Journal of Computer Science & Technology, Volume 18, Number 1, April 2018

- ORIGINAL ARTICLE -

Design and Implementation of a Computer Vision System for an Autonomous Chess-Playing Robot

Guillermo Larregay[1], Federico Pinna[1], Luis Avila[1,2], and Daniel Morán[1]

[1]*Laboratorio de Mecatrónica, Universidad Nacional de San Luis, Villa Mercedes, San Luis, Argentina.*
[2]*Laboratorio de Investigación y Desarrollo en Inteligencia Computacional, Universidad Nacional de San Luis, San Luis, Argentina.*
{golarregay, fpinna, loavila, dmoran}@unsl.edu.ar

Abstract

This work describes a mechatronic system composed by a robot arm that can play chess autonomously. The system is based on an industrial-grade robot manipulator, a computer vision system, and an open source chess engine. Classification algorithms were implemented in order to detect whether a given chessboard square is occupied, and in that case, if the piece is black or white. Such algorithms were compared in terms of their complexity of implementation, execution time and accuracy of predictions. To achieve an uniform illumination of the chessboard, a theoretical model of an LED illuminance curve was used to find the best orientation for each diode using a genetic algorithm. Both the support base for the LEDs and the chess pieces were made using a 3D printer. This implementation demonstrates the capabilities of the proposed vision-based system, whose complexity can be increased in the future for a number of applications.

Keywords: chess, computer vision, decision making, human-robot cooperation, mechatronics.

1 Introduction

In general, chess games require the location and identification of a set of pieces on a board, knowledge of the game rules -the set of posible valid moves to execute-, a representation of the actual state of the game for future decision making, and the manipulation of physical parts; all this while coordinating with the human opponent. This work presents a mechatronic system consisting of a robotic arm that plays chess in an autonomous manner against a human opponent.

Briefly, this implementation is based on an industrial-grade robot manipulator, a computer vision system, and an open source chess game engine. The chess engine, based on the GNU Chess open source package, is used as the back-end engine for the chess game. In order to achieve a proper and uniform level of illumination on the chessboard surface, a theoretical model of the illuminance curve of an LED was used as the evaluation function for a genetic algorithm, implemented to optimize each LED orientation using heuristic techniques so as to uniformly distribute the average illumination over the chessboard surface. Once a good enough light distribution was found, a 3D support base model was generated and 3D printed, where the LEDs were mounted according to the optimal orientation obtained. The system's hardware might be easily increased, so that more demanding tasks can be performed, attending or the uncertainty and physical restrictions of real environments [1].

A number of works describing the human-robot interaction have extensively analyzed the decision-making by turns problem in different contexts [2], [3]. There also exist diverse implementations of systems involving industrial type robots in strategy games such as chess and checkers among others. One of the first implementations, the Marine-Blue [4] is undoubtedly one of the most recognized. Some applications, such as in [5], use a simplified XY linear slide system. However, those autonomous systems commonly use instrumented boards and parts specially co-designed with the manipulator in order to simplify the tasks of piece detection and manipulation [6]. This issue results in specific designs that do not incorporate, in the design phase, the complexities introduced by the use of arbitrary pieces and boards on real environments. The implementations in Gambit [6] and Baxter [7] seems to be the closest systems to the one here presented in terms of functionality. Gambit's use SVMs to learn the piece types and besides it uses two cameras including depth information. Similarly

Citation: G. Larregay, F. Pinna, L. Avila and D. Morán. *Design and Implementation of a Computer Vision System for an Autonomous Chess-Playing Robot*. Journal of Computer Science & Technology, vol. 18, no. 1, pp. 1-11, 2018.

DOI: 10.24215/16666038.18.e01

Received: August 2, 2017 **Revised:** October 04, 2017 **Accepted:** December 29, 2017.

to our work, Baxter takes a simpler approach to the perception problem since its computer vision system is based on a single camera embedded in the robot's arm.

Particularly, the mechatronic system here presented does not require of any special instrumentation or specific modeling of parts in order to perform piece detection, which guarantees the feasibility of performing multiple and different tasks in future applications.

Since the chessboard is not referenced to the robot arm, the board corners can be located anywhere inside the image. Using edge and corner detection algorithms, the region of interest is dynamically extracted on each picture, see Fig. 3.

The chessboard state is monitored before and after the human player makes his movement, in order to detect it. In particular, we determine the difference between a known state of the chessboard, prior to the movement of the human player, and a new state to be inferred from a picture captured with a low-cost camera. The detection of the different pieces is carried out by color and not by shape. Because of that, the detection process, in addition to white and black pieces, has to classify two extra colors corresponding to the empty squares, i.e. light and dark squares. Therefore, the classification problem can be summarized as determining the content of each square of the chessboard, from the set of valid results: $\{$*white piece*, *black piece*, *nothing*$\}$, taking into consideration that there are two possible values for *nothing*. It should be noted that it is not necessary to determine which piece is in the square (e.g. pawn, bishop, etc.) since the game engine is the responsible for tracking the movements of each piece.

Results demonstrates the capabilities of the proposed vision-based chess playing system in uncertain environments, such as with uncontrolled lighting conditions. The system's complexity is able to be increased in the future for a number of applications. This work also contributing to numerous studies on human-computer interactions.

2 Methods

One of the main characteristics of the mechatronic system presented in this paper is that it is based on a commercial, industrial-grade robotic arm. This enables a broader and more diverse set of tasks that can be carried out. A picture of the system can be seen in Fig. 1.

