INTEGRACIÓN DE MÉTODOS DE INTELIGENCIA ARTIFICIAL Y PROCESAMIENTO DE IMÁGENES PARA JUGAR AJEDREZ EN EL ROBOT BAXTER

Julio Esteban Asiain Neno
Departamento de Ingeniería Informática
Universidad de Concepción
Julio Erasmo Godoy del Campo

30 de Agosto del 2018

RESUMEN

La robótica, inteligencia artificial y procesamiento de imágenes son áreas muy ligadas entre sí. Los robots utilizan procesamiento de imágenes para interactuar visualmente con su entorno acompañado de inteligencia artificial para tomar decisiones. Baxter es un robot que cuenta con estas capacidades y está siendo utilizado en diversas áreas. En esta memoria de título se diseñó un método que permite al robot Baxter jugar ajedrez reconociendo el tablero y las piezas con procesamiento de imágenes, tomando decisiones mediante inteligencia artificial con búsqueda alfa beta y moviendo las piezas utilizando las herramientas que Baxter posee y la programación en robótica. Los resultados comprobaron que Baxter puede jugar ajedrez contra él mismo o contra personas, aunque aún depende de una consola para ingresar las jugadas de las personas.

1. INTRODUCCIÓN

Inteligencia Artificial (IA/AI) es el comportamiento demostrado por máquinas, en contraste a la inteligencia natural demostrada por humanos o animales. En informática, se define inteligencia artificial como el estudio de los agentes inteligentes [1], esto es, entidades que perciben el ambiente y lo toman en consideración para tomar las decisiones que maximicen la probabilidad de lograr sus objetivos. Enfatiza la creación de máquinas que se comporten como humanos o animales.

En el mundo moderno, esta disciplina está siendo utilizada en una amplia variedad de dominios, desde juegos hasta robótica e incluso trabajos rutinarios. Algunos ejemplos importantes son el reconocimiento de rutas óptimas para el transporte [2], crear experiencias más reales en los juegos [3] o encontrar enfermedades a través de patrones específicos que presentan los síntomas [4].

Un campo muy estudiado en el área de la inteligencia artificial son los juegos [5, 6], ya que la amplia variedad de juegos permite diferentes tipos de simulaciones para diferentes investigaciones. Cuentan con una vasta cantidad de datos, facilitando la obtención de éstos de manera rápida [7]. Estos datos varían dependiendo del juego, pero en términos generales, se puede obtener información como: cantidad de turnos jugados, tiempo de juego, el resultado del juego, entre otros; datos más específicos pueden ser la distancia recorrida por el jugador en un juego de carreras o la cantidad de piezas capturadas en una partida de ajedrez. Existen juegos clásicos como el Go [8] que proveen información perfecta del ambiente del juego o juegos muy complejos como StarCraft II [9] donde la información entregada al jugador es incompleta y en tiempo real, creando una gran diferencia en complejidad entre estos tipos de juegos. Esta diferencia se debe a varios factores como el grado de conocimiento del juego, la cantidad de jugadas disponibles, si es en tiempo real o por turnos, las reglas del juego, cantidad de jugadores y otros. La razón más importante por la cual se estudian estos juegos es porque estas simulaciones son extrapolables a otras áreas, por ejemplo, neurociencia [10]. Dentro del mismo StarCraft II mencionado anteriormente o juegos como Grand Theft Auto, mover las unidades o al jugador de un lugar a otro puede ser considerado una simulación de encontrar rutas seguras o en tiempo óptimo, evitar colisiones con estructuras u otras unidades o elementos del juego que podrían aplicarse a la realidad. En lo práctico, esto podría aplicarse, por ejemplo, en investigaciones relacionadas con la navegación autónoma de los autos inteligentes [11].

Por otro lado, la robótica, área interdisciplinaria que investiga y desarrolla máquinas que puedan replicar acciones humanas [12], está avanzando a pasos agigantados. Empresas utilizan robots automatizados para realizar tareas mundanas que requerirían muchos operadores, como empaquetar [13] o trasladar peso [14]. A pesar de los esfuerzos tempranos de los investigadores, por mucho tiempo, la robótica y la inteligencia artificial se investigaron por separado [15, 16], si bien en los inicios de la inteligencia artificial ésta estaba conectada fuertemente con la robótica[17], factores como la falta de recursos, lo costoso que eran[18] y lo complicado que era investigar ambas áreas juntas causaron que

se investigasen por separado, por lo que una de sus metas tempranas, crear robots inteligentes, se veía muy distante [19].

Actualmente, la robótica está muy ligada a la inteligencia artificial. Si bien existen robots automatizados cuyas funciones no varían, se está avanzando gradualmente en robots inteligentes, es decir, robots con inteligencia artificial que aprenden por sí mismos cómo realizar el trabajo de manera óptima usando diferentes métodos de aprendizaje de inteligencia artificial [20].

Estos temas no están siendo investigados de manera extensiva en Chile y menos en Concepción, salvo por pocos talleres que se realizan en fechas específicas [21]. Existiendo diversas casas de estudio no se han visto suficientes proyectos relacionados con inteligencia artificial y la robótica, como sí es el caso en otros países del mundo como Estados Unidos, Japón, Alemania u otros [22].

En la actualidad, la facultad de ingeniería de la Universidad de Concepción cuenta con un robot desarrollado por Rethink Robotics llamado Baxter [23] para realizar investigaciones o tareas de aprendizaje. Aprovechando esta oportunidad, y estudiar la unión de inteligencia artificial con la robótica, se realiza esta memoria de título que consiste en lograr que el robot Baxter juegue ajedrez. Para lograr esto, se deben utilizar técnicas de reconocimiento de imágenes para reconocer tanto el tablero del juego como sus piezas, además de conceptos de inteligencia artificial para permitirle al robot Baxter jugar y así investigar las capacidades del robot.

1.1. OBJETIVO GENERAL

El objetivo de esta memoria es transferir métodos de inteligencia artificial, reconocimiento de imágenes y de manipulación de objetos a un robot humanoide (Baxter) para permitirle jugar ajedrez de manera semi-autónoma, extrapolando estas técnicas desde lo virtual a lo físico (Computador -> Robot).

1.2. OBJETIVOS ESPECÍFICOS

- Analizar técnicas de inteligencia artificial para búsqueda adversaria existentes, en este caso, minimax y alfabeta (Ver sección 2.1). Implementar uno de los algoritmos analizados para que juegue ajedrez a un nivel de dificultad básico.
- Estudiar técnicas de reconocimiento de imágenes como detección de borde, colores y figuras e implementar las técnicas estudiadas necesarias.
- Lograr que el robot Baxter juegue ajedrez mediante el uso de los algoritmos implementados de ambos campos (inteligencia artificial y reconocimiento de imágenes) utilizando sus capacidades físicas (brazos, cámaras y pinzas).

1.3. EFECTOS ESPERADOS

- Utilizar el proyecto como demostración de las capacidades tecnológicas de la Universidad y de la región.
- Apoyar la obtención de financiamiento por parte del país o de la Universidad para la realización de proyectos más ambiciosos y de avance tecnológico.
- Fomentar la creación de talleres, cursos, asignaturas, magíster o doctorados en el campo de la robótica.

2. DISCUSIÓN BIBLIOGRÁFICA

En esta sección se presentan trabajos con objetivos similares que utilizaban el robot Baxter para jugar ajedrez realizados por otros equipos de trabajo. Además, se mencionan brevemente las investigaciones que influyeron a la realización de este proyecto de memoria de título.

2.1. INTELIGENCIA ARTIFICIAL EN AJEDREZ

Búsqueda adversaria es un tipo de búsqueda competitiva multiagente en la que los objetivos de los agentes se oponen entre sí [24]. Esta búsqueda se basa en la construcción de una estructura llamada "árbol de juego", un grafo dirigido donde sus nodos son posiciones, sus vértices son movimientos, cada nivel una jugada(ply) y los nodos hoja representan un estado final del juego [24]. Los niveles pueden ser niveles MAX o niveles MIN, lo que representa a los jugadores. Para el caso del ajedrez, un jugador es MAX y el otro MIN. Este tipo de búsquedas suele ser poco prácticas al momento de analizar juegos con una cantidad de jugadas muy grande, porque termina generando un árbol muy grande, tardando demasiado en recorrerlo completo. Debido a esto, se utilizan funciones de evaluación para limitar la búsqueda. Estas funciones permiten saber cuándo una acción es buena o mala asignando puntajes a cada acción. Un algoritmo de búsqueda adversaria implementado en este proyecto es la búsqueda alfa beta, que utiliza funciones de evaluación para reducir el espacio de búsqueda, podando nodos y subárboles cuando se estima que sus valores no afectarán la decisión final.

Como el ajedrez cumple con los criterios para ser atacado con búsqueda adversaria, se han realizado varias investigaciones y desarrollado distintas implementaciones de inteligencia artificial para abordar el problema, como es el caso de DeepBlue [25] o StockFish [26].

Un trabajo que influyó considerablemente este proyecto fue la implementación de ajedrez realizada por Jean-François el año 2014 [27], ya que fue su implementación con pocas modificaciones la que se implementó en esta memoria de título.

Esta implementación se utilizó en esta memoria de título debido a que cumplía con los requisitos esperados, es decir, puede jugar ajedrez con sus reglas básicas y un adversario de no muy alto nivel, de manera que el robot pueda jugar con personas que no son expertas en el juego. Utiliza poda alfa beta para realizar la búsqueda, ya que el ajedrez es un juego con un espacio de búsqueda muy grande (35^{100} aproximadamente) y la poda alfa beta permite reducirlo (Ver Figura 1) considerablemente de 35^{100} a 35^{50} en el mejor caso gracias a las funciones de evaluación. Además, esta implementación permite cambiar la profundidad de búsqueda del algoritmo, lo que permite disminuir o aumentar (a costa del tiempo), la dificultad.

