robotics

MDPI

Article

Playing Checkers with an Intelligent and Collaborative Robotic System †

Giuliano Fabris, Lorenzo Scalera * and Alessandro Gasparetto

Polytechnic Department of Engineering and Architecture, University of Udine, 33100 Udine, Italy; giuliano.fabris@uniud.it (G.F.); alessandro.gasparetto@uniud.it (A.G.)
* Correspondence: lorenzo.scalera@uniud.it

† This paper is an extended version of our paper published in Fabris, G.; Scalera, L.; Gasparetto, A. An Interactive Collaborative Robotic System to Play Italian Checkers. In Proceedings of the IFToMM World Congress on Mechanism and Machine Science, Tokyo, Japan, 5-10 November 2023; pp. 74–84.

Abstract: Collaborative robotics represents a modern and efficient framework in which machines can safely interact with humans. Coupled with artificial intelligence (AI) systems, collaborative robots can solve problems that require a certain degree of intelligence not only in industry but also in the entertainment and educational fields. Board games like chess or checkers are a good example. When playing these games, a robotic system has to recognize the board and pieces and estimate their position in the robot reference frame, decide autonomously which is the best move to make (respecting the game rules), and physically execute it. In this paper, an intelligent and collaborative robotic system is presented to play Italian checkers. The system is able to acquire the game state using a camera, select the best move among all the possible ones through a decision-making algorithm, and physically manipulate the game pieces on the board, performing pick-and-place operations. Minimum-time trajectories are optimized online for each pick-and-place operation of the robot so as to make the game more fluent and interactive while meeting the kinematic constraints of the manipulator. The developed system is tested in a real-world setup using a Franka Emika arm with seven degrees of freedom. The experimental results demonstrate the feasibility and performance of the proposed approach.

Keywords: collaborative robotics; computer vision; trajectory planning; manipulation; Italian checkers

check for updates

Citation: Fabris, G.; Scalera, L.; Gasparetto, A. Playing Checkers with an Intelligent and Collaborative Robotic System. *Robotics* **2024**, *13*, 4. https://doi.org/10.3390/robotics13010004

Academic Editor: Chee-Kong Chui

Received: 22 October 2023
Revised: 13 December 2023
Accepted: 20 December 2023
Published: 21 December 2023

1. Introduction

In recent years, collaborative robotics has been increasingly adopted in the manufacturing industry to allow machines and humans to work and interact in a shared space for a common task while still ensuring human safety [1,2]. However, collaborative robotics is not a privilege of the industrial sector but has also been applied for entertainment purposes thanks to the development of proper AI algorithms. Several examples of entertainment robotic systems can be found in the literature, demonstrating that robots can safely cooperate with humans while playing board games (such as Tic Tac Toe [3] and Connect 4 [4]), painting [5], or playing music [6], just to mention a few applications of collaborative robotics in this field. Particularly, the possibility of playing chess and checkers against an artificial player has been widely investigated, developing several solutions in order to accomplish this purpose [7–9].

Developing a robot that autonomously plays checkers or chess is a complex multidisciplinary problem since it involves elements from different technical fields, like computer science, electronics, and mechanics, which have to be combined for executing a real-world task. This problem is relevant not only for entertainment reasons but also for educational purposes because playing board games helps to develop brain functions and logical thinking. Furthermore, the techniques and methodologies developed for robotic systems in the field of board games can be easily applied in the industry, such as vision systems for recognizing objects or trajectory planning approaches for pick-and-place tasks. Essentially, the main requirements of a robotic system for playing checkers or chess are three:

1. the recognition of the board and positions of pieces with respect to the robot, piece color, and type, as well as the human player move;
2. the decision of which is the optimal move to perform among all the possible ones according to a proper game strategy;
3. the trajectory planning and control of the robot in order to physically execute the chosen move.

In this work, we pursued the goal to develop a collaborative robotic system for interactively playing Italian checkers using a Franka Emika robot with seven degrees of freedom (DOFs) as a piece manipulator and a vision system for game state acquisition (Figure 1). The robotic system is able to identify the current game state, employ a developed decision-making algorithm to determine the best move, and physically manipulate the game pieces on the board through pick-and-place actions. This work extends our previous conference paper [10] by minimizing the motion time online of the robot during each pick-and-place operation while meeting its kinematics constraints. In this way, minimum-time trajectories are optimized on the fly for each move of the robot so as to avoid long waiting times for the human player and make the game more fluent and interactive.

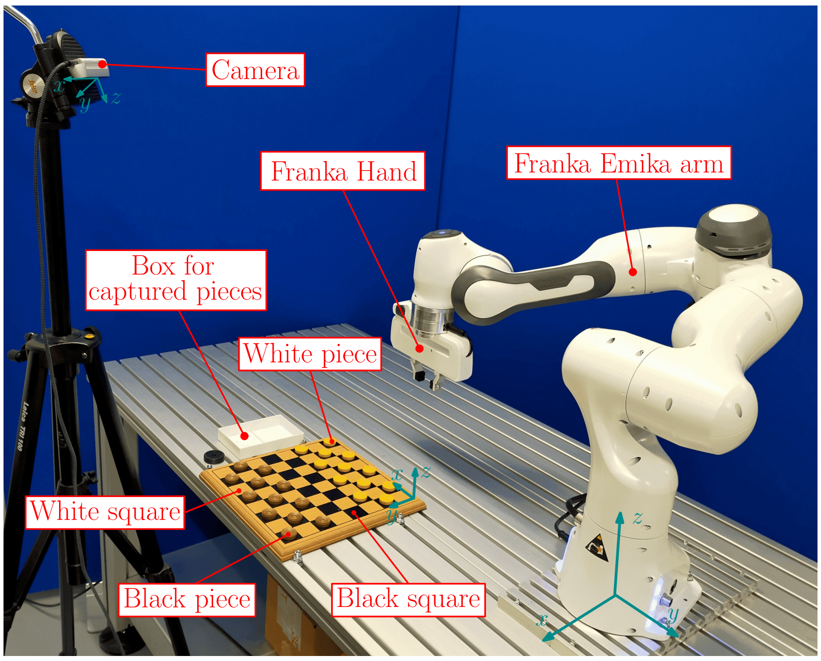


Figure 1. Experimental setup.

This paper is organized as follows: Section 2 describes the related works, whereas the materials and methods are illustrated in Section 3. In more detail, Section 3.1 outlines the rules of Italian checkers, Section 3.2 describes the computer vision approach, and Section 3.3 illustrates the game logic for the selection of the optimal move. Furthermore, the trajectory planning approach is illustrated in Section 3.4, whereas Section 3.5 presents the experimental setup. In Section 4, the experimental results are reported. Finally, Section 5 provides the conclusions and possible future developments of this work.

2. Related Works

The recognition of the board and pieces is usually challenging since it occurs mainly through computer vision, which can be influenced by lighting conditions. The board is usually less problematic to be recognized since computer vision libraries, like the open-source OpenCV, provide many tested and robust algorithms based on model recognition suitable to fulfill this purpose. Examples are Harris corner detection [11,12], Canny edge detection, and Hough Line Transform [13,14]. These algorithms are suitable when the relative position between the robot and board can change since they find the board position fairly accurately and quickly. However, they require a proper calibration process that has to be performed before the game starts, and they increase the total computational time. Differently, piece position and color are often more difficult to obtain. There are different solutions to reduce this issue, such as using colored pieces (they are normally black and white) in order to increase the contrast with squares (also usually black and white) [15,16], adopting a dedicated lighting system [17], or implementing strategies to compensate for distortions and lighting variations [13,18].

Nevertheless, computer vision is not the only way to obtain board position and configuration; other methods to solve this problem exist, such as those that exploit electro-magnetic phenomena. An example is reported in [19], where the authors installed light-dependent resistors (LDRs) under squares. LDRs are sensors that exhibit a high resistance when they do not receive light (occupied square), whereas their resistance decreases when they are enlightened (empty square). Therefore, by applying LDRs, whether a square is occupied or not can be known, but not the type and color of the piece. This issue is easily solved by knowing the starting board configuration and tracking the game state. Another example is described in [20], where the authors propose to place Hall sensors under squares for detecting the pieces on the board. Hall sensors recognize the presence and magnitude of a magnetic field. Using magnetic pieces, this system can detect only the presence or absence of a piece in a certain square. Similar to the previous case, the actual configuration is known by the initial one and tracking the game state. Finally, in order to know if the human player has made a move, detecting a change in the board configuration can be sufficient [18].

