than the *slowing_distance*, than its magnitude will be more and more reduced as the character will be approaching the target.

```
$target\_offset \leftarrow target - position$
$distance \leftarrow length(target\_offset)$
$ramped\_speed \leftarrow max\_speed * (distance/slowing\_distance)$
$clipped\_speed \leftarrow minimum(ramped\_speed, max\_speed)$
$desired\_velocity \leftarrow (clipped\_speed/distance) * target\_offset$
$steering \leftarrow desired\_velocity - velocity$
```

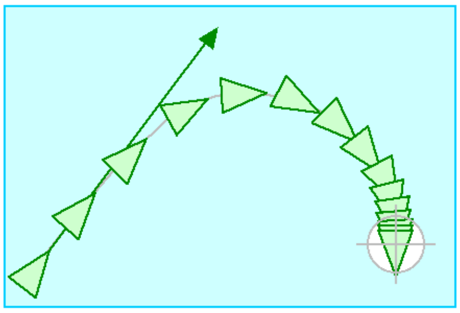
A character moving under the influence of the arrival behavior will follow a path similar to the one presented in figure 4.2b

4.1.5 Evasion

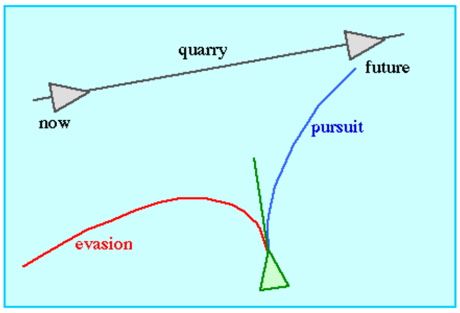
Evasion is similar to the flee, but the target is not fixed: at each time step the target to evade from is updated. This produced a non optimal path to escape, but better resembles natural systems where evasion is non optimal in order to be unpredictable. A visualization of the followed path can be seen in figure 4.2c



(a) Seek and Flee behaviors



(b) Arrival behavior



(c) Pursuit and Evasion behaviors

Figure 4.2: Trajectory followed by an actor under the influence of steering behaviors

4.2 Architecture

One of the most important features of the robot is that it has to show and act following a rational behavior: this means that each movement must be tied to a purpose, an intention that the robot wants to carry out. From this requirement arouses the need for a planning component capable of reasoning at a high level: this component should be in charge of defining the semantic meaning of subsequent decisions to be taken by lower level components that have to translate them into motion.
Each intention can be fulfilled by means of different actions: we need another component to reason at this middle level in order to choose the actions to be then actuated by the robotic platform.
Lastly, we need a component in charge of actuating the action planned at the previous level: it should be able to choose the proper steering behavior to actuate the action and calculate the velocity command to be sent to the system.
These would result in a three layer architecture as represented in figure 4.3 where each layer reasons at a different level of granularity about the interaction with the player.

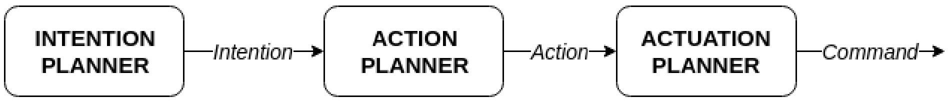


Figure 4.3: Planning pipeline

4.2.1 Intention Planner

The planning procedure at the intention level is regulated by a transition graph like the one shown in figure 4.4. Each state, given the current state of the game, can generate an intention between the ones allowed by the graph. At each iteration, we evaluate whether there should be a transition between states: if so, we change the current state and generate the new intention from it. In case no state transition happens, we evaluate if the current intention is still running: if it is no longer running, i.e. has been aborted or completed, we generate a new intention from the current state. The pseudocode for the planning procedure is presented in algorithm 2

Algorithm 2: Intention planning routine

```
Data: gameState
Result: Intention to carry out
initialize the intention graph;
set the initial currentState;
generate initial currentIntention;
while game is running do
    get current gameState;
    if state transition is triggered then
        update currentState;
        generate currentIntention from currentState;
    else
        if currentIntention is not running then
            generate currentIntention from currentState;
    publish currentIntention;
```

4.2.2 Action Planner

The planning procedure for actions should listen to the intention planned at the higher level of the hierarchy and plan the actions to be carried out. It hosts a lookup table to filter the actions meaningful for the current intention

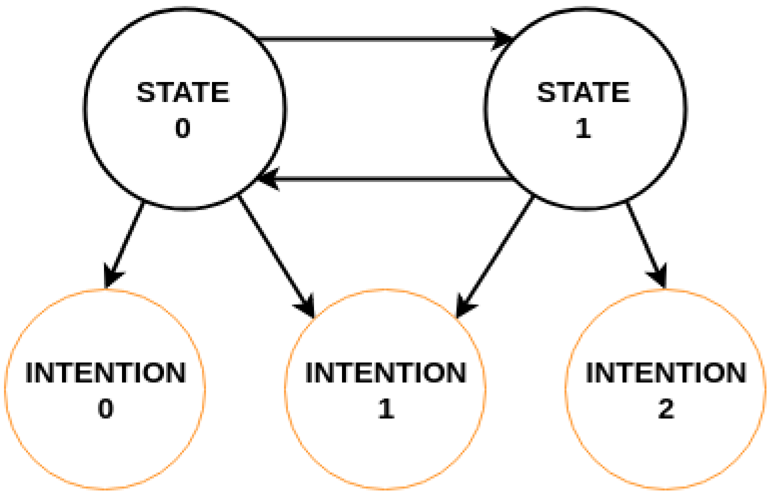


Figure 4.4: Intention planning transition graph example

between all the possible ones. The pseudocode for the planning procedure is presented in algorithm 3

Algorithm 3: Action planning routine

```
Data: gameState, currentIntention
Result: Action to actuate
initialize the intention-action lookup table;
wait for first intention;
while game is running do
    get current gameState;
    get currentIntention;
    if currentIntention has changed then
        lookup possible actions for currentIntention;
        choose best action given the gameState;
        publish action;
```

4.2.3 Actuation Planner

The lower level planner takes as input the action to actuate and outputs a stream of velocities command to make the robot move according to a steering behavior. Again, it hosts a lookup table to filter the steering behaviors meaningful for the current action between all the possible ones. Along with that, it holds a representation of the vehicle to be used to generate velocity commands according to steering behaviors. The pseudocode for the main routine can be found in algorithm 4.

Algorithm 4: Actuation planning routine

```
Data: gameState, currentAction
Result: Steering behavior to follow
initialize the action-behavior lookup table;
initialize vehicle representation;
wait for first action;
while game is running do
    get current gameState;
    get currentAction;
    if currentAction has changed then
        lookup possible behaviors for currentAction;
        choose best behavior given the gameState;
        set behavior to the vehicle representation;
    generate command given vehicle representation;
    publish command;
```

4.3 Implementation

We implemented a version of this architecture with states, intentions, actions and behaviors limited to the ones necessary to satisfy the needs of our research and to have a working game. The cascading pipeline decision process in the end could be summarized by the decision tree shown in figure 4.5. In the following paragraphs we're going to discuss the elements and decision in detail.

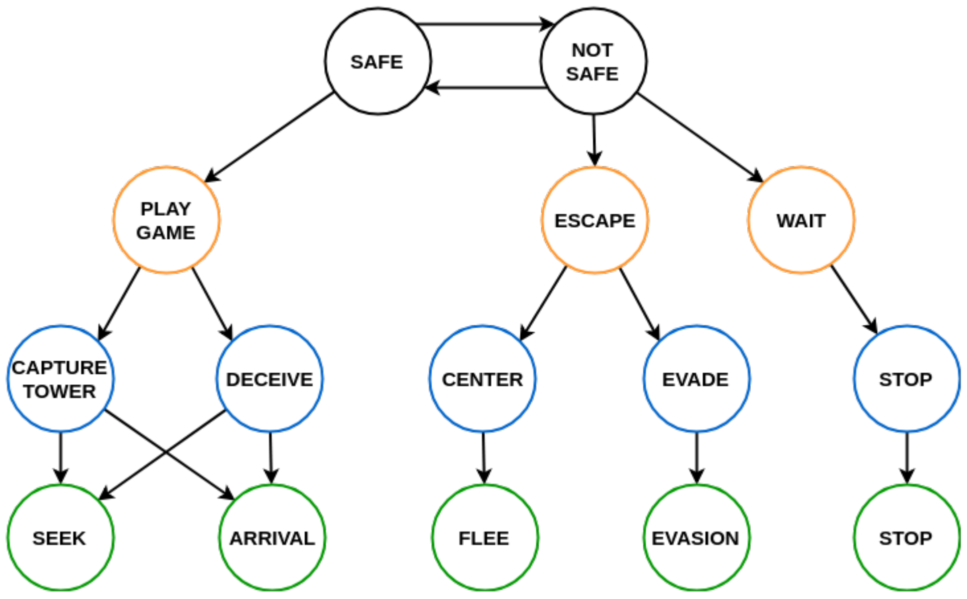


Figure 4.5: Decision tree derived from game movements planning. The Intentions are highlighted in orange, the Actions in blue and the Steering behaviors in green

4.3.1 State to Intention

The transitions between states are driven by safety decision: we would go from a *Safe* state to a *Not safe* state if we detect an imminent collision with the player. We would remain in a *Not safe* state until there is no obstacle (either player or wall) near the robot. The mapping between each safety state and the intentions they are able to produce in the following table.

STATE	**INTENTION**
Safe	PlayGame
Not safe	Escape Wait

Table 4.1: State to intention mapping

When we're in a *Safe* state, we always carry out the intention to play the game. When we're on a *Not safe* situation, we'll initially carry out the *Escape* intention, with the object of restoring a safe situation: if after a timeout of 3 seconds we're still in a *Not safe* situation, we'll switch to the *Wait* intention until the player gets far enough.

4.3.2 Intention to Action

We limited the set of actions to those useful to fulfill our implemented intentions, that are: *Capture Tower*, *Deceive*, *Center*, *Escape*, *Stop.* The mapping between intentions and actions can be found in table 4.2.

INTENTION	**ACTION**
PlayGame	CaptureTower Deceive
Escape	Center Evade
Wait	Stop

Table 4.2: Intention to action mapping

CaptureTower represents a direct attack to a given tower: each tower is assigned a utility value using the following function:

$$\begin{aligned} utility_i = & advantage\_factor_i * blocking\_factor_i - \\ & - led\_factor_i - recentness\_factor_i \end{aligned}$$

The $advantage\_factor_i$ takes into consideration the advantage that the robot has over the player to reach tower i. This is simply the difference of their distances from the tower, which is going to be positive if the robot is closer to the tower than the player, negative otherwise.

$$advantage\_factor_i = player\_distance_i - robot\_distance_i$$

The $blocking\_factor_i$ is going to represent how much the position of the player is blocking the robot's way towards tower i

$$blocking\_factor_i = \frac{alpha\_player - alpha\_tower_i}{\pi}$$

Where $alpha\_player$ is the angle between the robot and the player, and $alpha\_tower_i$ is the angle between the robot and the tower. The $blocking\_factor_i$ could take values between 0 and 1, where 1 means not blocked at all and 0 means highly blocked.

The led_factor_i is going to account for the completion state of a tower, since the complete ones should not be attacked by the robot.

$$led\_factor_i = -100 * (leds\_on_i == 4)$$

The $leds\_on_i$ is the number of leds currently on for tower i.

The $recentness\_factor_i$ has been introduced to avoid constant repetition in the choice of the same towers by the procedure: we noticed that the robot often kept going along the diagonal (being the most advantageous choice) resulting in a boring and highly predictable behavior. Thus we decided to always penalize the last tower that was chosen as a target and to penalize with a probability of $\frac{2}{3}$ the second to last.

$$recentness\_factor_i = -100 * (i == last\_target) -$$
$$-50 * (i == second\_to\_last\_target) * (random\_float < \frac{2}{3})$$

Where $random\_float$ is a real number randomly generated between 0 and 1. The penalty for the second to last tower will be dropped when there will be only two towers left.
An example of the trajectory followed by this action and of the results of this utility function can be seen in figure 4.6.

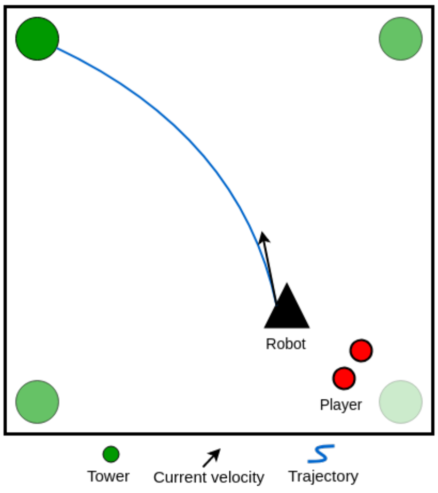


Figure 4.6: The tower with the brightest color has the highest utility

Deceive represents an attack to a tower after a fake attack to another one. The robot will be initially headed towards the fake target and then it will change target accelerating a little towards the real one, following a path like the one showed in figure 4.7.

