Computer Vision Based Chess Playing Capabilities for the Baxter Humanoid Robot

Andrew Tzer-Yeu Chen and Kevin I-Kai Wang
Embedded Systems Research Group, Department of Electrical and Computer Engineering
The University of Auckland
Auckland, New Zealand
e-mail: {andrew.chen, kevin.wang}@auckland.ac.nz

***Abstract*—This paper presents a project that allows the Baxter humanoid robot to play chess against human players autonomously. The complete solution uses three main subsystems: computer vision based on a single camera embedded in Baxter's arm to perceive the game state, an open-source chess engine to compute the next move, and a mechatronics subsystem with a 7-DOF arm to manipulate the pieces. Baxter can play chess successfully in unconstrained environments by dynamically responding to changes in the environment. This implementation demonstrates Baxter's capabilities of vision-based adaptive control and small-scale manipulation, which can be applicable to numerous applications, while also contributing to the computer vision chess analysis literature.**

Keywords-computer vision; uncontrolled environments; mechatronics; human-robot interaction; chess robot

I. INTRODUCTION

The Baxter robot developed by Rethink Robotics is designed to be a safe, flexible alternative to existing fixed manufacturing automation solutions, and can work in collaboration with humans. However, applications for Baxter do not need to be restricted to the manufacturing industry. The humanoid nature of the Baxter robot makes it a suitable platform for human-robot interaction applications, such as mental disease rehabilitation or autism treatment.

Our project looked at extending Baxter's capabilities to different domains, one of which was playing board games. Chess is often used in board game based research due to its reasonably high state-space complexity and complex ruleset that make it a challenging yet achievable problem for researchers and developers.

Our main goal was to develop a system that allows Baxter, shown in Fig. 1, to play chess against a human user, with minimal configuration. A secondary goal was to allow the robot to play in unstructured environments, such as with uncontrolled lighting conditions, dynamic board position, and no additional markers. We used only one arm and one camera, so that Baxter could potentially play two games of chess simultaneously. Robustness and consistency were also key considerations. A driving motivation was to make Baxter play chess in the same way a human would.

There are three main subsystems: computer vision (CV) for perception, chess engine for computation, and mechatronics for actuation.

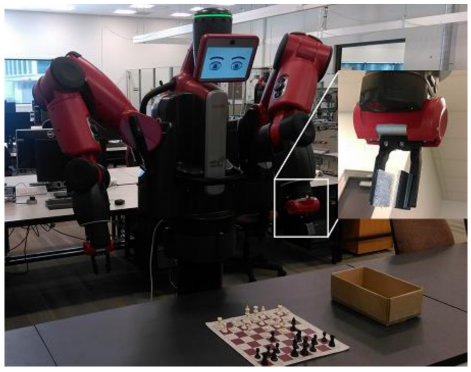

Figure 1. The baxter robot, with the camera and gripper magnified.

II. RELATED WORKS

A number of existing implementations of chess playing robots exist in the literature. However, there are often a number of significant constraints that simplify the problem greatly. Most commonly, a Digital Game Technology (DGT) board is used, which uses sensors embedded in the board and the pieces to report the current location of each piece to a computer, thus removing any need for external perception. Examples of this approach include Chesska from Russia and the KUKA Monstr from Germany [1]. However, DGT boards can be expensive and are not as common as standard non-digital chess sets.

When computer vision is used, common simplifications include using non-standard colours to help the camera differentiate between the board and pieces [2], and mounting the camera directly above the board to reduce perspective distortion [3], [4]. CVChess [5] uses Harris corner detection and heatmaps to support board and piece movement detection at acute angles from a laptop webcam, but requires manual move triggering. Neufeld and Hall [6] propose a probabilistic matching algorithm based on image masks to detect and classify the pieces, with lower accuracy than our method. Tam, Lay, and Levy [7] use a combination of Hough line transforms and domain knowledge to segment squares from populated chessboards, although no piece detection is implemented.