The second element of autonomous robot players is the game engine, which first aims at finding all the possible available moves that the artificial player can make. Furthermore, its purpose is to select the best one among all according to a chosen game strategy. Moreover, the development of such algorithms is not a simple task, especially if the aim is to develop an efficient and competitive system. To easily solve this problem, some authors preferred to implement an open-source game engine like Stockfish [14,21] for chess and Raven [16] for checkers. Instead, other researchers tried to develop a game engine by themselves. An example is the development of a checkers-playing GUI based on a *minimax* algorithm, presented in [22]. It consists of a game tree search algorithm that aims to find the path that leads to the most favorable configuration, but its calculation time increases exponentially with the tree depth. In order to decrease it, an alpha–beta pruning algorithm can also be considered [23]. This algorithm "prunes" the branches that definitely cannot lead to a better situation with respect to another previously evaluated, thus reducing the tree breadth. Another method to reduce computation time is illustrated in [24], where the authors develop a hybrid tree search algorithm on parallel CPU and GPU in order to exploit the full potential of the computer. A method different from the tree search is to adopt a neural network, as outlined in [20].

The last component of these systems has the purpose of driving the robot for physically executing the selected move. This part has a certain importance because, although checkers and chess do not necessarily need a physical implementation of a robotic player (a GUI interface can be sufficient), it is proved that this improves the user experience and the game attractiveness to human players, making the game more realistic [25,26]. Nevertheless, pieces are often arbitrarily shaped (especially in chess), so their manipulation is not always straightforward. The manipulation of the pieces on the board can be performed by adopting custom grippers that adapt to piece shape [26], pneumatic grippers [27], or electro-magnetic grippers [28]. However, the last two solutions require custom pieces in order to successfully grasp them.

Another challenge regarding the physical implementation of a robotic player is correctly driving the robot to move the pieces and remove them in the case of capture. The accuracy of the vision system has a particular influence in this context since it has to detect as accurately as possible the pieces' positions. In this manner, the gripper can correctly reach and grasp the pieces. As robotic agents, robotic arms are mainly used [12,14,18] thanks to their dexterity and manipulation capability. Cartesian mechanisms are also used [19,26] due to their easy control, but their structure is often large and bulky. Moreover, there has been an exploration of the potential to improve the user experience by enabling the robot to engage with the human player using sounds and facial expressions. One instance of this can be seen in Baxter, a two-armed robot equipped with a display that reflects facial expressions corresponding to its game status [13,16]. Another example is NAO, a humanoid robot that mimics real person movements, making the game more realistic [11,15].

Table 1 reports an overview on the state of the art of robotic systems for playing checkers or chess. In the table, a comparison of the proposed approach with similar works is also illustrated.

Table 1. State of the art on robotic systems for playing checkers of chess.

First Author	Year	Ref.	Robot	Game	Game State Estimation	Game Engine
Barakova	2018	[15]	Humanoid (NAO)	Checkers	Camera, computer vision, based on relation between the coordinate systems of camera and board	Minimax + alpha–beta pruning
Bernbaum	2018	[9]	Franka Emika robot	Chess	Camera, computer vision, based on model recognition	Sunfish chess engine
Brooks	2015	[16]	Baxter robot	Checkers	Camera, computer vision, based on model recognition	Raven Checkers
Carrera	2017	[26]	Cartesian robot	Chess	Camera, computer vision, pieces recognition	Not specified
Chen	2019	[13]	Baxter robot	Chess	Camera, computer vision, based on model recognition	Stockfish
Del Toro	2019	[14]	Custom robot with four DOFs	Chess	Camera, computer vision, based on model recognition	Stockfish
Elnaggar	2014	[24]	Lynxmotion AL5D	Checkers	Camera, computer vision, based on color information	Negamax
Gupta	2015	[19]	Cartesian robot	Chess	LDR sensors under squares for pieces detection	Minimax + alpha–beta pruning
Juang	2022	[11]	Humanoid (NAO)	Chess	Camera, computer vision, based on model recognition	Not implemented
Kołosowski	2020	[12]	UR5 robot	Chess	Camera, computer vision, based on model recognition	Minimax + alpha–beta pruning
Kopets	2020	[20]	Custom robot with three DOFs	Russian checkers	Hall sensors under black squares detect magnetic pieces	Neural network based on AlphaZero
Larregay	2018	[17]	ABB model IRB120	Chess	Camera, computer vision, dedicated lighting system	GNU Chess
Luqman	2016	[18]	Custom robot with four DOFs	Chess	Camera, computer vision, based on color information	Not specified
Manurung	2023	[28]	Gantry robot	Checkers	Camera, computer vision, based on color information	Minimax + alpha–beta pruning
Matuszek	2011	[29]	Gambit robot	Chess	Camera, computer vision, based on model recognition and point cloud information	Not specified
Rath	2019	[21]	Cartesian robot	Chess	Camera, computer vision, based on model recognition	Stockfish
Rodriguez-Sedano	2016	[25]	Baxter robot	Checkers	The operator sees the board configuration through a camera	Not implemented
Proposed approach			Franka Emika robot (minimum-time trajectories)	Italian checkers	Camera, computer vision based on color information	Minimax

3. Materials and Methods

3.1. Italian Checkers Rules

In the following, the rules of Italian checkers are recalled [30]. In this game, the board consists of an eight-by-eight grid of alternating light and dark squares, typically white and black. Players use the dark squares for their moves, and each player begins with twelve pieces, one set in light color and the other set in dark color, also usually white and black. The player with the light pieces makes the first move. There are two types of pieces in the game: *men* and *kings*. Each player begins with only men (each composed of one piece), and these can be promoted to kings when they reach the opposite side of the board. Kings are recognized by stacking two men on top of each other. The distinction between these two types of pieces lies in their movement capabilities. Men can move one square diagonally only forward, whereas kings have the possibility to move one square diagonally both forward and backward.

When a player piece, be it a man or a king, encounters an opponent piece with an empty square behind it (in a direction in which it can move), it must capture it. The capture is performed by jumping over the opponent piece to the empty square. The captured piece is then removed from the game board. If there are additional opportunities to capture opponent pieces from the target position, the capturing piece will continue until it reaches a position where no further captures are possible. A man can capture only other men, while a king has the ability to capture both men and kings. In scenarios where there are multiple capture possibilities, a specific order of priority must be followed: capturing the greatest number of pieces takes precedence, followed by capturing with a king, and then capturing the higher-value pieces (kings). If multiple capture opportunities still exist, priority is assigned to capture where the higher-value pieces are encountered first. Finally, the objective of each player is to either capture or block all their opponent's pieces so that the opponent has no more possible moves. If a player succeeds, the game is won; otherwise, the game ends in a draw.

3.2. Game State Evaluation

In the following, we describe the proposed computer vision algorithm, which is the first part of the system. This algorithm aims at identifying the positions and types of pieces starting from the image of the board, received as an input. First, the image analysis process is described. Then, we outline how the matrix that represents the board is built.

To obtain a 2D image that just contains the game board, the initial procedure is to switch from the plane of the camera photo to that of the board. To accomplish this, the input image is modified through a function called *homography*, obtaining the desired board image. This process requires the contribution of the user, who has to manually point the position of the board corners in the image. This operation does not demand high precision since, even if the user fails to indicate the corners perfectly, possible imprecision can be corrected thanks to the regularity of square divisions.

The next step is to distinguish and classify the different elements of the board. First, the setup of the colors is executed, assuming that there are only four categories of objects on the board: white/black pieces and white/black squares. Therefore, it is important to manage the board illumination carefully in order to minimize the impact of shadows and reflections on color recognition. This issue can be mitigated by increasing the image resolution and implementing a dedicated lighting system. All the pixels of the board image are arranged into a list where each entry contains the corresponding RGB value. These values are then analyzed to generate another list in which each original RGB value is replaced with the closest of the four predominant colors (corresponding to the four categories of objects present on the board), represented by a value from 1 to 4. Then, to obtain a black and white image, each pixel is assigned a shade of gray based on the values derived from the previously obtained list.

