Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Máster en Electrónica Industrial y Automática

# MÁSTER EN ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

## ESCUELA DE INGENIERÍAS INDUSTRIALES

## UNIVERSIDAD DE VALLADOLID

## TRABAJO FIN DE MÁSTER

# DISEÑO DE UN SISTEMA ROBÓTICO EDUCATIVO PARA JUGAR AL AJEDREZ CON ROBOTS INDUSTRIALES

Autor: D. Carlos Jiménez Jiménez
Tutor: D. Alberto Herreros López
Departamento de Ingeniería de Sistemas y Automática

Valladolid, 1 de julio de 2019

Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Máster en Electrónica Industrial y Automática

# MÁSTER EN ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

## ESCUELA DE INGENIERÍAS INDUSTRIALES

## UNIVERSIDAD DE VALLADOLID

## TRABAJO FIN DE MÁSTER

# DISEÑO DE UN SISTEMA ROBÓTICO EDUCATIVO PARA JUGAR AL AJEDREZ CON ROBOTS INDUSTRIALES

Autor: D. Carlos Jiménez Jiménez
Tutor: D. Alberto Herreros López
Departamento de Ingeniería de Sistemas y Automática

Valladolid, 1 de julio de 2019

## Resumen

El proyecto consiste en el desarrollo de un sistema robótico educativo capaz de jugar al ajedrez con un robot industrial. Los movimientos de las piezas en el tablero los realiza un robot ABB-IRB-120. El robot está comunicado, empleando el protocolo TCP/IP, con un PC en el cual se utiliza Matlab como software central. Este se comunica con motores de ajedrez que deciden la jugada del ordenador, y con un entorno gráfico para que el usuario pueda definir su jugada y visualizar el tablero. Asimismo, tanto las piezas de ajedrez como la garra del robot han sido diseñadas en el proyecto para optimizar los movimientos.



## Summary

The proyect consists in a robotic system development that is able to play chess using an industrial robot. The ABB-IRB-120 made the movements of the pieces on the chessboard. Using TCP/IP protocol, the robot communicates with the PC where resides the main software, Matlab. This also communicates with chess engines that decide the play of the computer and with graphic environment so that the user can define his play and visualize the chessboard. Likewise, both the chess pieces and the tool robot have been designed in the project to optimize the movements.

# Agradecimientos

Agradecer, principalmente, el apoyo y la ayuda constante de Alberto Herreros. El entusiasmo que ambos mostramos por la idea de este proyecto nos ha llevado a desarrollarlo con pasión, esfuerzo y gusto. La ayuda mutua y la puesta en común de nuestros conocimientos ha dado como resultado el trabajo que se plasma en este documento.

Mostrar también el agradecimiento a mi familia, que con su esfuerzo y motivación constante han hecho posible que llegue hasta aquí.

# ÍNDICE

# CAPÍTULO 1. INTRODUCCIÓN

## 1.1 Justificación del proyecto

Este proyecto pretende desarrollar un sistema robótico plenamente operativo que pueda ser usado con fines docentes y de investigación. En estos campos permitirá, por ejemplo, el desarrollo de nuevas estrategias de control, el perfeccionamiento de algoritmos, la comparación de motores de ajedrez, la distribución e integración del software, el análisis de trayectorias y el diseño y fabricación de modelos 3D.

Las posibilidades que ofrece la plataforma en su conjunto son muy variadas, dando lugar a un sistema muy abierto que permitirá incorporar y probar nuevas tecnologías al robot de una forma muy dinámica y sencilla.

Además, la robótica es una de las áreas de mayor crecimiento en la industria ya que, el diseño y la implementación de procesos cada vez más automatizados, es lo que ha permitido fabricar en cantidades inimaginables, a precios muy competitivos y aumentando la calidad y seguridad. Un robot industrial siempre está conectado a otros dispositivos, como los autómatas (PLC's), a los que da y de donde recibe información para que el sistema robótico general funcione. En este proyecto de tipo educativo se trata de crear un sistema robotizado con el que se puede jugar al ajedrez usando muchos dispositivos diferentes que deben ser comunicados entre sí.

En cuanto al ámbito educativo, este proyecto abre la puerta a una amplia gama de oportunidades. Los alumnos podrían aprender sobre esta nueva plataforma e incluso, introducir avances, mejoras o alternativas.

En el campo de la computación y la inteligencia artificial, facilitará a los desarrolladores progresar en los algoritmos de ajedrez, depurar su código sobre partidas reales, enfrentar a diferentes motores y analizar sus jugadas y funcionamiento.

## 1.2 Alcance del proyecto

El proyecto comprende el desarrollo un sistema robótico capaz de jugar al ajedrez utilizando un robot de tipo industrial que debe emplear motores de análisis del juego junto con una interfaz gráfica hombre-máquina. Será preciso la implementación de un sistema de comunicación entre los diferentes módulos del proyecto. Se debe definir la estructura de las piezas, así como la garra del robot con el fin de optimizar los movimientos y asegurar la precisión. Esta etapa comprende el diseño asistido por ordenador y su posterior fabricación.

## 1.3 Antecedentes

En 2014, San José Miguel Ángel Mato [1] llevó a cabo un proyecto final de carrera que consistía en el desarrollo de una interfaz gráfica desde Matlab que, empleando la librería "Robotic ToolBox", movía un robot ABB IRB 120 simulado y de forma simultánea el robot real mediante comunicación OPC.

Posteriormente, entre 2015 y 2016, Juan Antonio Ávila Herrero [2] y Álvaro Galindo de los Santos [3], desarrollaron sus proyectos de manera simultánea dando lugar a la estación robótica con la que se cuenta actualmente en la escuela. Por un lado, Juan Antonio Ávila Herrero, se encargó del modelado de la célula robótica utilizando el programa "RobotStudio" y, por otro lado, Álvaro Galindo de los Santos, centró su proyecto en el control remoto del robot mediante tecnología WIFI. Ambos trabajaron con fines docentes diseñando diferentes escenarios de educación. Entre estos, cabe destacar el juego de las tres en raya, con el cual se podía interactuar de forma remota con una Tablet a través de la tecnología WIFI.

Por otra parte, profesores del Departamento de Automática de la Escuela de Ingenierías Industriales han desarrollado también algunos proyectos educativos sobre esta plataforma. Es el caso de Alberto Herreros López, que con su programación dotó al robot de la capacidad de escribir sobre un papel el texto leído de un fichero. Se podían emplear distintos tipos de letra y distintos planos de escritura.

## 1.4 Objetivos

Partiendo de estas líneas de trabajo, se ha desarrollado un entorno robótico para que un alumno aprenda a usar diferentes dispositivos con un fin común. Por un lado, está el robot con su programación; por otro, el HMI con su programación; y por otro, los motores de ajedrez. El conjunto y comunicación de estos elementos dan lugar a un sistema general con el que se puede jugar al ajedrez con un robot industrial.

Para lograr este hito, se han marcado una serie de objetivos intermedios, que son:

- Implementar un código que sea capaz de gestionar todos los aspectos que influyen en una partida de ajedrez, tales como, el modo de juego, la colocación inicial de las piezas en el tablero, la legalidad de los movimientos, el sistema de turnos, las amenazas al rey, las capturas de piezas, etc.