One of the common simplifications used by hobbyists for the mechatronics subsystem is to use an XY linear slide system. However, this requires a fixed board position, and can sometimes occlude the board from a human player. They may alternatively use specially designed boards and pieces to

978-1-4673-9859-6/16/$31.00 ©2016 IEEE

support manipulation. Other implementations require that the board be in a fixed position, so that the physical location of each square (and piece) is known to the robot beforehand [3].

Gambit [8] is the closest implementation to ours in terms of functionality, with a stated goal of achieving as much generality as possible. It is an autonomous system that uses an infrared depth camera in conjunction with a secondary camera to perceive the board, and a custom 6-DoF robot arm for actuation. However, there are still significant differences in the approaches, such as Gambit's use of SVMs to learn the piece types, and using two cameras including depth information instead of just one camera. Our implementation takes a simpler approach to the perception problem.

III. Computer Vision Subsystem

The main focus of the project was developing the ability for Baxter to "see" the board. Baxter has a camera built into the arm, underneath the gripper as shown in Fig. 1. This provides a 1280×800 widescreen image, although some of the image is obscured by the gripper. There are a number of sub-problems to solve as shown in Fig. 2: finding the chessboard from the wide image, locating the individual squares on that chessboard, and finding the pieces in those squares. We utilised OpenCV [9], an open source image processing library, to help accelerate the development. The desired output of the CV subsystem is the current position of all the pieces on the board, and the coordinates of a move if there is one. The arm (and therefore the camera) moves to a fixed position each time the CV subsystem needs to read the state of the board. In our testing, the camera is held at a 0.4 radian angle from the perpendicular to the table, so that the camera is not directly above the board.

Firstly, pre-processing (morphological closing and Gaussian blurring) reduces the noise for later processing. Then we attempt to find the chessboard. Otsu thresholding allows the algorithm to dynamically respond to lighting changes by recalculating the threshold. Canny edge detection is used to identify the boundaries of objects, and dilation fills in small discontinuities in any edges to form continuous contours. Square contours are identified by iterating through all contours and finding those with four sides, at approximately right angles to each other, and with approximately equal length and width. The largest square is designated the chessboard, and we slice the original image so that only the chessboard is present in order to reduce processing in later stages. We also calculate the orientation of the chessboard by determining the angles of the four chessboard edges in comparison to the x and y axes of the image (and therefore Baxter). This allows us to account for situations where the board may not be exactly parallel to Baxter, such as seen in Fig. 3. Our measurements indicate that the orientation can deviate by up to 20 degrees

Contrast limited adaptive histogram equalization (CLAHE) is used to boost the differences in intensity; otherwise the white pieces on white squares or black pieces on black squares can be almost indistinguishable in a greyscale image. The adaptive equalisation also helps avoid undesirable information loss encountered after global histogram based contrasting.

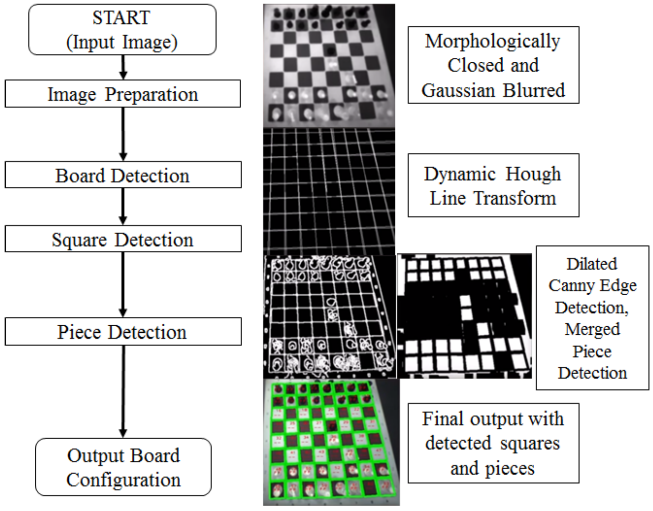

Figure 2. Top level CV algorithm flowchart with intermediate screenshots at each stage of the algorithm.