To decide between a *Deceive* action and a *Capture tower* we'll look at the utilities of towers: if we see that there are two tower with approximately the same utility, we'll choose a the first one. If this situation does not occur, we choose the latter action.

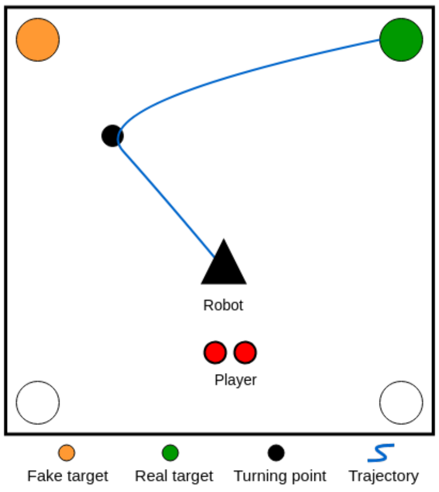
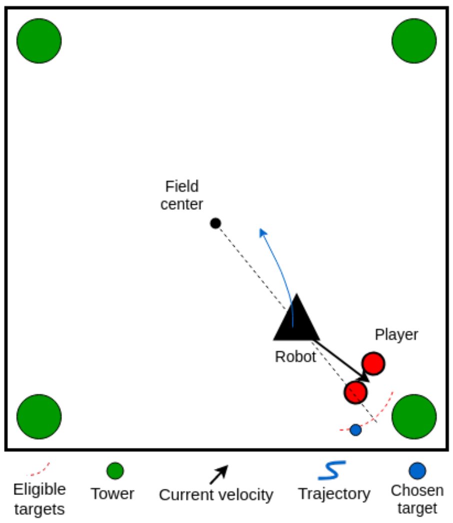
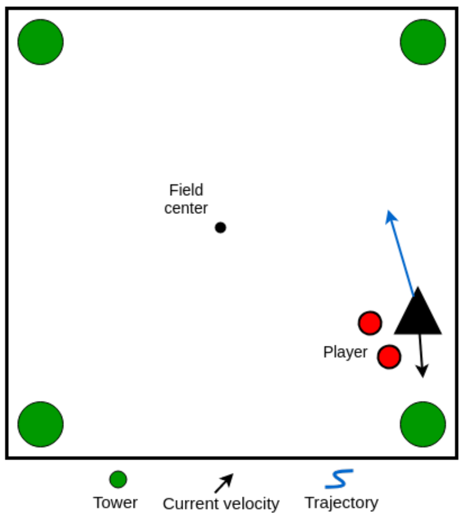


Figure 4.7: Example of a deceptive trajectory

Escape intention should lead to the simulation of a getaway by the robot when it's facing a dangerous situation, trying to restore a safe situation. We consider two actions for this intention: in the first one, called *Center*, the robot will try to get closer to the center of the field, where it will have more room for manoeuvre; in the second one, called *Evade*, it will just escape from the obstacle closest to it.
To determine which is the most suitable action, we take a look at what is surrounding the robot: if the way towards the center is not obstructed, we will choose the *Center* action, which will be implemented in the following way. Firstly, we define a set of targets eligible to be the ones to flee from: this set is composed by points distributed along an arc centered in a point in the opposite direction with respect to the center, as shown in figure 4.8a. Then the target is chosen randomly, in order to avoid repetitive and predictable movements. In case this behavior is not feasible, we will choose the *Evade* action, where we simply run away from the closes obstacle. A sample of a situation where we would choose it and a possible trajectory are shown in figure 4.8b.



(a) Example of a trajectory for the Center action



(b) Example of a trajectory for the Evade action

Figure 4.8: Difference between actions tied to the Escape intention

4.3.3 Action to Steering Behavior

We decided to characterize the robot motion through the usage of steering behaviors. In order to perform the actions described above, we implemented the *seek*, *arrival*, *flee*, *evasion* and *stop* behaviors; a detailed description of these behaviors had been given at the beginning of this chapter. The mapping between actions and behaviors can be found in table 4.3.

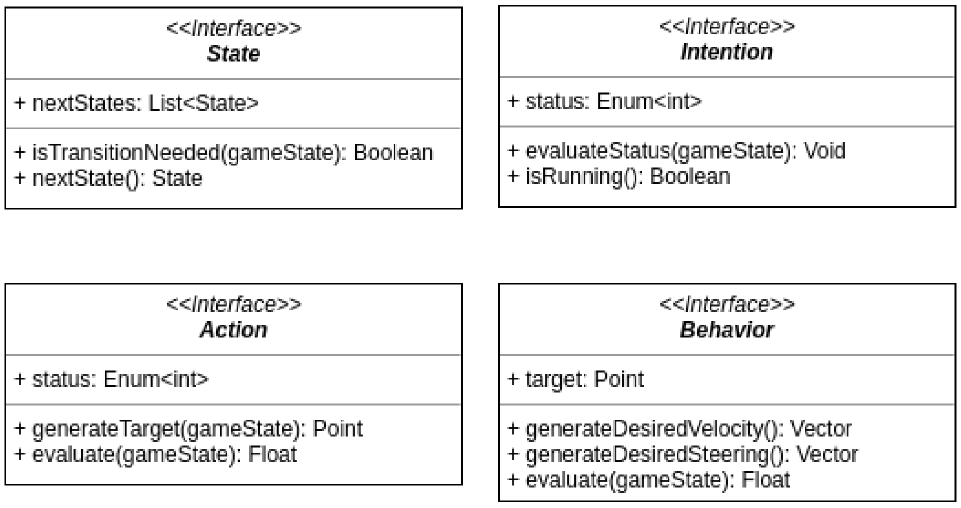
ACTION	**BEHAVIOR**
CaptureTower	Seek Arrival
Deceive	Seek Arrival
Center	Flee
Evade	Evasion
Stop	Stop

Table 4.3: Action to behavior mapping

In order to choose between *seek* and *arrival* behaviors, we'll simply look at the maximum speed currently set for the robot: in presence of a higher speed, we will select the *arrival* behavior, while we'll go with the *seek* one with lower speed values. The reason behind this decision is a merely safety issue. We experienced that most of the collisions between human and robot happen when the human gets suddenly in the way of the robot attacking one tower: thus, if we use an *arrival* behavior, decelerating a little when approaching the tower, it will be easier to change direction if the player suddenly steps in the way. For lower speed values this adjustment is not necessary to avoid collision.

4.4 Extending the planner

Given the design of the planning system, it is easy to extend the capabilities of the robot adding new states, intentions, actions or behaviors. We exploited inheritance and polymorfism to reason at a high level of abstraction: we designed the interfaces these objects must implement in order to be taken in consideration by the planner without changing the planning core loops. The only change required would be a remap of the lookup tables defining intention to action mapping and action to behavior mapping. As an example, if we wanted to add a new intention to interact with the player, not finalized to play the game, we would just need to define the condition upon which the *Safe* state could output the *Interact* intention. Then we would define the actions to fulfill this intention. For example, if we would like to follow the player movement, and the steering behavior to implement such action, we can simply use the interfaces in figure 4.9



<<Interface>>
State
+ nextStates: List<State>
+ isTransitionNeeded(gameState): Boolean
+ nextState(): State
<<Interface>>
Intention
+ status: Enum<int>
+ evaluateStatus(gameState): Void
+ isRunning(): Boolean
<<Interface>>
Action
+ status: Enum<int>
+ generateTarget(gameState): Point
+ evaluate(gameState): Float
<<Interface>>
Behavior
+ target: Point
+ generateDesiredVelocity(): Vector
+ generateDesiredSteering(): Vector
+ evaluate(gameState): Float

Figure 4.9: Required interfaces

After this sample extension, the decision tree would look like figure 4.10. All this would not require changes in the main loops of the components, if not for the initialization that should initialize the new items as well.

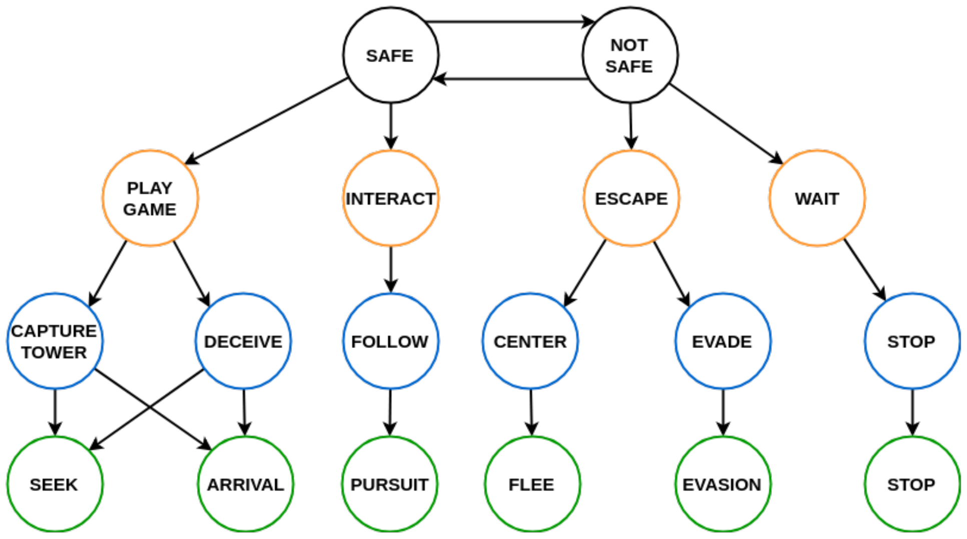


Figure 4.10: Decision tree after the introduction of the Interact intention

Chapter 5

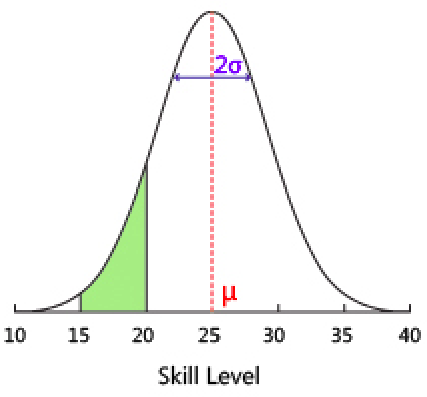
Adaptive difficulty selection procedure

This chapter will describe the procedure used in order to change the exhibited ability of the robot. The purpose of our research is to find a proper way to select a motion profile for the robot in order to maximize the engagement of the human player: given the assumption that a game is more engaging when even, our purpose becomes to find the motion profile that maximizes the probability to draw the game.
Finding such a profile in a continuous space introduces unnecessary overhead difficulties: the range of speed we can allow is so narrow that small changes would not be relevant. Thus, we defined three sets of parameters which are going to characterize three motion profiles for the robotic platform.
To help us in identifying the most suitable configuration for the player, we exploit and adapt to our needs the TrueSkill ranking system.

5.1 TrueSkill

TrueSkill ranking system is a skill based ranking system for Xbox Live developed at Microsoft Research. The purpose of a ranking system is to both identify and track the skills of gamers in a game (mode) in order to be able to match them into competitive matches. The TrueSkill ranking system only uses the final standings of all players in a game in order to update the skill estimates (ranks) of all gamers playing in this game.
This system models the skill of the player using two numbers:

- μ: the average skill of the player
- σ: the degree of uncertainty in the player's skill



Measuring the degree of uncertainty in the player's skill enables the system to perform updates that will have an high impact on the ranking during the initial phase, when the uncertainty is high, and to reach a stable ordering when the uncertainty is lower.
Starting from this representation, this system is able to calculate the hypothetical chance of drawing a game between two players given its current belief.

5.1.1 Skill update

After each game, the system updates its belief about the players involved according to its beliefs about the skills before the game. As we can see from equation 5.1, the winner is going to get an increment of a multiple of $\mathrm{v}\left(\frac{\mu_{\text{winner}}-\mu_{\text{loser}}}{c}, \frac{\epsilon}{c}\right)$, while the loser is going to get a decrement of a multiple of the same factor.

$$\begin{aligned}\mu_{\text{winner}} \leftarrow \mu_{\text{winner}} + \frac{\sigma^2{}_{\text{winner}}}{c} * \mathrm{v}\left(\frac{\mu_{\text{winner}} - \mu_{\text{loser}}}{c}, \frac{\epsilon}{c}\right) \\ \mu_{\text{loser}} \leftarrow \mu_{\text{loser}} - \frac{\sigma^2{}_{\text{loser}}}{c} * \mathrm{v}\left(\frac{\mu_{\text{winner}} - \mu_{\text{loser}}}{c}, \frac{\epsilon}{c}\right)\end{aligned} \tag{5.1}$$

To have a better understanding of what this update equations imply, we should focus our attention on what the difference $\mu_{\text{winner}} - \mu_{\text{loser}}$ tells us. If the believed mean skill of the winner was higher than the loser's one, the outcome matched our expectations: the function v would output a small increment in skill for the winner along with a small decrement for the loser. On the contrary, if $\mu_{\text{winner}} - \mu_{\text{loser}}$ was negative, meaning that we believed that the loser was a better player than the winner, the function v would output a major increment in the mean skill for the winner along with a

great decrement for the loser.
The uncertainty σ over the mean skill μ is used as a weighting factor in the update process: beliefs with higher uncertainty will get higher increments on equal values computed by v.