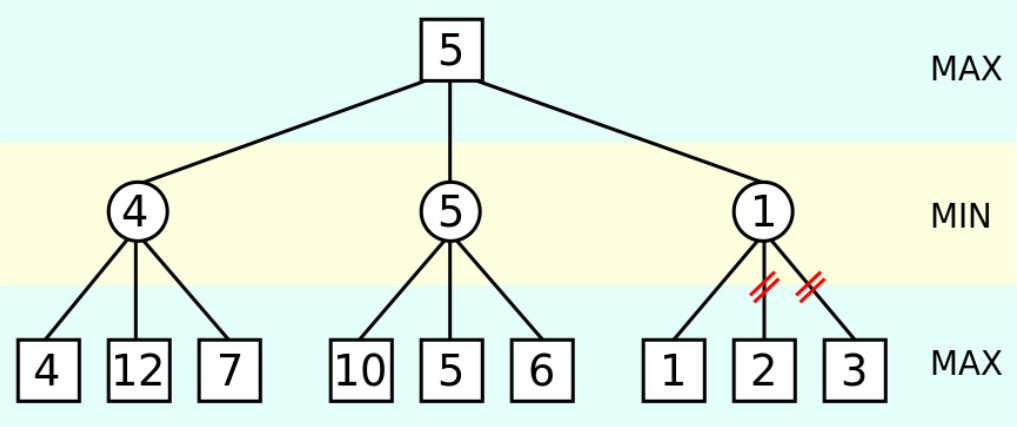

Figura 1: Ejemplo de Poda Alfa Beta explicado en el blog de Jean-François [27]

Puede ejecutar los siguientes movimientos:

- Mover una pieza
- Capturar una pieza
- Enroque (Roque)
- Al paso (En passant)
- Promover una pieza

Permite jugabilidad tipo:

- Jugador vs Jugador
- Jugador vs Inteligencia Artificial
- Inteligencia Artificial vs Inteligencia Artificial

El tablero está mapeado como una lista que guarda las 64 casillas en orden, de izquierda a derecha y de arriba abajo como se muestra en la Figura 2. Esta implementación además permite ver el tablero por consola para poder seguir el juego con normalidad.

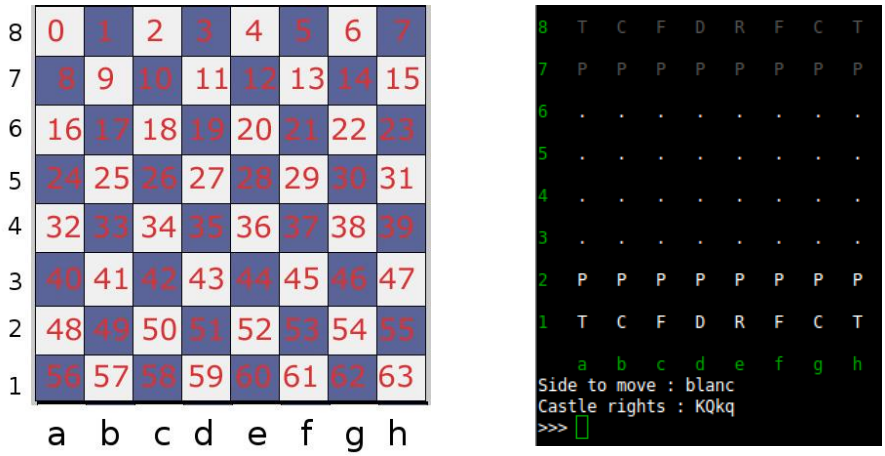

Figura 2: Mapeo del tablero según [27] (izquierda), Tablero impreso en consola (derecha)

2.2. RECONOCIMIENTO Y PROCESAMIENTO DE IMÁGENES

Los robots han estado en desarrollo por mucho tiempo. Uno de los primeros robots, Eric, fue construido el año 1928 por el Capitán William H. Richards, veterano de la primera guerra mundial [28]. Sus funciones eran básicas, se paraba, se sentaba, reverenciaba y daba un discurso. Eric no necesitaba herramientas para percibir su entorno, ya que dependía de él. A medida que el tiempo ha ido avanzando los robots se han vuelto más complejos. El año 1967 crearon a Shakey, el primer robot con inteligencia artificial [29]. Shakey fue muy importante para el campo de la robótica en general, de él surgieron avances en múltiples áreas, tales como, inteligencia artificial, robótica, reconocimiento de imágenes y navegación y fue de los primeros en relacionarse con su entorno de manera inteligente, utilizando cámaras y ruedas para moverse alrededor de su laboratorio y reconociendo qué hacer con los obstáculos. Más adelante, en el año 2000, Honda creó al robot Asimo [30], un robot humanoide capaz de subir y bajar escaleras, patear una pelota e incluso abrir una botella y servirle jugo a alguien. Si bien este último no parecerá como una gran meta para un ser humano normal debido a lo trivial que es abrir una botella y servir la bebida, para la robótica, es una meta de gran magnitud. Asimo utiliza diversas herramientas para realizar sus acciones en su entorno. Posee cámaras para distinguir lugares y obstáculos, medidores que le permiten saber si va a caerse o no para luego estabilizarse y mantenerse de pie o no caer al caminar, entre otros.

Los robots han ido avanzando, y con cada avance significativo también debe avanzar su reconocimiento con el entorno, ya sea para desenvolverse mejor en él, o para adaptarse a este.

Baxter es un robot creado originalmente para manufactura (Ver sección 2.3). En ese ambiente, se debe lidiar constantemente con el contacto humano y para ello, debe aprender a convivir con ellos. El robot Baxter puede desenvolverse en su entorno con facilidad gracias a su diseño compuesto de cámaras y motores que le permiten tener una buena relación con el entorno humano, evitando contacto accidental con humanos, minimizando su uso de fuerza y deteniéndose en contacto con un humano [31].

Para solucionar los problemas de percepción que este trabajo conlleva, se utilizaron algoritmos en reconocimiento de imágenes como se explica en la sección 3.2.

El reconocimiento de imágenes en robótica es diferente al que se está acostumbrado a escuchar [32]. Un robot, a diferencia de una cámara estática, cambia las posiciones de su cámara, generando ruido y problemas en la percepción, debido a problemas como la perspectiva o la fuente de luz cuya posición cambia respecto al movimiento de la cámara [33]. Esto es muy importante sobre todo en robots que usan este reconocimiento para actuar o moverse [34].

El reconocimiento de imágenes de este proyecto se basó en un trabajo realizado por Active Robots con fines educativos [35].

El trabajo consiste en que el robot Baxter pueda tomar pelotas de golf y colocarlas en orden dentro de una bandeja de huevos. Para esto, utiliza técnicas de reconocimiento de imágenes como Canny Edge Detector [36] para reconocer la bandeja de huevos y HoughCircles [37] para encontrar todas las pelotas de golf como se aprecia en la Figura 3. Este último fue el más importante para la realización de este proyecto de memoria de título, ya que los métodos implementados para reconocer las piezas y las casillas del tablero utilizan la función HoughCircles implementada en OpenCV [38].

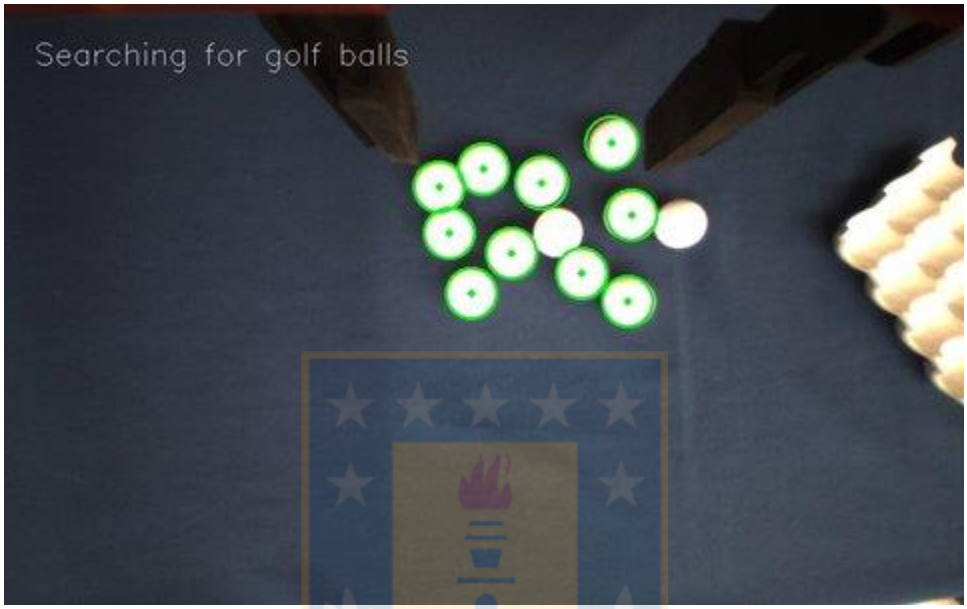

Figura 3: Reconocimiento de las pelotas de golf

2.3. BAXTER JUGANDO AJEDREZ

Baxter (Figura 4) es un robot creado por Rethink Robotics [23] el año 2012 con el fin de ser utilizado en manufactura usualmente en empaque, sin embargo, también se utiliza en instituciones de educación superior e investigación [39].

Figura 4: Baxter en empresa de manufactura

El tema específico de jugar ajedrez con el robot Baxter no ha sido estudiado ampliamente.

Las investigaciones realizadas en [40] toman un enfoque muy distinto a esta memoria de título. En primer lugar, debido a la dificultad de reconocer piezas de ajedrez con el robot Baxter, no utilizan piezas de ajedrez clásicas, sino que usan sus propias piezas que consisten en cubos de espuma con un código arriba y una pequeña foto de la pieza. Además, utilizan un tablero completamente blanco con líneas que dividen las casillas en vez de un tablero blanco y negro clásico como se puede ver en la Figura 5.

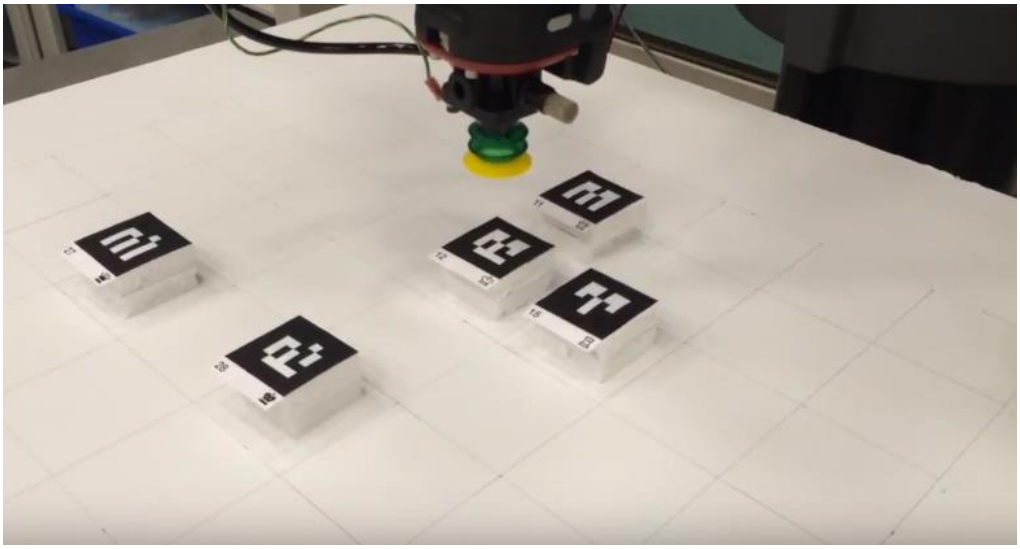

Figura 5: Pieza de ajedrez para y tablero utilizados en [40]

En [40], el robot Baxter utiliza ambos brazos para jugar, el izquierdo para mover piezas fuera del tablero si es que fueron capturadas, y el derecho para ejecutar el resto de los movimientos, esto lo realiza utilizando succión en ambos brazos en vez de pinzas. El resultado (Ver Figura 6) carece de precisión absoluta, no deja las piezas centradas e incluso quedan en intersecciones de casillas, provocando que estas después deban ser movidas por el jugador humano para que queden bien posicionadas y no terminen rotadas indebidamente.