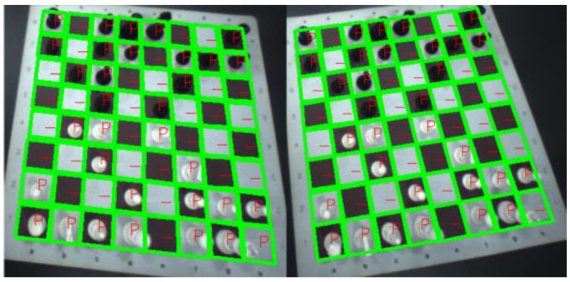

Figure 3. The computer vision algorithm can detect the board, squares, and pieces when the board is in different orientations.

As the camera is not directly above the board, some of the square boundaries are occluded by the pieces, so edge detection alone is not sufficient. A dynamic Hough line transform is conducted to find the lines in the edge image. We call this transform "dynamic" because in our approach we segment the image and apply transforms with different vote thresholds in order to create the most usable set of lines. This accounts for the fact that the body of the robot can create a shadow over the chessboard, making squares further away from Baxter lighter than squares closer to Baxter. By iteratively selecting different vote thresholds, we can adjust the number of lines that appear in the transform output. Additionally, the angles calculated earlier for determining the orientation of the chessboard are used to constrain the line selection to ensure that lines are roughly parallel to the x and y axes of the chess board.

The same approach as earlier is used to detect board square contours. We then use the domain specific knowledge that there should be 64 squares on a chessboard. If there are less than 64, then we reject the image. If there are more than 64, then the smallest squares are removed as they usually originate from noise at the edges of the board. Masks are created for each square to build an 8x8 occupancy grid.

For piece detection, we process the chessboard in two different ways and merge the outputs. Firstly, we use the output of the Canny edge detection in the square detection

step and morphologically close the edges. This can provide a good indication of where the pieces are, but is susceptible to misinterpreting small shadows on empty squares as pieces. Secondly, we use morphological edge detection on the earlier contrasted image, and again morphologically close the edges. This is resistant to the small shadows created by the pieces, but can misinterpret broader shadows across multiple empty squares as pieces. By merging these two images, we can more accurately determine which squares have pieces.

For each square in the occupancy grid, we then calculate the average intensity of the square using the mask and compare it to a hardcoded threshold to determine if there is a piece present or not. We also use the original greyscale image to determine if the piece is white or black. In contrast to approaches from other implementations (see Section II), we do not attempt to determine the type of piece. Firstly, the pieces appear too small in the image for sufficient detail to be extracted, and secondly we found that the chess engine does not need to know the type of piece as it can keep track of the pieces and their types as they move around the board, assuming that they start in their initial positions. A differential image approach is taken, where each frame is compared to the previous frame to see if a piece has moved. From this analysis, an origin square and destination square (both from 0 to 63) can be determined.

IV. Chess Engine Subsystem

Rather than developing a chess engine from scratch, we use the open source engine Stockfish[1], one of the strongest chess engines in the world. At the beginning of the game, we ask the user to select a difficulty level that corresponds to the depth of the engine, which is the number of moves that the engine searches ahead in order to find the best move. We also use Pystockfish[2], an open-source wrapper, to integrate Stockfish into our program. We additionally use Chessnut[3], an open-source chess modelling application written in Python, which allows us to verify individual moves to check if they are legal. If a user plays an illegal move, then the program informs them that they must try again. We also keep track of the moves made in order to generate a report at the end of the game. Once the origin and destination square numbers are provided by the CV subsystem, these are converted to their algebraic notation equivalents (from a1 to h8), verified by Chessnut, and passed to Stockfish. The chess engine responds with its next move, which is converted back to square numbers (from 0 to 63) and passed to the mechatronics subsystem.