- Desarrollar una aplicación que sea capaz de comunicarse con los motores de ajedrez empleando el protocolo UCI.

- Realizar un programa en RAPID con trayectorias que gestionen el movimiento de las piezas.

- Diseñar y programar una aplicación visual que se comporte como la interfaz hombre-máquina del proyecto.

- Implementar los sistemas de comunicación entre los diferentes módulos de los que se compone el sistema.

- Llevar a cabo el diseño asistido por ordenador (CAD) de las piezas de ajedrez y la garra del robot y su posterior fabricación.

## 1.5 Estructura de la memoria

La memoria se estructura en 8 capítulos bien diferenciados, como son:

- Capítulo 1. Introducción: Aquí se incluyen la justificación del proyecto, el alcance de este, los antecedentes y los objetivos. Pretende ofrecer una visión global del proyecto.

- Capítulo 2. Estado del arte: Hace referencia a la base teórica sobre la que se sustenta el proyecto. Es frecuente que el segundo capítulo de una tesis o un proyecto en ingenierías se denomine "Estado del arte" donde se hace un repaso de las técnicas empleadas.

- Capítulo 3. Diseño Previo: En este capítulo se lleva a cabo un análisis de soluciones, se define la estructura del software, se determina el sistema de comunicaciones y, por último, se establecen las premisas del diseño 3D.

- Capítulo 4. Desarrollo del proyecto: Se corresponde con la etapa de ejecución. Se implementa lo diseñado en la etapa anterior y se detalla cada una de las partes del proyecto.

- Capítulo 5. Resultados: En este capítulo se llevan a cabo todas las pruebas y experimentos que hayan sido necesarios para ajustar y asegurar el correcto funcionamiento de todas las partes. El objetivo final de este capítulo es determinar que se han alcanzado los objetivos iniciales.

- Capítulo 6. Manual de Usuario. Pequeña guía de información para que cualquier usuario pueda ser capaz de ejecutar las diferentes funcionalidades.

- Capítulo 7. Estudio económico. Informe detallado del coste que ha supuesto o supondrá la realización de este proyecto.

- Capítulo 8. Conclusiones: En este último capítulo se extraen las conclusiones obtenidas tras la realización del proyecto. Además, se introducirán pequeñas líneas de mejora que puedan ser adoptadas en proyectos futuros.

Además, se incluyen la Bibliografía y los Anexos.

# CAPÍTULO 2. ESTADO DEL ARTE

## 2.1 El ajedrez

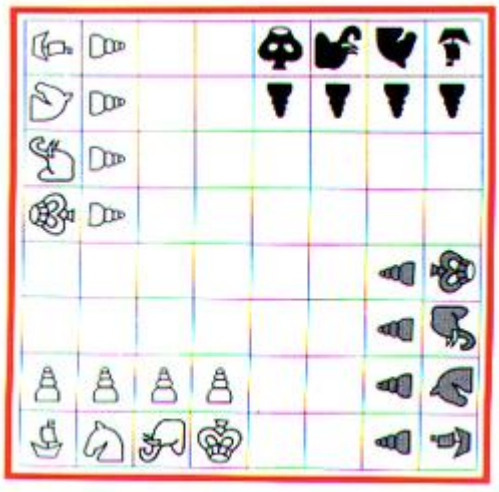
### *El juego*

El ajedrez es un juego entre dos personas, cada una de las cuales dispone de 16 piezas móviles que se colocan sobre un tablero dividido en 64 casillas o escaques. En su versión de competición, está considerado como un deporte.

Se juega sobre un tablero cuadriculado de 8×8 casillas (llamadas escaques), alternadas en colores blanco y negro, que constituyen las 64 posibles posiciones de las piezas para el desarrollo del juego. Al principio del juego cada jugador tiene dieciséis piezas: un rey, una dama, dos alfiles, dos caballos, dos torres y ocho peones. Se trata de un juego de estrategia en el que el objetivo es «derrocar» al rey del oponente. Esto se hace amenazando la casilla que ocupa el rey con alguna de las piezas propias sin que el otro jugador pueda proteger a su rey interponiendo una pieza entre su rey y la pieza que lo amenaza, mover su rey a un escaque libre o capturar a la pieza que lo está amenazando, lo que trae como resultado es el jaque mate y el fin de la partida. [4]

### *Historia*

En la historia del ajedrez el origen del juego sigue siendo controvertido, pero la versión más aceptada es que el ajedrez fue inventado en Asia, probablemente en la India, con el nombre de "*chaturanga*", y desde allí se extendió a China, Rusia, Persia y Europa, donde se estableció la normativa vigente. Sin embargo, investigaciones recientes indican un posible origen chino, en la región entre Uzbekistán y la antigua Persia, que se podría remontar hasta el siglo III a. C.



Tras la conquista de Persia por los árabes, estos asimilaron el juego y lo difundieron en Occidente, llevándolo al norte de África y Europa, e incluso la actual España e Italia alrededor del siglo X, desde donde se extendió al resto del continente llegando a la región de Escandinavia e Islandia.

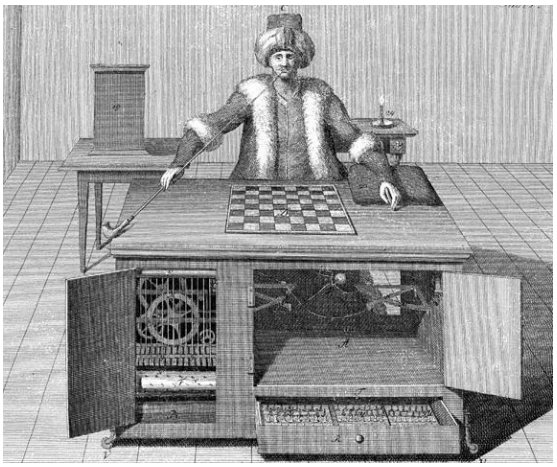
Hacia finales del siglo décimo quinto comenzaron a cambiar las reglas de manera decisiva. Se regularían los movimientos del peón, el alfil y la dama. El peón podría ahora dar dos pasos en el primer movimiento (hasta entonces uno solo); el alfil, a cualquier distancia (hasta entonces solo saltaba dos escaques); se introdujo la regla conocida como *"en passant"* ('al paso'), que permite capturar el peón que sigue su marcha y no come la ficha que se le ha ofrecido por una determinada estrategia; el revolucionario concepto del enroque; y la dama, a discreción en cualquiera de las ocho direcciones (antes avanzaba únicamente un escaque en dirección diagonal); con lo que de repente pasó de ser una figura relativamente débil a convertirse en la más importante del tablero y a dotar de gran espectacularidad al juego.

Mediante estos ajustes se cambió el juego completamente. Se trata del nacimiento del ajedrez moderno. El nuevo juego exigía distintas tácticas y aperturas. El ejercicio ganó en velocidad y, al mismo tiempo, en popularidad. Estas novedades se introdujeron probablemente en Valencia entre los años 1470 y 1490.