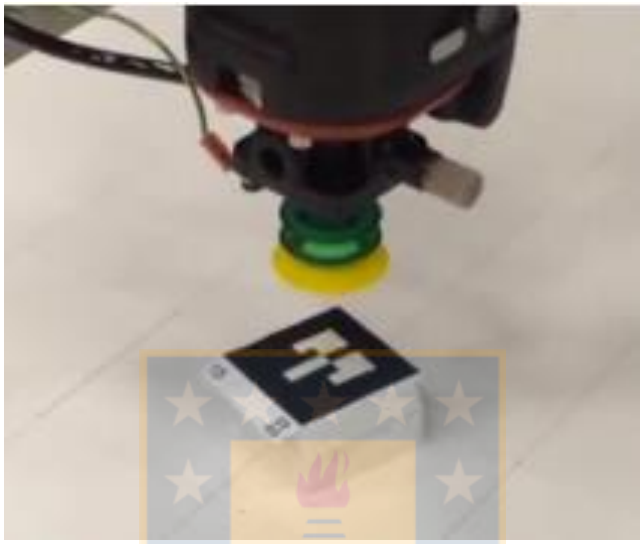

Figura 6: Resultados de la investigación en [40], pieza puesta en intersección de casillas

Por otra parte, para simular el ajedrez utilizaron Stockfish Chess [26], un sofisticado motor de ajedrez con distintas funciones de puntaje y patrones e instrucciones específicas para cada tipo de piezas y jugada que fue considerado el mejor software de ajedrez hasta fines del año 2017 cuando fue vencido por AlphaZero, inteligencia artificial creada por Google [41].

El segundo experimento realizado por vteams [42], funciona con dos jugadores remotos, uno en cada computador y Baxter sólo ejecuta el movimiento de los jugadores (Ver Figura 7), es decir, no juega utilizando una inteligencia artificial y es sólo un puente entre lo virtual y lo físico.

Para jugar, cada jugador abre su juego de ajedrez en su computador y juegan por conexión remota. Cada jugada realizada es entonces emulada por Baxter en el plano físico utilizando un tablero y piezas clásicas.

No se especifica en [42] cómo se construyó el proyecto de Baxter jugando ajedrez, pero se puede ver en el experimento que consta con una interfaz gráfica para ayudar a los jugadores a guiarse correctamente durante el juego. Tampoco detallan mucho en las jugadas posibles de su ajedrez, si es que puede realizar primer paso o enroque, además de reglas como no realizar la misma jugada tres veces, o la regla de los 50 turnos que especifica que si ningún jugador captura una pieza o mueve un peón en los últimos 50 turnos, se puede declarar el juego como empate.

En vez de reconocimiento de imágenes, a Baxter se le entregan las coordenadas de la ubicación de la pieza a mover y su destino, luego esas coordenadas se transforman a coordenadas de Baxter (coordenadas cartesianas en metros y ángulos en radianes) que dependen en este caso de el tamaño de las casillas y su distancia respecto al centro del torso de Baxter. En cambio, en el presente trabajo, se trabajó con procesamiento de imágenes para reconocer el tablero y las piezas.

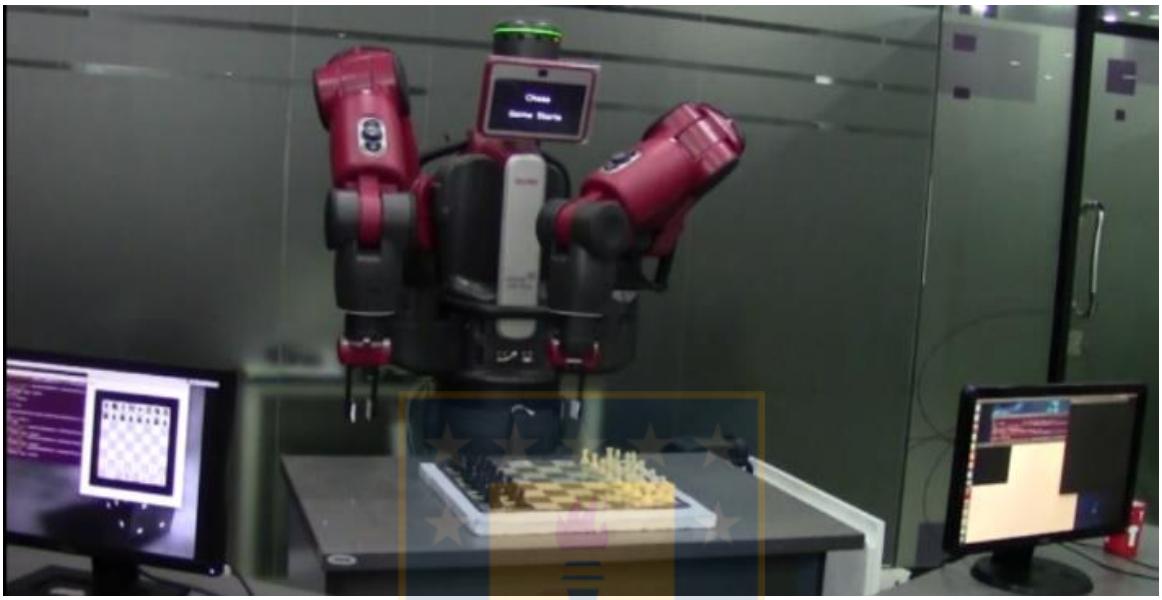

Figura 7: Experimento realizado por [42]

3. MÉTODO DESARROLLADO

El trabajo desarrollado se divide en tres partes. En primer lugar, se encuentra la preparación del ambiente, esto incluye la configuración del área de trabajo, del tablero y del robot. La segunda parte consiste en el reconocimiento del tablero y las piezas de ajedrez, los algoritmos utilizados para ello, problemas y técnicas implementadas. Por último, se encuentra la sección del ajedrez, que aborda el cómo juega ajedrez el robot y cómo procesa y enlaza sus percepciones con sus acciones.

Se escogió el robot Baxter en particular debido a que cuenta con las capacidades necesarias para poder jugar ajedrez, puede tomar las piezas, reconocer imágenes con sus cámaras y además tiene forma humanoide lo que lo hace más interesante a la vista. Para hacerlo jugar se estudiaron diversas técnicas de reconocimiento de imágenes además de las librerías y herramientas que ofrece ROS (Robot Operating System) [43].

Para jugar ajedrez el robot primero debe reconocer el tablero de ajedrez y la posición de las piezas descrito en la sección 3.2, luego, mediante el algoritmo basado en [35] Baxter escoge la pieza a mover y su destino. Mediante técnicas de movimiento en robótica implementadas en ROS, el robot Baxter mueve la pieza como corresponde con las coordenadas entregadas por el algoritmo de ajedrez y el reconocimiento de imágenes. En el caso de capturar una pieza, primero saca la pieza capturada del tablero y luego mueve la pieza capturadora a su destino.

A continuación, se muestran de manera más detalladas la preparación, el reconocimiento del tablero, el algoritmo para jugar ajedrez y los movimientos del robot.

3.1. PREPARACIÓN

La primera gran diferencia entre este trabajo y las investigaciones realizadas por otros equipos es el ambiente de trabajo. Las investigaciones citadas en la sección 2.1, constan con un laboratorio equipado para su investigación, a diferencia de este proyecto en la que se tuvo que improvisar un ambiente para trabajar, ya que, en vez de un laboratorio real, es una sala de clases común y corriente sin equipamiento especial para el robot o el área de la robótica en general.

El laboratorio improvisado en el cual se encuentra Baxter tiene ventanales en un lado tapado con cortinas café claro, disminuyendo la iluminación proveniente de las ventanas. En el techo, se encuentran 6 luces blancas que iluminan todo el laboratorio. Estas luces pueden ser encendidas en pares, permitiendo distintos tipos de iluminación, esto se puede hacer prendiendo las dos luces al frente del robot, las dos luces encima del robot o las dos detrás de él, además se pueden prender combinaciones distintas como las dos de atrás y las dos de adelante al mismo tiempo.

En la sala, al frente del robot, se encuentran tres mesas juntas con una tabla de 1.5 x 2.5m color negro sobre ellas. Esta tabla ayuda al reconocimiento de imágenes ofreciendo un fondo oscuro y evitando que se enfoquen cosas aparte de ella (el piso, los cables, etc.), además, sobre la tabla se encuentran cartulinas negras para disminuir el brillo producido por las luces del techo sobre la tabla. Al lado izquierdo de la mesa se encuentra una lámpara con luz amarilla que se utiliza para entregar una mejor iluminación en caso de ser necesario. Por último, se dispone de una caja transparente a la izquierda para dejar las piezas capturadas.

El robot tiene una cámara en cada brazo, una cámara y pantalla frontal en su cabeza y una pinza en cada mano. Para este proyecto se trabaja utilizando el brazo izquierdo sin pinzas y el brazo derecho con pinzas, esto debido a que la cámara izquierda es la que realiza el reconocimiento de imágenes, por lo que se optó por remover las pinzas que obstruían la imagen percibida (Ver Figura 8).

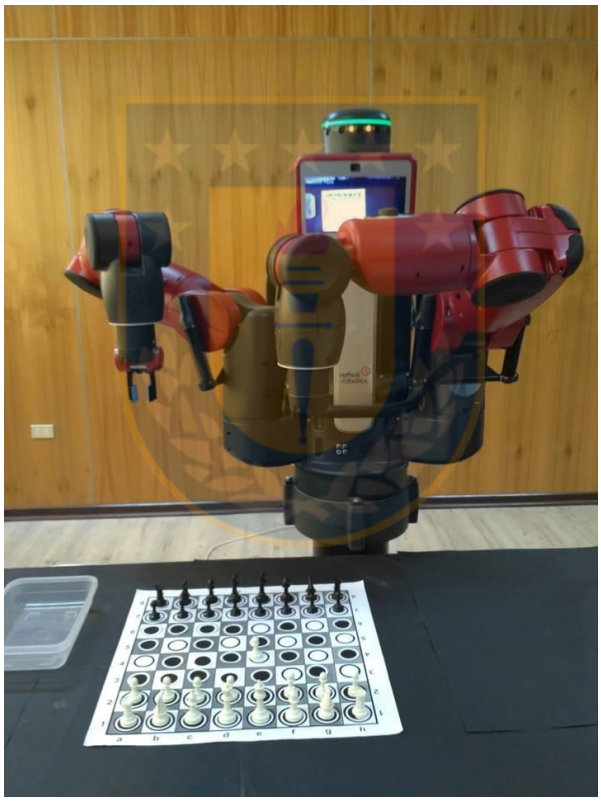

Figura 8: Ambiente de trabajo del robot Baxter

El tablero está impreso en una hoja con un tamaño de 50 x 50 cm en blanco y negro, dónde las casillas son grises para que se haga más fácil el reconocimiento de las piezas. Consta con las 64 casillas y los identificadores de coordenadas, estas son, números a los costados del 1 al 8 y letras de la A a la H en la parte inferior y superior del tablero. Cada casilla tiene impresa en ella dos círculos (Ver Figura 11), uno exterior del color opuesto a la casilla en la que está y un círculo interior de color igual al de la casilla como se ve en la Figura 9. Eso con el fin de poder reconocer las piezas utilizando HoughCircles.