V. Mechatronics Subsystem

The Baxter arm has seven degrees of freedom, which can make control of the end effector complicated. Thankfully, we are able to use IKFast[4], an inverse kinematics solver that allows us to simply input the desired end effector co-ordinates and orientation. All co-ordinates are defined relative to the center of Baxter's torso. Unlike a number of other implementations (see Section II), we do not need the board to be in a fixed position. During gameplay with a standard chess set, the board may move slightly when touched by the human. Therefore, the robot dynamically recalculates the co-ordinates of the squares before each move based on the camera image.

As Baxter is designed to be a compliant robot that is safe to use around humans without a safety cage, there are two key problems. Firstly, the movement of the arm is slow in comparison to other robot arms due to torque and safety limitations. Secondly, the accuracy of the arm is not always perfect, as it allows deviations from the equilibrium position. To address this, the mechatronics subsystem calculates intermediary waypoints between the current position and the desired endpoint, which causes the arm to make smaller movements (and thus reducing overshooting) at the cost of slower speed. This is important to ensure that Baxter is robust and errorless when the board is not in a fixed position, although he position of the board is still ultimately constrained by the overall reach space of Baxter's arm.

Given the origin and destination squares, the mechatronics system uses the dynamically calculated co-ordinates to pick and place the piece. Baxter's arm has an electric two-finger gripper that closes and opens to pick up and release the chess pieces as shown in Fig. 1. If a piece is already at the destination, then Baxter removes this piece first and puts it in a box beside the board. The program also checks if the move from the chess engine is castling; if so, Baxter moves the second piece as well.

VI. Overall System Control

A finite state machine approach, as shown in Fig. 4, is used to manage the different actions that Baxter should take. During initialisation, Baxter and its peripherals including the cameras and the screen (i.e. Baxter's face) are set up. During the calibration state, the CV and mechatronics subsystems are calibrated for the position of the board, and a chess engine difficulty level is set. The CV subsystem detects whether the white or black pieces are closest to Baxter, and therefore which side Baxter is playing.

If it is Baxter's move, the program asks the chess engine for the move, the mechatronics subsystem executes the move, and the CV subsystem verifies that the move made matches the expected move (otherwise it asks for human intervention). It also checks if the win condition or lose condition (checkmate for each side) has been reached. For the human move state, the arm moves to an observation position, and Baxter indicates to the human that it is their turn by nodding the head and changing the face image. The CV subsystem then continuously samples the camera to see if a move has been made; no manual move triggering is required as the CV subsystem rejects images where it cannot find 64 squares because part of the board is obscured (e.g. by a hand). Once a move is detected, it is verified by the chess engine for legality.

For each of the game over states, drawing some inspiration from [10], Baxter responds in a different way;

[1] https://stockfishchess.org/
[2] https://github.com/iamjarret/pystockfish
[3] https://github.com/cgearhart/Chessnut
[4] http://openrave.org/docs/0.8.0/openravepy/ikfast/

when he wins, the arms move to a neutral position and the screen flashes different colours to indicate celebration; when he loses, Baxter crosses his arms and the screen shows an upset face to indicate sadness. In our testing, Baxter has never reached the lose state.

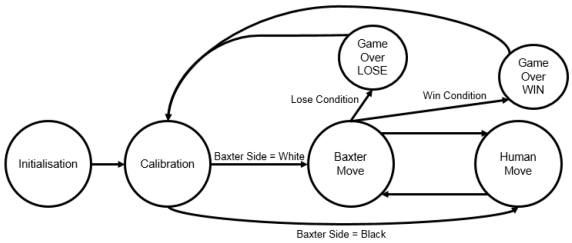

Figure 4. Finite state machine of the overall system control.

VII. Results

Stockfish is a very strong chess engine, and the average person is not able to beat it since we use a minimum depth of 6 (most humans only plan two or three moves ahead). As we do not have any chess Grandmasters in our department, no human has beaten Baxter in almost 50 games.