Regarding the updates for the uncertainty σ over the mean skill μ, it will depend again from the matching between expected and observed outcome: if there's a match, both the uncertainties will be reduced by an higher factor; if the observed outcome is different from the expected one, the uncertainties will decrease less. The equations are shown in 5.2.

$$\begin{aligned} {\sigma^2}_{\text{winner}} &\leftarrow {\sigma^2}_{\text{winner}} * \left[1 - \frac{{\sigma^2}_{\text{winner}}}{c^2} * \mathrm{w}\left(\frac{\mu_{\text{winner}} - \mu_{\text{loser}}}{c}, \frac{\epsilon}{c}\right)\right] \\ {\sigma^2}_{\text{loser}} &\leftarrow {\sigma^2}_{\text{loser}} * \left[1 - \frac{{\sigma^2}_{\text{loser}}}{c^2} * \mathrm{w}\left(\frac{\mu_{\text{winner}} - \mu_{\text{loser}}}{c}, \frac{\epsilon}{c}\right)\right] \end{aligned} \tag{5.2}$$

In all the equations the c factor is

$$c^2 = 2\beta^2 + {\sigma^2}_{\text{winner}} + {\sigma^2}_{\text{loser}}$$

where the β^2 factor represents the variance of the performance around the skill of each player, and its default value is 4.16. Moreover, ϵ is the aforementioned draw margin which depends on the game mode. The plots of the functions v and w for varying values of $\frac{\epsilon}{c}$ can be found in figure 5.1

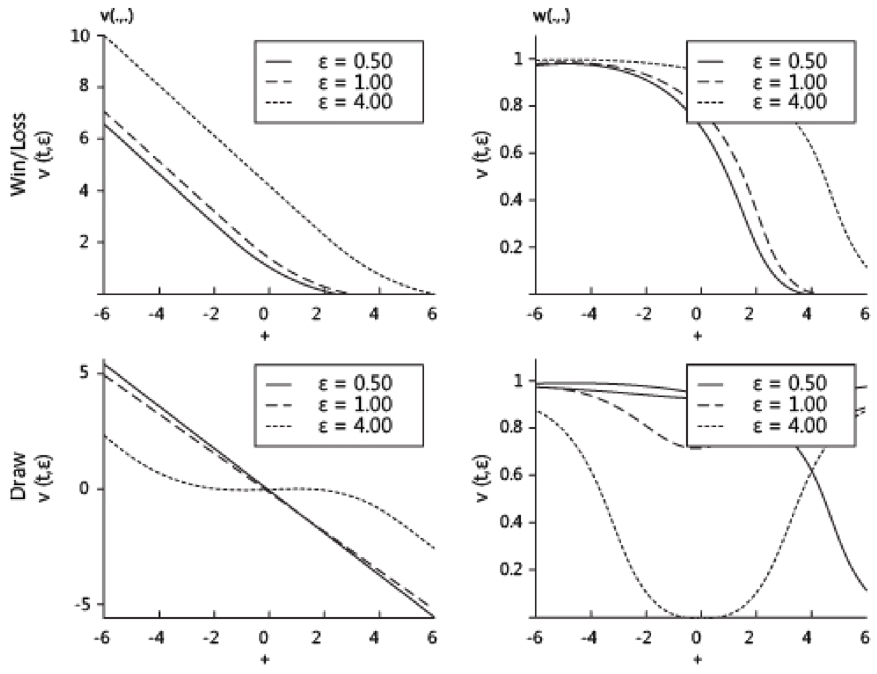


Figure 5.1: Different plots for functions v and w given different $\frac{\epsilon}{c}$ parameters

5.2 Approach idea

The TrueSkill ranking system was developed to be used in a setting where two players are playing the same game against each other. In our scenario, we have a slightly different setting: the goal of the robot is different from the goal of the player, but we can still treat this as an adversarial game where we can elect the winner. If we think of the robot playing with each different set of parameters describing a motion profile as a different opponent for the human player, we can apply the TrueSkill approach to model the skill corresponding to each set of parameters and to select the one with the highest chance of drawing the game against the current player.

Before applying this approach to the physical game, we built a simulation environment that could match the characteristics of our scenario. The environment we needed should have had the following three characteristics:

1. Two opponents competing to reach two separate goals
2. The possibility to modify the ability of one of them

3. The possibility to define winning, losing and drawing situation

Given an environment with such characteristics, in order to simulate our scenario we need a set of players that are able to compete autonomously in the game with a different level of ability.
The purpose of this environment was to run simulations to gain insights about our approach, to build an estimate of its quality before applying it to the physical robot.

5.2.1 Environment

The OpenAI Gym project is a well known environment to build and train agents to solve a wide variety of tasks. Going trough the environments offered by the projects, we identified the BipedalWalker-v2 environment as the most suitable for our needs. In this environment a bipedal agent has to travel a given distance without loosing its balance: the path the agent has to face is created at run time and could feature hills, valleys, stairs and steps.
While this environment is mainly used and studied as an environment to train an agent, we should change our point of view and interpret it in a different way. We should see it as a game between two agents: one is the walking agent and the other one is the agent in charge of creating the path. The goal of the walking agent is to maintain the balance and proceed along the path, while the goal of the path agent is to make the walking agent fall. From here, we define win, draw, and loss situations for both agents: the walking agent wins if he succeeds in reaching the end of the path without falling; the game is drawn if the walking agent completes half of the path and then falls; the path agent wins if the walking agent falls before the mid point of the path.
To complete the simulation environment we need a set of walking agents that can play the game autonomously with different performances and a way to characterize and control the ability of the path agent.

5.2.2 Path agent

The path agent is the one we want to control trough a set of parameters, simulating the robot in our scenario. We have the ability of controlling what kind of obstacles the agent is able to create along the path. We defined three sets of parameters that enable the path agent to:

1. Create no obstacle - Easy configuration

2. Create hills and valleys - Medium configuration
3. Create stairs and steps - Hard configuration

This is also an ordering in terms of performance of the agent, having in mind that the goal is to make the walking agent loose its balance. Examples of the types of paths that the agent could create are shown in figure 5.2

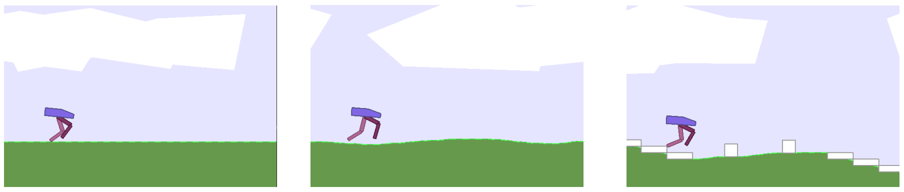

(a) Path created with no obstacle (b) Path created with hills and valleys (c) Path created with steps and stairs

Figure 5.2: Different paths created by the path agent

5.2.3 Walking agents

The walking agents will have to play the role of human players in our scenario. We need a set of players that are able to play with different levels of performance. To create this set, we followed a reinforcement learning approach to train several walking agents.
To ensure a tangible difference in the performance, we used two measures: first, we trained the walking agents against each parameter characterization of the path independently, taking the assumption that an agent trained on a path with no obstacles would fail to face a stair or a step; second, we saved the model produced every 10000 training steps, taking the assumption that more training steps would produce a better performance.

5.2.4 Ground truth

Now that we have an adequate simulation environment that reproduces the distinctive features of our scenario, we can build a ground truth to evaluate our approach for the selection of the best set of parameters to characterize the ability of the agent opposing to the player. Given the assumption that a game is more engaging when even, we define the best set of parameters as the one yielding to the highest drawing rate of a game. To identify it, for each walking agent we played several games against each parameter characterization of the path agent and collected the outcomes. The results of the draw rates can be found in figure 5.3

Now that we know the set of parameter that maximizes the draw rate for each walking agent, we can evaluate the accuracy of our approach in identifying it.

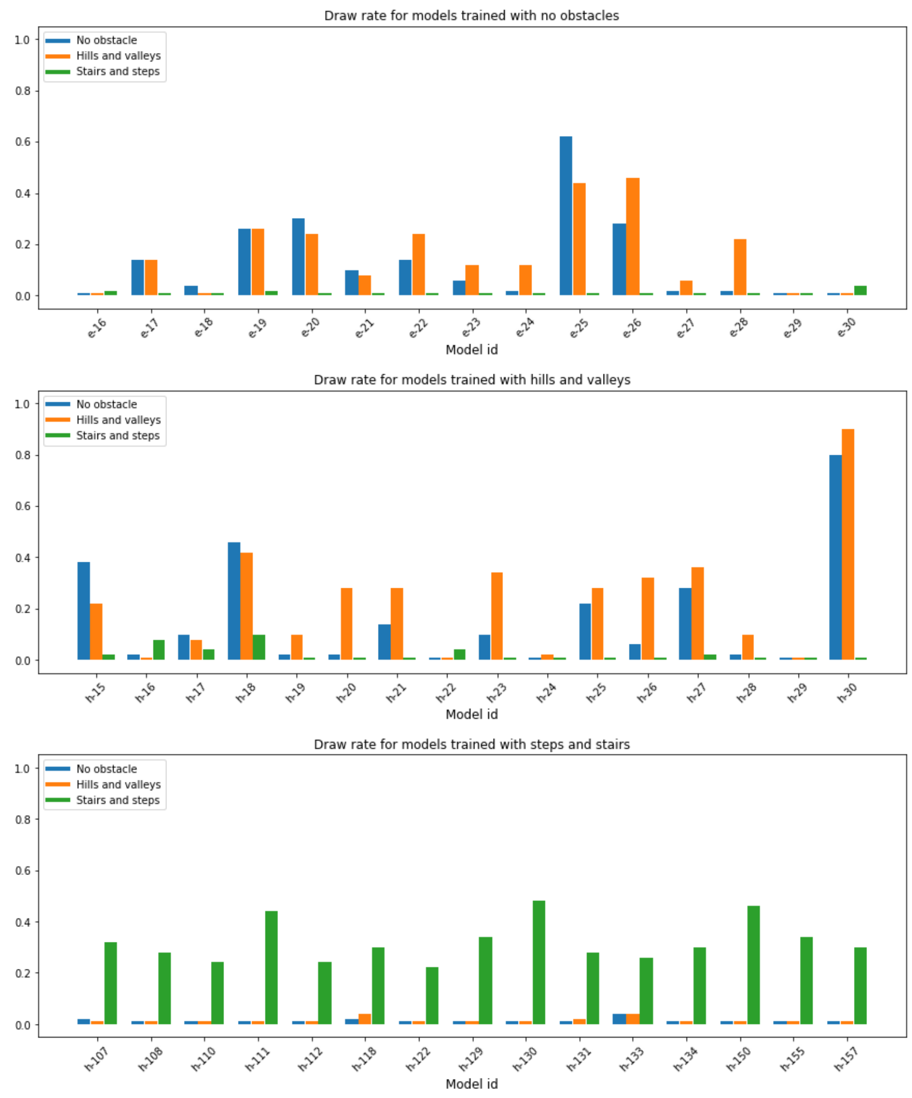


Figure 5.3: Here are displayed, for each walking agent, the draw rate against each parameter characterization of the path agent, represented by the corresponding color. The walking agents have been divided into three sets with respect to the characterization of the path agent they had been trained against

5.2.5 Simulation procedure

Before discussing the approaches we tried for the online selection of the parameters, it's worth presenting the simulation procedure followed and explain a bit how to read and interpret the produced plots which vehicle a lot of information.
We iterate the same routine over 30 games: before each game, we select the parameters according to the devised procedure; we play the game, update the beliefs of the system according to the outcome and collect their new values. After collecting these values after each game, we plot the sequence to evaluate the evolution of beliefs through time, producing a plot with two components like the one reported in figure 5.4.