Las partidas comenzaron a ser registrados con mayor frecuencia y se han publicado más estudios teóricos. En el siglo XVIII se fundaron los primeros clubes para la práctica de ajedrez y federaciones deportivas en Europa, y debido a la gran cantidad de pequeños torneos que ocurren por todo el continente, en 1851 se celebró el primer torneo internacional en Londres. La popularidad de las competiciones internacionales ha llevado a la creación del título de campeón del mundo, ganado por Wilhelm Steinitz en 1886, y, en 1924, se fundó la Federación Internacional de Ajedrez (FIDE), en París, que organiza la primera Olimpiada de Ajedrez y el mundial femenino. [5] [6] [7]

## 2.2 Ajedrez por ordenador. El hombre contra la máquina.

Ya en el siglo XVIII comenzó a difundirse la idea de crear una máquina que jugara al ajedrez contra una persona. En 1769 se hizo famoso un jugador de ajedrez autómata llamado "El Turco". Tenía forma de una cabina de madera de 1.20 cm x 60 cm x 90 cm, con un maniquí vestido con túnica y turbante sentado sobre él. Fue construido y revelado por Wolfgang von Kempelen. "El Turco" era capaz de jugar una partida de ajedrez contra un jugador humano a un alto nivel.



[8]

No obstante, se cree que la cabina era una ilusión óptica bien planteada que permitía a un maestro del ajedrez de baja estatura esconderse en su interior y operar el maniquí. Los ojos de "El Turco" reflejaban mediante espejos las posiciones de las piezas del tablero para informar al maestro de ajedrez escondido. Realmente cualquiera de los dos podía ganar, pero el maestro escondido contaba con la ventaja de poder asustar a su oponente haciéndole creer que el Turco en realidad era autómata, lo cual ponía nervioso al contrincante, impidiéndole desempeñar sus conocimientos sobre el juego con totalidad. [9]

Más tarde, el español Leonardo Torres Quevedo construyó, en 1912, un autómata capaz de jugar al ajedrez llamado *"El Ajedrecista".* Hizo su debut durante la Feria de París de 1914, donde generó gran expectativa. Utilizando electroimanes bajo el tablero de ajedrez, jugaba automáticamente un final de rey y torre contra el rey de un oponente humano. No jugaba de manera muy precisa y no siempre llegaba al mate en el número mínimo de movimientos, a causa del algoritmo simple que evaluaba las posiciones, pero lograba la victoria en todas las ocasiones. [10]

Después de estos dos sucesos, el tema del ajedrez mecánico no se volvió a mencionar y cayó en el olvido hasta la aparición de las primeras computadoras en la década de los 50. Desde entonces, los aficionados del ajedrez y de la informática han construido máquinas y programas que juegan al ajedrez.

### *Causas*

El origen del ajedrez computarizado fue motivado por varias causas:

- El entretenimiento propio de los ajedrecistas, pudiendo permitir que los jugadores practiquen y se diviertan cuando no hay ningún oponente disponible.
- Emplearlo como una herramienta de análisis.
- Con relación a lo anterior, como soporte en investigación y para profundizar el conocimiento humano.

- Competición entre computadoras de ajedrez, enfrentando diferentes maquinas, algoritmos y programadores.

*Enfrentamientos*

Todo comenzó en 1968 cuando el Maestro Internacional David Levy realizó una apuesta, afirmando que ninguna máquina capaz de jugar al ajedrez en condiciones de torneo le derrotaría en un periodo de 10 años. Ganó la apuesta en 1978, venciendo 3,5-1,5 en un combate a 5 partidas enfrentándose a "Chess 4.7" considerada por aquel entonces la computadora más fuerte. Sin embargo, reconoció que no pasaría mucho tiempo para que una máquina le superase. Un año más tarde, Levy fue derrotado por "Deep Thoght" en una partida de exhibición.

No obstante, esta máquina aún no estaba a la altura de los grandes campeones del ajedrez, como demostró Gary Kasparov en ese mismo año ganando en dos ocasiones. Finalmente, en 1996 la computadora "*Deep Blue"* de IBM, derrotó a Kasparov bajo condiciones de torneo con un ritmo de juego lento, si bien es cierto que después Kasparov consiguió vencer 3 y empatar 2, siendo el resultado final de 4-2 favorable a Kasparov.

En 1997, una versión mejorada de "*Deep Blue*" derrotó a Kasparov en un encuentro a 6 partidas por 3,5-2,5 generando el debate de si el jugador más fuerte de ajedrez en ese momento era una máquina.

A principios de los años 2000, los programas comercialmente disponibles como Junior y Fritz eran capaces de entablar partidas contra el campeón del mundo Garry Kasparov y el campeón del mundo de ajedrez clásico Vladímir Krámnik.

En 2005, Hydra, una computadora de ajedrez dedicada con hardware personalizado y sesenta y cuatro procesadores, que es capaz de calcular 40 millones de posiciones por segundo, ganadora del 14º IPCCC en 2005, ganó a Michael Adams (séptimo en las listas mundiales) con un contundente 5.5-0.5. Algunos comentaristas creían que Hydra será de manera definitiva claramente superior a los mejores jugadores humanos o si no lo será su directo sucesor

Fue en 2006 contra Deep Fritz, un software que acabó ganando 4-2 (dos derrotas y cuatro empates para Kramnik) y que desató un comentario especialmente llamativo del profesor Monty Newborn, uno de los expertos que ayudaron a organizar el encuentro original entre Kasparov y Deep Blue. Al hablar de futuros enfrentamientos hombre-máquina, este académico lo dejó claro: "the science is done". "El trabajo estaba hecho" y la ciencia se había impuesto.

Desde entonces ningún gran maestro pareció estar interesado en competir contra las máquinas. Aquello, simplemente, no tenía emoción.

## 2.3 Notación Algebraica

Se trata del sistema de anotación de partidas de ajedrez más extendido en la actualidad. Fue introducido por "Philipp Stamma de Alepo" en su libro "El noble juego del ajedrez" publicado en 1937. Desde la reforma de las leyes del ajedrez en 1997, esta notación desbancó a la notación descriptiva. [11]

Algunas de las razones por las que se utiliza esta notación son las siguientes:

- No se permite a un jugador participar en un torneo de ajedrez si no conoce las reglas básicas de anotación.
- Si hay algún problema en un juego, el árbitro debe estar al tanto de la progresión del juego.
- Los registros de notación pueden ser útiles para revisar los momentos críticos de las partidas y analizar donde se puede mejorar.
- Para jugadores profesionales, guardar partidas en las bases de datos de ajedrez puede resultar muy interesante.

[12]

Para identificar cada una de las 64 casillas se emplean dos caracteres de manera única. El primero debe ser una letra minúscula (de la "a" a la "h") y se identifica con las columnas, mientras que el segundo es un número (del "1" al "8") que indica la fila.