In particular, the manipulator consists of an ABB model IRB120 manipulator with 6 degrees of freedom (DOF). The first three DOF are rotational and provide position control, while the last three provide control and orientation of the end effector of the robot, which consists of a 4-finger parallel gripper and can be easily replaced to accomplish other particular tasks. The use of a commercial and industrial-grade robotic manipulator also allows the creation of an open and flexible platform that may support the future expansion both in terms of hardware and software.

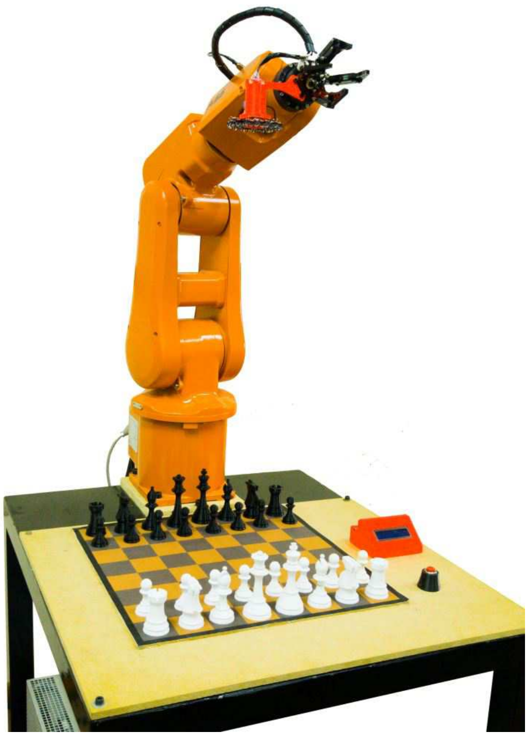


Figure 1: 6 DOF robot manipulator with controller, and the chessboard used.

For visual detection of the chess pieces, the camera support was associated with an LED arrangement to guarantee a uniform distribution of illumination on the chessboard. Regarding the architecture of the system, it consists of the following subsystems:

- Computer vision subsystem
- Open source chess game engine
- Central Control Unit software
- Communication interface with robot controller
- Robot arm

This architecture is organized around a software module called the *Central Control Unit* (CCU), as shown in Fig. 2. The goal of the computer vision subsystem is to monitor the current state of the chessboard. Part of this process consists in detecting the location of the chessboard with respect to the robot's base coordinate system, as it is not fixed it can be accidentally moved during gameplay. Once the system correctly locates the chessboard and identifies the pieces in each square, the state of the game in the chess engine is updated and evaluated. Following, the engine determines its next movement, sends its decision to the CCU, which in turn converts the chess

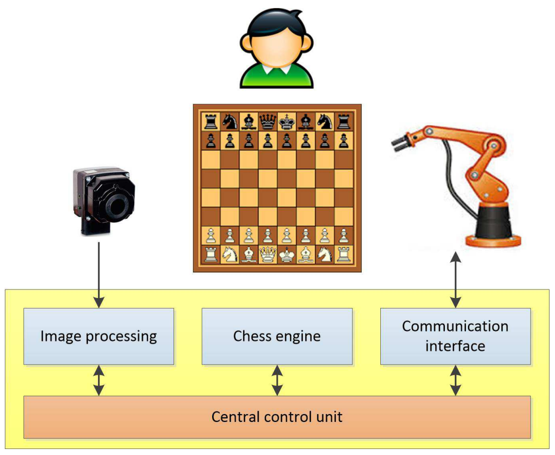


Figure 2: Brief description of the architecture of the autonomous chess-playing system.

notation into physical coordinates and commands the robot controller to execute the actions. Then, the industrial manipulator moves and/or captures the chess pieces involved. Finally, the system passes the turn to the opponent, and the robot stands by in its rest position until the human player moves and passes the turn activating a button associated with the play timer.

2.1 Computer vision algorithms

2.1.1 Image processing

To allow for proper recognition of the pieces on the chessboard, a photograph of the chessboard is captured and then processed to obtain the most important features of the squares and the pieces located in each them. To achieve this, a 1080p Full-HD camera with a resolution of 640x480 was mounted on the robot gripper. Low-cost cameras quality is usually very low, with high amounts of color noise, most notably in low-light areas of the scene. Once the photograph has been captured, the first step of processing consists in extracting the region of interest (the chessboard), improving the contrast and color saturation of the image, and finally sampling the colors present in the center of each square. The full sequence is detailed in Fig. 3.

OpenCV (Open Source Computer Vision Library) was used to process the obtained image. Therefore, at the beginning of a game, the manipulator moves to a position that allows the entire chessboard to be visible to the camera. Since the aspect ratio of the captured image is 4:3, the picture must be cut to retain only the region containing the chessboard. In addition, due to the chromatic characteristics of the LED illumination, it is also necessary to correct the color temperature and increase the color saturation value.