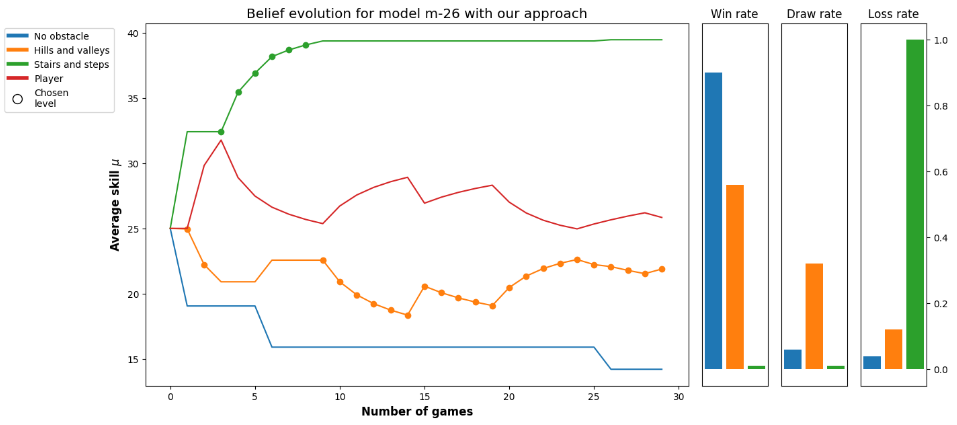


Figure 5.4: Sample plot for explanation purposes

On the left component, we will plot the evolution of the belief regarding the mean skill μ of both the walking agent and the path agent's characterizations. An example of the plot presented in this component can be found in figure 5.5. Along the x axis we find the timeline, expressed by means of number of games played, while along the y axis we have the estimated mean skill μ after n number of games. The colored dots represent the configuration chosen to characterize the path agent for the given game, with the same color code as the one showed in the legend. In order to read each decision, we should slice the plot vertically. Let's have a look at what happens with the first game: we added the arrows to the plot to highlight the changes in the belief. Before playing, the estimated mean skills for each characterization are approximately:

- 33 - Stairs and steps
- 25 - Hills and valleys
- 25 - Player
- 19 - No obstacle

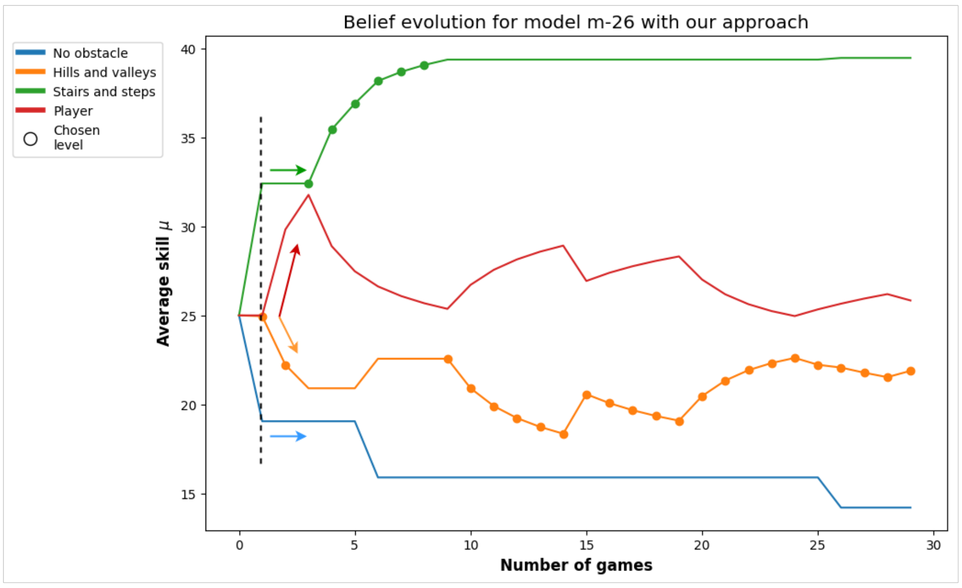


Figure 5.5: Focus on the belief update after the first game

The procedure selects the *Hills and valleys* configuration as the most appropriate to characterize the path agent, as the orange circle represents. After the game, we see an increase in the estimated skill for the player and a decrease in the *Hills and valleys* configuration one, highlighted by the arrows for explication purposes. This means that the player won this particular game. Since the *Stairs and steps* and *No obstacle* configurations did not participate, their skill estimates are not updated. For the second game, again the *Hills and valleys* had been chosen, while from the third one the selected one was the *Stairs and steps*.

The left component of the plot is the representation of the ground truth built before the experiment. It represents the win, draw, and loss rate of the walking agent against each configuration of the path agent. To give an example, let's look at figure 5.6. The color code is the same used in the belief plot, being blue for *No obstacles*, orange for *Hills and valleys* and green for *Stairs and steps*. In this case, we can see that the walking agent has approximately a win rate (first bar graph) of 0.9 against the *No obstacles* configuration, a win rate of 0.6 against the *Hills and valleys* and a win rate of 0.0 against the *Stairs and steps*. From the plot we can also see that the configuration yielding the highest draw rate (second bar graph) is the *Hills and valleys* one. Having this graph helps understanding the evolution of the

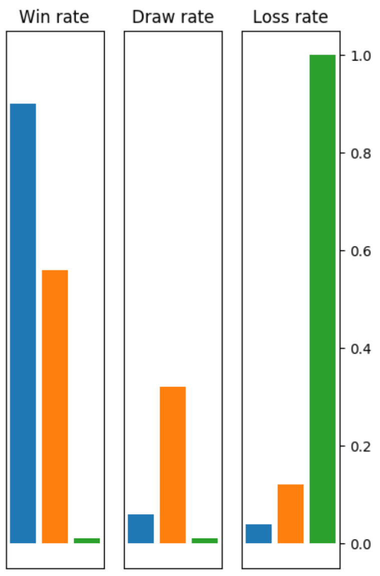


Figure 5.6: Ground truth sample

belief of the mean skill μ, and that's why we present them together: from this one is clear that the *Stairs and steps* configuration is too difficult for this walking agent, and that's why it looses games from 2 to 7.

5.2.6 Baseline approach

First of all, we tried a bare metal approach of the TrueSkill skill representation and update procedure in order to create a baseline to reference subsequent refinements.
If we have no prior knowledge about the player we're currently facing, we have to initialize the database records relevant to the player: to do so, we have to select starting values of σ_{start} and μ_{start}. Given that the belief is represented as a Gaussian, the values that the average skill μ is going to take will fall in the interval $[\mu_{\text{start}} - 3 * \sigma_{\text{start}}, \mu_{\text{start}} + 3 * \sigma_{\text{start}}]$ with a confidence of 99.7%. Usually the starting values in the TrueSkill framework for any players are an average skill μ of 25 and a degree of uncertainty σ of 8.3, so that the expected range of meaningful values for μ is [0, 50]. We stick to these values as well, since they can be arbitrarily chosen. For each player we're going to maintain separate beliefs regarding the skill associated to the parameter settings: this is because the skill of each parametric configuration is estimated with respect to the single player, not globally.

Next, we're going to evaluate the probability of drawing a game given the current beliefs of the skills using the TrueSkill approach. For the baseline approach, we're just going to choose the set of parameters maximizing this probability estimate. After the decision, we set all the parameters of the path agent and run a game against the current walking agent. After the game, we update the beliefs of the two opponents given the outcome of the game, and repeat the procedure. The control flow of this procedure can be found in figure 5.7.

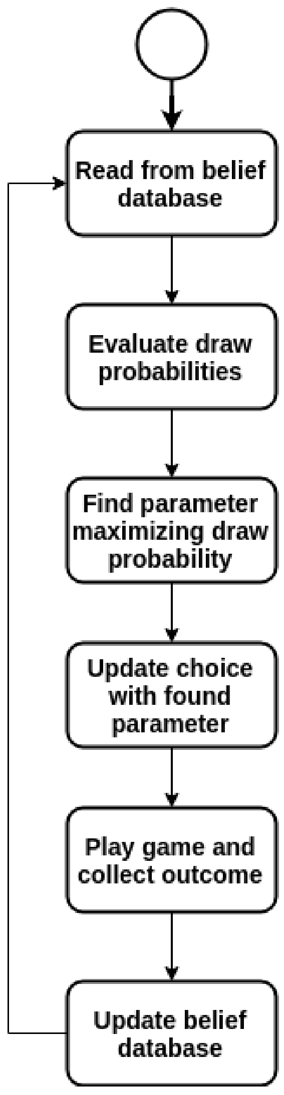

Read from belief database
Evaluate draw probabilities
Find parameter maximizing draw probability
Update choice with found parameter
Play game and collect outcome
Update belief database

Figure 5.7: Control flow for baseline approach to choose the parameters

To understand the evolution trough time of the beliefs of the skills of the involved agents, we can have a look at figure 5.8. On the y axis are represented the average skills of each participant to the game, while on the x axis are enumerated the games played. The colors of the circles represent the parameters chosen for the ith game. As we can see there are some problems about this approach:

- Slow convergence: the decision becomes consistent after about 8 games, which are too many for our scenario, and has an high variance during the early stage
- Wrong result: the set of parameters the system converged to is not the one maximizing the draw rate according to the ground truth we built

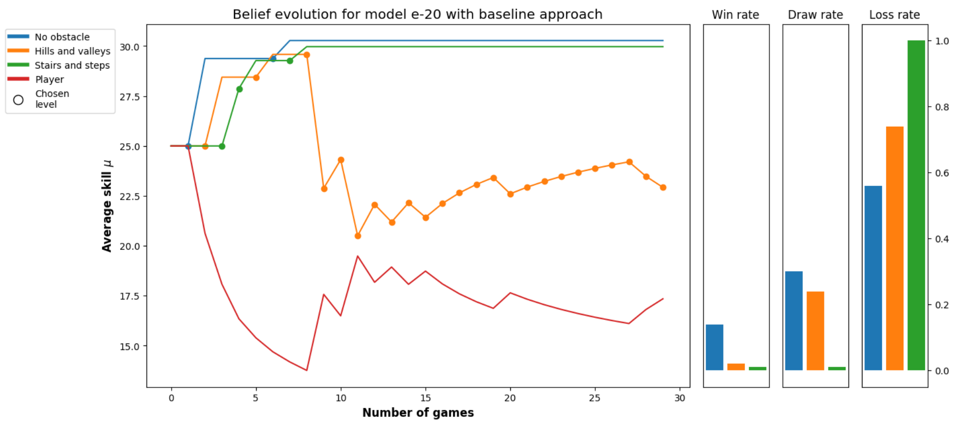


Figure 5.8: Evolution of the belief for the average skill μ about walking agent e-20. On the right side we can see the ground truth for the model expressed in win, draw and loss rate for the walking agent against each parameter characterization of the path agent

The root cause for these results is that the assumption we made about the ordering of path agent skills is not represented in the beliefs of the system: as we can see, the parametric characterization of the path agent that can create hills and valleys has an average skill estimate lower than the one that cannot create any obstacles. But if we look at the ground truth, we see that the assumption is holding: the walking agent has a higher win rate against the path agent that creates no obstacle.
This discrepancy between belief and reality is caused by the basic usage of the TrueSkill framework: the belief updates take in consideration only the skill representation of the two opponents that have just competed against each other. This implies that if two opponents never play a game against each other, they're relative ranking is not reliable. In this scenario, two

path agents are never going to face each other in a game, making the whole ranking unreliable.

5.2.7 Our approach

To address this problem, we had to find a way to infuse this prior knowledge about relative ranking between parameter sets into the belief update procedure. Even if we cannot actually run a game between two path agents with different parameter configurations, we can craft the outcome of a hypothetical game between them to update their relative skills. This procedure would act as a regulator for the average skill of parameter settings, reducing the uncertainty σ related to the estimate, while leaving freedom to the estimate of the skill of the player.
Keeping the same backbone of the baseline procedure, we perform this regulating procedure at a decaying rate during the decision process: we'll do it more frequently at early stages when the estimates are less reliable and repeat it less and less frequently as the reduction in σ will itself prevent abrupt changes in the average skill estimations. The enhanced control flow can be found in figure 5.9.