After having recognized the four main colors, the setup of the types is performed. It is needed since the assignment of the shades of gray to each color is random. It means that a lighter shade may not necessarily correspond to white and a darker shade to black. For the setup of the types, the algorithm asks the user to select the position of a white square, black square, white piece, and black piece in sequential order. This information allows the program to identify the location of an element for each type of the four present on the board. For each selected element, an image of its square is cropped and analyzed to generate a histogram depicting the frequency of each color within it (Figure 2). Finally, the color with the highest occurrence in the histogram is determined. This process enables the program to link the user-indicated real colors to the initially randomly assigned color classes.

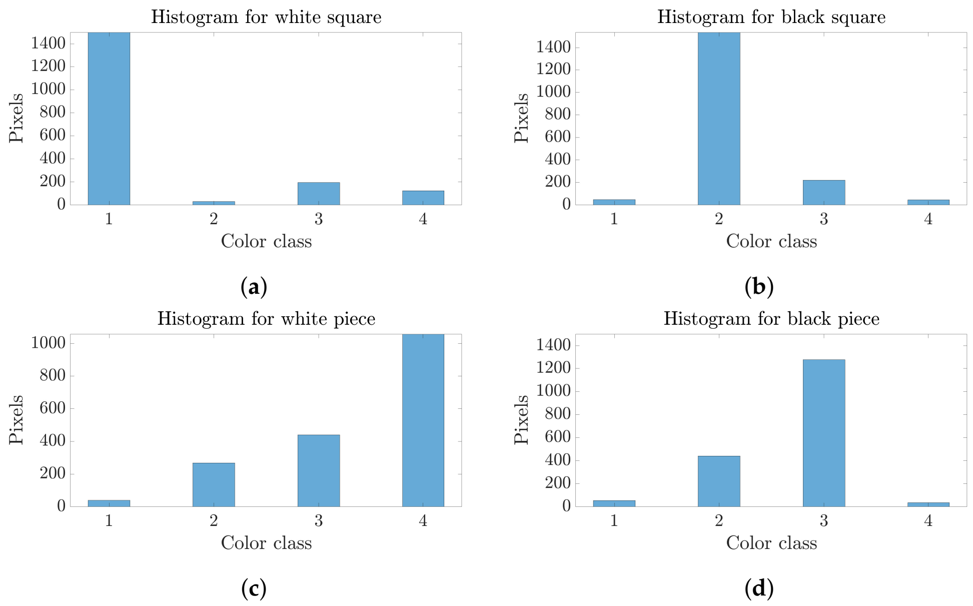
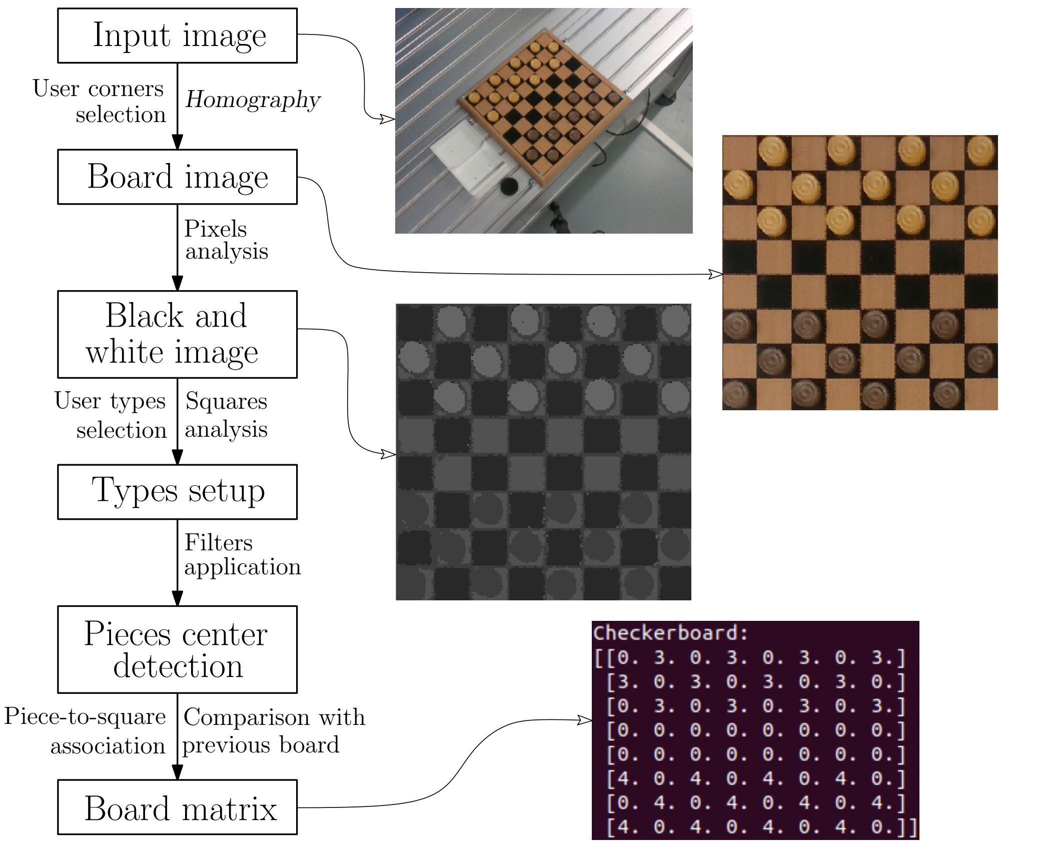


Figure 2. Examples of the histograms representing the frequency of each color class within selected elements: white square (**a**), black square (**b**), white piece (**c**), and black piece (**d**).

In order to develop the gaming algorithm, it is useful to represent the board configuration as a matrix since, in this manner, every square and its content can be identified by a couple of coordinates and an identifier. For this purpose, once the association between each color and its corresponding element is established, the location of the center of the pieces needs to be determined. This can be accomplished by first considering the pixels corresponding to the white pieces, then the ones of the black pieces. For each color, filters are applied to the respective pixels with the aim of removing imperfections and enhancing the accuracy of identifying the actual centers of the pieces. Through this process, the developed computer vision algorithm can locate the real centers of the pieces with a maximum error in the order of a few millimeters. These positions are then recorded using x and y coordinates in the board reference system, normalized between 0 and 1. These coordinates are also employed for robot control.

Finally, the process of constructing the matrix that contains the positions and types of the pieces can be executed. Then, each piece has to be associated with the corresponding square. For this process, the software computes the distance between the center of each piece and the centers of closer black squares, associating each piece to the closest one. However, from only the current image, it is possible to establish the color of the piece but not its type (man or king) since only the colors are considered. To solve this problem and determine piece types, it is sufficient to compare the current board image with the previous one, keeping in mind that, if a man reaches the opposite side of the board, it becomes a king. Then, identification numbers are assigned accordingly (0 for an empty square, 3 for a white man, 4 for a black man, 5 for a white king, and 6 for a black king), resulting in the final matrix. Figure 3 provides an overview of the key steps that compose the computer vision algorithm and depicts their main outputs.



```
Checkerboard:
[[0. 3. 0. 3. 0. 3. 0. 3.]
 [3. 0. 3. 0. 3. 0. 3. 0.]
 [0. 3. 0. 3. 0. 3. 0. 3.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [4. 0. 4. 0. 4. 0. 4. 0.]
 [0. 4. 0. 4. 0. 4. 0. 4.]
 [4. 0. 4. 0. 4. 0. 4. 0.]]
```

Figure 3. Workflow diagram of the developed computer vision algorithm with examples of the main stages' outputs.

3.3. Game Engine

After obtaining the matrix that represents the board, it is necessary to choose the move to perform, which must respect the Italian checkers rules described in Section 3.1. In the following, we present the structure of the developed gaming algorithm to select the best move.