Before starting the analysis, it is necessary to compensate for variations in the illumination of the surface of the chessboard, in order to avoid possible problems in the piece detection task. To this end, an algorithm to equalize the image luminance histogram

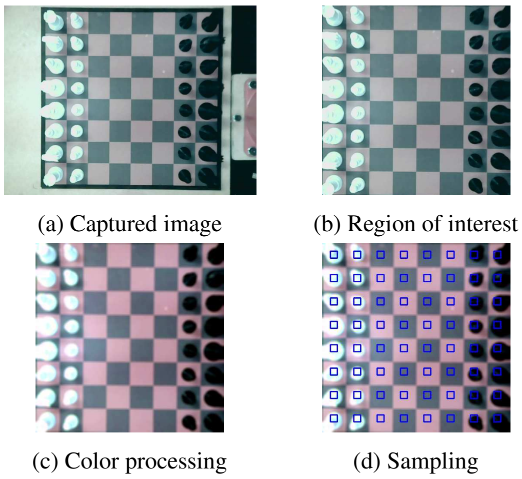


Figure 3: Sampling process for the color detection of the pieces.

was implemented. In particular, the *Contrast-Limited Adaptive Histogram Equalization* (CLAHE) was used [8]. This algorithm divides the image into sectors and subsequently transforms the contrast of each pixel according to its environment, so as to achieve a maximum contrast range in each sector of the image. The correct distribution of illumination on the surface of the chessboard is carried out by a set of LEDs, whose optimal arrangement was obtained by means of a genetic algorithm. The optimum illumination distribution obtained can be observed in Fig. 9, while the methodology used is described in section 2.2.

To compute the equalization on the illumination differences, the image has to be previously converted from the RGB color space to a color space with a luminance channel, and then apply CLAHE on this channel. In the present work the HSL (Hue - Saturation - Lightness) color space was used, since the Lightness channel gives a good approximation to the luminance, and it also has the advantage that the color information from the Hue channel makes it relatively easy to classify the contents of the squares. However, a known disadvantage of histogram equalization algorithms is that, when there is luminance noise in the image, the results obtained are far from optimal. To overcome this problem, the image was pre-filtered using Gaussian blur to suppress the thermal noise characteristic of the camera sensor and to soften the image colors.

2.1.2 Square content detection

Three different classification algorithms to detect the pieces were implemented and compared in terms of their complexity of implementation, execution time and accuracy of predictions.

Multisampling and voting

This technique consists of a supervised learning algorithm that takes multiple color samples from each square (7 samples per square) and predicts the class for each sample, determining if it corresponds to a white, black, or empty square given a distance metric. This algorithm needs a quick calibration process carried out automatically at the beginning of each game, and it consists of getting a *reference color* for the white pieces, black pieces and empty squares. This reference color is calculated as the average color of the 9 center pixels of each square containing white pieces, black pieces or nothing.

The multiple samples approach is used to improve the detection rates in the cases where, during the game, the pieces are not perfectly centered in the square, or in some cases, to avoid light reflections on some pieces. When all the samples are taken, a simple voting task is performed:

1. If there are at least two samples closer to the white reference than to the empty squares reference, and the number of samples closer to the white reference is greater than the number of samples closer to the black reference, then it is considered that the square is occupied by a white piece.

2. If there are at least two samples closer to the black reference than to the empty squares reference, and the number of samples closer to the black reference is greater than the number of samples closer to the white reference, then it is considered that the square is occupied by a black piece.

3. In any other case, then it is considered that the square is empty.

The distance metric used was the Euclidean distance in the HSL color space.

Modified k-medoids

This algorithm implements a modification of the original k-medoids algorithm [9]. In particular, our implementation needs a calibration process similar to the one used in the *Multisampling and voting* algorithm. In this case, the references are used as initial medoids, in order to optimize the convergence time of the algorithm and the reliability of the detection process. Unlike the k-means clustering algorithm that results sensitive to extreme values, since the mean is easily influenced by extreme values, clustering with k-medoids is more robust when noise and outliers are present in the data [10]. This is due to the fact that instead of using the midpoint as the center of a group, k-medoids uses a real point in the group to represent it. The medoid is the most centric object of the cluster and with the minimum sum of distances to other points. For this implementation k = 4. That is, it is classified into four groups, according to white pieces, black pieces, empty light squares and empty dark squares. However, the two groups belonging to empty squares were unified in a single group.

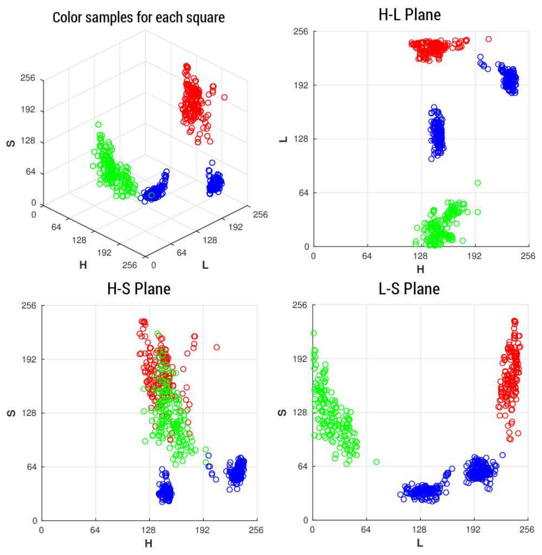


Figure 4: Training set used for the decision tree algorithm. White pieces (red), black pieces (green) and empty squares (blue).

Decision tree

Finally, a decision tree classifier was implemented. The decision tree was trained using selected data and the WEKA tool. Data was divided into three classes: white pieces, black pieces, and empty squares. The data file had 704 instances, of which 170 belonged to the white pieces class, 166 to the black pieces class and the remaining 368 to the empty squares class. Training points and classes are shown in Fig. 4.

All algorithms were implemented in the C++ language. In order to evaluate the robustness of each of them, a test set consisting of 80 images of different chessboard states and different lighting situations (natural light, LED and fluorescent) was employed. In addition, some images were purposelly taken with partial shading on the chessboard or with subtle lighting differences between one side of the chessboard and the other. Finally, some pieces were deliberately placed outside the center of the squares, so as to simulate real game conditions.