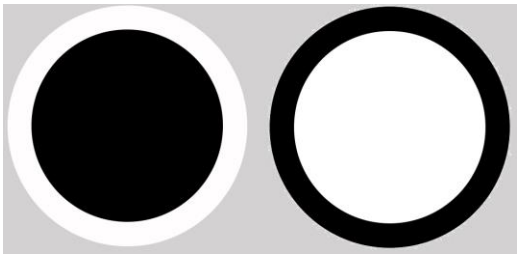

Figura 9: Círculos en las casillas del tablero, a la izquierda, círculo para casillas negras, a la derecha, círculo para casillas blancas.

Las piezas utilizadas son piezas clásicas blancas y negras obtenidas de un juego de ajedrez perteneciente al centro de alumnos de la facultad. En la base de cada pieza se colocó un círculo triple en su posición inicial, esto es, centro y exterior del color opuesto a la pieza separados por un color intermedio igual al color de la pieza (Ver Figura 10). De esta forma, los círculos son reconocidos en las casillas, respecto al color de las mismas.

Figura 10: Círculos en las bases de las piezas en su posición inicial, a la izquierda, círculo para piezas negras, a la derecha, círculo para piezas blancas.

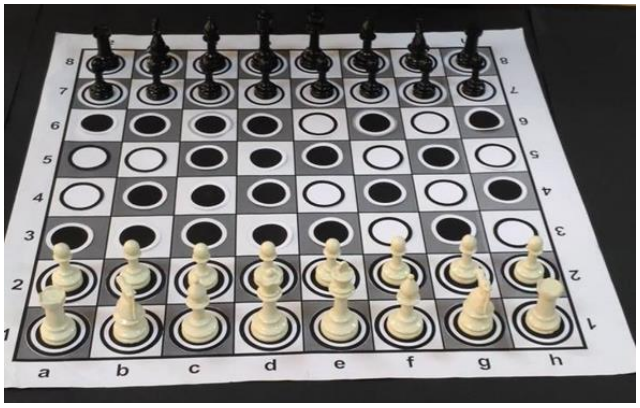

Figura 11: Tablero de ajedrez utilizado durante la experimentación

3.2. RECONOCIMIENTO DEL TABLERO

El reconocimiento digital de imágenes es un proceso que involucra diversas fuentes de incertidumbre. Por ejemplo, efectos externos como cuando una nube pasa durante un día soleado y genera sombras o bloquea el sol por completo, pudiendo así generar problemas en la calidad del reconocimiento. El brillo generado por las luces artificiales es variable dependiendo de la hora y el clima, si está soleado, este brillo se notará menos, de la misma manera si es de día, por otro lado, si está nublado el brillo se notará más y si es de noche se verá mucho más. La iluminación en general también depende mucho de las condiciones del día o de las mismas luces artificiales que pueden comenzar a fallar, o en el caso de la luz lateral, estar ligeramente movida, generando distintas sombras que pueden causar problemas. Las técnicas basadas en pixeles pueden ser problemáticas al momento de detectar un pixel erróneo que puede acarrear mucho error.

Se utilizó OpenCV en sus diferentes versiones (OpenCV 2 y OpenCV 3) para el reconocimiento de imágenes.

OpenCV (Open Source Computer Vision Library), es una librería open source que posee métodos de machine learning y reconocimiento de imágenes o visión artificial (computer vision) [29]. Es compatible con C++, MatLab, Python y Java, lo que lo hace idóneo para trabajar con el robot Baxter.

Para reconocer el tablero se considera lo siguiente:

- El tablero es el mencionado en la sección 3.1
- Las piezas son las mencionadas en la sección 3.1
- El ambiente de trabajo es el descrito en la sección 3.1

De lo contrario, no se garantizan los resultados mencionados posteriormente.

El reconocimiento del tablero utiliza técnicas en reconocimiento y procesamiento digital de imágenes. A continuación, se describen los pasos para ello:

1) Se inicializa el robot en un estado de espera en el que ubica el brazo izquierdo sobre el tablero a una distancia de 45 cm, lo que permite ver todo el tablero. Luego, el brazo derecho se mueve para no molestar el reconocimiento. La cámara ve y muestra en pantalla lo que se encuentre a su alcance.

2) Se obtienen las coordenadas de las casillas y la posición inicial de las piezas en pixeles utilizando la función de OpenCV *HoughCircles* (Ver Anexo 7.1) que detecta todas las circunferencias de la imagen (Ver Figura 12), considerando solo aquellas que estén a una distancia determinada, de manera que no se identifiquen circunferencias innecesarias. Estos círculos son determinados en un orden azaroso. Se realiza la detección en 100 iteraciones, para asegurarse que encuentre todos los círculos al menos una vez, el algoritmo tiene una precisión entre 68% y 100% (Ver tabla 3 en la sección 4.3 (resultados)). Si se encuentran más de 64 círculos (cantidad de casillas), se omite el resultado y se procede a iniciar la siguiente iteración, lo mismo ocurre si encuentra menos.
3) Se ordenan los círculos de izquierda a derecha y de arriba abajo al igual que el mapeo original del tablero en [23], utilizando las coordenadas del centro de estos. Para ordenarlos, se ordenó primero respecto al eje y, pero si había pequeños cambios en las coordenadas del eje Y el ordenamiento fallaba, por ejemplo, al ordenar la primera fila, entregaba este resultado → 0,1,2,3,5,4,6,7; en vez de el deseado 0,1,2,3,4,5,6,7.
   Para corregirlo, una vez ordenados los círculos en el eje Y, se reordena cada fila en el eje X, de esta manera se garantiza el resultado deseado.
4) Se convierten estas coordenadas en pixeles a coordenadas de Baxter.
5) Finalmente se guardan estas coordenadas de Baxter ordenadas y se utilizan al momento de mover las piezas.

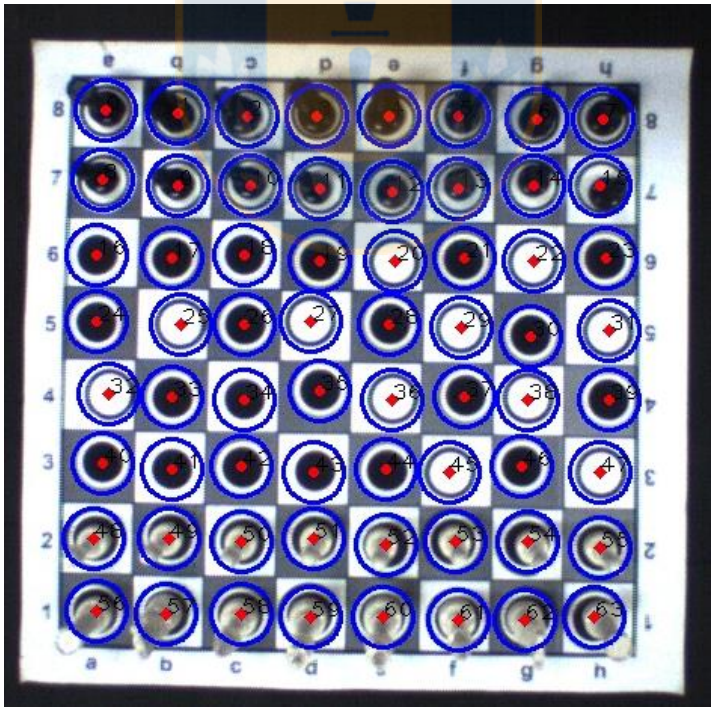

Figura 12: Reconocimiento de círculos

3.3. AJEDREZ

El algoritmo de ajedrez está dividido en 3 partes, tablero, piezas y el motor del ajedrez, estas partes están separadas en tres ficheros distintos, uno para cada parte. A continuación, se explicará la función de cada uno de ellos.

3.3.1 PIEZAS

Cada pieza está formada por un nombre, un valor y un color. El color depende de qué jugador sea el controlador de la pieza mientras que el nombre y valor es específico para cada tipo de pieza, pero los valores se pueden repetir entre ellos. Los nombres dependen del tipo de pieza y poseen los mismos nombres que en el juego original, es decir, un caballo tiene nombre 'CABALLO', un peón tiene nombre 'PEÓN'. La casilla vacía se identifica con el símbolo '.'. Los valores asignados (puntaje) a cada pieza son los siguientes: Reina = 9, Torre = 5 Caballo = 3, Alfil = 3, Peón = 1, Rey = 0, Casilla vacía = 0.

Para el movimiento de las piezas, se utilizan dos matrices.

La primera matriz de 12×10 tiene una submatriz de 8×8 (Ver Figura 13) y sirve para verificar que las piezas no se salgan del tablero al ejecutar una jugada, por ejemplo, si una torre se encentra en 3 en la matriz 12×10 y trata de ir hacia arriba, no podrá realizar el movimiento, ya que hay un -1.

La segunda matriz de 8×8 (Ver Figura 13) sirve para mover las piezas y es la que se utiliza para navegar en el tablero durante el juego.

```
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1,  0,  1,  2,  3,  4,  5,  6,  7, -1,
-1,  8,  9, 10, 11, 12, 13, 14, 15, -1,
-1, 16, 17, 18, 19, 20, 21, 22, 23, -1,
-1, 24, 25, 26, 27, 28, 29, 30, 31, -1,
-1, 32, 33, 34, 35, 36, 37, 38, 39, -1,
-1, 40, 41, 42, 43, 44, 45, 46, 47, -1,
-1, 48, 49, 50, 51, 52, 53, 54, 55, -1,
-1, 56, 57, 58, 59, 60, 61, 62, 63, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1
```

```
21, 22, 23, 24, 25, 26, 27, 28,
31, 32, 33, 34, 35, 36, 37, 38,
41, 42, 43, 44, 45, 46, 47, 48,
51, 52, 53, 54, 55, 56, 57, 58,
61, 62, 63, 64, 65, 66, 67, 68,
71, 72, 73, 74, 75, 76, 77, 78,
81, 82, 83, 84, 85, 86, 87, 88,
91, 92, 93, 94, 95, 96, 97, 98
```

Figura 13: A la izquierda, matriz 12×10. A la derecha, matriz 8×8

Para poder navegar en la matriz 8×8, cada pieza debe tener una cantidad de movimientos realizables al igual que en el ajedrez original, los movimientos son los siguientes:

- Torre: -10, 10, -1, 1.
- Alfil: -11, -9, 11, 9
- Caballo: -12, -21, -19, -8, 12, 21, 19, 8
- Rey y Reina: Torre + Alfil
- Peón:
    - Si es negro: +10 para avanzar, +20 si nunca se ha movido (opcional), +9 y +11 para capturar.
    - Si es blanco: -10 para avanzar, -20 si nunca se ha movido (opcional), -9 y -11 para capturar.