The gaming algorithm, built upon AI principles, has been designed as a *minimax* algorithm. This is a recursive algorithm frequently employed in decision-making processes and zero-sum games, like Italian checkers. It enables the artificial player to select the optimal move when facing a human opponent, who is expected to make optimal moves as well [22,31]. The algorithm implemented in this work relies on the recursive optimization of an evaluation function that considers rewards and penalties derived from the game strategy.

Evaluation Function (1) is employed to assess the current board configuration, associating it with a score (i.e., the outcome of the evaluation function). This function considers the number of one's own pieces (N_m and N_k, respectively, for men and kings) and of the opponent's ones (N_{om} and N_{ok}, respectively). It is specified that, for notation convenience, the contributions for the four types of pieces are indicated with proper subscripts, respectively m for own men, k for own kings, om for opponent men, and ok for opponent kings. The board configuration is also taken into account in the score computation. For instance, the closer an own man is to becoming a king (i.e., with high $line_m$), the higher the score. Conversely, the closer an opponent man is to becoming a king (i.e., with high $line_{om}$), the lower the score. The symbols $line_m$ and $line_{om}$ mean the row of the square occupied by a man from its point of view. It means that, if a man is in the nearest row to its starting side, its *line* value is equal to 0; instead, if it occupies a square in the row corresponding to its opposite side of the board, its *line* value its equal to 7; intermediate positions have *line* values between 0 and 7. The score is also influenced by the men that are protected from a possible capture (i.e., that do not have an empty square behind them in one or more directions, indicated by pr_m and pr_{om}). Moreover, king position is considered in the score definition, favoring configurations in which the kings occupy the most central squares of the board (c_k and c_{ok}) in order to have a greater chance of movement. A weight is assigned to each factor contributing to the score (ω_m weights the number of men, ω_k the number of kings, $\omega_{m,pos}$ the man position, $\omega_{m,pr}$ the man protection, and $\omega_{k,pos}$ the king position). By varying them, the different contributions can be taken into account differently, thus allowing the game strategy to be modified.

$$\begin{aligned}\text{score} = \omega_m \cdot (N_m - N_{om}) + \omega_k \cdot (N_k - N_{ok}) + \\ \omega_{m,pos} \cdot \left[\sum_{m=1}^{N_m} line_m^2 - \sum_{om=1}^{N_{om}} line_{om}^2\right] + \\ \omega_{m,pr} \cdot \left[\sum_{m=1}^{N_m} pr_m - \sum_{om=1}^{N_{om}} pr_{om}\right] + \\ \omega_{k,pos} \cdot \left[\sum_{k=1}^{N_k} c_k^2 - \sum_{ok=1}^{N_{ok}} c_{ok}^2\right]\end{aligned} \tag{1}$$

The algorithm considers only the top five moves out of all the possible ones that lead to board configurations with higher scores. This restriction is imposed to limit the breadth of the search tree and reduce the computation time, under the assumption that other moves do not result in favorable conditions. For each of these selected moves, the algorithm is then applied from the perspective of the human player, determining their available and better moves. This process is iteratively repeated up to a specified depth, constructing the search tree. Increasing the search depth enhances the artificial player performance, but at the cost of longer computation times, as shown in Section 4.

The proposed gaming algorithm aims to identify the move leading to more favorable game situations, i.e., those where the chance of victory is greatest. This search process involves traversing the tree in a backward manner, starting from the leaf nodes and assuming that the opponent is also making optimal moves from his point of view, as illustrated in Figure 4. When the algorithm encounters a parent node during a turn corresponding to the human player, it assigns a value equal to the minimum among the values of its child nodes (MIN). Differently, if the parent node corresponds to an artificial player turn, it assigns a value equal to the maximum of the values associated with its children (MAX). Finally, the developed gaming algorithm determines the best move to be executed by the robotic manipulator.

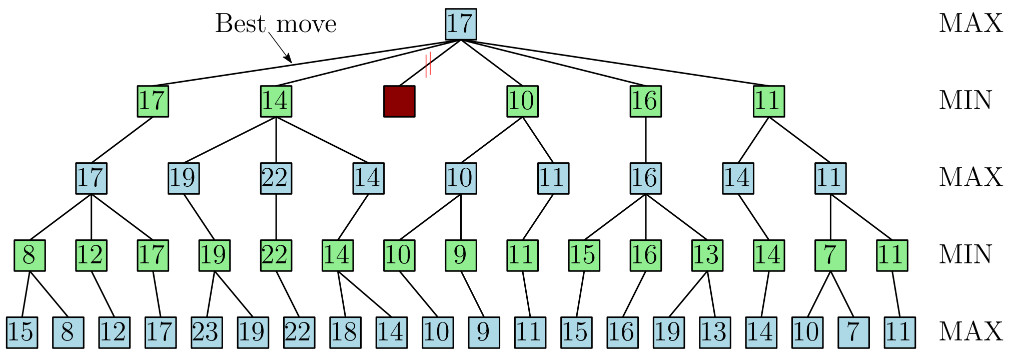


Figure 4. Illustration of structure and operation of the developed gaming algorithm. The light blue color indicates the robot turn, the green color the human turn, the red a not considered move, whereas the numbers correspond to the score values.

3.4. Trajectory Planning

The trajectories for the robot are planned as point-to-point motion in the joint space of the robot using five-degree polynomials with zero velocity and acceleration at the initial and final joint state. The orientation of the end-effector of the robot is maintained constant during each trajectory. The time duration of each point-to-point motion of the robot is minimized online so as to avoid long waiting times for the human player and make the game more fluent and interactive. The optimization problem that computes the minimum time duration online t for each trajectory $\boldsymbol{q}(t)$ for the robot is defined as follows:

$$\min_{t} \quad w_0\, t \tag{2}$$

subject to

$$\begin{aligned} q_{i,min} \leq q_i \leq q_{i,max} \\ |\dot{q}_i| \leq \dot{q}_{i,max} \\ |\ddot{q}_i| \leq \ddot{q}_{i,max} \\ |\dddot{q}_i| \leq \dddot{q}_{i,max} \\ i = 1, ..., N \end{aligned} \tag{3}$$

where w_0 is a positive weight, $q_{i,min}$ and $q_{i,max}$ are the position limits at the i-th joint, whereas $\dot{q}_{i,max}$, $\ddot{q}_{i,max}$, and $\dddot{q}_{i,max}$ are the velocity, acceleration, and jerk limits, respectively. Finally, N represents the number of the degrees of freedom of the robot.

During a typical move, the robot moves from the homing configuration and grasps the piece that has to be moved. Subsequent moves depend on the specific kind of play to be executed:

- in the case of a simple movement, the robot positions the piece in the target square and then returns to the neutral pose;
- if a capture (including multiple ones) has to be made, the robot, after placing its piece in the target square, proceeds to eliminate from the board the opponent piece (or pieces) captured;
- when a move results in a man becoming a king, the robot avoids stacking pieces to make king by itself; instead, it removes the man from the board and takes the king, which is prepared in a reference position outside the board.

An example of a simulation of a pick-and-place operation is shown in Figure 5.

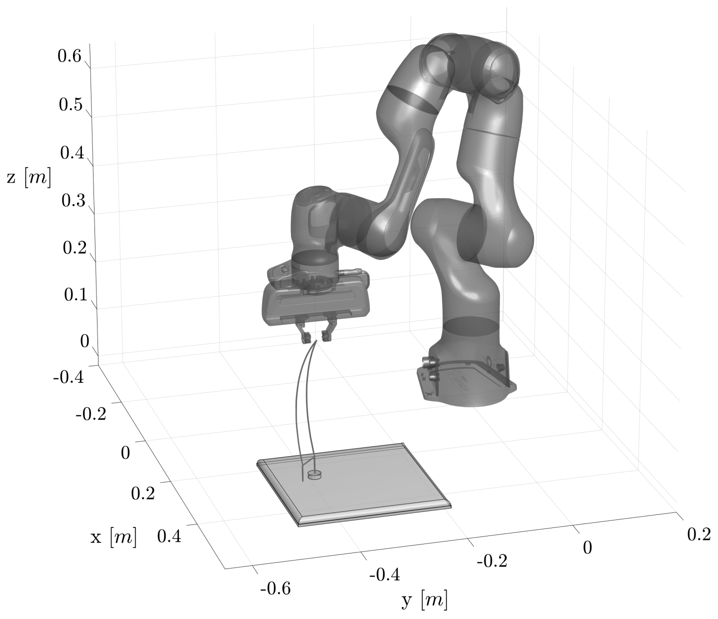


Figure 5. Exemplary robot path in the operational space to pick and place a piece.