2.2 Board illumination

It is clear that for the correct implementation of any computerized vision system it is necessary to obtain a uniform illumination on the work plane [11]. For this purpose have been developed lenses with multiple LEDs and arbitrary shapes [12], diverse lens arrange-

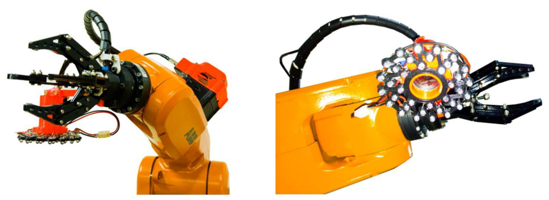


Figure 5: Left, robot gripper and camera. Right, LED support mounted around the camera.

ments [13], varying the optical wavelengths [14] and with specific reflectors for color uniformity [15]. In other works the location of each LED in the illuminator is varied to reach a uniform illumination [16]. The present work follows this idea since it is simpler to develop a prototype of lighting with a 3D printer and then evaluate its performance.

To achieve a uniform illumination on the chessboard, an arrangement of LEDs was designed and placed around the camera. The spatial location of each LED was fixed, but the orientation (i.e. the location on the chessboard where it points) can be varied. In order to hold the LEDs, a 3D printed plastic support was designed and placed around the camera, next to the gripper (see Fig. 5).

To guarantee the uniformity of illumination over the surface of the chessboard, the orientation of each LED was determined using a genetic algorithm, the illumination model of an LED and the spatial position of the illuminator with respect to the chessboard. The goal is to generate and evaluate multiple orientation configurations and combine them to improve initial generation [17].

Once the genetic algorithm found a good enough solution, a set of vectors containing the optimal orientation of each LED was used to build a 3D model of a support base. This was achieved using the open source 3D programming and modelling language OpenSCAD and other free tools such as Cura and Slic3r for model slicing and subsequent generation of the G-code, which is specific to the Prusa i3 Steel printer with Marlin firmware. In Fig. 6 a flowchart of the complete process for building the illuminator is shown, where the red box indicates the software used in each section and the blue box shows the implementation of the genetic algorithm.

Although an LED cannot be considered a perfect Lambertian source, as it is not a coherent point source of light, it is still possible to approximate its illumination model by means of a Lambertian equation [18] in the form of Eq. 1. This is valid since the photon emission area is very small compared to the distance to the object to be illuminated.

$$E(r,\theta) = E_0(r)cos^m\theta \tag{1}$$

where the expression $E_0(r)$ represents the irradiance of the LED on the axis $\theta = 0$ and at a distance r from

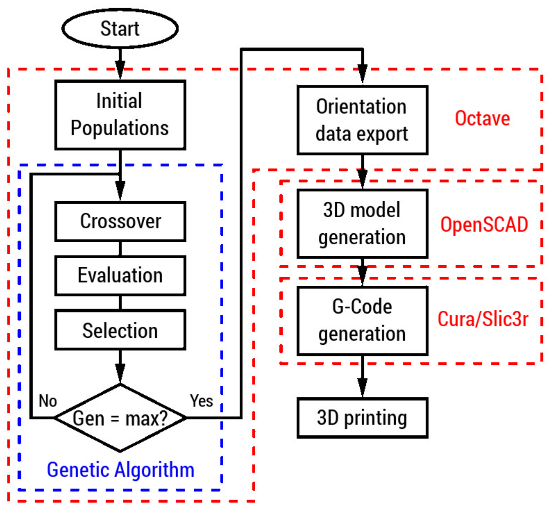


Figure 6: Flowchart of the illuminator design and printing process.

the diode. The exponent m depends on the illumination angle of the diode according to

$$m = -\frac{ln(2)}{ln\left(cos\left(\theta_{\frac{1}{2}}\right)\right)} \tag{2}$$

Both the value of E_0 and the value of m are specified in the datasheets of the LED manufacturer. However, after extensive testing, it was determined that the values provided by these datasheets deviate from real conditions and those variations affect the calculated illumination model. To solve this problem, different measurements of the luminance values of an LED were taken so as to calculate more accurate values for the parameters E_0 and m. In order to get the measurements, a lux meter was used at each of the corners of a grid located on a normal plane and at a known distance of the LED, as shown in Fig. 7 (left). The illuminance measured at each point is shown in Fig. 7 (right). By fitting the known model to the data, the values $E_0 = 63.81$ and $m = 6.767$ were obtained. Once the model for the LEDs illumination was obtained, the total illuminance has to be calculated on the chessboard. For this, the horizontal component of the illumination of each LED on each point of the chessboard is calculated and added to the final estimate.

$$E_H = \frac{Icos^3\alpha}{h^2} \tag{3}$$

This algorithm was implemented as a function within the Octave environment in order to evaluate the illumination levels for the arrangement of LEDs. The data obtained from the simulation includes the minimum, maximum and average levels of illuminance, and the mean square error between the illuminance of each point and the mean value.