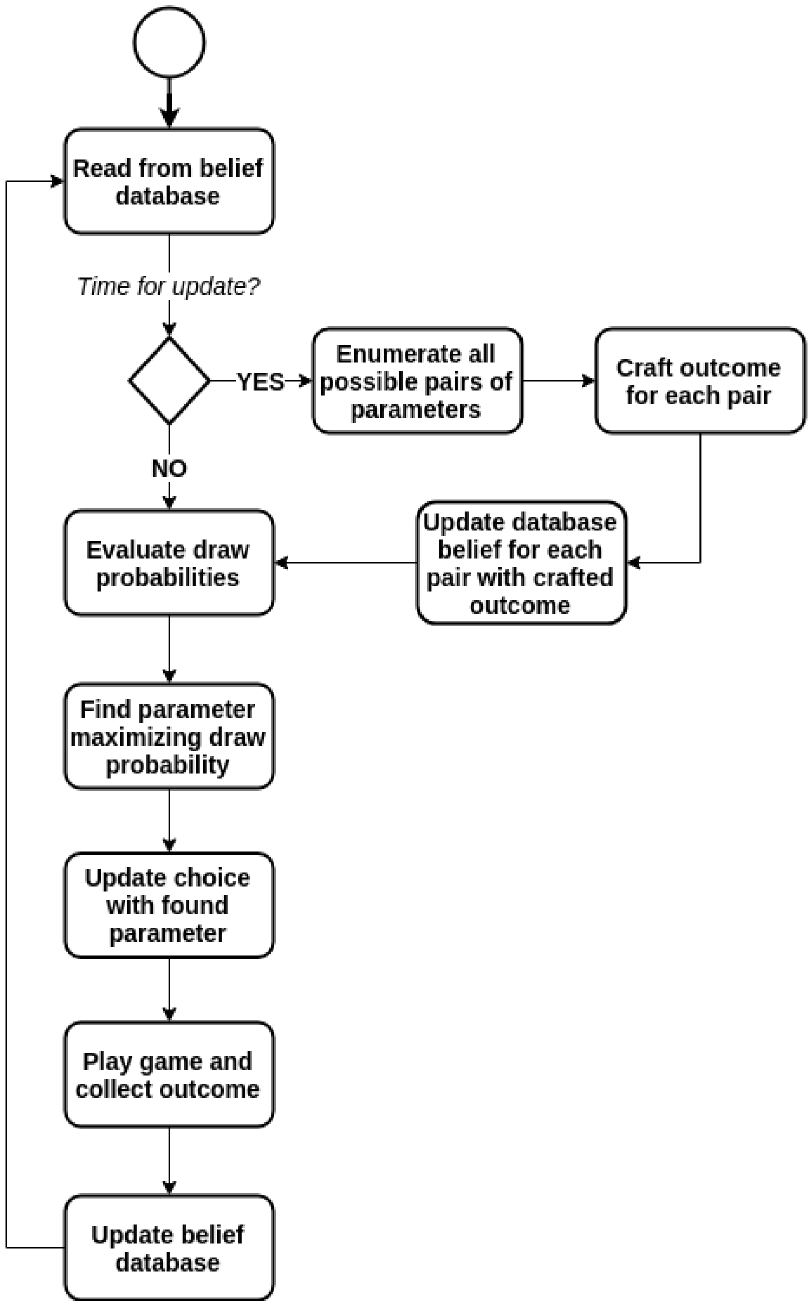

Read from belief database
Time for update?
YES
NO
Enumerate all possible pairs of parameters
Craft outcome for each pair
Update database belief for each pair with crafted outcome
Evaluate draw probabilities
Find parameter maximizing draw probability
Update choice with found parameter
Play game and collect outcome
Update belief database

Figure 5.9: Control flow enhanced with regulation of relative ranking between skills of agent parametric characterization

Figure 5.10 shows the evolution of the system beliefs for the walking agent e-20 using the novel approach: we can see now that the relative ranking between parametric characterizations of the path agent are stable and coherent with the ground truth we have. The arrows highlight the effect of the regularization procedure: if we focus our attention on how the belief is updated before games 0 and 5, we can see that the average skill of each characterization of the path agent is updated according to their relative ranking. This results in an increase of the believed skill of the hardest configuration to beat and a reduction of the skill of the easiest to beat.

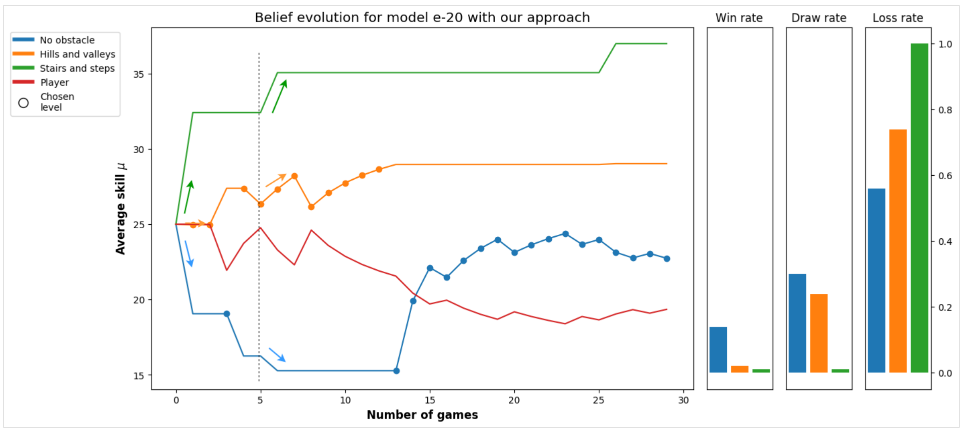


Figure 5.10: Average skill belief evolution for model e-20, with focus on the regularization procedure

The system converged to the correct set of parameters that maximize the draw probability, but still it takes too many games to reach a stable situation. This approach has another drawback that is better visible for another walking agent, number e-16. If we take a look at figure 5.11, we see that the algorithm does not converge to a stable decision. The reason for this becomes clear when we look at the ground truth for this agent: while it outperforms both the path agents able to create no obstacle and hills and valleys, it performs extremely poorly against the one able to create stairs and steps. This means that there is not a suitable set of parameters, among the ones we can consider, to offer an even game for the agent. While we could consider that on average the walking agent would win and loose the same number of games with this approach, in our scenario this would translate into a robot frequently changing playing style: this could transmit a sense of randomness to our users, which is undesired.

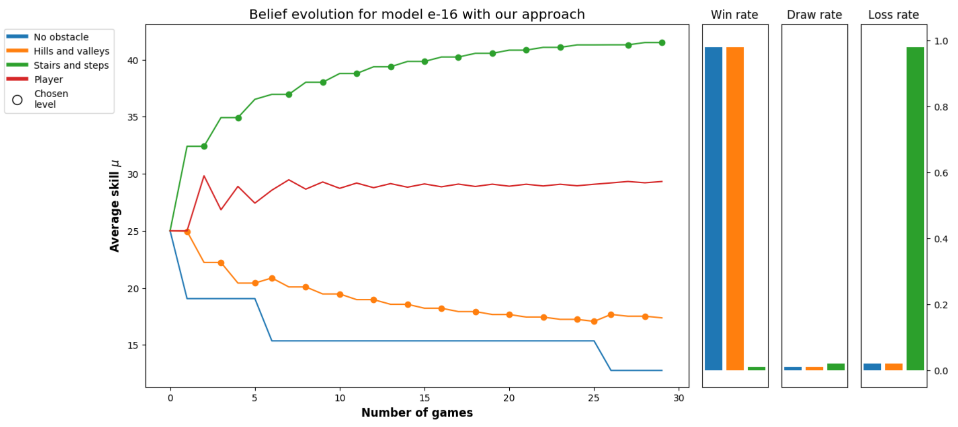


Figure 5.11: Average skill belief evolution for model e-16

5.2.8 Regulating oscillation

We needed a way to regulate changes of decision in order to reduce the oscillating behavior. We are going to change decision if the new choice satisfies two requirements:

1. The estimated probability of a draw with the new parameters is higher than a given threshold with respect to the probability of a draw for the current parameters
2. The uncertainty σ of the skill estimate for the new parameters is higher than a given threshold with respect to uncertainty σ of the skill estimate for the current parameters

The first requirement enforces the fact that we change decision for one that can give a relevant, potential benefit; the second one favours the exploration of a set of parameters that we have been less exploited. The enhanced control flow can be found in figure 5.12

The results of this regulation for model e-16 is shown in figure 5.13: there's clearly more consistency in the decision. Moreover this regulation provided faster convergence for model e-20, as shown in figure 5.14

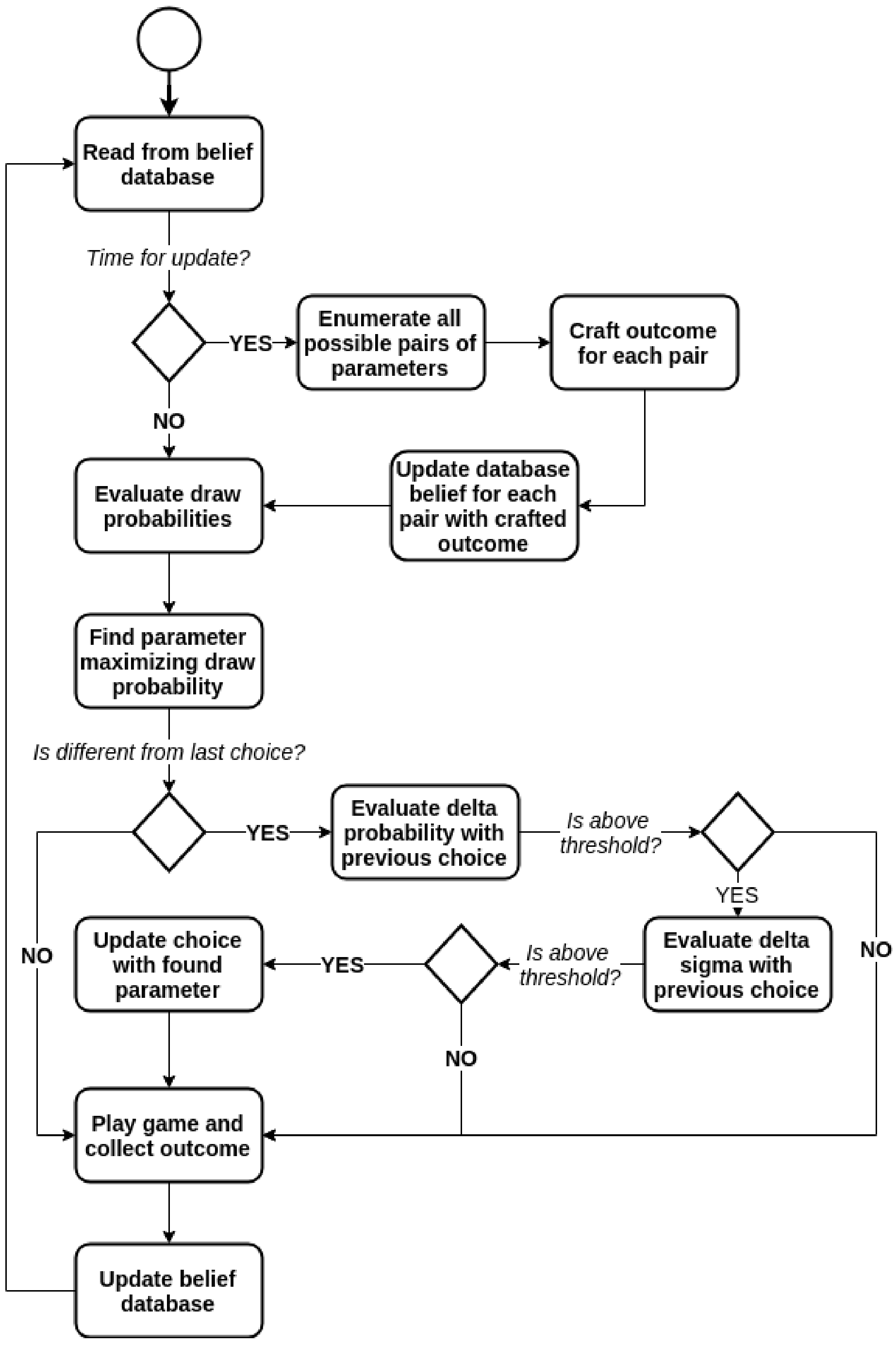

Read from belief database
Time for update?
YES
Enumerate all possible pairs of parameters
Craft outcome for each pair
NO
Evaluate draw probabilities
Update database belief for each pair with crafted outcome
Find parameter maximizing draw probability
Is different from last choice?
YES
Evaluate delta probability with previous choice
Is above threshold?
YES
NO
Update choice with found parameter
YES
Is above threshold?
Evaluate delta sigma with previous choice
NO
NO
Play game and collect outcome
Update belief database

Figure 5.12: Control flow augmented with both regulation for relevant ranking between parametric characterization and for decision change