En cuanto a las piezas, cada una se asocia con una letra mayúscula que varía en función del idioma anotador (ver Tabla 1).

| | Rey | Dama | Torre | Alfil | Caballo | Peón |
|---|---|---|---|---|---|---|
| Alemán | **K** = König (*rey*) | **D** = Dame (*dama*) | **T** = Turm (*torre*) | **L** = Läufer (*corredor*) | **S** = Springer (*saltador*) | **(B)** = Bauer (*peón*) |
| Español | **R** = Rey | **D** = Dama | **T** = Torre | **A** = Alfil | **C** = Caballo | **(P)** = Peón |
| Esperanto | **R** = Reĝo (*rey*) | **D** = Damo (*dama*) | **T** = Turo (*torre*) | **K** = Kuriero (*mensajero*) | **Ĉ** = Ĉevalo (*caballo*) | **(P)** = Peono (*peón*) |
| Francés | **R** = Roi (*rey*) | **D** = Dame (*dama*) | **T** = Tour (*torre*) | **F** = Fou (*bufón*) | **C** = Cavalier (*caballero*) | **(P)** = Pion (*peón*) |

| | | | | | | |
|---|---|---|---|---|---|---|
| Hindi | **R** = Rājā (*rey*) | **V** = Vazīr (*visir*) | **H** = Hāthī (*elefante*) | **O** = Ūṁṭ (*camello*) | **G** = Ghoṛā (*caballo*) | **(P)** = Pyādā (*infantería*) |
| Inglés | **K** = King (*rey*) | **Q** = Queen (*reina*) | **R** = Rook (*castillo*) | **B** = Bishop (*obispo*) | **N** = K**n**ight (*caballero*) | **(P)** = Pawn (*peón*) |
| Italiano | **R** = Re (*rey*) | **D** = Donna (*dama*) | **T** = Torre (*torre*) | **A** = Alfiere (*alférez*) | **C** = Cavallo (*caballo*) | **(P)** = Pedone (*peón*) |
| Neerlandés | **K** = Koning (*rey*) | **D** = Dame (*dama*) | **T** = Toren (*torre*) | **L** = Loper (*corredor*) | **P** = Paard (*caballo*) | Pion (*peón*) |
| Polaco | **K** = Król (*rey*) | **H** = Hetman (*comandante*) | **W** = Wieża (*torre*) | **G** = Goniec (*mensajero*) | **S** = Skoczek (*saltador*) | **(P)** = Pion (*peón*) |
| Ruso | **Кр** = Korol' (*rey*) | **Ф** = Ferz' (*visir*) | **Л** = Lad'ya (*barco*) | **С** = Slon (*elefante*) | **К** = Kon' (*caballo*) | **(П)** = Peshka (*peón*) |

*Tabla 1 Notación piezas*

Para identificar los movimientos basta con indicar la letra de la pieza que se mueve y la casilla de destino. Un movimiento de captura se representa con la letra "x" inmediatamente antes de la casilla destino.

En cuanto a los enroques, se representan con "0-0" si es un enroque corto (flanco del rey) o bien "0-0-0" si es un enroque largo (flanco de la dama).

Finalmente, si un movimiento provoca una situación de jaque, se añade opcionalmente el signo "+" como sufijo del movimiento anotado. Si el movimiento es jaque mate se emplea "++", o bien, "#".

### *Signos especiales*

- !: jugada buena.
- ?: jugada mala.
- !!: jugada muy buena.
- ??: jugada muy mala.
- !?: jugada interesante.
- ?!: jugada dudosa.
- =: igualdad de oportunidades para ambos jugadores.
- +/=: ligera ventaja blanca.

- =/+: ligera ventaja negra.
- +/- (o también ±): ventaja blanca.
- -/+: ventaja negra.
- +-: ventaja decisiva blanca.
- -+: ventaja decisiva negra.
- ∞: posición incierta.
- =/∞ (o también ∞=): juego compensado a pesar de diferencia de material.
- N: novedad teórica.

[13]

## 2.4 Motores de ajedrez

Uno de los hitos más revolucionarios en la historia del ajedrez ha sido, sin duda, la aparición de los motores de ajedrez conocidos como "engines" o módulos de análisis. Se trata de un programa informático (software) que se ejecuta en un ordenador el cual es capaz de jugar al ajedrez y que, además, implementa un lenguaje de comunicación que le permite intercambiar la información con otros programas.

### 2.4.1 Principios de funcionamiento

*Algoritmo minimax*

En teoría de juegos, "minimax" es un método de decisión para minimizar la pérdida máxima esperada en juegos con adversario y con información perfecta. Se trata de un algoritmo recursivo.

En otras palabras, el funcionamiento de este algoritmo puede entenderse coloquialmente como escoger el mejor movimiento para uno mismo suponiendo que tu contrincante escogerá el peor para ti Figura 1.

También existe la estrategia "maximin". Esta intenta maximizar la ganancia propia, es decir, busca que el jugador obtenga el mejor resultado, mientras que "minimax" intenta minimizar la ganancia del rival, o sea busca que el rival tenga el peor resultado.

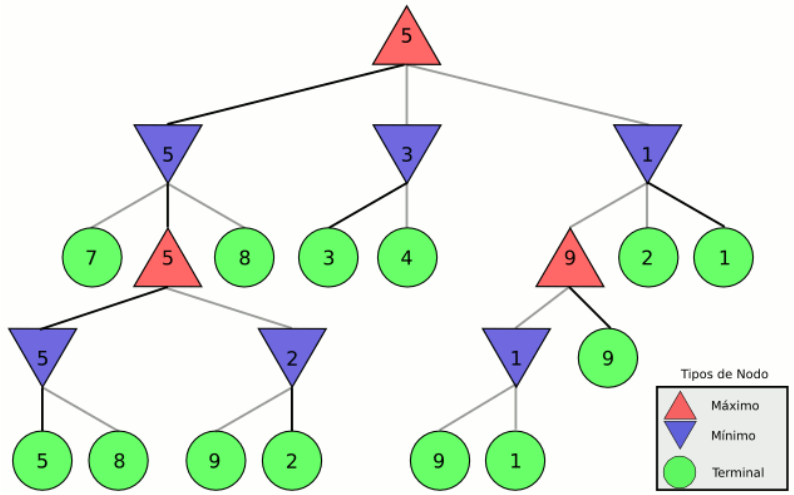



*Figura 1 Estrategia minimax*

Suele atribuirse a John von Neumann el principal mérito de la concepción del principio minimax, ya que fue él quien, en su artículo de 1928 "Sobre la teoría de los juegos de sociedad" puso las bases de la moderna teoría de juegos y probó el teorema fundamental del "minimax", por el que se demuestra que para juegos de suma cero con información perfecta entre dos competidores existe una única solución óptima.

Cabe destacar dos afirmaciones que realizó este matemático. Así explicó la noción de lo que era un juego:

*"Un juego es una* ***situación conflictiva*** *en la que uno debe tomar una decisión sabiendo que los demás también toman decisiones, y que el resultado del conflicto se determina, de algún modo, a partir de todas las decisiones realizadas."*

Además, añadió que:

*"Siempre existe una* ***forma racional de actuar*** *en juegos de dos participantes, si los intereses que los gobiernan son completamente opuestos."*

La demostración a esta afirmación es lo que se denomina teoría "minimax". [14]

### *Juego de suma cero*

En teoría de juegos no cooperativos, un juego de este tipo describe una situación en la que las ganancias o pérdidas de un participante se equilibran con exactitud con las perdidas o ganancias respetivamente de los otros participantes. El "go", el "ajedrez" y el "poker" son algunos ejemplos de juegos de suma cero.

El teorema "minimax" establece que en los juegos bipersonales de suma cero donde cada jugador conoce de antemano la estrategia de su oponente y sus consecuencias, existe una estrategia que permite a ambos jugadores minimizar la pérdida máxima esperada. Por ello, cuando se examina cada posible jugada, se

deben considerar todas las respuestas posibles del jugador adversario y la pérdida máxima que se puede acarrear.