3.5. Experimental Setup

The checkers-playing robotic system presented in this paper consists of the following main components:

- robotic arm;
- computer and control system;
- camera;
- board and pieces.

As pieces manipulator, a Franka Emika robotic arm with seven degrees of freedom is used. This robot exhibits an industrial-grade pose repeatability of ± 0.1 mm, a payload capacity of 3 kg, and an operational reach of 855 mm. A Franka hand is adopted as robot end-effector, a two-finger gripper needed for grasping and manipulating pieces. This gripper has a travel range of 80 mm and a maximum force of 70 N. The computer used to implement the whole system runs Ubuntu 18.04 with an Intel Core i5-10600K CPU @ 4.10 GHz and 31.2 GB of RAM. Python 3 and ROS (Robot Operating System) Melodic are used to control and drive the manipulator. Particularly, the MoveIt motion planning framework is used to send the control values to the robot. The gaming algorithm is implemented using Python 3, whereas the developed computer vision algorithm is written in GNU Octave. Through the `oct2py` library, these Octave functions can be conveniently invoked from the main Python program.

The optimization problem in Equations (2) and (3) is implemented using the IPOPT algorithm of the open-source tool for nonlinear optimization CasADi [32]. The maximum number of iterations of the optimization is set equal to 7, whereas the lower and upper bounds for the time duration of each robot trajectory are 0.2 s and 1.5 s, respectively. For the sake of simplicity, in this work, the dynamics of the robot are not considered so as to minimize the solution time of the optimization problem. However, in case the torque constraints of the robot need to be verified, the dynamics parameters of the Franka Emika arm identified in [33] can be used.

To implement the computer vision system and capture the board image, we employed an Intel RealSense D435 depth camera, equipped with a maximum resolution of 1920×1080 pixels. The camera is mounted on a tripod and placed above the board in order to clearly distinguish the different squares and their content in a position that does not obstruct the robot and human moves.

The relative positioning of the robot and the board is held constant. This approach avoids the need for continuous detection and tracking of the board position, allowing to streamline the algorithms, particularly the computer vision one, and to decrease the total computational time. Additionally, we have opted for the use of a commercial checkerboard and game pieces with standard colors and shapes. The checkerboard has dimensions equal to 256×256 mm, and each square equal to 32×32 mm, whereas the pieces have a diameter of 25 mm and a height of 8 mm. This choice is made to maintain the authenticity of the game and avoid any alterations to the traditional appearance and characteristics of the board and pieces. Additionally, a box for depositing the captured pieces and a reference where to go to pick up the prepared kings is added to the system. The experimental setup is illustrated in Figure 1, whereas Figure 6 depicts how the different sections of the system work together during a game. Table 2 summarizes the main characteristics of the hardware and software components.

A calibration procedure is required for the robot before the game start. This process allows the algorithm to determine the board position relative to the robot base reference frame. For the calibration, it is sufficient to acquire the position of the four corners of the board in the robot base reference frame. This is accomplished using a custom calibration tip as robot end-effector, printed in 3D using an Ultimaker S5 Pro Bundle. Touching the four corners with the tip, as illustrated in Figure 7a, their location in the Cartesian space is determined using the direct kinematics of the manipulator. The procedure starts from the corner to the left of the robot and continues in a counterclockwise direction (Figure 7b).

However, it should be noted that the position of only three corners is strictly required for the robot calibration, as outlined in the following; the fourth corner position is acquired only to ease the computation of the coordinates of the board center.

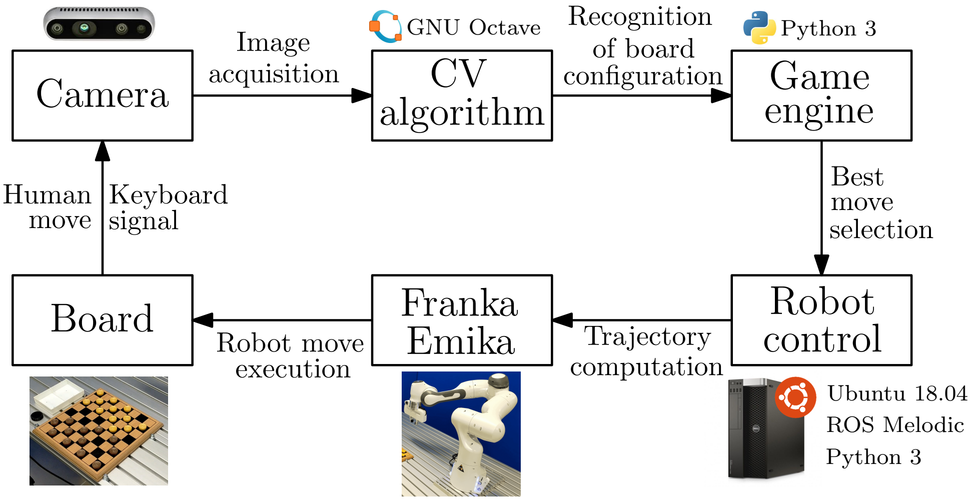
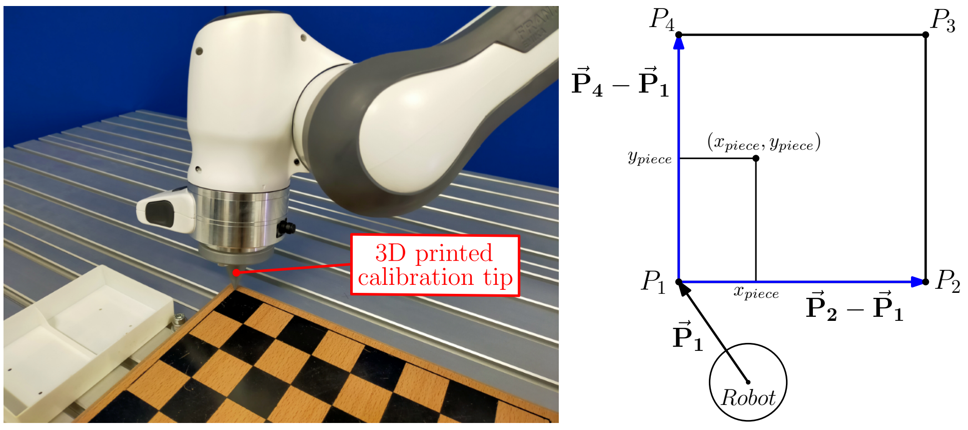


Figure 6. Workflow diagram of the developed system.

Table 2. Specifications of the hardware and software components.

Component	Main Features
Computer	• Operating system: Ubuntu 18.04 • CPU: Intel Core i5-10600K @ 4.10 GHz • RAM: 31.2 GB
Vision system	• Camera: Intel RealSense D435 • Resolution: 1920 × 1080 pixels • Programming language: GNU Octave
Game engine	• Structure: *minimax* algorithm • Programming language: Python 3
Robotic player	• Manipulator: Franka Emika arm • Gripper: Franka hand • Control framework: ROS Melodic • Programming language: Python 3
Checkerboard	• Total dimensions: 256 × 256 mm • Square dimensions: 32 × 32 mm
Pieces	• Diameter: 25 mm • Height: 8 mm


3D printed calibration tip
P_4
P_3
$\vec{\mathbf{P}}_4 - \vec{\mathbf{P}}_1$
(x_{piece}, y_{piece})
y_{piece}
P_1
x_{piece}
P_2
$\vec{\mathbf{P}}_2 - \vec{\mathbf{P}}_1$
$\vec{\mathbf{P}}_1$
Robot

(**a**) (**b**)

Figure 7. The robot touches one of the board corners with the calibration tip as end-effector (**a**); representation of the change in coordinates from board to robot reference system (**b**).