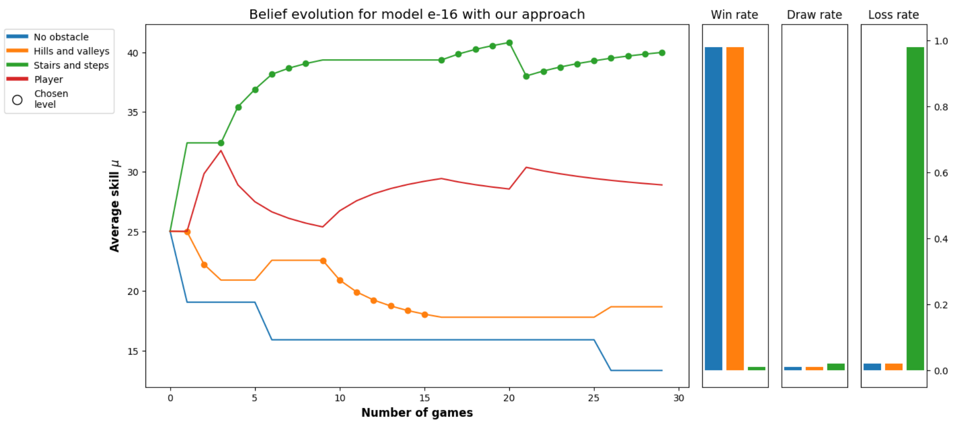


Figure 5.13: Average skill belief evolution for model e-16

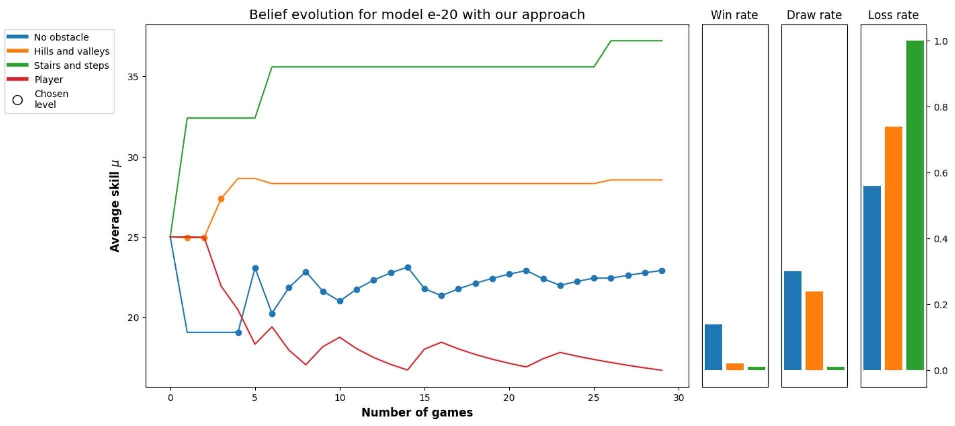


Figure 5.14: Average skill belief evolution for model e-20

5.2.9 Accuracy

Finally, we evaluate the accuracy of the procedure in selecting the correct characterization of the path agent. In figure 5.15 we can see the accuracy expressed as $\frac{number\_of\_correct\_selections}{number\_of\_walking\_agents}$ evolving trough time, as more and more games are played. As we can see from the figure, the regulation of oscillation helps to achieve both higher and faster accuracy when compared to the only regulation on relative ranking. After about 4 games played, the accuracy is around 0.7, meaning that we are able to identify the set of parameter matching the ground truth for 7 walking agents out of 10, and remains more or less stable as the number of played games grows.

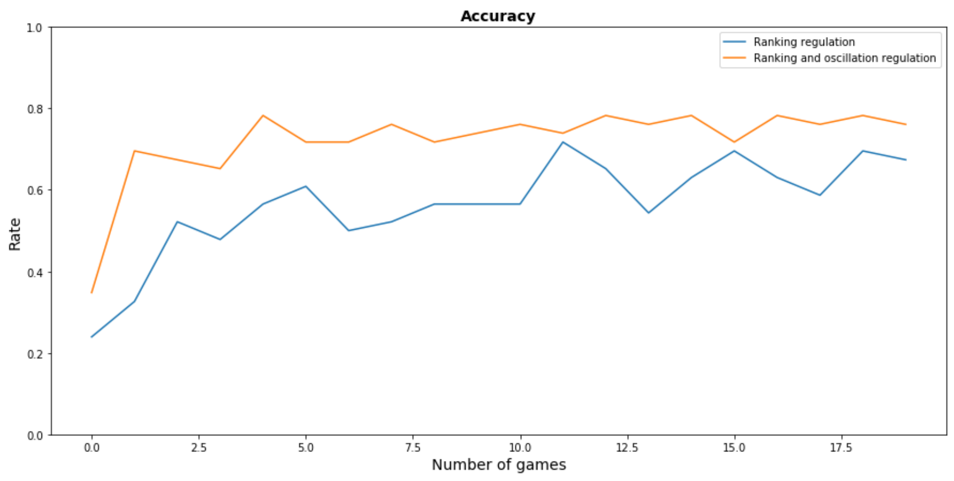


Figure 5.15: Accuracy for the proposed procedures

5.3 Implementation in our scenario

To transfer this approach to our scenario, we had to redefine some concepts due to its intrinsic limitations. Firstly, we have to define how to characterize different abilities for the robot; secondly, we have to define win, draw, and loose situations; thirdly, we want to be able to choose the right parametric characterization for the robot during one game, not after many games.

5.3.1 Robot parameters

Since the motion of the robot is driven by the steering behavior framework, there are two main parameters that characterize its motion profile: the maximum speed and the mass of its representation, introduced in section 4.1.1.

We identified three sets of parameters that can be considered to describe three levels of ability:

Difficulty	**Parameters**
Easy	speed: 0.6 m/s mass: 10
Medium	speed: 0.75 m/s mass: 8
Hard	speed: 1.0 m/s mass: 6

Table 5.1: Different sets of parameters to characterize the ability of the robot.

Lowering the mass of the representation allows a crisper movement and a higher acceleration under the influence of a force of the same magnitude, as already discussed in paragraph 4.1.1. Since the goal of the robot, in order to win, is to reach the tower before the player, we assume that the type of motion derived from a lower mass value would result in a behavior closer to the optimal one, where the optimal robot is the one going full speed following a straight path towards the target.

5.3.2 Microgame

Since we want to find the parameters while playing a single a game, we decided to segment a full game into microgames: a microgame is a sequence of 2 attacks to towers performed by the robot.
A single attack is won by the player if he succeeds to make a progress in the game: we consider a progress the lighting of a led on a tower. Conversely, an attack is won by the robot if it succeeds in preventing the player from progressing. A microgame is won by the participant winning the largest number of attacks. A complete game is usually composed by 6 to 8 microgames, which is a fair amount to perform a good selection of the robot parameters in one single game.
In order to capture these microgames, we have to enhance the planning architecture presented in chapter 4 with a couple of modules. We need a module in charge of recognizing when a led is lighted, which we will name *Tower activity monitor*: this module is deployed on each tower, and sends out information about the pressing of the button present on top of them. Another module in charge of capturing the beginning of an attack and elect the winner of the microgame named *Microgame monitor*: it will listen to the pressing of buttons and output whether after two attacks the player

was able to make progress in the game or not. The resulting architecture is presented in figure 5.16.

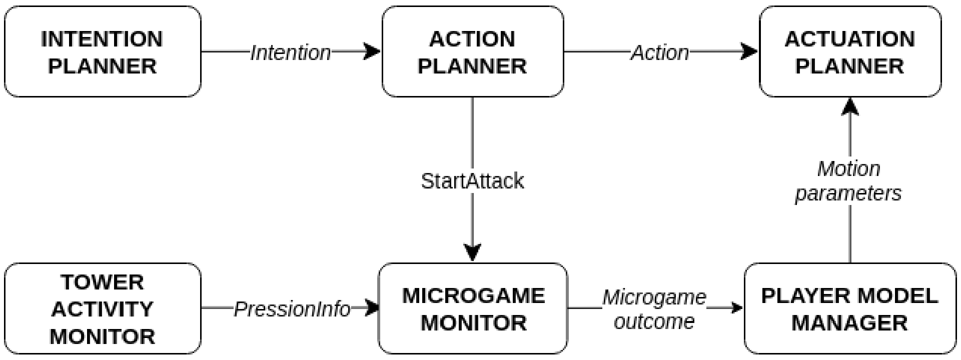


Figure 5.16: Supporting architecture to capture in game evolution

5.3.3 In game parameter estimation

During the experiments, the procedure was able to distinguish and differentiate the decision for different types of players: we had players whose best parameters corresponded clearly to an easy configuration, like the one in figure 5.17, to a medium one as in figure 5.18 and to a hard one as in figure 5.19. Even in the case of player with an unclear evaluation of their skill, the algorithm managed to find a suitable configuration after a bit of exploration, as in figure 5.20.