Se debe jugar, por tanto, con la estrategia que resulte de minimizar la máxima pérdida. Esta estrategia se denominará óptima para ambos jugadores en el caso de que sus "minimaxes" sean iguales en valor absoluto y contrarios en signos. Si el valor común es cero, el juego no tendría sentido. [15]

### *Poda Alpha-beta*

El principal problema de la búsqueda "minimax" es que el número de nodos a explorar crece de manera exponencial al número de movimientos (niveles de profundidad). A nivel computacional es inviable explorar todos los nodos posibles hasta el final del juego.

La técnica "alfa-beta" surge precisamente para enfrentar este problema ya que reduce el número de nodos evaluados en un árbol de juego. Es muy utilizada en juegos entre adversarios como el ajedrez, el Go o el tres en raya.

Esta variante intenta eliminar partes grandes del árbol de búsqueda que no sean susceptibles de ser exploradas, de modo que el resultado final sea el mismo que el que devolvería el algoritmo "minimax", ya que la poda de estas ramas no influye en la decisión final (ver Figura 2).

El nombre de esta técnica proviene de los dos parámetros utilizados para realizar la poda:

- Alfa (α): es el valor de la mejor opción hasta ese momento a lo largo del camino para MAX, lo cual implica la elección del valor más alto.
- Beta (β): es el valor de la mejor opción hasta ese momento a lo largo del camino para MIN, lo cual implica la elección del valor más bajo.

La búsqueda de estos valores va actualizándose a medida que se recorre el árbol. Se lleva a cabo una poda de las ramas restantes en el momento en que el valor actual de alfa (α) sea mayor que el de beta (β). [16]

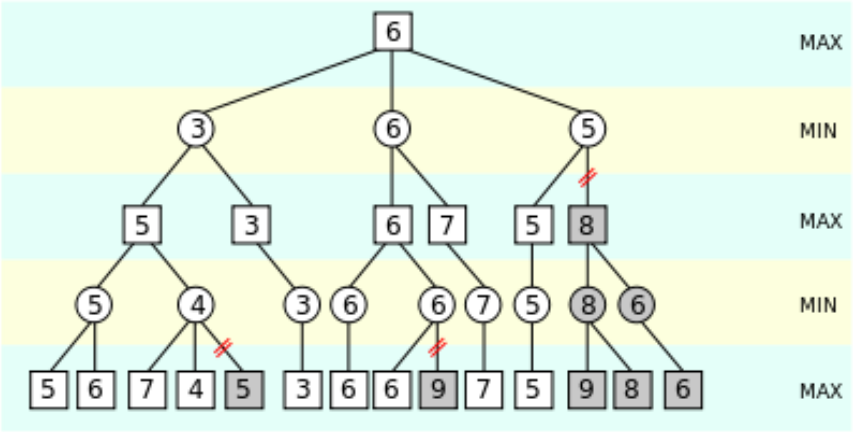



*Figura 2 Ejemplo poda alfa-beta*

*Negascout*

Negascout es una variante del algoritmo "minimax" que combina la relación matemática del "Negamax" y el uso de ventana nula.

El "Negamax" es también una variante del algoritmo "minimax" donde cada nodo toma el valor máximo de sus hijos independientemente de si el nivel es MAX o MIN, a diferencia de que en los niveles MAX se cambian de signo. Realmente, se consigue el mismo efecto ya que tomar el mayor valor, pero cambiado de signo es en realidad tomar el menor. [17]

En cuando a "Negascout" también llamado "búsqueda de la variante principal", puede ofrecer mejores rendimientos que la poda "alfa-beta" si los nodos se encuentran correctamente ordenados.

La eficacia de la poda "alfa-beta" depende del orden en el que se examinan los nodos sucesores, es decir, que el algoritmo se comportará de manera más eficiente si examinamos primero los sucesos que tengan más probabilidad de ser mejores. Si se fuese capaz de hacer esto, la técnica "alfa-beta" con "Negascout" podría mirar hacia delante aproximadamente dos veces más que el algoritmo "minimax" en la misma cantidad de tiempo.

Si en lugar de elegir el mejor suceso probable se recurre a una elección aleatoria, la cantidad de nodos examinados sería aproximadamente de $^4/_3$ más que con el algoritmo original.

En el caso del ajedrez se puede realizar una función de ordenación sencilla teniendo en cuenta primero los nodos en los que se produce una captura de ficha, después nodos en los que se producen amenazas, nodos de movimientos hacia delante y, por último, movimientos hacia detrás. Con esto conseguiríamos un resultado similar al de escoger el mejor suceso probable, es decir, el doble que con el algoritmo "minimax". [18]

### 2.4.2 Tipos de motores

Existen numerosos motores de ajedrez como son "Stockfish", "Komodo", "Fire", "Hannibal", entre otros. Todos son diferentes entre sí, por la forma en que están programados, como toman decisiones, es decir, en sus detalles más específicos. No obstante, todos comparten el principio general con el que funcionan y deben implementar el mismo protocolo de comunicación para que sean fácilmente comparables y sustituibles.

Para poder entender cómo funciona un motor de ajedrez es imprescindible entender el concepto de árbol en el ámbito de la computación. Es decir, se trata realmente del concepto de ramificación, una estrategia que permite revisar todas las posibles jugadas y determinar a qué estado del juego nos puede llevar cada una de ellas.

Al inicio del juego, por ejemplo, existen 20 posibles movimientos para las piezas blancas (16 de los peones y 4 de los caballos) y lo mismo para las negras. Por tanto, hay 400 posibilidades solo para la primera jugada (se entiende por jugada cuando ha transcurrido un turno de blancas y negras, es decir, ambos jugadores mueven pieza. Sin embargo, tras solo cuatro jugadas este número se dispara a más de 288 mil millones de posibilidades. Por esta razón, es imposible resolverlo revisando todas las jugadas y se debe recurrir a la heurística, una manera de buscar la solución de un problema mediante métodos no rigurosos. [19]

El primer artículo sobre este tema fue escrito por Claude Shannon, y publicado en 1950 incluso antes de la existencia de una computadora que jugara al ajedrez. Predijo acertadamente las dos posibles formas de búsqueda de cualquier programa, a las que nombró “Tipo A” y “Tipo B”.

*Tipo A*

Los programas de “Tipo A” son más rudimentarios y utilizan una búsqueda basada en la fuerza bruta, es decir, examinan todas las posiciones posibles de cada rama del árbol de movimientos usando el algoritmo “minimax”,

Un módulo de este tipo consiste básicamente en dos subprogramas: uno de ellos se encarga de calcular todas las jugadas legales a partir de la posición actual del tablero; el otro toma cada posición y la evalúa mediante una función de evaluación. De esta manera, las posiciones se convierten en números de evaluación. Finalmente, el algoritmo ordena todas de mayor a menor y devuelve la mejor jugada situada en dicha lista.

En la jerga de la computación, a cada posición se le llama nodo y, por lo general, en la información que nos ofrecen los motores de ajedrez cuando responden con una jugada suele contener el número de posiciones que ha evaluado nuestro ordenador por segundo en kN (Kilo Nodos). De esta forma también podemos hacernos una idea de cuando de rápido es nuestro pc.