Since the goal is to achieve uniform illumination

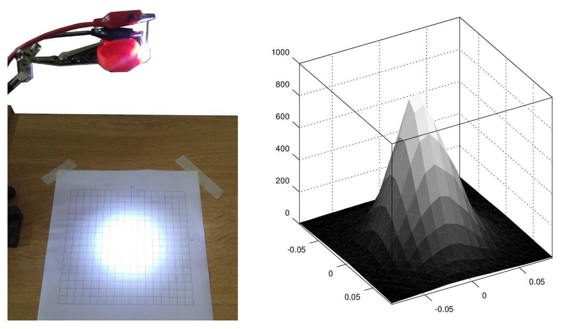


Figure 7: Left, setup used to estimate the LED parameters. Right, values measured in each point of the grid.

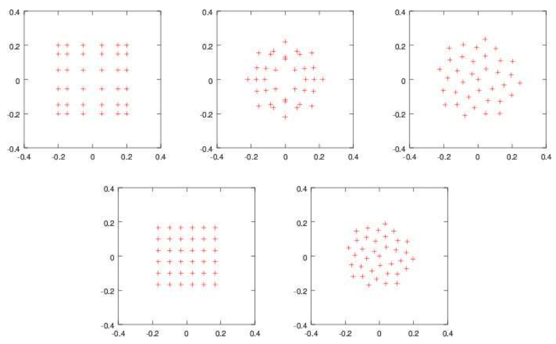


Figure 8: Some of the initial populations for the genetic algorithm.

on the chessboard surface, a good illuminator configuration should provide an average illuminance value as high as possible, and a mean square error as low as possible. As a design constraint, it was decided to use 36 LEDs in the illuminator, located in three concentric circles.

2.2.1 Genetic algorithm and design of the illuminator support

A genetic algorithm is a computational optimization model that uses combination and selection operators to generate new search points on a space of solutions [17]. These algorithms, originally inspired by evolution and natural selection, look for minimal heuristic solutions in terms of an evaluation function. These functions are applied on each *chromosome*, which represents in the case of the present work, a certain configuration for the orientation of each LED in the illuminator.

In order to implement a genetic algorithm, three requirements have to be met: an initial population, an evaluation function and a crossover function between the elements in the population. From the 15 initial populations used in this work, 5 were generated manually, and the remaining 10 were generated in a random manner. The initial manually generated populations are shown in Fig. 8. Noteworthy, the initial populations represent the coordinates of the points on the chessboard to which each of the LEDs is directed to.

This genetic algorithm operates in a genotype-phenotype mode. The genotype is represented by a data structure with 2 fields. The first field indicates the x-coordinate of the point on the board surface where the LED's light is directed, and the second field indicates the y-coordinate of the same point. The phenotype consists in the illumination level calculated on the board surface.

As stated in the previous section, the variations of the illumination levels on the chessboard come from the variation of the orientation of each LED in the illuminator. This means that a set of orientations (i.e. an element of the population) can be evaluated through a function that calculates the maximum and minimum values of illumination on the surface of the chessboard. The crossover or descendant function for two given configurations is calculated as the average between the coordinates of the equal index points in both configurations, while a random value (mutation) is added to increase diversity in the new generation and to avoid the case that all solutions are linear combinations of the initial populations.

The algorithm runs according to the steps outlined in blue in Fig. 6, for a certain number of generations. In the design of the illuminator the limit was 10000 generations. However, it is possible to establish a stop criteria based on the uniformity of the light distribution obtained, or limit the maximum time of execution of the algorithm. The best distribution achieved by the algorithm in this implementation is shown in Fig. 9 (left). The ratio of the minimum to the maximum illumination in this configuration is 1.0472, i.e. less than 5%. The Fig. 9 (right) shows, in dotted line, the location of the illuminator LEDs and the points on the chessboard to which each one is directed to.

Once a good enough distribution is obtained, the coordinates of the location of the LEDs in the support and the orientation vector for each LED are exported into a text file, which is in turn imported into an OpenSCAD script that is responsible for generating the 3D model of the support of the diodes, generating three-dimensional holes in a user-specified shape where each LED will be mounted with the optimal orientation. The shape can be indicated by the user, as shown in Fig. 10, where different support shapes are

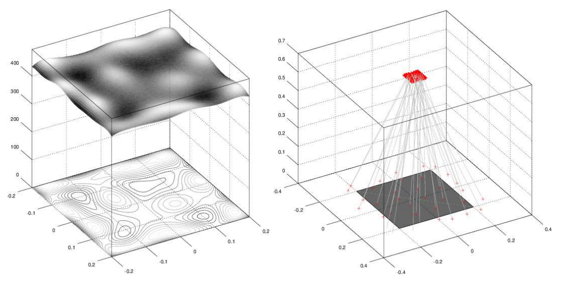


Figure 9: Left, illumination level on the chessboard. Right, orientation of each diode.

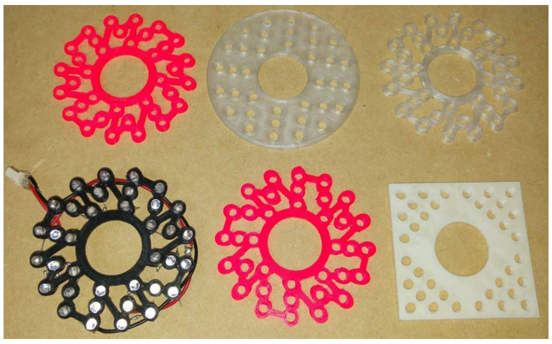


Figure 10: Some of the LED supports printed and tested.

shown. The supports shown were generated by different runs of the genetic algorithm, and once printed were tested and compared according to the uniformity of the light distribution on the chessboard under real world conditions.