Knowing the location of the board with respect to the robot, the position of the pieces can be easily determined since the vision algorithm records the positions of the pieces in the board coordinate system using coordinates normalized between 0 and 1. These coordinates can be transformed in the robot system as shown in Figure 7b. This transformation is achieved by multiplying the x coordinate of the pieces by vector $\vec{\boldsymbol{P}}_2 - \vec{\boldsymbol{P}}_1$ and their y coordinate by vector $\vec{\boldsymbol{P}}_4 - \vec{\boldsymbol{P}}_1$ and then adding vector $\vec{\boldsymbol{P}}_1$. Vectors $\vec{\boldsymbol{P}}_i$ with $i = 1, \ldots, 4$ represent the coordinates of the four corners in the robot system. Moreover, a homing configuration is established from which the robot begins its movement and returns to after a move is made (Figure 1). Care must be taken to ensure that this configuration does not occlude the view of the camera.

Through the previous procedure, the system knows the position of the board with respect to the robot, but the board position in the camera reference frame is still unknown, so a camera calibration process is needed. Since the position of the camera can be varied until a suitable position is found, a board-camera calibration is chosen, which is faster and simpler than a robot-camera calibration. The positions of the board corners are manually identified on the image taken by the camera, maintaining the same order as used during the robot calibration. This step ensures the alignment of the real-world corners with their counterparts in the image. Next, the user assesses whether the *homography* and the black and white image are acceptable. In that case, the user proceeds to indicate the positions of a white square, black square, white piece, and black piece within the image. Alternatively, if the *homography* or black and white image quality is deemed unsatisfactory, the camera setup process is restarted, involving adjustments to lighting and/or camera position until acceptable results are achieved. This camera-board calibration process will need to be repeated only when the camera position is altered.

Once this setup is completed, the game can start. For safety reasons, the robot does not start its movements right after the human player moves; instead, it waits for an input signal from the keyboard.

4. Results

The principal operations executed during a game are illustrated in Figure 8. In particular, Figure 8a,b show a piece that is been picked, Figure 8c,d depict the moving of a piece, and in Figure 8e,f the placing of a piece can be seen. Figure 8g shows a captured piece that is been grasped, whereas, in Figure 8h, a captured piece being removed from the board is depicted. Finally, a prepared king picked up from the reference point can be seen in Figure 8i. The system functionality and performance are also illustrated in a video available online (https://www.youtube.com/watch?v=KiR5qAI5S2M, accessed on 12 December 2023).

In the following, the performance of the entire algorithm in terms of computational time and success rate, from image acquisition to selection and execution of the move, is analyzed. The run time of the developed computer vision algorithm remains relatively constant, typically about 1 s. This stability is due to the fact that the operations performed by the algorithm are practically always the same for each image to be analyzed. Moreover, it can be noted that eliminating board detection from the algorithm results in a low calculation time, thus reducing the total execution time of the move.

Regarding its success rate, it is important to note that the algorithm is sensitive to lighting conditions, particularly with regard to the presence of shadows and reflections on the board. To solve these issues, the system is illuminated with a light source placed directly above the board. This setup ensures a consistent and controlled lighting environment, thus significantly reducing the impact of lighting variations. As a result, under standard lighting conditions, the developed computer vision algorithm demonstrates a good degree of accuracy in the piece detection.

As outlined in Section 3.3, the gaming algorithm is developed on the basis of a *minimax* algorithm. The run time of this algorithm is influenced by both the chosen search depth and the maximum number of moves taken into account (which affects the breadth of the decision tree). Figure 9 provides an overview of the average time required varying these two parameters. A duration of approximately 1–2 s can be deemed suitable for a smooth gaming experience. Choosing, for example, six levels for the search depth and five considered moves, the computational times for the gaming algorithm are below one second.

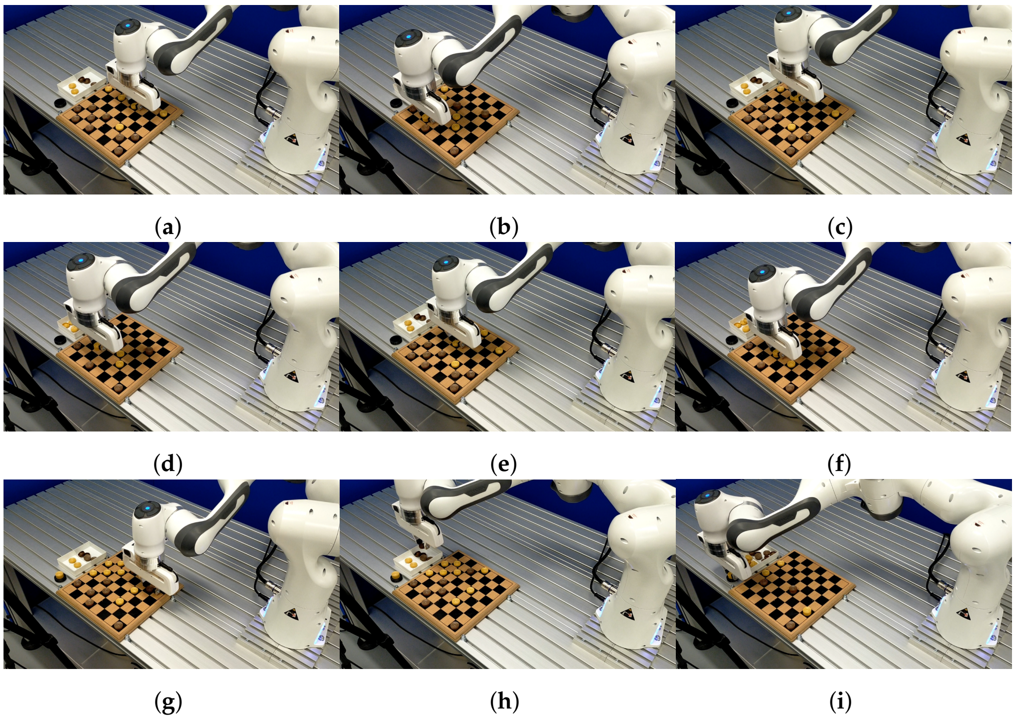


Figure 8. Main actions executed during a game: pick a piece (**a**,**b**), move a piece (**c**,**d**), place a piece (**e**,**f**), pick a captured opponent piece for removing it (**g**), remove a captured opponent piece from board (**h**), and pick a king from the reference point (**i**).

Furthermore, based on several games played, we observed that, against players who are not highly skilled, a search depth of four is often sufficient to provide a sufficiently challenging game. This is due to the fact that, in such games, most nonprofessional human players typically plan only two or three moves ahead at most [13].