La función de evaluación es el alma de cada motor, y es lo que caracteriza esencialmente a cada módulo. Los números que asigna la función de evaluación a cada posición o pieza son totalmente arbitrarios. Cada programador tiene en cuenta un tipo de ponderación diferente (por ejemplo: peón = 1 punto; reina = 9 puntos …) y, por eso, no se pueden comparar las funciones de evaluación de diferentes motores directamente, que es uno de los errores más comunes.

Pero ¿en qué momento se para de calcular y evaluar posiciones? El ordenador va calculando en niveles de profundidad, a medida que le dejamos tiempo para calcular una posición, va llegando más lejos en su cálculo. Es decir, la profundidad de cálculo dependerá de dos aspectos esencialmente, uno es el parámetro del tiempo que le asigna el programador al motor por jugada y el otro la capacidad hardware del pc que se esté empleando, ya que, para un mismo tiempo

asignado, un ordenador puede calcular más posiciones que otro en función de su potencia hardware.

Los niveles de profundidad se miden en "ply", que es lo que en ajedrez se considera una "media jugada", es decir, que juegue el blanco o que juego el negro. Por tanto, una profundidad de 2 ply es equivalente a una jugada. El ordenador va progresando en esta profundidad. En primer lugar, genera todas las posibles jugadas legales y, por tanto, todos los posibles nodos a los que podemos llegar desde la posición actual. A continuación, ordena todos esos nodos y los ordena de mejor a peor evaluación. Con esto, el pc ha terminado de calcular la profundidad 1. Acto seguido, toma todas las posiciones generadas a 1 ply y calcula todas las posibles jugadas legales, generando todos los posibles nodos a los que se puede llegar después de 2 ply, y los evalúa; con esto se llega a la profundidad 2; y así sucesivamente. [20]

*Tipo B*

Los programas de "Tipo B" en lugar de gastar potencia examinando movimientos malos o triviales, emplean una especie de "inteligencia artificial estratégica" para analizar únicamente las mejores jugadas de cada posición, algo parecido a lo que hacen los seres humanos.

Lo que realmente diferencia a jugadores expertos de jugadores mediocres es la habilidad del reconocimiento de patrones, que se va adquiriendo con la experiencia. Esto permite analizar más profundamente las mejores líneas y no perder el tiempo con otras peores. Los jugadores de ajedrez recuerdan muchas de las posiciones jugadas en anteriores partidas y aprenden de la experiencia, sin embargo, los algoritmos de computación no lo tienen tan fácil.

*Estrategia contra fuerza bruta*

Inicialmente se pensaba que los programas de "Tipo A" eran muy poco prácticos por la escasa potencia de computación con la que contaban los primeros ordenadores, ya que con aproximadamente 30 movimientos posibles en una posición típica de medio juego para examinar las $30^6$ (más de 700.000.000) posiciones contenidas en las tres primeras jugadas se empleaban unos 16 minutos en el caso más optimista.

Por el contrario, los programas de "Tipo B" al emplear la inteligencia artificial examinaban menos jugadas lo cual permitía analizar las líneas significativas de manera más profunda en un tiempo razonable.

El problema de los programas "Tipo B" es que se confía demasiado en que el programa pueda decidir qué movimientos son suficientemente buenos y cuáles no, lo que es un problema bastante más grave que en programas de 'Tipo A' con un hardware de gran velocidad.

En 1973, la Universidad de Northwestern, que era la principal encargada de la creación de programas de "Tipo B", dejó de programarlos, pasando al bando de los programas de "Tipo A". El resultado fue "Chess 4.0", ganador del torneo ACM durante 5 años seguidos. Una de las razones de este cambio fue porque los programas de "Tipo B" eran poco estimulantes durante los torneos, debido a que es muy difícil predecir lo que va a mover y el por qué. Otra de las razones es que en un programa de "Tipo A" es mucho más fácil detectar fallos del programa y depurarlos.

Además, el desarrollo del hardware y los avances tecnológicos hicieron que los sistemas de fuerza bruta ("Tipo A") siguieran al alza y se intensificaran en la década de los años 90. Fue en el 97 cuando "Deep Blue" ("Tipo A") ganó por primera vez al Campeón del Mundo Gary Kasparov.

No obstante, a finales de 1990, los programadores empezaron a preferir los programas de "Tipo B". "Rebel 10" fue uno de los resultados, el cual se proclamó segundo motor de ajedrez más fuerte del mundo.

A principios del siglo XXI siguieron desarrollándose ambos sistemas, en muchos casos complementándose el uno al otro. Sin embargo, el hecho más reciente que ha revolucionado los motores de ajedrez es el caso de "AlphaZero".

### 2.4.3 AlphaZero

Se trata de un programa informático desarrollado por "DeepMind", una compañía de inteligencia artificial inglesa creada en 2010 y que fue adquirida por Google en 2014. Este software emplea el enfoque generalizado de "AlphaGo Zero", el software de inteligencia artificial desarrollado por la misma compañía para jugar al juego de mesa Go. [21]

A diferencia de otros programas, "AlphaZero" no está basado en el conocimiento humano. Su comprensión sobre el ajedrez, más allá de conocer las reglas básicas, proviene únicamente de su capacidad de autoaprendizaje. Tras casi 5 millones de partidas contra sí mismo durante 4 horas, "AlphaZero" obtuvo el mismo conocimiento que los humanos en 1400 años.

El 5 de diciembre de 2017, el equipo de "DeepMind" lanzo una preimpresión presentando "AlphaZero", que logró en 24 horas un nivel de juego sobrehumano en ajedrez, shogi y Go al derrotar a los campeones del mundo en cada disciplina como "Stockfish", "Elmo" y la versión de tres días "AlphaGo Zero" respectivamente.

"Stockfish" es una máquina programada por los ingenieros más brillantes del planeta desde 10 años atrás con el objetivo de jugar un ajedrez perfecto. Si se comparan las búsquedas de árbol de búsqueda de Monte Carlo, "AlphaZero" busca solo 80.000 posiciones por segundo en ajedrez mientras que "Stockfish" busca 70 millones. "AlphaZero" lo compensa mediante el uso de su red neuronal profunda para centrarse mucho más selectivamente en la variación más prometedora.

"AlphaZero" dominó a "Stockfish" en tan solo 4 horas de autoaprendizaje, sin ningún tipo de acceso a libros de apertura o bases de datos de tablas de finales. Partiendo de una hoja en blanco, fue capaz de entrenar la red neuronal en la que se basa para aprender hasta límites insospechados sin aportación humana. El programa alimentado de conocimiento humano fracasó de manera estrepitosa e histórica, perdiendo de manera humillante contra "AlphaZero" en un duelo a 100 partidas (perdió 28 y empató las restantes). [22] [23]

### 2.4.4 UCI (Universal Chess Interface)

El lenguaje de comunicación o protocolo manejado por un motor le permite ser usado por un programa o interfaz gráfica de usuario (GUI). De esta manera, se permite el desarrollo de aplicaciones gráficas de manera independiente, que empleando el protocolo de comunicación que implementan los motores, se pueda comunicar con cualquiera de ellos arbitrariamente.

La GUI permite al usuario jugar contra un motor en un ambiente gráfico más agradable e intuitivo. Gracias a esto, se han desarrollado multitud de software gratuitos que permiten jugar al ajedrez incluso en línea a través de la red con otros usuarios. Caben destacar las interfaces "XBoard" en Linux y "WindBoard" y "ChessBase" en Windows.