Además, algunos movimientos se pueden multiplicar dependiendo la pieza que se esté jugando, por ejemplo, a una torre se le puede dar un multiplicador máximo de 7, que lo permitirá moverse de una esquina adyacente a otra (Ver Figura 14). Las piezas con multiplicador son: Torre, Alfil, Reina. Para el caso de las piezas con multiplicador se utiliza la siguiente función:

$$Movimientos\ disponibles * multiplicador, \text{ donde } multiplicador = 1,2,3,4,5,6,7$$

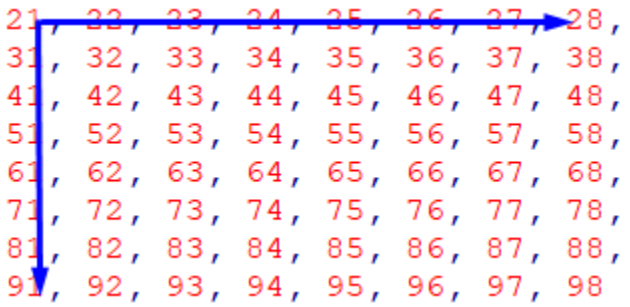

Figura 14: Ejemplo de movimiento de torre de a8 a h8 y de a8 a a1. Los números tocados por la flecha azul son los espacios a los cuales la torre se puede mover si está en a8 (21 en la tabla)

Un peón tiene la capacidad de ser promovido, esto ocurre sólo cuando un peón llega al extremo opuesto del tablero, por ejemplo, un peón blanco debe llegar a la primera fila del tablero. Al promoverse, se convierte en cualquier pieza que desee (excepto rey o peón) y sus atributos pasan a ser los de la nueva pieza y su posición inicial será la casilla actual en la que se encuentra al momento de ser promovido.

3.3.2 TABLERO

En esta sección se crea un tablero con las coordenadas estándar (a1, b1, c1, etc…) y se crea una lista con el estado inicial del tablero, dónde se colocan las piezas en su lugar indicado para jugar y se llenan las casillas vacías con Piece(), que tiene nombre '.' y valor 0.

Se crean marcas(flags) para saber qué movimientos especiales van quedando disponibles a medida que avanza el juego, estos son los diferentes enroques (enroque corto o largo, negro o blanco) y primer paso. Se le da el primer turno al jugador blanco, se inicializa el historial de movimientos como una lista vacía y la cantidad de turnos jugados en 0.

Las tres funciones más importantes en este fichero son gen_move_list y domove (Ver Algoritmo 1) y evaluate (Ver Algoritmo 2).

La primera función genera una lista con todas las jugadas posibles para el jugador que está jugando, revisa casilla por casilla ignorando las vacías y las que tengan piezas que no corresponden a su color. Esto sirve para otras funciones como is_attacked, is_checked y para almacenar el historial. Además, también sirve para entregar las jugadas legales que tiene el jugador si éste las pide por consola.

La segunda función, domove, es la que permite el movimiento de las piezas en el tablero.

Algoritmo 1: domove

Input: departure, arrive, promote

Output: Boolean (indica se mueve o no)

self.cases[arrivee] ← self.cases[depart]

self.cases[depart] ← Piece()

self.ply ← self.ply + 1

checks en passant

checks castling

return True if moved

return False if not moved

Esta función recibe por parámetros el punto de origen, el destino, y si es promovida o no. Con estos datos recibidos, procede a realizar el movimiento. Para ello, a la casilla de destino le asigna los datos de la pieza de la casilla de origen y a la pieza de origen le asigna los datos de la casilla vacía, ya que, cuando una pieza se mueve, deja el espacio de origen (de dónde se movió) se movió, vacío. Luego, le suma 1 a ply (ply es medio turno o una jugada básicamente). Por último, revisa el estado de las jugadas especiales, para saber si fueron realizadas o no, si no lo fueron, revisa si todavía son realizables, ya que estas pueden dejar de ser realizables si, por ejemplo, el rey está en jaque, o se comieron una torre para un enroque. La función retorna “True” si la pieza se movió y retorna “False” si no se movió. Es posible que una pieza no se mueva si se entregó por consola una jugada ilegal o si deja al propio rey en jaque.

La tercera función es la función evaluate, correspondiente a la función de evaluación.

Algoritmo 2: evaluate

Input: -

Output: score

```
WhiteScore ← 0
BlackScore  ← 0
for pos1,piece in enumerate(self.cases) do
    if piece.color=='white' then
        WhiteScore ← WhiteScore + piece.value
    else
        BlackScore ← BlackScore + piece.value
If self.side2move=='white' then
     return WhiteScore-BlackScore
else
      return BlackScore-WhiteScore
```

Por cada movimiento que la inteligencia artificial del ajedrez realiza, la función de evaluación busca todas las piezas que sean del mismo color que el color del jugador que está llamando a la función. Para cada una de estas piezas, obtiene su valor asignado y los suma todos. La suma de todos los valores de las piezas equivale al puntaje de un jugador. Finalmente, resta su puntaje con el puntaje del oponente para verificar cuál tiene puntaje más alto. Dependiendo de la profundidad de búsqueda, puede decidir que tipo de jugada realizar, ya sea moverse normalmente, comerse una pieza, sacrificar una de sus propias piezas para una jugada más estratégica, entre otras.

Otras funciones presentes en este fichero son:

render: Muestra el tablero en la terminal

caseStr2Int y caseInt2Str: Transforma el número de la casilla a coordenadas o las coordenadas a el número de la casilla respectivamente (e2 → 52, 52 → e2)

showHistory: Muestra el historial

is_attacked: Revisa si las piezas pueden ser atacadas. Función utilizada para el jaque.

is_check: Revisa si hay jaque

3.3.3 MOTOR DE AJEDREZ

La última parte del ajedrez, es aquella que tiene las instrucciones para las distintas entradas al momento de correr el programa. Entre las funciones más importantes destacan, usermove, search y alphabeta (Ver Algoritmo 3).

usermove permite al jugador entregar por consola la coordenada de la jugada y esta función la interpretará y ejecutará. Los pasos que realiza la función son los siguientes:

1) Revisa si la partida terminó, luego revisa si se ingresó un comando correcto
2) Transforma lo ingresado de coordenadas a números. e2 → 52
3) Revisa si hay promoción
4) Revisa si la jugada ingresada está dentro de la lista de movimientos posibles, si no lo está, pide que ingrese otra jugada, ya que la ingresada es inválida, si está, realiza la jugada.
5) Imprime por consola el tablero como se muestra en la Figura 2

search es análogo a usermove, pero en vez de recibir una jugada como input humano por consola, escoge el movimiento a realizar usando la función alphabeta (Ver Algoritmo 3) y lo ejecuta.

La búsqueda alfa beta implementada (Ver Algoritmo 3), utiliza una Triangular PV-Table (Triangular Principal Variation Table) para guardar los movimientos. Triangular PV-Table es un arreglo de Principal Variations (o variaciones principales en castellano), indexadas por el número de jugadas (ply). Es utilizada para almacenar la variación principal de los mejores movimientos dentro de alfa beta, propagando el mejor puntaje hasta la raíz del árbol como se ve en la Figura 15.

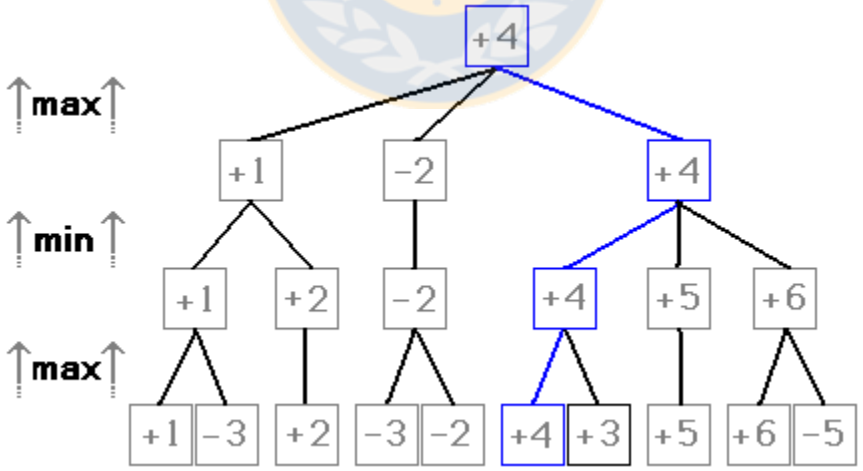

Figura 15: Ejemplo de variación principal (imagen obtenida de [44])

Algoritmo 3: alphabeta

Input: depth, alpha, beta, b

Output: score

```
if  depth==0 then
        return b.evaluate()
self.nodes ← self.nodes + 1
self.pv_length[b.ply] ← b.ply
if b.ply >= self.MAX_PLY-1 then
        return b.evaluate()
chk ← b.in_check(b.side2move)
if chk then
         depth ← depth +1
mList ← b.gen_moves_list()
f ← False
for i,m in enumerate(mList) do
        if not b.domove(m[0], m[1], m[2]) then
                continue
        f ← True
        score ← -self.alphabeta(depth-1,-beta,-alpha,b)
        b.undomove()
        if score > alpha then
                if score >= beta then
                        return beta
                alpha = score
                self.pv[b.ply][b.ply] ←  m
                j ← b.ply + 1
                while j < self.pv_length[b.ply+1] do
                        self.pv[b.ply][j] ←  self.pv[b.ply + 1][j]
                        self.pv_length[b.ply] ←  self.pv_length[b.ply + 1]
```

```
                    j ← j + 1
    if not f then
        if chk then
            return -self.INFINITY + b.ply
        else
            return 0
return alpha
```

Inicialmente, el algoritmo revisa si la profundidad de búsqueda es 0, si lo es, retornará como puntaje el resultado de la función de evaluación (Algoritmo 2), de lo contrario, asignará el valor correspondiente a su posición en la lista pv_length. Si la profundidad es mayor a 30, entonces retorna el puntaje otorgado por la función de evaluación (Algoritmo 2). Luego, revisa si el rey está en jaque, si es así, entonces aumentará en uno la profundidad para tener más posibilidades al momento de salir del jaque. Genera todos los movimientos que puede realizar el jugador de turno y se comprueba que al menos se realice una jugada de esta lista. Se asigna el puntaje de esta jugada recursivamente. Si una jugada genera una poda, se le dará más prioridad (subirá de nivel). Finalmente se actualiza la Triangular PV-Table con la secuencia de mejores movimientos encontradas como se ve en la Figura 16.