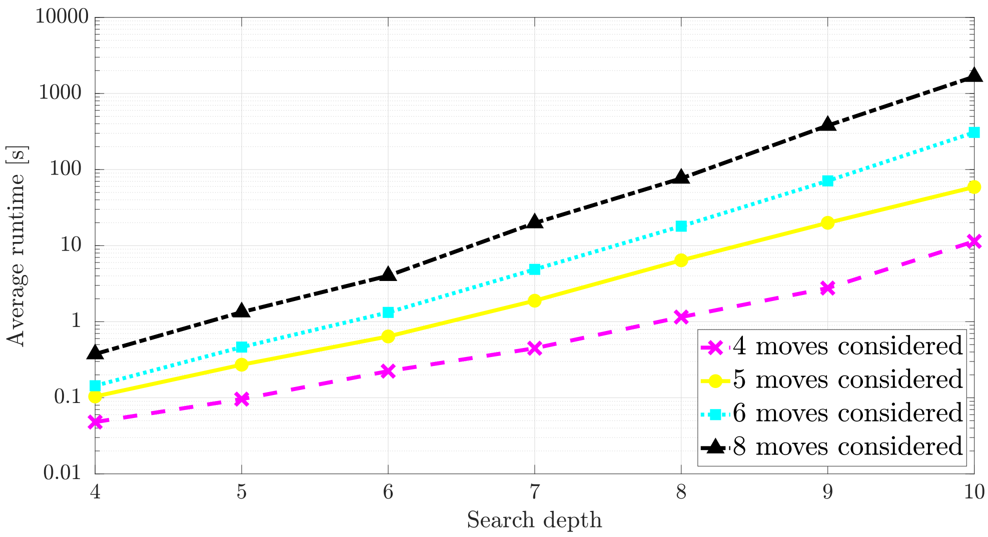
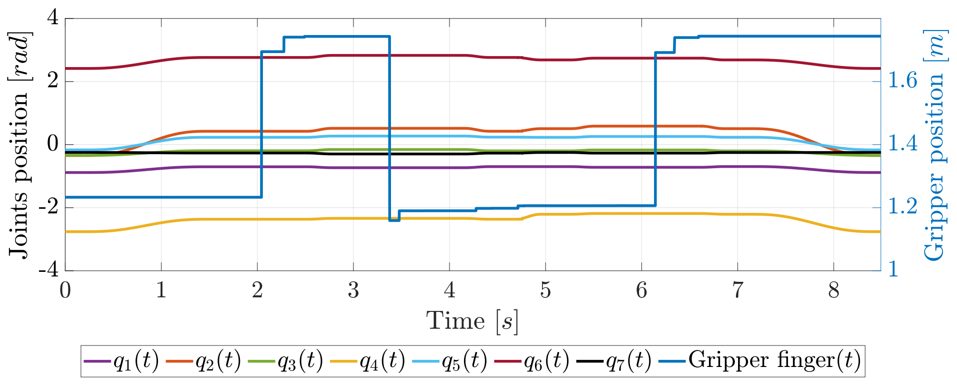
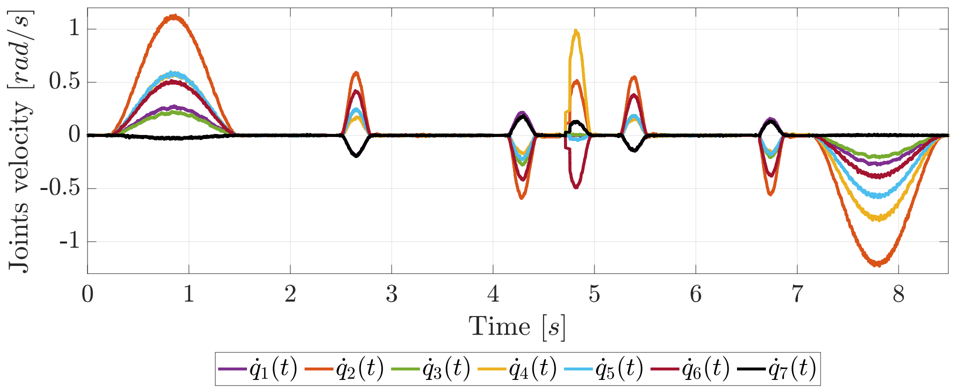


Figure 9. Average execution time of the developed gaming algorithm by varying the search depth and the maximum number of considered moves.

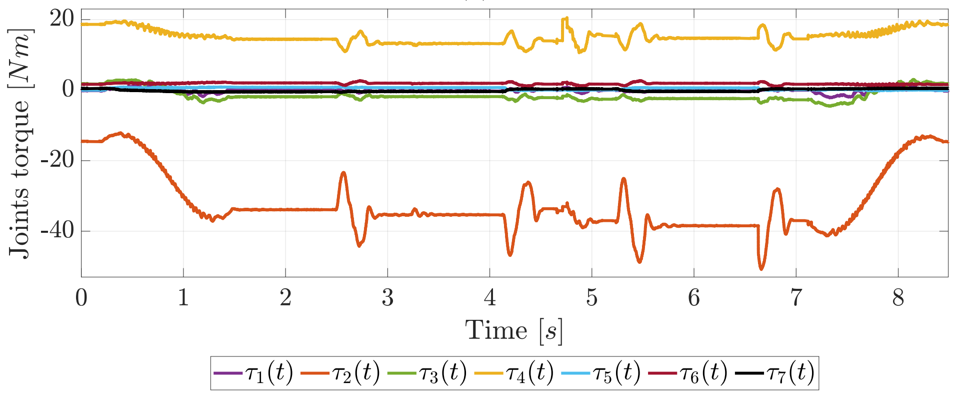
By considering the approach for trajectory planning, the optimization of the time duration of each point-to-point motion of the robot guarantees a maximum execution time equal to 1.5 s while meeting the joint position, velocity, acceleration, and jerk limits of the manipulator. An exemplary pick-and-place trajectory is shown in Figure 10, where the joints and gripper positions, joints velocities, and joints torques over time are reported. This trajectory corresponds to the motion of the robot in the Cartesian space shown in Figure 5. In particular, from Figure 10, it can be seen that the total duration of this exemplary pick-and-place trajectory for a single piece of the board is equal to 8.5 s. This time comprises the computational time for the optimization of each trajectory segment (usually below 10 ms for each point-to-point motion), as well as the time to open and close the gripper for picking and releasing the piece on the board.



(**a**)



(**b**)



(**c**)

Figure 10. Exemplary pick-and-place trajectory: joints and gripper positions (**a**), joints velocities (**b**), and joints torques (**c**) over time, corresponding to the motion in the Cartesian space shown in Figure 5.

Finally, we can state that the computer vision algorithm for identifying the position of the pieces and the use of the Franka hand for grasping and moving them on the gaming board provide high performance without failures. Indeed, no episodes of loss of pieces on the board during the game have been recorded and no necessity of customizing the gripper fingers has been raised.

5. Conclusions

This paper introduced an intelligent and collaborative robotic system designed for playing Italian checkers. The system is capable of perceiving the game state through a camera, assessing the optimal moves, and physically manipulating the game pieces. Minimum-time trajectories are optimized online for pick-and-place operation of the robot so as to make the game more fluent and interactive while meeting the kinematic constraints of the manipulator.

Extensive experimental tests have been successfully performed, completing full games without errors and demonstrating the viability of the proposed approach. The three modules that compose the described system, i.e., game state evaluation, game engine, and robot control, are distinct and independent of each other. This modular approach offers flexibility for future improvements and developments in all three parts as each module can be modified independently of the others and without affecting their functionality.

In future works, we plan to further improve all the algorithm modules of our intelligent and collaborative robotic system for playing Italian checkers. In more detail, to provide a pathway for subsequent studies, we will consider the following aspects:

- Vision system: the robustness and reliability of the computer vision algorithm will be improved through the implementation of an approach based on model recognition, which is more robust with respect to illumination, and an ad hoc illumination system;
- Computational times: the alpha–beta pruning algorithm will be implemented to speed up the execution times of the gaming algorithm and enhance its overall performance;
- Safety of the human player: safety strategies will be implemented to track the human position in real time and stop the robot in the case of a potential collision, as in [2]. Furthermore, the safety of the gripper will also be considered to avoid accidents, such as the one in 2022, when a chess robot broke a finger of 7-year-old boy during a tournament in Moscow [34].

Author Contributions: G.F., L.S. and A.G. contributed to the study conception and design. Material preparation, data collection, and analysis were performed by G.F. and L.S. The first draft of the manuscript was written by G.F. and L.S. All authors read and approved the final manuscript.

Funding: This work has been partially supported by iNEST-Interconnected NordEst Innovation Ecosystem, funded by PNRR (Mission 4.2, Investment 1.5), NextGeneration EU—Project ID: ECS 00000043, and developed within the Laboratory for Big Data, IoT, Cyber Security (LABIC), funded by Friuli Venezia Giulia region, and the Laboratory for Artificial Intelligence for Human–Robot Collaboration (AI4HRC), funded by Fondazione Friuli.

Data Availability Statement: Data are contained within the article.