El primer protocolo estándar que se utilizó fue la interfaz de línea de comandos de "GNU Chess", que se llamaba "Protocolo de Comunicación de Motores de Ajedrez". No obstante, a partir de noviembre del 2000 se creó un protocolo más nuevo cuyas siglas son UCI ("Universal Chess Interface"). Algunos motores soportan ambos protocolos, debido a que el "Protocolo de Comunicación de Motores de Ajedrez" es más popular, aunque muchos de los desarrolladores piensan que el UCI es más expresivo.

El protocolo UCI es abierto y puede utilizarse sin ningún tipo de licencia. Hoy en día, los motores de ajedrez más modernos soportan el protocolo UCI de manera que es muy sencillo intercambiar un motor de ajedrez por otro en una aplicación gráfica. [24] [25]

# CAPÍTULO 3. DISEÑO PREVIO

Desde un primer momento se tenía por buen seguro la idea de que uno de los objetivos principales del proyecto no iba a ser el desarrollo del algoritmo del ajedrez. Esto contempla dos grandes ámbitos: por una parte, está la gestión del juego, es decir, administrar el tablero, controlar los movimientos, determinar los turnos, etc. Y, por otra parte, se encuentran los algoritmos que, ante una jugada dada, son capaces de responder con un movimiento adecuado, estos son los motores de ajedrez.

Los motores de ajedrez no suponen ningún problema, ya que la mayoría de ellos son públicos, no de código abierto, pero si se puede disponer gratuitamente de sus ejecutables. En cuanto a la gestión del juego, se ha basado en el libro de A. M Vozmediano [26], que propone una solución sencilla basada en el lenguaje de programación C.

Además, en esta etapa de diseño se plantean también las diferentes alternativas en cuanto al empleo de diferentes robots industriales, interfaces gráficas de usuario, comunicaciones y diseño 3D.

## 3.1 Análisis de soluciones

### *Robot*

En la Escuela de Ingenierías Industriales de Valladolid, se dispone de dos modelos diferentes de robots industriales, uno de ellos colaborativo. Se trata del "ABB IRB 120" y del "UR3" (Figura 3 y 4).

*Figura 3 UR3 (colaborativo)*


*Figura 4 ABB IRB 120*

Por la finalidad del proyecto, la idea principal era utilizar el robot colaborativo "UR3. No obstante, tras realizar algunas pequeñas pruebas sobre el tablero se comprobó que no iba a ser posible llegar de forma adecuada a todas las posiciones con el robot y, menos aún, si se incluía la colocación y dejada de piezas comidas en el exterior del tablero.

Se realizaron pruebas similares sobre el robot "ABB IRB120" para confirmar la viabilidad del proyecto y se verificó que era capaz de llegar a todas las posiciones del tablero.

Al escoger este robot, la implementación del software se llevará a cabo sobre la plataforma de desarrollo de ABB "Robot Studio". Esta herramienta no solo nos permite programar el robot si no también llevar a cabo simulaciones del entorno de trabajo y las trayectorias programadas. Se programa en un lenguaje propio de ABB que se denomina "RAPID" ("Roboics Application Programming Interactive Dialogue").

*IDE de desarrollo*

En este punto se busca un entorno de desarrollo en el cual desarrollar todo el código correspondiente al algoritmo del ajedrez, tanto la gestión del juego como la implementación de los motores de ajedrez. Además, se requiere una herramienta que sea capaz de implementar sistemas de comunicación con otros programas, o bien, con el propio robot.

Ante esta tesitura, la primera solución que se plantea es el entorno de desarrollo integrado de "MATLAB". Es un sistema de cómputo numérico que ofrece un entorno de desarrollo con un lenguaje propio y está disponible para todas las plataformas (Windows, Mac OS X y GNU/Linux).

Otra de las posibilidades planteadas ha sido Visual Studio, que también es un entorno de desarrollo integrado para Windows, Linux y Mac OS. Compatible con muchos lenguajes de programación tales como C++, C# y Visual Basic es una herramienta ideal para el desarrollo de aplicaciones de este tipo.

No obstante, ante el desconocimiento de esta última, se optó por utilizar "MATLAB" para integrar los cimientos del código del ajedrez. Asimismo, "MATLAB" dispone de multitud de librerías, entre ellas una de comunicación TCP/IP muy estable y sencilla de utilizar, lo cual puede ser muy útil para la transferencia de información con el robot.

*Interfaz gráfica de usuario (GUI)*

Se trata de un programa informático que actúa de interfaz de usuario, utilizando un conjunto de objetos gráficos para representar la información y acciones disponibles. Es considerado un punto clave en el desarrollo de este proyecto, ya que supone, de alguna manera, el enmascaramiento del código, de forma que cualquier usuario pueda interactuar sin necesidad de conocimientos previos en programación.

"MATLAB" dispone de su propio GUI denominado "GUIDE" que proporciona herramientas para diseñar interfaces de usuario para aplicaciones personalizadas. Mediante el editor de diseño es posible diseñar gráficamente la interfaz con el empleo de botones, controles deslizantes, menús, barras de herramientas, etc. La sencillez se ve reflejada en la generación automática del código a través del "GUIDE".

De este modo, al haber escogido “MATLAB” para la implementación del código de ajedrez, la opción de desarrollar el GUI sobre este entorno cobra fuerza.

Otras alternativas pasan por entornos de desarrollo como “Visual Studio” o “Qt”, más orientados al diseño de aplicaciones gráficas que “MATLAB”.

*Figura 5 Logo herramienta Qt creator*

*Figura 6 Logo IDE Visual Studio*

El entorno de desarrollo de “Qt” es más abierto en el sentido de que cualquier aplicación que se realice puede ser compilada indistintamente para cualquier sistema operativo. Por otra parte, “Visual Studio” se apoya sobre los pilares de “Microsoft .NET Framework” una tecnología que admite la compilación y ejecución de la última generación de aplicaciones y servicios web. Por consiguiente, “Visual Studio” dota de más robustez a la aplicación si esta va a ser ejecutada sobre el sistema operativo Windows.

### *Diseño 3D*

En cuanto al software para realizar diseños en 3D existen multitud de alternativas. No obstante, se han centrado los intereses en tres:

- Autodesk Inventor: es un paquete de modelado paramétrico de sólidos en 3D producido por la empresa de software AutoDesk. Se tienen conocimientos de esta herramienta debido a la formación adquirida durante el grado.

- Catia: es un programa informático de diseño, fabricación e ingeniería asistida por computadora comercial realizado por Dassault Systèmes. El programa está desarrollado para proporcionar apoyo desde la concepción del diseño hasta la producción y el análisis de productos. También se tienen conocimientos sobre esta herramienta gracias a la formación adquirida durante el Máster.

- FreeCad: en este caso se trata de una aplicación libre de diseño asistido por ordenador en tres dimensiones destinado a la asistencia en ingeniería mecánica y el diseño de elementos mecánicos. Está basado en Open CASCADE y programado en los lenguajes C++ y Python.


*Figura 7 A. Inventor*


*Figura 8 Catia*

*Figura 9 FreeCad*

Ante estas opciones, se ha decidido utilizar Catia para este proyecto ya que es una herramienta muy completa, robusta, que ofrece grandes resultados y con gran importancia en el mundo laboral.