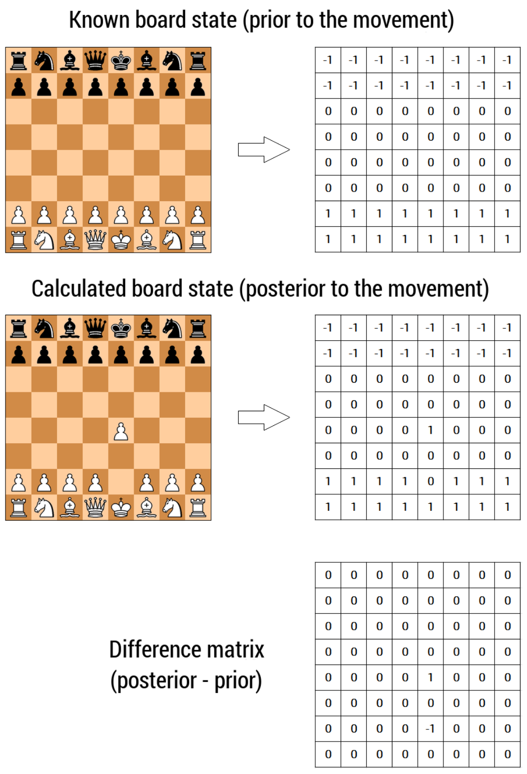
2.3 Chess movements

2.3.1 Detection of human player movements

The GNU Chess open source package is used as the back-end engine for the chess game and, in consequence, a software interface had to be developed for the interaction between the CCU and the chess engine. This interface allows to determine the movements made by the human player and also allows the robot to execute the next movement returned by the analysis engine. To detect the last movement executed by the human player, two matrices containing the state of the chessboard before and after the human player movement are compared. In Fig. 11 an example of detection is provided. The first matrix contains the known chessboard state before the human movement. The second matrix contains the calculated chessboard state after the human movement, as a result of executing the classification algorithm on the image obtained with the camera.

An advantage of working with matrix arrays is that it is possible to perform the subtraction operation between pair elements to identify any change in the chessboard between movements, thus obtaining a matrix of changes in terms of a difference matrix.

Based on the difference matrix, the system can now detect all the legal movements in a game of chess: basic movement, capture, short and long castling, pawn promotion and *en-passant* capture –this is a particular case of capture for the algorithm, since the destination square of the capturing pawn is not that of the captured piece–. It is important to notice that each of these movements has a particular signature in the difference matrix and thus allows obtaining the human movement in chess notation with little additional work. In the difference matrix, non-variant squares are represented as zeros.


Known board state (prior to the movement)
Calculated board state (posterior to the movement)
Difference matrix
(posterior - prior)

Known board state (prior to the movement)

-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

Calculated board state (posterior to the movement)

-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	0	1	1	1
1	1	1	1	1	1	1	1

Difference matrix (posterior - prior)

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	-1	0	0	0
0	0	0	0	0	0	0	0

Figure 11: Difference matrix for a basic movement.

The behavior of the difference matrix for each type of movement is analyzed given the case that the human opponent plays with white pieces. If this is not the case, the only difference is that the signs of the matrix elements are inverted. The matrix signature for all types of movements are given by:

1. Basic movement: The origin square of the movement is represented by the value -1, while the destination square has the value 1.

2. Capture: The origin of the movement is represented, as in the previous case, by the value -1. However, in this case, the destination square has the value 2.

3. Castling: Castling detection is relatively simple because there are only two possibilities per color of pieces: short castling and long castling. The first or last rows of the chessboard is analyzed (for the case that the human plays white or black pieces, respectively) and compared with the known values [-1 1 1 -1] and [-1 0 1 1 -1] for the case of short and long castling respectively.

4. *En-passant* capture: The origin square of the pawn that captures is represented by a value of -1. However, because the captured pawn is not in the destination square, there will be two squares with value 1 in the difference matrix. This is easily solved because it is known that the pawn capturing has to move diagonally.

5. Pawn promotion: This happens when a white pawn arrives at row 8, or when a black pawn arrives at row 1. By default it is considered that the promotion is to a Queen, however, this behavior can be changed to allow the coronation to any other piece.

2.3.2 Robot movement execution

In order for the CCU to communicate with the robot, a communication interface was developed between the robot controller and the embedded computer. Such interface runs under Windows and was programmed in C# language using the VisualStudio environment, as the robot manufacturer provides a tool named PCSDK to communicate with the robot controller based on C#. This tool allows system integrators, third parties or end users to develop their own custom user interfaces to the IRC5 controller. Therefore, PC applications can be run independently, communicating with the robot controller through a wired or wireless TCP/IP network.

When started, the interface software searches for any IRC5 controller on the local network. Once the search is finished, the user must select which of the detected controllers will be used to start the communication. Here, the system waits for the user to configure the controller in Automatic mode and to power the actuators. Once this configuration is finished, the controller is connected remotely to access the program and control it from the CCU. In the next step, initialization and calibration parameters are sent and a confirmation from the human player is expected, by pressing the end-of-turn button. Once the button is activated, the interface sends a signal to the CCU so that it begins to process the movement. Once processed, the interface waits for a data packet (in XML format) to send a motion execution command to the robotic arm. The data contained in the package is then converted into a group of symbols that the controller can process. These symbols include the type of movement to be executed, coordinates of the pieces in the chessboard and heights of the pieces involved.