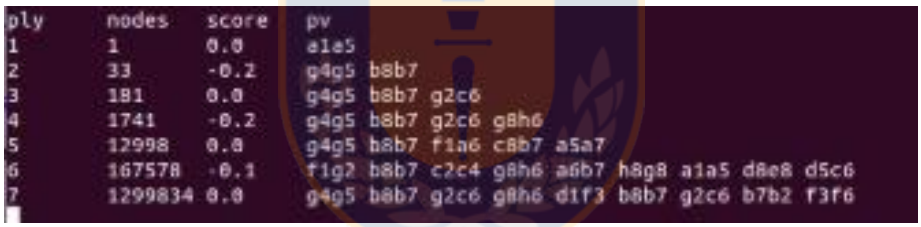

Figura 16: Inteligencia artificial imprimiendo su análisis en consola

3.4. ROBÓTICA

A través de las librerías y herramientas que aporta ROS, se ejecutan los movimientos necesarios para mover las piezas de ajedrez como corresponda. Para ello, se utilizan funciones implementadas en ROS que permitan activar las cámaras, abrir o cerrar las pinzas y mover los brazos. El algoritmo de ajedrez entrega el movimiento a realizar al algoritmo de reconocimiento de imágenes, este convierte esas instrucciones en coordenadas de Baxter como se explicó en la sección 3.2, se le aplica una pequeña desviación calculada mediante ensayo y error debido a la falla de exactitud en la precisión y se mueve la pieza o piezas en el caso que capture una.

Para que Baxter mueva sus brazos, se le debe entregar el destino en sus coordenadas (x, y, z y sus ángulos respectivos roll(x), pitch(y), yaw(z)). Lo único que sabe Baxter entonces, es su punto de inicio y su destino. Esto significa que Baxter puede tomar el recorrido que él encuentre óptimo para moverse a su destino. Por esta razón, Baxter en muchas ocasiones toma un recorrido que choque su brazo con otras cosas, como piezas, el tablero o la mesa, rayándolo como se ve en la siguiente figura (Figura 17) y moviéndolo.

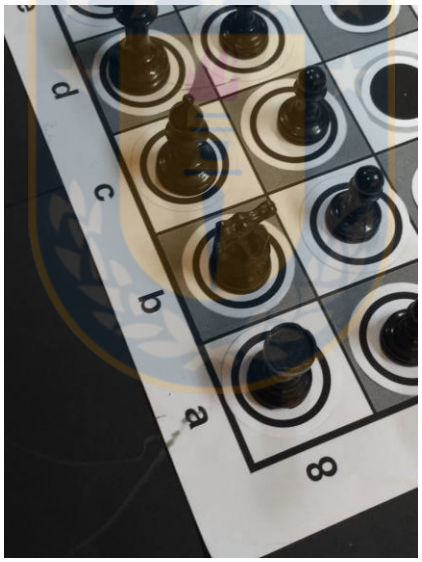

Figura 17: Marcas generadas por el choque de las pinzas con el tablero o la mesa

Para solucionar esto, cada vez que Baxter mueve una pieza, se realizan movimientos cortos entre el origen y el destino, en total son 12 los pasos que Baxter realiza para llegar a su destino. Para describir los pasos, se utilizará como ejemplo el movimiento e2e4.

Los pasos son los siguientes:

1) Abrir la pinza: Simplemente revisa si tiene la pinza abierta, si está cerrada, la abre.
2) Ir a la posición inicial: Posición fija a la que Baxter siempre vuelve al comenzar la secuencia, esto se hace para reducir los movimientos no deseados descritos anteriormente. Revisa si el movimiento a ejecutar involucra la captura de una pieza. Si es así, proceder a I., de lo contrario, seguir en 3)
    I. Se sitúa arriba de la pieza a capturar
    II. Baja hasta poder tomarla
    III. Cierra las pinzas y la agarra
    IV. Levanta la pieza a una altura determinada por ensayo y error
    V. Se mueve afuera del tablero y se sitúa sobre una caja transparente
    VI. Abre las pinzas y bota la pieza en la caja
    VII. Vuelve a la posición inicial y retoma los pasos en 3)
3) Situarse arriba del origen(e2): Baxter mueve su brazo derecho desde la posición inicial a una altura determinada del origen de la pieza a mover y espera un segundo para que se sitúe correctamente en la posición. Esta altura se determinó como la mejor por ensayo y error.
4) Bajar hasta el punto de origen(e2): Baxter baja su brazo derecho a la altura indicada para poder tomar la pieza. Altura determinada por ensayo y error.
5) Tomar la pieza(e2): Baxter cierra las pinzas en su brazo derecho y espera un segundo para asegurar que ha tomado correctamente la pieza.
6) Subir con la pieza tomada(e2): Una vez tomada la pieza, Baxter sube su brazo derecho nuevamente a una altura distinta del paso 3 (un poco más alto), esto lo hace para evitar colisión con las piezas a su alrededor al momento de realizar el paso 6.
7) Situarse arriba del destino(e4): Sin cambiar la altura del paso 5, se sitúa arriba de la posición de destino.
8) Bajar hasta el punto de destino(e4): Cambia solamente su altura y baja para quedar a la distancia adecuada de su destino.
9) Suelta la pieza(e4): Abre sus pinzas soltando la pinza y espera medio segundo para asegurarse que no mueva la pieza por moverse muy rápido apenas abra las pinzas.
10) Situarse arriba del destino(e4): Mismas coordenadas que el paso 6, pero esta vez no lleva la pieza
11) Subir más: Sube un poco más manteniendo las coordenadas de 10, esto se hace para evitar movimientos no deseados. Revisa si se produjo enroque o primer paso, de ser así, proceder a 1, de lo contrario, seguir en 12).
    1. Si es enroque, ejecuta los pasos 3) al 11) y sale en 12)
    2. Si es primer paso, ir a I. y seguir hasta VI., Luego, ir a 12).
12) Moverse arriba de la posición inicial: Se mueva arriba y no directamente a la posición inicial para evitar los movimientos no deseados.

Así termina la secuencia de pasos para mover una pieza. Cada vez que mueva una pieza deberá repetir estos 12 pasos. La Figura 18 muestra a Baxter tomando un peón para levantarlo.

Figura 18: Baxter levantando un peón

4. EXPERIMENTOS Y RESULTADOS

En esta sección se muestran los experimentos realizados por Baxter y sus resultados.

Los experimentos realizados consistieron en analizar la eficacia del algoritmo de ajedrez, comprobar el funcionamiento de los algoritmos de reconocimiento de imágenes y analizar el comportamiento del robot Baxter.

Para realizar los experimentos, se ejecutó cada parte por separado hasta que funcionasen correctamente. Una vez ocurrido esto, se comenzaron las pruebas de Baxter jugando ajedrez mediante el uso de una inteligencia artificial y procesamiento digital de imágenes.

4.1. ALGORITMO AJEDREZ

Se realizaron dos experimentos para probar la efectividad de la búsqueda alfa beta implementada. En primer lugar, se hizo que Baxter jugase contra sí mismo.

Se corrió el algoritmo durante 14hrs (aprox) y se obtuvieron los siguientes resultados:

Profundidad	Turnos jugados	Resultado final	Número de turnos al término
3	21561	Loop infinito	13
4	7884	Loop infinito	15
5	1389	Loop infinito	13
6	55	Loop infinito	50
7	15	Loop Infinito	12
8	6	No terminó	6

Tabla 1: Resultados de Baxter contra Baxter

Donde:

- Profundidad: Indica el número de jugadas que predice el algoritmo.
- Turnos jugados: Un turno son dos jugadas (una blanca y una negra)
- Resultado final: Indica el resultado final de la partida, este puede ser victoria de un lado empate, loop infinito (llega un punto en que repiten jugadas) o no termina.
- Número de turnos al término: Cuántos turnos tardó en obtener los resultados de la columna anterior (Resultado final)

Los resultados de Baxter jugando contra sí mismo son preocupantes, se esperaba que al menos uno ganase, pero en todos los casos (excepto el de profundidad 8, pero se cree que fue por los pocos turnos jugados) se llegó a un loop infinito.

Se estima que el loop infinito sucede debido a que llega un punto en que ambos jugadores juegan defensivo, y como no se implementó la restricción de las 3 jugadas seguidas, ninguna deja de jugar defensivo.

Baxter, o más bien, su algoritmo de búsqueda, jugó contra personas reales usando una profundidad igual a 4. En total se realizaron 8 juegos contra 5 personas con distintos niveles (malo, decente, bueno, según sus propios comentarios) de desempeño en el ajedrez (Julio, Carlos, Jorge, Gatica, Jonathan). Los resultados se ven en la siguiente tabla:

Jugador (Color)	Turnos jugados	Ganador
Julio (Blanco)	34	Baxter
Julio (Negro)	31	Baxter
Carlos (Blanco)	35	Baxter
Jorge (Blanco)	44	Baxter
Jorge (Negro)	23	Baxter
Gatica (Blanco)	38	Empate
Gatica (Negro)	32	Gatica
Jonathan (Blanco)	14	Jonathan

Tabla 2: Resultados de Baxter contra personas

Los resultados finales de este algoritmo fueron bastante satisfactorios, se pensó que el robot iba a perder siempre, pero en las muestras de la Tabla 2 se demostró lo contrario. Si bien la inteligencia artificial implementada es lenta (puede tomar hasta 30 segundos aproximados en pensar una jugada con profundidad 4), tiene un buen nivel de dificultad para principiantes y jugadores amateurs.

4.2. RECONOCIMIENTO DEL TABLERO Y SUS PIEZAS

La efectividad del reconocedor de círculos se evaluó de dos maneras. La primera consiste en ver cuántos círculos reconoce, para ello, se agregaron tres contadores distintos, uno para 64 círculos, uno para 63 círculos y uno para 62 círculos. Esto se realizó para poder medir con efectividad la precisión en la cantidad de círculos encontrados. Si faltan uno o dos círculos, está dentro del margen esperado, ya que se toman en consideración los factores mencionados anteriormente en la sección 3.2 que pueden influir en el reconocimiento de los círculos.

Para realizar estas pruebas, se corrió el algoritmo 27 veces. Se tomaron en consideración el peor y mejor resultado para cada caso a estudiar (62, 63, 64 círculos), la cantidad de círculos totales durante las 100 iteraciones, donde el máximo son 2700 círculos. Por último, se midió el promedio de círculos detectados en cada caso. En la Tabla 3 se aprecian estos resultados.