Acknowledgments: The authors would like to thank Diego Masotti, Jacopo Foltran, Alberto Ragazzon, and Federico Lozer for their help in the experimental setup.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Vicentini, F. Collaborative robotics: A survey. *J. Mech. Des.* **2021**, *143*, 040802. [CrossRef]
2. Scalera, L.; Giusti, A.; Vidoni, R.; Gasparetto, A. Enhancing fluency and productivity in human-robot collaboration through online scaling of dynamic safety zones. *Int. J. Adv. Manuf. Technol.* **2022**, *121*, 6783–6798. [CrossRef]
3. Dell'Ariccia, A.; Bremers, A.W.; Lee, W.-Y.; Ju, W. "Ah! he wants to win!": Social responses to playing Tic-Tac-Toe against a physical drawing robot. In Proceedings of the 16th International Conference on Tangible, Embedded, and Embodied Interaction, Daejeon, Republic of Korea, 13–16 February 2022 ; pp. 1–6.

4. Carbonari, L.; Forlini, M.; Scoccia, C.; Costa, D.; Palpacelli, M.-C. Disseminating Collaborative Robotics and Artificial Intelligence Through a Board Game Demo. In Proceedings of the 18th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA), Taipei, Taiwan, 28–30 November 2022; pp. 1–5.
5. Karimov, A.; Kopets, E.; Leonov, S.; Scalera, L.; Butusov, D. A Robot for Artistic Painting in Authentic Colors. *J. Intell. Robot. Syst.* **2023**, *107*, 34. [CrossRef]
6. Lin, J.-Y.; Kawai, M.; Nishio, Y.; Cosentino, S.; Takanishi, A. Development of performance system with musical dynamics expression on humanoid saxophonist robot. *IEEE Robot. Autom. Lett.* **2019**, *4*, 1684–1690. [CrossRef]
7. Samuel, A. L. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.* **1959**, *3*, 210–229. [CrossRef]
8. Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature* **2020**, *588*, 604–609. [CrossRef] [PubMed]
9. Bernbaum, A.; Greenberg, B.; Latreille, J.; Mistry, S.; Pattison, L.; Ruegg, P.; Zhang, S. De3-rob1 Chess Group Documentation, Chess Project for the Robotics 1 Module in Design Engineering, Imperial College London. 2018. Available online: https://de3-rob1-chess.readthedocs.io/en/latest/ (accessed on 5 December 2023).
10. Fabris, G.; Scalera, L.; Gasparetto, A. An interactive collaborative robotic system to play Italian checkers. In *IFToMM World Congress on Mechanism and Machine Science*; Springer: Berlin/Heidelberg, Germany, 2023; pp. 74–84.
11. Juang, L.-H. Humanoid robots play chess using visual control. *Multimed. Tools Appl.* **2022**, *81*, 1545–1566. [CrossRef]
12. Kołosowski, P.; Wolniakowski, A.; Miatliuk, K. Collaborative robot system for playing chess. In Proceedings of the International Conference Mechatronic Systems and Materials (MSM), Bialystok, Poland, 1–3 July 2020; pp. 1–6.
13. Chen, A.T.-Y.; Wang, K.I.-K. Robust computer vision chess analysis and interaction with a humanoid robot. *Computers* **2019**, *8*, 14. [CrossRef]
14. del Toro, C.; Robles-Algarín, C.; Rodríguez-Álvarez, O. Design and construction of a cost-effective didactic robotic arm for playing chess, using an artificial vision system. *Electronics* **2019**, *8*, 1154. [CrossRef]
15. Barakova, E.I.; De Haas, M.; Kuijpers, W.; Irigoyen, N.; Betancourt, A. Socially grounded game strategy enhances bonding and perceived smartness of a humanoid robot. *Connect. Sci.* **2018**, *30*, 81–98. [CrossRef]
16. Brooks, D.J.; McCann, E.; Allspaw, J.; Medvedev, M.; Yanco, H.A. Sense, plan, triple jump. In Proceedings of the IEEE International Conference on Technologies for Practical Robot Applications (TePRA), Woburn, MA, USA, 11–12 May 2015; pp. 1–6.
17. Larregay, G.; Pinna, F.; Avila, L.; Morán, D. Design and Implementation of a Computer Vision System for an Autonomous Chess-Playing Robot. *J. Comput. Sci. Technol.* **2018**, *18*, 1–11. [CrossRef]
18. Luqman, H.M.; Zaffar, M. Chess brain and autonomous chess playing robotic system. In Proceedings of the International Conference on Autonomous Robot Systems and Competitions (ICARSC), Bragança, Portugal, 4–6 May 2016; pp. 211–216.
19. Gupta, V.; Kumar, A.; Agrawal, S.; Jaiswal, S. Autonomous Chess Playing Robot. *Int. J. Eng. Res. Technol.* **2015**, *4*.
20. Kopets, E.E.; Karimov, A.I.; Kolev, G.Y.; Scalera, L.; Butusov, D.N. Interactive Robot for Playing Russian Checkers. *Robotics* **2020**, *9*, 107. [CrossRef]
21. Rath, P.K.; Mahapatro, N.; Nath, P.; Dash, R. Autonomous Chess Playing Robot. In Proceedings of the 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), New Delhi, India, 14–18 October 2019; pp. 1–6.
22. Escandon, E.R.; Campion, J. Minimax checkers playing GUI: A foundation for AI applications. In Proceedings of the XXV International Conference on Electronics, Electrical Engineering and Computing (INTERCON), Lima, Peru, 8–10 August 2018; pp. 1–4.
23. Nasa, R.; Didwania, R.; Maji, S.; Kumar, V. Alpha-Beta Pruning in Mini-Max Algorithm–An Optimized Approach for a Connect-4 Game. *Int. Res. J. Eng. Technol.* **2018**, *5*, 1637–1641.
24. Elnaggar, A.A.; Gadallah, M.; Aziem, M.A.; Aldeeb, H. Autonomous checkers robot using enhanced massive parallel game tree search. In Proceedings of the 2014 9th International Conference on Informatics and Systems, Cairo, Egypt, 15–17 December 2014; pp. 35–44.
25. Rodrıguez-Sedano, F.J.; Esteban, G.; Inyesto, L.; Blanco, P.; Rodrıguez-Lera, F.J. Strategies for haptic-robotic teleoperation in board games: Playing checkers with Baxter. *Strategies* **2016**, 31–37.
26. Carrera, L.; Morales, F.; Tobar, J.; Loza, D. MARTI: A robotic chess module with interactive table, for learning purposes. In Proceedings of the World Congress on Engineering and Computer Science, San Francisco, CA, USA, 25–27 October 2017; pp. 25–27.
27. Lukač, D. Playing chess with the assistance of an industrial robot. In Proceedings of the 3rd International Conference on Control and Robotics Engineering (ICCRE), Nagoya, Japan, 20–23 April 2018; pp. 1–5.
28. Manurung, E.B. Gantry Robot System Checkers Player. *ADI J. Recent Innov.* **2023**, *5*, 9–19. [CrossRef]
29. Matuszek, C.; Mayton, B.; Aimi, R.; Deisenroth, M.P.; Bo, L.; Chu, R.; Kung, M.; LeGrand, L.; Smith, J.R.; Fox, D. Gambit: An autonomous chess-playing robotic system. In Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 4291–4297.
30. Italian Checkers Rules. Available online: https://www.boardgamecentral.com/rules/checkers-rules-italian.html (accessed on 2 October 2023).
31. Diez, S.G.; Laforge, J.; Saerens, M. Rminimax: An optimally randomized MINIMAX algorithm. *IEEE Trans. Cybern.* **2012**, *43*, 385–393. [CrossRef] [PubMed]

32. Andersson, J.A.; Gillis, J.; Horn, G.; Rawlings, J.B.; Diehl, M. CasADi: A software framework for nonlinear optimization and optimal control. *Math. Program. Comput.* **2019**, *11*, 1–36. [CrossRef]
33. Gaz, C.; Cognetti, M.; Oliva, A.; Giordano, P.R.; De Luca, A. Dynamic identification of the franka emika panda robot with retrieval of feasible parameters using penalty-based optimization. *IEEE Robot. Autom. Lett.* **2019**, *4*, 4147–4154. [CrossRef]
34. Chess Robot Grabs and Breaks Finger of Seven-Year-Old Opponent. Available online: https://www.theguardian.com/sport/2022/jul/24/chess-robot-grabs-and-breaks-finger-of-seven-year-old-opponent-moscow (accessed on 5 December 2023).