3 Results

Robustness and runtime comparisons for the three classification algorithms implemented are shown in Fig. 12 and in Table 1. The multisampling and voting classification algorithm is the most robust but it is also the slowest. This is explained by the fact that it has to perform a large number of operations on each of the 7 samples, both in the computing of color averages and in the actual voting process. In addition, it can be seen that the time curve is quite irregular. Such variations may be caused by external factors, e.g. given the considerable amount of time needed to perform the image processing and voting, or the operating system having

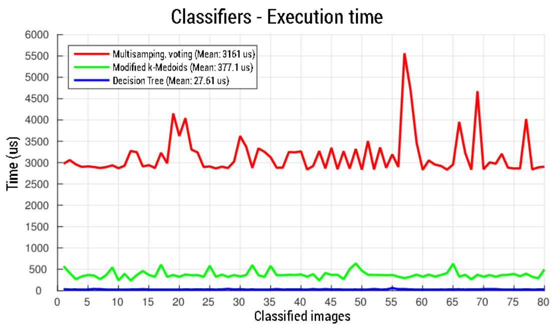


Figure 12: Algorithm execution time comparison.

Table 1: Classification matrices for the implemented algorithms.

Multisampling and voting

		Real color		
		W	*B*	*E*
Class	*W*	1084	0	0
	B	0	982	0
	E	0	0	3054

Modified k-Medoids

		Real color		
		W	*B*	*E*
Class	*W*	1084	0	0
	B	0	964	0
	E	0	18	3054

Decision Tree

		Real color		
		W	*B*	*E*
Class	*W*	1084	1	0
	B	0	974	0
	E	0	7	3054

to attend other tasks simultaneously. The second reason may be related to a greater number of accesses to memory, explained naturally by the greater amount of data used by this algorithm.

Although the k-Medoids classifier achieved a significantly lower execution time, the results are far from expected. Since a classification error may cause the chess match to be interrupted it might considerably affect the concentration of the human player. A number of small peaks can be observed in the time graph, and these correspond to those chessboard images that have been a little more challenging for the classification algorithm to process and needed more iterations in the main cycle to converge.

Finally, the decision tree classifier resulted to be the fastest. This is explained by the fact that the classification process consists, in the worst case, of only 3 logical comparisons. Moreover, although results are not optimal, its performance was very good, with even fewer errors than the k-Medoids algorithm.

All classifiers used the same 80 images, and exactly the same image processing work was performed before the classification step. At this point, future work may include improvement of the classification techniques or the use of alternative methods to process the images, since in some cases, slight changes in illumination led to detection errors. In addition, the calibration operation used for voting classifiers and k-Medoids may take into account not only the color of the square and its contents but also the amount of light present in the chessboard at the moment of calibration. This problem might be even worse when playing on outdoor environments with natural light or even with time-varying illumination (near a window for example). Moreover, alternative metrics of distance should be implemented that can take into account variance in the samples, such as the Mahalanobis distance, to improve precision in the classification process.

To verify the proper operation of the computer vision algorithm for pieces detection in a real situation, 10 complete chess games were played against a human opponent using the multisampling and voting classifier, and errors were quantified. Results are shown in Table 2. To make a comparison, detection errors were also evaluated in 10 games using just the artificial illumination present in the laboratory (fluorescent tubes, with highly non-uniform illumination on the playing surface). Results can be seen in Table 3.

A false positive (FP) means the system detects a piece (of any color) in an empty square, a false negative (FN) means the system detects an empty square instead of a piece, and a classification error (CE) is obtained if the classification algorithm incorrectly determines the color of the piece. The total number of movements performed by the manipulator in each game was saved. For cases where movement errors were detected, the detection matrix was manually corrected so as to be able to continue the game. A significant improvement in performance can be seen in Table 2 and Table 3 when the level of illumination used on the chessboard was uniform. The error rate was reduced from 11.76% using only the artificial illumination present in the laboratory, to 2.86% with uniform LED illumination.

During experiments, some inappropriate behaviors were observed. First, when the illumination was not uniform, the shadows caused by the pieces on the empty squares or on pieces of lesser height affected the correct detection of the color or even the detection of a false presence of a piece in a square. False positives are explained mostly by empty squares of dark color that, when partially shaded, are mistaken for black colored pieces. False negatives largely consist of shaded white pieces, which are confused with clear empty squares, while color errors are entirely shaded white pieces, in areas with poor illumination.

When uniform illumination was used, false negative errors were completely eliminated, mainly because there is a greater separation between the color thresholds of an empty square and the occupied ones. However, due to the location of the LED support with respect to the camera, reflections on black pieces (for example black rook, black knight) may lead to a color detection error. False positives, although smaller when using LEDs, still remained remarkably high. A likely cause may be that even with the LED illuminator in operation, the laboratory lights were on. A solution to this problem would be to increase the number of LEDs, in order to achieve a higher level of illumination on the chessboard thus minimizing external incidences and, therefore, shadows caused by fluorescent tubes or any other external form of illumination.

Table 2: Results using LED illuminator

Game	**Movements**	**FP**	**FN**	**CE**
1	24	1	0	0
2	31	0	0	0
3	19	0	0	0
4	22	0	0	1
5	21	2	0	0
6	37	0	0	0
7	23	1	0	1
8	19	0	0	0
9	21	0	0	0
10	28	1	0	0
Total	*245*	*5*	*0*	*2*

Table 3: Results not using LED illuminator

Game	**Movements**	**FP**	**FN**	**CE**
1	19	0	1	3
2	24	1	0	1
3	22	2	0	2
4	15	0	0	1
5	34	3	1	1
6	29	1	0	3
7	19	1	0	1
8	24	0	0	2
9	24	2	1	0
10	28	0	0	1
Total	*238*	*10*	*3*	*15*

4 Conclusions

This work presents a system based on an industrial robot that is capable of playing chess with a human opponent using 3D printed pieces and a non-instrumented board. The interaction with a human opponent is achieved by means of a simple button and a game timer. The main advantage of our mechatronic implementation corresponds to its general structure, without the need of an instrumented chessboard or specially designed pieces, so that it could be easily adapted to other tasks.