Cantidad	Peor Resultado	Mejor Resultado	Círculos Totales	Promedio (Círculos totales / 27)
64	68	100	2246	83.19
≥63	88	100	2620	97.04
≥62	94	100	2690	99.63

Tabla 3: Efectividad del reconocimiento de los círculos

Donde:

- Cantidad: Indica la cantidad de círculos reconocidos
- Peor resultado: Muestra la cantidad más baja de veces en la que reconoció la cantidad de círculos de la columna anterior en 100 iteraciones.
- Mejor resultado: Muestra la cantidad más alta de veces en la que reconoció la cantidad de círculos de la primera columna en 100 iteraciones.
- Círculos totales: La suma de todos los círculos encontrados en cada iteración
- Promedio: El promedio de círculos encontrados por iteración

Los resultados fueron bastante satisfactorios, si bien se puede mejorar, el objetivo de este experimento era que, por lo menos una vez de las 100 iteraciones encontrase todos los círculos de manera consistente. Si se encontraban los 64 círculos al menos una vez en cada iteración sin fallar, se podía trabajar usando las coordenadas de esos círculos encontrados.

El segundo criterio a evaluar fue el ver qué tan preciso reconocía el centro de los círculos. Si bien en la mayoría de los casos no encontraba los centros de los círculos con precisión exacta, los resultados variaban tan poco de los centros reales que se consideraban como resultados positivos (Ver Figura 19). De todas maneras, también se encontraron múltiples veces círculos que no satisfacían con las necesidades del experimento. Se encontraron varios círculos descentrados como muestra la Figura 20.

Figura 19: Resultados positivos al identificar centros.

Figura 20: Resultados no satisfactorios de 3.2. Centros desviados.

Los resultados finales del reconocimiento del tablero y sus piezas fueron bastante satisfactorios, si bien tiene problemas, como se demostró en la Tabla 3 y en la Figura 20, en general funciona como se esperaba.

4.3 COMPORTAMIENTO DE BAXTER

Esta fue la parte más complicada de analizar, ya que como los brazos de Baxter carecen de precisión fina, se dificulta la predicción exacta de sus movimientos. Un estudio visual acompañado de cifras relativamente correctas permitió tomar las decisiones para evaluar el desempeño de Baxter.

Si bien en general Baxter se mueve como debería, existen ocasiones, más vistas en las piezas negras, en las que el brazo de Baxter rota de manera no deseada, golpeando otras piezas en el camino o moviendo el tablero. A diferencia de lo visto en la Figura 8 donde los brazos de Baxter están en una posición correcta, existían casos en los que, debido a que no se le especifica el recorrido entre origen y destino, Baxter llegaba con sus cámaras a la posición deseada, pero realizando movimientos no deseados y que perjudicaban el progreso del método implementado. (Ver Figura 21).

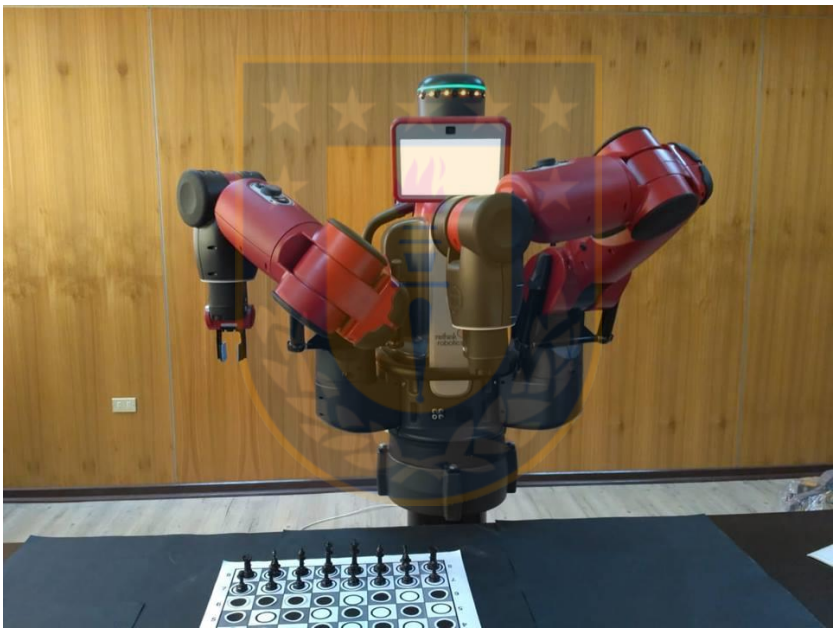

Figura 21: Baxter con el “codo” invertido (con respecto a la Figura 8 que es su posición deseable)

La precisión con la que dejaba las piezas era, en ocasiones, imperfecta. Los problemas de precisión fina brazo robótico impedían que las piezas siempre se dejasen en el lugar exacto en el que se deseaba como se ve en la Figura 22.

Figura 22: A la izquierda, caballo dejado con imprecisión. A la derecha, caballo movido con precisión satisfactoria

La Figura 23 muestra a Baxter moviendo una pieza de ajedrez satisfactoriamente.

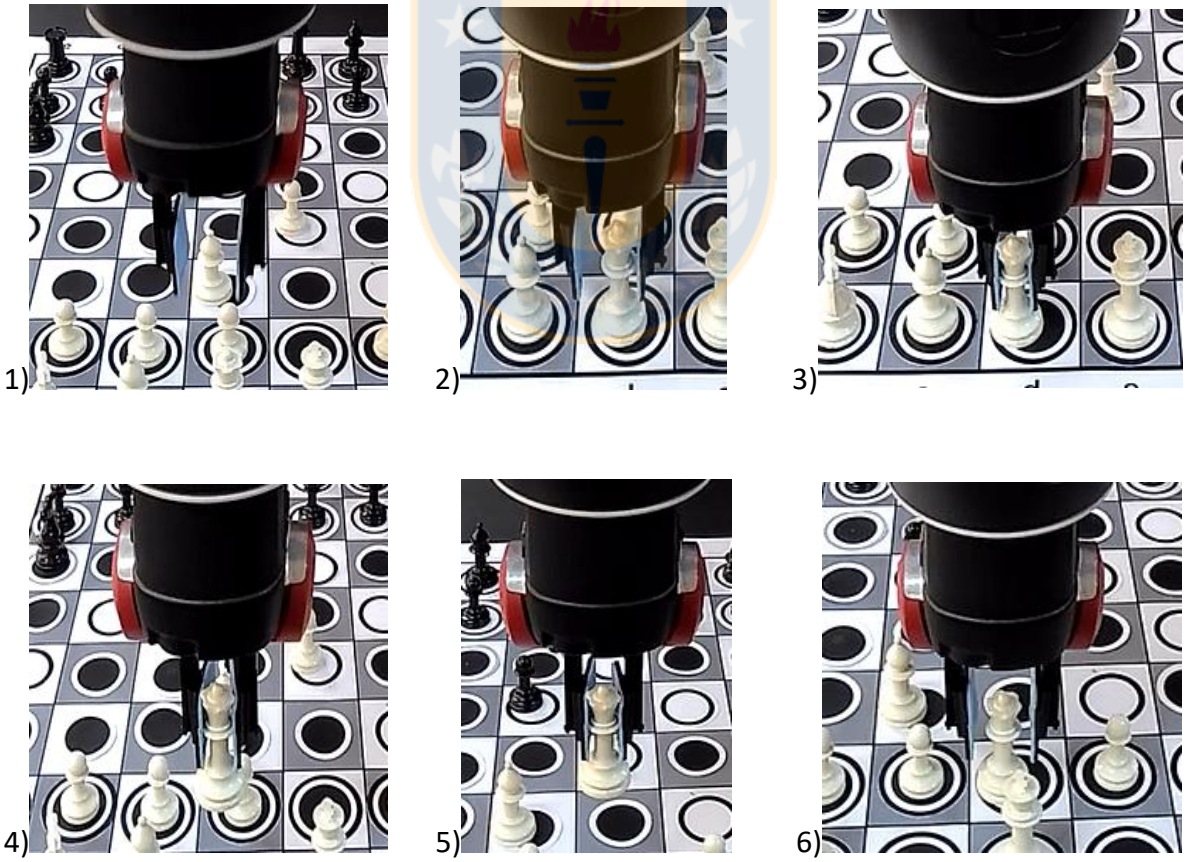

Figura 23: Baxter moviendo una reina (d1e2). 1) Arriba de la reina. 2) En la reina. 3)Toma a la reina. 4) Levanta a la reina. 5) Sobre el destino. 6) Baja y la suelta

4.4 RESULTADO FINAL

Los resultados finales son satisfactorios y es lo que se esperaba lograr, si bien hay cosas que se pueden mejorar, como el juego contra él mismo, el robot Baxter ahora juega ajedrez a un nivel mejor del esperado contra personas y con menos errores de los que se pensaron que iban a ocurrir.

Por el lado del reconocimiento de imágenes, este aspecto también resultó ser satisfactorio. La implementación de HoughCircles para reconocer las piezas y el tablero obtuvo muy buenos resultados como se pudo ver en la Tabla 3 de la sección 4.2.

En la parte de robótica, los resultados fueron los esperados, el robot puede mover las piezas como corresponde y ejecutar jugadas como: capturar una pieza, enroque y primer paso.

Debido a ciertas limitaciones, los objetivos se lograron, pero no de la manera deseada al momento de diseñar el proyecto. Originalmente, no se planeaba utilizar círculos para reconocer las piezas. El modelo inicial consistía en reconocerlas tal y como son de manera natural. Se pensó inicialmente hacerlo mediante el entrenamiento de una red neuronal, pero las piezas vistas desde arriba son tan parecidas, que conseguir los datos para su entrenamiento, entrenar la red y ponerla en uso, tardaría mucho tiempo. Además, no se podía saber si funcionaría o no.

Para el tablero ocurrió un caso similar, se esperaba reconocer el tablero como un objeto cuadrado con casillas en su interior, de manera que se pudiese saber, sin la necesidad de guardar coordenadas iniciales, el estado de cada casilla, es decir, si estaba vacía o si tenía una pieza. Este método serviría para saber qué pieza se movió y su destino. Estos objetivos debieron ser reemplazados por el uso de HoughCircles y el guardado de coordenadas. Tomando esto en consideración, el resultado final como un todo fue realmente lo esperado. Baxter juega ajedrez de manera semi-autónoma contra sí mismo o contra personas.

5. CONCLUSIONES

En este proyecto de memoria de título se propuso que el robot Baxter jugase ajedrez utilizando procesamiento digital de imágenes e inteligencia artificial para lograrlo. A pesar de los problemas encontrados durante la investigación, se logró el objetivo satisfactoriamente. Como se pudo ver en la sección de resultados (sección 4), los resultados fueron los esperados.