For most humans, the perception time for detecting and analysing a move is very short, the computation time of deciding the next move is comparatively long, and the actuation time of moving a piece is very short. For Baxter, this is exactly the opposite; the CV subsystem processes a few frames per second (depending on how fast the dynamic Hough transform finds an acceptable solution), the chess engine subsystem usually returns a move in less than a second (depending on the difficulty level/search depth), but the mechatronics subsystem can take 45-90 seconds to actuate depending on if Baxter needs to move one piece or two (in the case of piece capture or castling). At the beginning of the game, a human player familiar with common chess openings can play significantly faster than Baxter, leading to the impression that Baxter plays very slowly. However, as the game progresses, Baxter can outpace the human if the player takes a few minutes to consider their move. Using the proposed approach, Baxter could be easily replaced by an alternative camera/robot arm arrangement if speed of play is an important factor.

In terms of errors, the CV subsystem on its own very rarely misinterprets shadows or changes in lighting as piece moves. However, these are usually illegal moves; with move validation using Chessnut, the move perception was 100% accurate over 300 moves (across 10 games). For the mechatronics subsystem, occasionally the arm accidentally knocks over other pieces as it actuates; human intervention is currently required, but some fault tolerance could be developed as described in section VIII.

VIII. Conclusions and Future Work

Chess playing capabilities are developed for the Baxter robot, utilising computer vision to allow the robot to play in largely unconstrained conditions. The computer vision, chess engine, and mechatronics subsystems co-operate to allow the robot to progress through a perceive-compute-actuate loop, emulating human behaviour. In comparison to other robot chess approaches, Baxter can play without the use of a DGT board, without non-standard board colours, without a fixed camera directly above the board, with variable lighting conditions, with small variations in board orientation, and with changes in board position.

Approaches that allow Baxter's arm to move faster without sacrificing accuracy or precision would improve the experience of playing against the currently slow robot. Some fault tolerance could be developed to help Baxter automatically correct situations where it accidentally knocks pieces over. Better modelling of the chess pieces, such as the use of deep learning, would be required in order to tell the difference between a knocked over piece and two or three pieces in adjacent squares. Baxter could feasibly pick up the piece and reorient it correctly, but this requires better sensing of the piece orientation while it is in Baxter's gripper.

Further development of the facial expressions shown by Baxter and other arm movements could improve human-robot interaction and make the experience more engaging and enjoyable for human players. This platform could also be used for other small scale manipulation tasks involving humans such as dispensing medication. Using alternative end effectors such as vacuum grippers could allow Baxter to play other board games or card games. In conclusion, our implementation of a chess playing robot is robust, and is a useful case study for further human-robot interaction research with Baxter, as well as improving upon existing computer vision approaches towards analysing chess games.

References

[1] E. Mukhamedov, "Chesska defends world champion title in robot chess," Chess News, May 23, 2012.

[2] C. Danner and M. Kafafy, Visual Chess Recognition, Stanford University, Stanford, 2015.

[3] D. Urtin and Y. Berbers, "Marine blue: A low-cost chess robot," IASTED International Conference on Robotics and Automation, Salzburg, 2003.

[4] E. Sokic and M. Ahic-Djokic, "Simple computer vision system for chess playing robot manipulator as a project-based learning example," International Symposium on Signal Processing and Information Technology, Sarajevo, 2008.

[5] J. Hack and P. Ramakrishnan, CVChess: Computer Vision Chess Analytics, Stanford University, Stanford, 2014.

[6] J. Neufeld and T. Hall, "Probabilistic location of a populated chessboard using computer vision," Midwest Symposium on Circuits and Systems, Seattle, 2010.

[7] K. Y. Tam, J. A. Lay, and D. Levy, "Automatic grid segmentation of populated chessboard taken at a lower angle view," Digital Image Computing: Techniques and Applications, Canberra, 2008.

[8] 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," International Conference on Robotics and Automation, Shanghai, 2011.

[9] G. Bradski, "The open CV library," Dr. Dobb's Journal of Software Tools, 2000.

[10] A. Pereira, C. Martinho, I. Leite, and A. Paiva, "iCat, the chess player: the influence of embodiment in the enjoyment of the game," International Conference on Autonomous Agents and Multiagent Systems, Estoril, 2008.