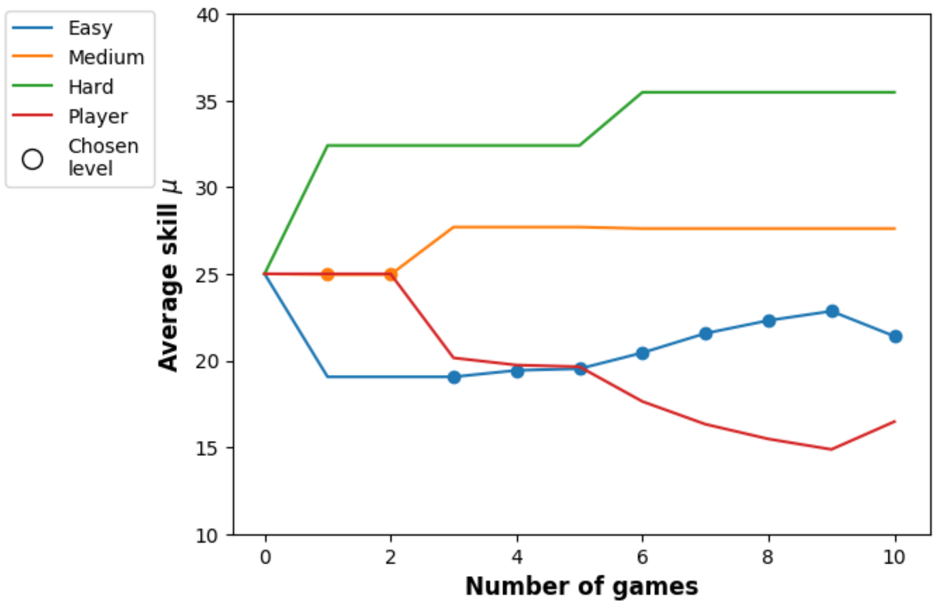


Figure 5.17: Player whose best configuration was Easy

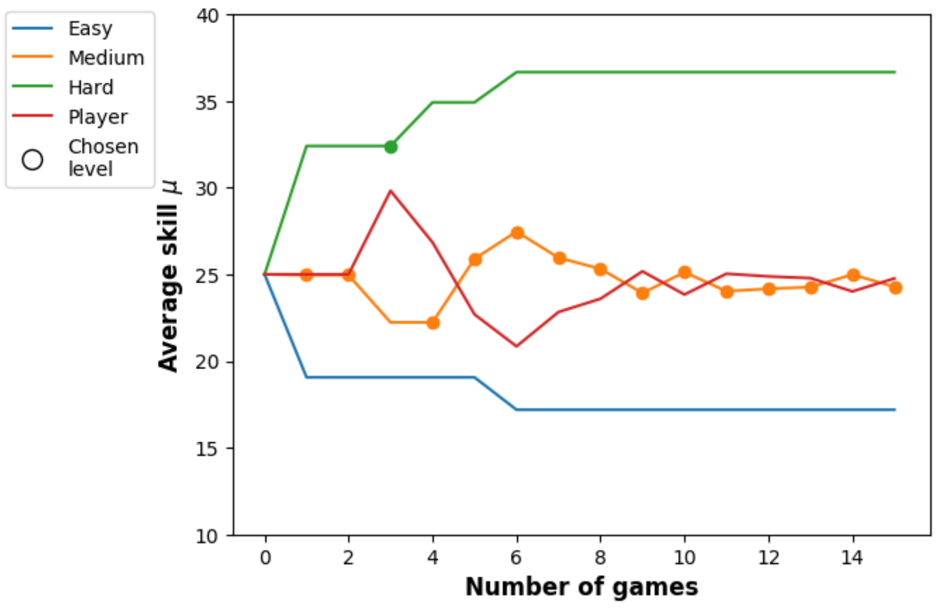


Figure 5.18: Player whose best configuration was Medium

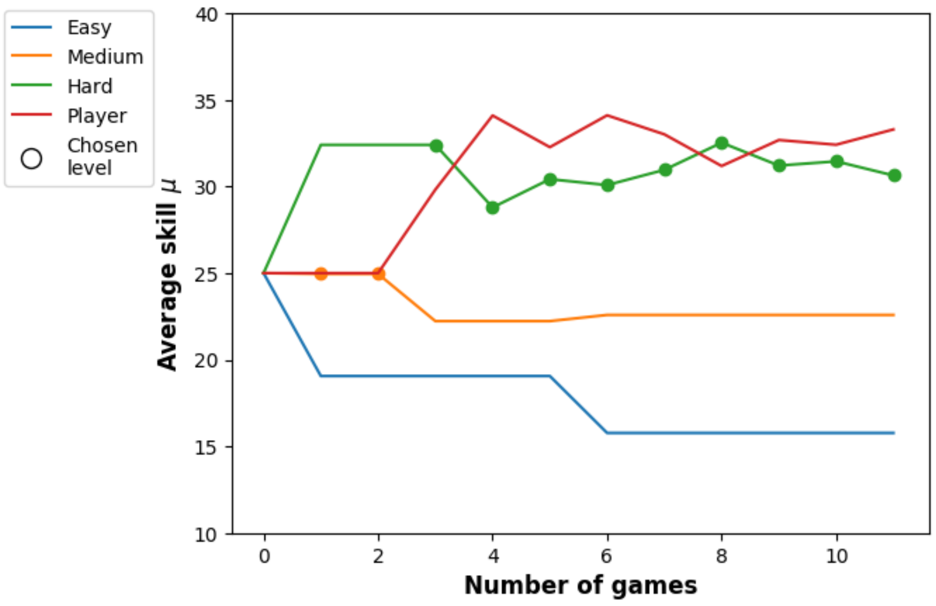


Figure 5.19: Player whose best configuration was Hard

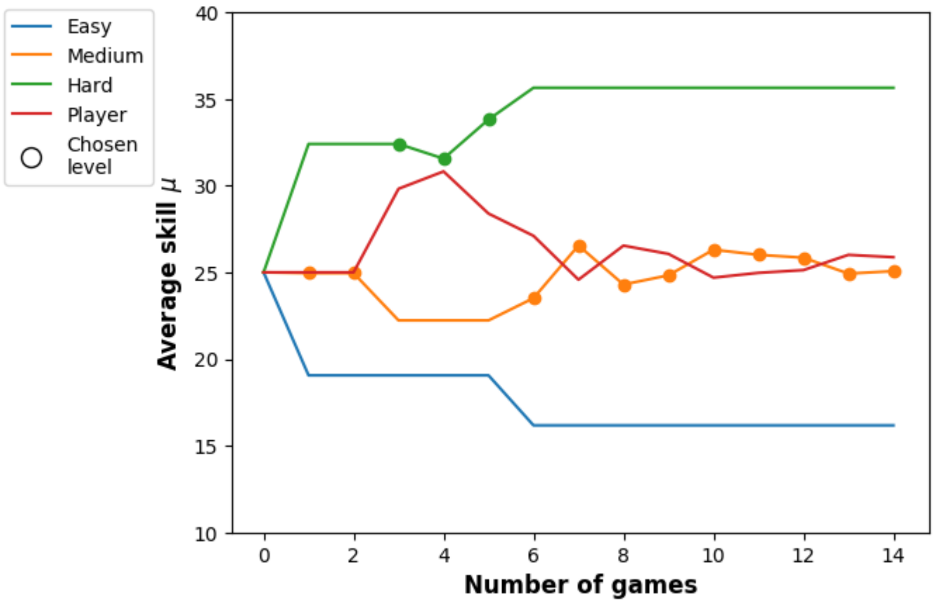


Figure 5.20: Player whose best configuration was mistaken as Hard and corrected as Medium

Chapter 6

Evaluation

"It's the best version ever of this game. Awesome."

Michele Bertoni

In the following section we're going to describe the experimental setting used to answer our questions regarding HRI. We will start by stating the questions we want to answer: after that, we will describe how the data had been collected, and which statistical analysis we performed. Then we will present the data visually and start to gain the first informal insights. In the last section we will provide the outcome of the statistical analysis and discuss the results.

6.1 Questions

The main aspect we want to investigate is if this approach to adapt to the player yields to a more engaging and pleasant interaction between the human and the robot. On a second note, we also want to investigate whether this approach results in a recognizable adaptation from the point of view of the player, or if a more extensive characterization of the robot's playing style should be conceived for this purpose. Finally, we also want to confirm that our approach is able to match the ability of the robot to the ability of the player.
Formalizing the questions we want to answer:

1. Does this adaptive behavior yield a better engagement than a fixed one?
2. Does the adaptation of motion profile yield to a recognizable adaptive behavior?

3. Does this approach produce an even game?

6.2 Statistical analysis

6.2.1 Experimental setup

We performed this study asking students of Politecnico di Milano to play Robotower in the controlled environment of the AIRLab. Each participant played 2 games and filled a questionnaire regarding the experience. The questionnaire can be found in appendix A.
We built the control group by having participants playing against a fixed configuration for the robot's motion profile, corresponding to:

- Maximum speed: 0.75 m/s
- Mass: 8

Later on, we will refer to this version of the game as *Fixed*.
We built the experimental group by having participants playing against a robot able to modify its motion profile during the game according to the procedure discussed in chapter 5. We will refer to this version as *Adaptive*.

6.2.2 Statistical tools

In order to evaluate whether the adaptation procedure we devised actually had an impact on the interaction between the human and the robot, we performed a statistical analysis on the answers collected. We used the non-parametric Mann Whitney U-test to test the null hypothesis that the two independent statistical samples come from the same distribution. This test is suitable for the analysis of ordinal values, like the ones we have collected. In case of small sample sizes, typically under 20 samples, the statistical significance of the study can be obtained by calculating the U value and matching it against provided tables. With bigger samples, as in our case, U can be approximated with a normal distribution: this gives us the ability to interpret the statistical significance by calculating the p-value of the U value obtained.
The statistical significance analysis tells us only what is the chance that we rejected the null hypothesis while it actually holds, but provides no information regarding the magnitude of impact of our work on the interaction: to gain insights on this, we calculated the Common Language Effect Size (CLES). This value is a measure of the difference between the two populations, which expresses the probability that a randomly selected score X_1

from one population will be greater than a randomly sampled score X_2 from the other population.

$$\begin{aligned} X_1 \in & experimental, X_2 \in control \\ CLES = & P(X_1 > X_2) \end{aligned} \tag{6.1}$$

In the end, we will provide a 95% confidence interval for the CLES we estimate, that is an interval where the true value of the CLES will lie with that given confidence.

6.3 Data Exploration

We split the total 52 participants evenly, thus we had 26 of them playing the *Fixed* version of the game and the other 26 the *Adaptive one*. The age of participant went from 21 to 30 years old, distributed as shown in figure 6.1.

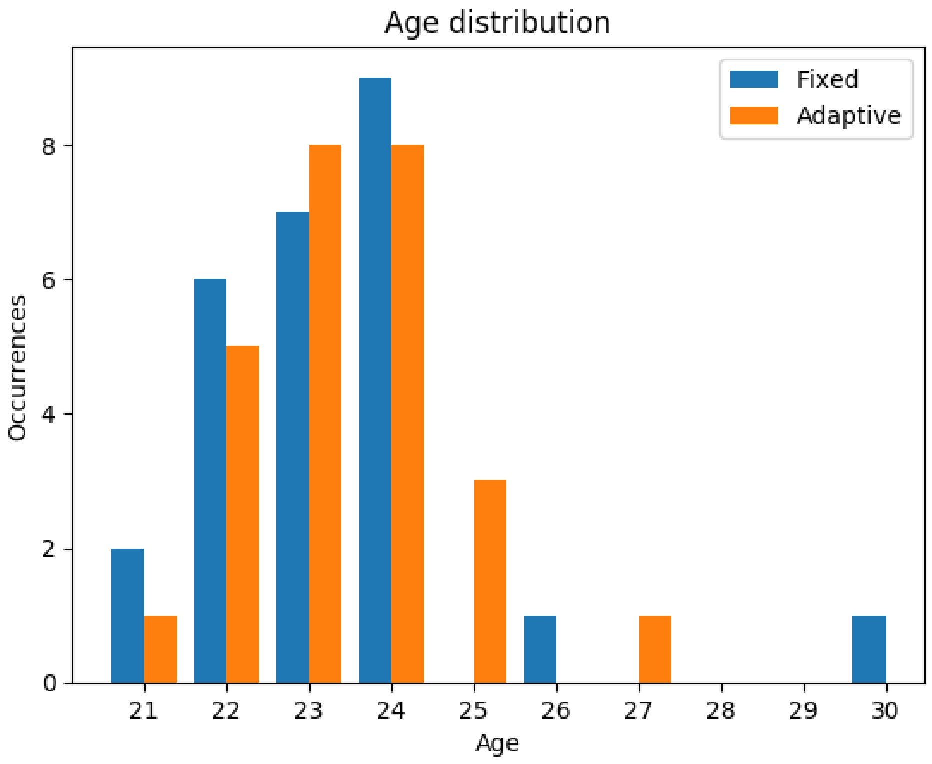


Figure 6.1: Age distribution of participants

More information about the participant can be found in table 6.1.

Version	Male	Female	Total	Age mean	Age variance
Fixed	25	1	26	23.3	2.99
Adaptive	19	7	26	23.4	1.55

Table 6.1: Participant details

Our work focused on evaluating the effect of providing an even game for the player. Figure 6.2 compares the win rates for the two versions of the game: one playing with the *Fixed* behavior and the other with the *Adaptive* behavior. While the game in the *Fixed* version resulted in an unbalanced win rate in favour of the human player, our approach managed to reach almost an equal partitioning of the number of wins, with 27 wins for the human and 25 for the robot, out of the 52 games played (2 for each person): we can say that we were able to provide an even game.

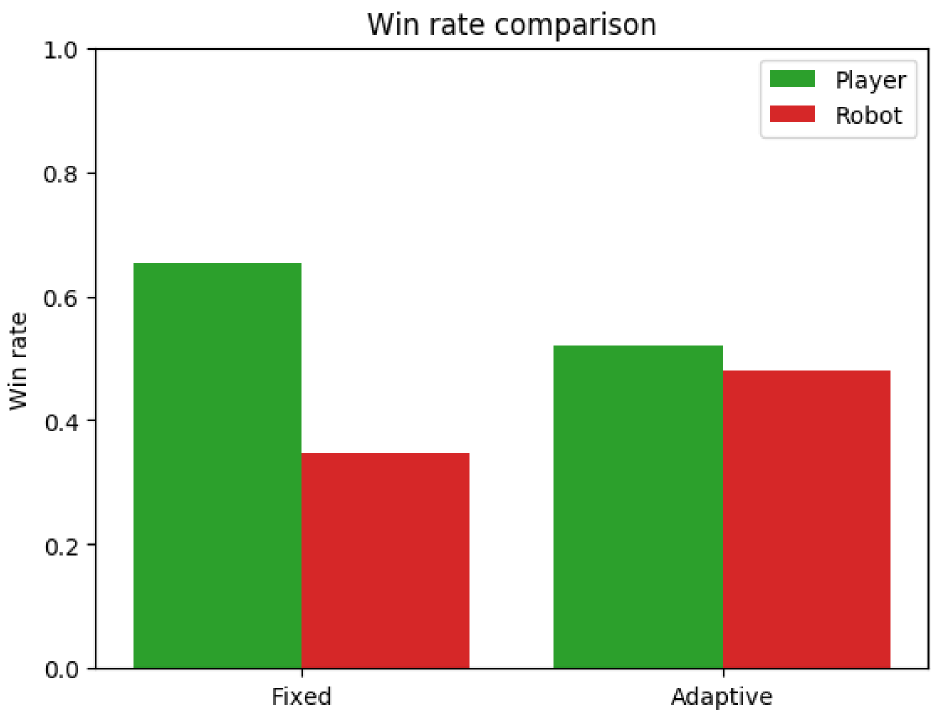


Figure 6.2: Comparison of win rate using Fixed and Adaptive behavior

One of the effect produced by the ability matching procedure was that we were also able to provide longer interactions: in table 6.2 we compare the average duration of a game for the two variants. We divided the analysis between games won by the player and the robot as in the latter case interactions are inherently shorter.

Version	**Fixed**	**Adaptive**
Number of player wins	32	27
Number of robot wins	17	25
Average time for the player to win	95s	118s
Average time for the robot to win	51s	77s

Table 6.2: Win shares and average duration of a game in seconds

In figure 6.3 we compared answers to the question *I had fun* between the two versions. We can see that people liked the game even in its *Fixed* version, mostly agreeing to the statement, but few people gave the maximum agreement rate. For the *Adaptive* version, people were more prone to strongly agree with the statement.

In figure 6.4 we compare agreement level to the statement *The robot changed playing style* to see if the adaptation was perceived by the human player. We can see high variance for the *Fixed* version, but most of the people neither agreed nor disagreed with the statement. The dynamic change of the robot action space during the game, given by the fact that a tower conquered by the player can no longer be attacked, had been sometimes interpreted as a change in playing style. For the *Adaptive* version, most people agreed to the statement recognizing the change of motion profile as change in playing style. For some people, like in figure 5.18, it happened that the selected motion profile never changed during the game.