De este proyecto se aprendieron técnicas y herramientas para proyectos futuros, además, puede ser utilizado como base para proyectos similares, como hacer que juegue otros juegos. Además, como se trabaja con áreas separadas (algoritmos de juegos, robótica y reconocimiento de imágenes), se reemplazan sólo una de las implementaciones de estas sin necesidad de generar cambios mayores en las otras. Por ejemplo, si se realiza un proyecto para que Baxter juegue damas, sólo se debería cambiar el algoritmo del juego, el resto recibiría pequeñas correcciones como el tamaño del tablero o las correcciones manuales hechas al movimiento de los brazos de Baxter.

Este experimento dio a notar la importancia de tener un ambiente de trabajo en las condiciones óptimas tanto para el robot Baxter como para su reconocimiento de imágenes.

Comparando este proyecto de memoria de título con los otros proyectos de Baxter jugando ajedrez [40] y [42], este trabajo presentaba un desafío mayor debido a las limitaciones y la complejidad agregada del problema.

Más allá de ser un proyecto de memoria de título de la carrera de ingeniería civil informática que demuestre lo aprendido durante la misma, este proyecto busca levantar el interés en los temas de robótica e inteligencia artificial dentro de la región, atrayendo más estudiantes a la casa de estudios, generando fondos para este campo que pueden ser gubernamentales o fondos propios de la universidad, posteriormente, creando más posibilidades de estudiar dentro de la región, ya sea con cursos, asignaturas, magíster, o doctorados.

5.1. TRABAJO FUTURO

5.1.1. CONSEGUIR UN AMBIENTE DE TRABAJO ÓPTIMO

Como se explicó en la sección 3.1, se necesitan ambientes de trabajo adecuados para trabajar con reconocimiento de imágenes, un ambiente lo más estático posible, es decir, que no cambie mucho y no se vea afectado sobremanera por motivos ajenos al laboratorio o experimento (nubes, hora del día, etc.)

5.1.2 MEJORAR LA PRECISIÓN

Se debe mejorar la precisión de los siguientes elementos en orden para que funcione de la mejor manera posible:

1) Controlar de mejor manera los movimientos finos de Baxter
2) Centrado de las cámaras
3) Reconocimiento de los círculos
4) Transformaciones de coordenadas
5) Movimiento de Baxter

5.1.3 MEJORAR EL ALGORITMO DE AJEDREZ

Para mejorar el ajedrez, se deben optimizar o agregar los siguientes aspectos:

1) Mejorar los tiempos de cálculo de los movimientos
2) Mejorar los movimientos calculados
3) Ofrecer distintos niveles de dificultad
4) Agregar restricciones que existen en el ajedrez y que no están implementadas
    a. 50 turnos
    b. 3 jugadas iguales

5.1.4 RENOVAR EL RECONOCIMIENTO DE IMÁGENES

Reconocer el tablero, casillas y piezas sin la necesidad de los círculos.

1) Hacer esto de manera constante, de manera que si se mueve el tablero Baxter lo sepa
2) Prescindir del computador para jugar tanto para el humano como para Baxter, para esto, el humano deberá jugar como si estuviese jugando contra otra persona y Baxter deberá ser capaz de reconocer cuándo se hizo una jugada y cuál fue. Esto se puede lograr con un reconocimiento de imágenes óptimo.

6. REFERENCIAS

[1] Poole, Mackworth & Goebel, *Computational Intelligence and Knowledge*, Chapter 1, (1998)

[2] Muneer-Ul-Haque, Mohammad S. Raunak, M Asifur Rahim, Tashfin Delwar, and Abul L Haque, *Artificial Intelligence Approach of Optimal Route Selection in Telematics* (January 2004)

[3] Jeff Orkin, *Three States and a Plan: The A.I. of F.E.A.R* (2006)

[4] AN Ramesh, C Kambhampati, JRT Monson, PJ Drew, *Artificial intelligence in medicine*, page 334-336 (2004)

[5] Thompson, T. (2015). *Why Study AI in Games?*. Retrieved from https://aiandgames.com

[6] Georgios N. Yannakakis and Julian Togelius, *Artificial Intelligence and Games* (2018)

[7] (2015, May 13). *Why AI researches like video games.* Retrieved from https://www.economist.com/

[8] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel & Demis Hassabis, *Mastering the game of Go with deep neural networks and tree search* (2018)

[9] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Kuttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, *StarCraft II: A New Challenge for Reinforcement Learning* (2017)

[10] Goodhill, G. (2017, June 5). *No more playing games: AlphaGo AI to tackle some real world challenges.* Retrieved from https://theconversation.com

[11] Knight, W. (2016, September 12). *Self-Driving Cars Can Learn a Lot by Playing Grand Theft Auto.* Retrieved from https://www.technologyreview.com

[12] Maureen R. Galan, *Introduction to Robotics* (2017)

[13] (2017, December 19). *Food Automation: Processing and Packaging Becomes Highly Automated*. Retrieved from https://www.robotics.org

[14] Thilmany, J. (2017, October) *Robotic Co-Workers Do the Heavy Lifting.* Retrieved from https://www.asme.org

[15] Killea, F. (2016, July). *Elektro the Robot.* Retrieved from https://www.ohiomagazine.com

[16] Anderson, M.R. (2017, March 17). *After 75 years, Isaac Asimov's Three Laws of Robotics need updating.* Retrieved from https://theconversation.com

[17] (2008, July 22). *Grey Walter and his tortoises.* Retrieved from http://www.bristol.ac.uk

[18] Anyoha, R. (2017, August 28). *The History of Artificial Intelligence.* Retrieved from http://sitn.hms.harvard.edu

[19] Hoggett, R. (2015, October 13).*1982 – RB5X the Intelligent Robot – Joseph Bosworth (American).* Retrieved from http://cyberneticzoo.com

[20] DO Won, BD Kim, HJ Kim, TS Eom, KR Müller, SW Lee, *Curly: An AI-based Curling Robot Successfully Competing in the Olympic Discipline of Curling* (2018)

[21] (2018, April 4). *Taller de Robótica para Principiantes | Centro Creación Concepción.* Retrieved from http://www.concepcioncultural.cl

[22] Crowe, S. (2018, February 8). *10 Most Automated Countries in the World.* Retrieved from https://www.therobotreport.com

[23] Guizzo, E., Ackerman E. (2012, September 18) *How Rethink Robotics Built Its New Baxter Robot Worker.* Retrieved from https://spectrum.ieee.org

[24] Stuart Russell, Peter Norvig, *Artificial Intelligence A Modern Approach (3rd edition)*, page 161-162 (2010)

[25] Murray Campbell, A. Joseph Hoane Jr. Feng-hsiung Hsu, *Deep Blue* (2002)

[26] Ray, C. (2014, March 8). *How Stockfish Works: An Evaluation of the Databases Behind the Top Open-Source Chess Engine.* Retrieved from http://rin.io

[27] François, J. (2014, September 22). *Comment programmer un jeu d'échecs?.* Retrieved from https://fr.jeffprod.com

[28] Hoggett,R. (2009, October 18). *1928 – Eric Robot – Capt. Richards & A.H. Reffell (English)*. Retrieved from http://cyberneticzoo.com

[29] Hoggett,R. (2010, January 5). *1967 – "Shakey" – Charles Rosen, Nils Nilsson, Bertram Raphael et al (American)*. Retrieved from http://cyberneticzoo.com

[30] Yoshiaki Sakagami, Ryujin Watanabe, Chaiki Aoyama, Shinichi Matsunaga, Nobuo Higaki, Kikuo Fujimura. *The intelligent ASIMO: System overview and integration*, page 2-5 (February 2002)

[31] Mraz,S. (2012, October 16). *Baxter: A robot for the rest of us.* Retrieved from https://www.machinedesign.com

[32] Owen-Hill, A. (2016, Julio 7). *Robot Vision vs Computer Vision: What's the Difference?*. Retrieved from https://robotiq.com

[33] Yaser Sheikh, Omar Javed, Takeo Kanade, *Background Subtraction for Freely Moving Cameras*, page 1-2 (2009)

[34] Sk. Zeeshan Murshed, Anwesha Dutta, Sombuddha Chatterjee, *Controlling an embedded robot through image processing based object tracking using MATLAB* (November 2016)

[35] Pugh, V. (2014, July 29). *Worked Example Visual Servoing.* Retrieved from http://sdk.rethinkrobotics.com

[36] John Canny, *A Computational Approach to Edge Detection* (November 1986)

[37] H.K. Yuen, J. Princen, J. Illingworth and J. Kittler, *A Comparative Study of Hough Transform Methods for Circle Finding* (1990)

[38] Bradski, G. Dr. Dobb's Journal of Software Tools, *The OpenCV Library* (2000)

[39] Benoit, B. (2016, January 26). *Robots in the classroom: teaching the next generation with Baxter*. Retrieved from https://www.rethinkrobotics.com

[40] Schack, S., Friedman, S. (2014). *UC Berkeley EE125 Fall 2014 Project*, Baxter Chess. Retrieved from http://spencerschack.com

[41] Klein, M. (2017, December 6). *Google's AlphaZero Destroys Stockfish In 100-Game Match*. Retrieved from https://www.chess.com

[42] (2015, July 29). *Baxter Research Robot – Chess Game Demonstration For Two Remote Players.* Retrieved from http://vteams.com

[43] Morgan Quigley, Brian Gerkey , Ken Conley , Josh Faust , Tully Foote , Jeremy Leibs , Eric Berger, Rob Wheeler, Andrew Ng, *ROS: an open-source Robot Operating System* (2009)

[44] AllUltima, (2007, December 21). *An illustration of the principle variation of a minimax game tree.* Retrieved from https://en.wikipedia.org

7. ANEXOS

7.1. HOUGH CIRCLES

Para definir un círculo se utilizan tres parámetros, la coordenada en X del centro, la coordenada en Y del centro y el radio del círculo.

Una vez definido lo que es un círculo, se carga la imagen de la cual se quieran obtener los círculos.

Se convierte esta imagen a escala de grises para poder aplicar un GaussianBlur, que permite reducir el ruido al generar una imagen un poco más borrosa.

Finalmente se aplica la transformación Hough Circle

```
HoughCircles( src_gray, circles, CV_HOUGH_GRADIENT, 1, src_gray.rows/8, 200, 100, 0, 0 )
```

Donde:

- Parámetro 1: La imagen en escala de grises
- Parámetro 2: La definición de los círculos (x, y, radio)
- Parámetro 3: Método de detección de círculos, actualmente CV_HOUGH_GRADIENT es el único en OpenCV, descrito en [33]
- Parámetro 4: Razón de resolución invertida
- Parámetro 5: Distancia mínima entre los centros de los círculos.
- Parámetro 6: Límite superior del Canny, es el parámetro 1 de CV_HOUGH_GRADIENT
- Parámetro 7: Segundo parámetro de CV_HOUGH_GRADIENT. Límite para la detección de los centros
- Parámetro 8: Radio mínimo de detección (círculo más chico detectable)
- Parámetro 9: Radio máximo de detección (círculo más grande detectable)