Since any piece misclassification may interrupt the game affecting the human player concentration different classification strategies were implemented and tested. The multisampling and voting strategy resulted to be the more robust to environmental variability but also the slower in terms of compute complexity. This may be caused by the large amount of processed data in memory that involves this algorithm. The obtained accuracy for the decision tree classifier was acceptable, with even fewer detection errors than using k-Medoids. Notwithstanding, better performance might be attained by improving the image processing to mitigate the effects of changes in the illumination or color.

The implementation of new algorithms is also planned in order to obtain a greater separation between the proposed classification classes. Currently an adaptive algorithm is being developed which, based on the number of pieces of each color, dynamically re-calculates the reference colors of the chessboard so as to enhance the detection performance. Regarding the communication between the CCU and the robot controller, a new software interpreter running directly on the controller is being tested. The last will allow to remove the computer running the C# interface, and making the system work faster due to less interactions. Lastly, as it would be more appropriate using a perceptually uniform color space rather than the Euclidean distance in the HSL color space, future work may use the color space L*a*b, since in addition to having a luminance channel L*, it has the advantage that distances between colors on the coordinates a* and b* are proportional to the perceptual differences between colors.

The mechatronic system implemented is feasible to be migrated to other contexts. A video of the system in use can be found at https://www.youtube.com/watch?v=cJwDVGww09Q. Algorithms and software models were uploaded to the GitHub open source repository.

Competing interests

The authors have declared that no competing interests exist.

References

[1] N. Aliane and S. Bemposta, "Checkers playing robot: A didactic project," *IEEE Latin America Transactions*, vol. 9, no. 5, pp. 821–826, 2011.

[2] C. Breazeal and B. Scassellati, "Infant-like social interactions between a robot and a human caregiver," *Adaptive Behavior*, vol. 8, no. 1, pp. 49–74, 2000.

[3] K. Dautenhahn and A. Billard, "Games children with autism can play with robota, a humanoid robotic doll," in *Universal Access and Assistive Technology: proceedings of the Cambridge Workshop on UA and AT'02, Springer Verlag*, Springer, 2002.

[4] D. Urting and Y. Berbers, "Marineblue: A low-cost chess robot.," in *Robotics and Applications*, pp. 76–81, 2003.

[5] S. Sarker, "Wizard chess: An autonomous chess playing robot," in *Electrical and Computer Engineering (WIECON-ECE), 2015 IEEE International WIE Conference on*, pp. 475–478, IEEE, 2015.

[6] C. Matuszek, B. Mayton, R. Aimi, M. P. Deisenroth, L. Bo, R. Chu, M. Kung, L. LeGrand, J. R. Smith, and D. Fox, "Gambit: An autonomous chess-playing robotic system," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 4291–4297, IEEE, 2011.

[7] A. Chen, I. Kevin, and K. Wang, "Computer vision based chess playing capabilities for the baxter humanoid robot," in *Control, Automation and Robotics (ICCAR), 2016 2nd International Conference on*, pp. 11–14, IEEE, 2016.

[8] A. M. Reza, "Realization of the contrast limited adaptive histogram equalization (clahe) for real-time image enhancement," *The Journal of VLSI Signal Processing*, vol. 38, no. 1, pp. 35–44, 2004.

[9] R. Chitrakar and H. Chuanhe, "Anomaly detection using support vector machine classification with k-medoids clustering," in *Internet (AH-ICI), 2012 Third Asian Himalayas International Conference on*, pp. 1–5, IEEE, 2012.

[10] C. Sammut and G. I. Webb, *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[11] F. Wu and Q. Huang, "A precise model of led lighting and its application in uniform illumination system," *Optoelectronics Letters*, vol. 7, no. 5, pp. 334–336, 2011.

[12] Y. Ding, X. Liu, Z. Zheng, and P. Gu, "Freeform led lens for uniform illumination," *Optics Express*, vol. 16, no. 17, pp. 12958–12966, 2008.

[13] A. J. Whang, Y. Chen, and Y. Teng, "Designing uniform illumination systems by surface-tailored lens and configurations of led arrays," *Journal of display technology*, vol. 5, no. 3, pp. 94–103, 2009.

[14] Y. Gu and N. Narendran, “Design and evaluation of an led-based light fixture,” in *Proc. SPIE*, vol. 5187, pp. 318–329, 2004.

[15] Z. Zhu, X. Qu, G. Jia, and J. Ouyang, “Uniform illumination design by configuration of led array and diffuse reflection surface for color vision application,” *Journal of Display Technology*, vol. 7, no. 2, pp. 84–89, 2011.

[16] I. Moreno, “Configurations of led arrays for uniform illumination,” in *Proc. SPIE*, vol. 5622, pp. 713–718, 2004.

[17] D. Whitley, “A genetic algorithm tutorial,” *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[18] F. L. Lewis and F. Lewis, *Optimal estimation: with an introduction to stochastic control theory*. Wiley New York et al., 1986.