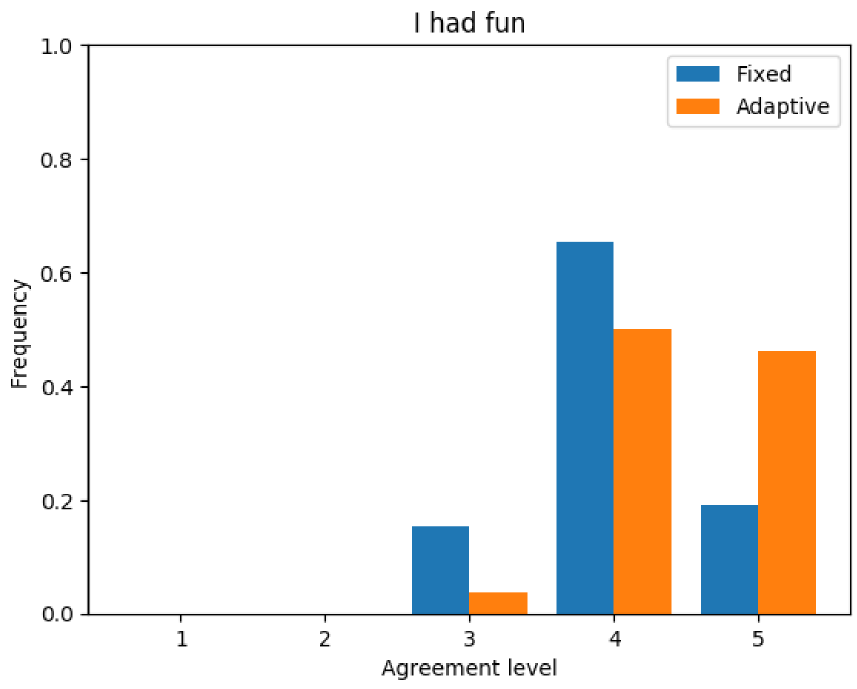


Figure 6.3: Comparison of agreement to the 'I had fun' statement

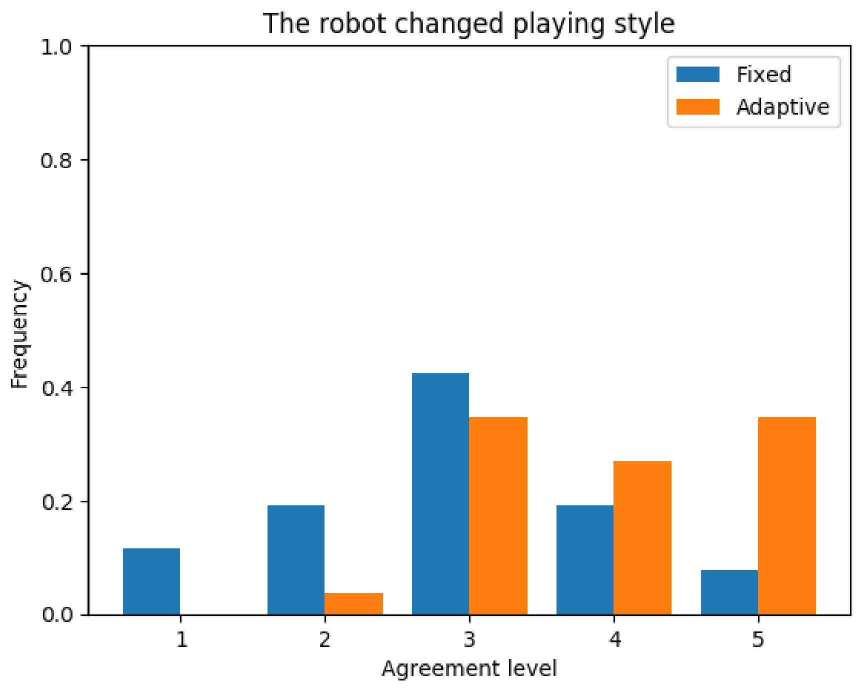


Figure 6.4: Comparison of agreement to 'The robot changed playing style' statement

6.4 Statistical results

Table 6.3 reports the outcome of the statistical analysis carried out to compare results of the agreement level about the statements *I had fun* and *The robot changed playing style*. We used the first statement to establish whether the interaction with the robot was more pleasant and engaging for the player and the second one to investigate if the adaptation was perceived by the player.

	I had fun	**The robot changed playing style**
U	229.5	173.5
mean	338	338
variance	679.5	239.0
p-value	0.024	0.002
CLES	0.66	0.74
CI CLES	[0.52, 0.80]	[0.60, 0.88]

Table 6.3: Statistical values for comparison on the engagement and perceived adaptation. Mean and variance refer to the normal approximation for the distribution of the U statistics. The p-value is computed for the two-tailed test, the Confidence Interval for the CLES is computed for a 95% confidence

About the engagement level, we can reject the null hypothesis that the two samples come from the same distribution at a significance level $\alpha = 0.05$: this means that the probability that we are rejecting the null hypothesis while it actually holds is 5%. The Common Language Effect Size found for the engagement analysis is 0.66, meaning that if we draw randomly an answer from people that played with the fixed version and another one from people that played the adaptive version, there's a probability of 66% that the second one answered with a score higher than the first one. While the magnitude of this result may not seem relevant, we can see from figure 6.3 that the game results appealing even in its fixed variant, leaving a small margin for improvement. We also compute a 95% confidence interval for this measure, resulting in a CI = [0.52, 0.80]: we can be highly confident that the approach had a positive impact, but still the appeal of the game in the *Fixed* version limited the magnitude of the impact our approach could bring.

Regarding the perception of adaptation from the player, our approach managed to bring a higher impact. We can reject the null hypothesis that the two samples come from the same distribution at a significance level

$\alpha = 0.01$. Also, the CLES we computed is 0.74, meaning that about three times out of four we would get a higher agreement level to the statement from randomly selected answer from people who played the *Adaptive* version against randomly selected answer from people who played the *Fixed* one. The computed 95% confidence interval is CI = [0.60, 0.88].

Chapter 7

Conclusions and Future Works

The study we conducted aimed at evaluating the effect of adaptation to the user in a competitive HRI scenario. While this question has already been investigated in collaborative scenarios, where an adaptation of the robot is aimed at easing the task of the human, this is one of the first study that aims at providing a challenging interaction for the user and evaluating its effects.

We characterized the ability level of the robot trough its motion abilities, both in terms of steering ability and maximum speed, and devised a procedure to modify the ability level of the robot during the game in order to match the estimated player ability. This procedure succeeded in providing an even game, where the number of wins for the robot was almost equal to the number of wins for the players. We performed a study inviting students to play the game and give their opinion trough a questionnaire: from a statistical analysis we found out that the adaptation produced a positive effect on the user engagement when compared to the interaction with a robot exhibiting a fixed behavior: the magnitude of the effect was limited by the appeal of the game even in its basic form. This result is nevertheless encouraging, and suggests that the adaptation of the robot behavior can lead to positive effects even in competitive scenarios.

Future works on the field may try to produce a deeper adaptation of the robot behavior. We adapted only the motion ability of the robot, but the adaptation could be brought to an higher reasoning level by recognizing and classifying actions from the player and counteracting them.

Bibliography

[1] M. I. Ahmad and O. Mubin. Applying adaptive social mobile agent to facilitate learning. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 407–408, March 2016.

[2] M. Bajones. Enabling long-term human-robot interaction through adaptive behavior coordination. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 597–598, March 2016.

[3] C. Balaguer, A. Giménez, J. M. Pastor, V. M. Padrón, and M. Abderrahim. A climbing autonomous robot for inspection applications in 3d complex environments. *Robotica*, 18(3):287–297, 2000.

[4] Anders Drachen, Alessandro Canossa, and Georgios N. Yannakakis. Player modeling using self-organization in Tomb Raider: Underworld. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 1–8. IEEE, 2009.

[5] Arpad E. Elo. *The rating of chessplayers, past and present.* Arco Pub., New York, 1978.

[6] Mark E. Glickman. Parameter estimation in large dynamic paired comparison experiments. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 48(3):377–394, 1999.

[7] Ralf Herbrich and Thore Graepel. Trueskill(tm): A bayesian skill rating system. Technical report, January 2006.

[8] M. Köseoğlu, O. M. Çelik, and Ö. Pektaş. Design of an autonomous mobile robot based on ros. In *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*, pages 1–5, Sep. 2017.

[9] F. Lamberti, F. G. Pratticó, D. Calandra, G. Piumatti, F. Bazzano, and T. R. K. Villani. Robotic gaming and user interaction: Impact of

autonomous behaviors and emotional features. In *2018 IEEE Games, Entertainment, Media Conference (GEM)*, pages 1–9, Aug 2018.

[10] G. D. S. Lee, K. S. Lee, H. G. Park, and M. H. Lee. Optimal path planning with holonomic mobile robot using localization vision sensors. In *ICCAS 2010*, pages 1883–1886, Oct 2010.

[11] M. Mariappan, J. C. Sing, C. C. Wee, B. Khoo, and W. K. Wong. Simultaneous rotation and translation movement for four omnidirectional wheels holonomic mobile robot. In *2014 IEEE International Symposium on Robotics and Manufacturing Automation (ROMA)*, pages 69–73, Dec 2014.

[12] Diego Martinoia, Daniele Calandriello, and Andrea Bonarini. Physically Interactive Robogames: Definition and design guidelines. *Robotics and Autonomous Systems*, 61(8):739–748, August 2013.

[13] Radek Pelánek. Applications of the elo rating system in adaptive educational systems. *Computers & Education*, 98:169 – 179, 2016.

[14] F. G. Pin and S. M. Killough. A new family of omnidirectional and holonomic wheeled platforms for mobile robots. *IEEE Transactions on Robotics and Automation*, 10(4):480–489, Aug 1994.

[15] J. Pripfl, T. Körtner, D. Batko-Klein, D. Hebesberger, M. Weninger, C. Gisinger, S. Frennert, H. Eftring, M. Antona, I. Adami, A. Weiss, M. Bajones, and M. Vincze. Results of a real world trial with a mobile social service robot for older adults. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 497–498, March 2016.

[16] L. Vanessa Cruz Quispe and J. E. O. Luna. A trueskill approach for movies recommendation. In *2015 Latin American Computing Conference (CLEI)*, pages 1–5, Oct 2015.

[17] Craig Reynolds. Steering behaviors for autonomous characters. 1999.

[18] Ali Sekmen and Prathima Challa. Assessment of adaptive human-robot interactions. *Knowledge-Based Systems*, 42:49 – 59, 2013.

[19] M. Tielman, M. Neerincx, J. Meyer, and R. Looije. Adaptive emotional expression in robot-child interaction. In *2014 9th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 407–414, March 2014.

[20] Georgios N. Yannakakis. How to model and augment player satisfaction: a review. In *First Workshop on Child, Computer and Interaction*, 2008.

[21] Georgios N. Yannakakis and John Hallam. Real-time game adaptation for optimizing player satisfaction. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(2):121–133, 2009.

[22] Georgios N. Yannakakis, John Hallam, and Henrik Hautop Lund. Entertainment capture through heart rate activity in physical interactive playgrounds. *User Modeling and User-Adapted Interaction*, 18(1):207–243, 2008.

Appendix A

Questionnaire

A.1 English version

Who are you?

1. **Age (completed years):** ________

2. **Gender:** ☐ M ☐ F

3a. **Have you already had a direct experience with robots?**
☐ No ☐ Yes

3b. **If you answered "Yes" to the previous question, please describe your experience:**

____

____

____

4. **What do you usually play?**
☐ Videogames ☐ Board games ☐ Cards ☐ Sport
☐ Other ________________________

With respect to the game just played, state how much do you agree with the following statements, where: 1 = "strongly disagree", 3 = "indifferent" e 5 = "strongly agree"

5. **I had fun**
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

6. **I wanted to play**
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

7. **While playing, I paid attention to the robot**
   ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
8. **The robot was paying attention to what I was doing**
   ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
9. **The robot wanted to win**
   ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
10. **I was scared by the robot**
    ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
11. **The game lasted too few**
    ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
12. **I understood immediately the game rules**
    ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
13. **I changed my playing style**
    ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
14. **The robot changed his playing style**
    ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
15. **The robot did some feints**
    ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
16. **I would enjoy more if the robot did some feints**
    ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
17. **It was easy to understand what the robot would have done**
    ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

Game outcome (to be completed by the researcher)

18. **Winner:** ________________
19. **Time:** __________________
20. **Winner:** ________________
21. **Time:** __________________
22. **Notes:**

____

____

A.2 Italian version

Chi sei?

23. Età (anni compiuti): ________

24. Genere: ☐ M ☐ F

25a. Hai già avuto esperienze dirette con dei robot? ☐ No ☐ Sì

25b. Se hai risposto "Sì" alla domanda precedente, racconta la tua esperienza:

____

____

____

26. Come giochi di solito?
☐ Videogiochi ☐ Giochi in scatola ☐ Carte ☐ Sport
☐ Altro ____________________

Rispetto al gioco appena fatto, indica quanto sei d'accordo con le seguenti affermazioni, dove: 1 = "per niente d'accordo", 3 = "indifferente" e 5 = "molto d'accordo"

27. Mi sono divertito
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

28. Avevo voglia di giocare
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

29. Quando giocavo stavo attento al robot
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

30. Il robot era attento a quel che facevo
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

31. Il robot voleva vincere
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

32. Avevo paura del robot
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

33. Il gioco dura troppo poco
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

34. **Ho capito subito le regole del gioco**
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

35. **Ho cambiato il modo di giocare**
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

36. **Il robot ha cambiato modo di giocare**
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

37. **Il robot ha fatto delle finte**
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

38. **Mi divertirei di piú se il robot facesse delle finte**
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

39. **Era facile capire cosa avrebbe fatto il robot**
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

Risultato gioco (a cura del ricercatore)

40. **Vincitore:** ________________

41. **Tempo:** ____________________

42. **Vincitore:** ________________

43. **Tempo:** ____________________

44. **Note:**
____
____