## 3.2 Estructura software

Como ya se viene advirtiendo a lo largo de este proyecto, será necesaria la separación del software en diferentes máquinas, como puede apreciarse en la Figura 10. Por una parte, se encuentra el propio software del robot, el cual será implementado sobre la computadora de ABB perteneciente al mismo. Por otra parte, estará el software del juego implementado en "Matlab", tanto la gestión de tablero y jugadores como los motores de ajedrez. Y finalmente, la interfaz gráfica, que al ser desarrollarla en "Visual Studio" es totalmente independiente del código de "Matlab" y puede ser ejecutada, o bien, en la misma máquina que "Matlab", o bien, en otra diferente.


*Figura 10 Módulos de programa*

Ante esta situación, se debe determinar que conexiones y comunicaciones gestiona cada nodo con el resto. Hay que tener en cuenta que, por la importancia que reside en su código el nodo de "Matlab" es imprescindible en cualquiera de los casos, es decir:

1) Se podrá utilizar el nodo de "Matlab" y el nodo del robot, prescindiendo en este caso de la interfaz gráfica, jugando simplemente con comandos desde "Matlab" y sin visualización gráfica.

2) Otra opción sería emplear el nodo de "Matlab" y la interfaz gráfica, es decir, en este caso estaríamos jugando al ajedrez virtualmente, sin que el robot tenga que intervenir.


3) Y la última opción, sería emplear los tres nodos simultáneamente, que es el objetivo final de este proyecto.


## 3.3 Comunicaciones

La comunicación entre los diferentes nodos se lleva a cabo mediante el modelo TCP/IP. El conocido Protocolo de Control de Transmisión (TCP) y el llamado Protocolo de Internet (IP) es una denominación que permite identificar al grupo de protocolos de red que respaldan a Internet y que hacen posible la transferencia de datos entre redes de ordenadores. Fue desarrollado por Vinton Cerf y Robert E. Kahn, en la década de 1970 y fue implantado en ARPANET, la primera red de área amplia (WAN). [27]

El primer protocolo (TCP) proporciona un transporte muy fiable de los datos mientras que el segundo, el protocolo IP, se identifica y define especialmente por ofrecer la posibilidad de dirigir los dato a otras máquinas. [28]

Para conseguir un intercambio fiable de datos entre dos equipos, se llevan a cabo muchos procedimientos separados. El resultado es un software muy complejo. Por ello, con un modelo de capas resulta más sencillo agrupar funciones relacionadas e implementar un software de tipo modular.

Las capas están jerarquizadas y cada capa se construye a partir de su predecesora. En cualquier red, la misión de una capa es proveer servicios a las capas superiores haciéndoles trasparentes el modo en que esos servicios se llevan a cabo. De esta manera, cada capa debe ocuparse exclusivamente de solicitar servicios a su nivel inferior y de prestarlos a su nivel superior.

En la Figura 11 podemos comparar la estructura de capas del protocolo TCP/IP con el modelo OSI (*Open System Interconexion* o modelo de interconexión de sistemas abiertos) el cual es un modelo de referencia para los protocolos de red.

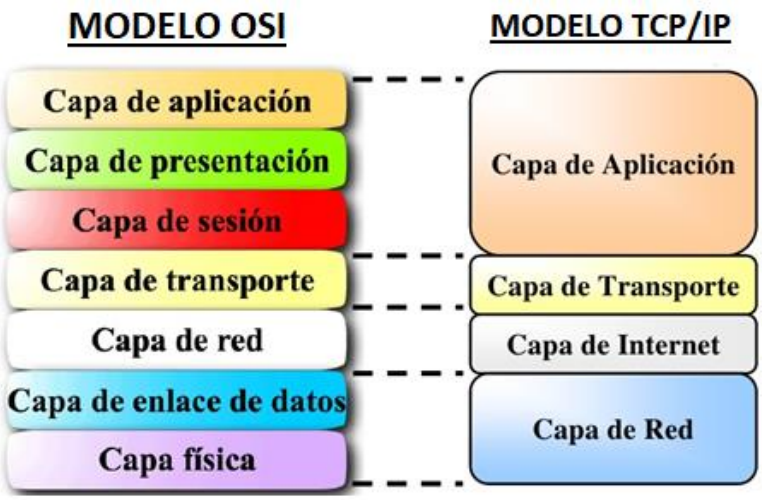



*Figura 11 Comparación estructura de capas entre Modelo Osi y TCP/IP*

[29]

Como se puede apreciar, el modelo TCP/IP cuenta con 4 capas o niveles:

- Capa 4 o nivel de aplicación: asimilable a las capas 5 (sesión), 6 (presentación) y 7 (aplicación), del modelo OSI. Es el nivel más alto dentro del protocolo que nos ocupa y aporta la comunicación entre procesos o aplicaciones en computadores distintos. Este nivel proporciona a las aplicaciones informáticas el acceso a la red de comunicaciones y facilita diferentes servicios de comunicaciones como, por ejemplo, transferencia de archivos, correo electrónico, etc.

- Capa 3 o nivel de transporte: asimilable a la capa 4 (transporte) del modelo OSI. Este nivel es el encargado de transferir datos entre ordenadores sin tener en cuenta la tecnología de la red sobre la que se transmiten. Esta capa garantiza una transmisión fiable; es decir, asegura que los datos llegan al destino sin errores y en el mismo orden que fueron enviados.

- Capa 2 o nivel de Internet: asimilable a la capa 3 (red) del modelo OSI. Este nivel se encarga de direccionar y guiar los datos desde el origen al destino a través de la red o redes intermedias. Las funciones de este nivel que caben destacar son:
  - Fragmentación de la información en fragmentos más pequeños, denominados paquetes.
  - Envío de los paquetes a la red. Cada uno de los paquetes viaja de forma independiente hacia el destino. Los diferentes paquetes pueden atravesar y viajar a través de redes diferentes.
  - Encaminamiento; es decir, buscar un camino entre todos los posibles para que los paquetes lleguen a su destino

- Capa 1 o nivel de acceso a la red: asimilable a la capa 2 (enlace de datos) y a la capa 1 (física) del modelo OSI. Este nivel depende de la tecnología de red

utilizada y no se especifica de antemano dentro de TCP/IP. Es el responsable del intercambio de datos entre el sistema final y la red a la que está conectado y de la definición de las características del medio, señalización y codificación de las señales

[30]

En un protocolo de este tipo orientado a conexión, no hay relaciones maestro/esclavo. Las aplicaciones, sin embargo, utilizan un modelo cliente/servidor en las comunicaciones.

El servidor, como su propio nombre indica, es una aplicación que ofrece un servicio a usuarios de Internet mientras que el cliente es el que solicita este servicio. Generalmente, un servidor puede tratar múltiples peticiones (múltiples clientes) al mismo tiempo. [31]

En este caso, para el proyecto que se está desarrollando, será necesario definir las comunicaciones entre los diferentes bloques anteriormente mencionados, partiendo de la premisa de que el bloque "Matlab" va a ser imprescindible en cualquier modo de juego. Atendiendo a los servicios que presta o solicita cada uno de los bloques se ha establecido lo siguiente:

- El bloque robot actúa como **servidor** frente al bloque de "Matlab" que será su **cliente**. "Matlab" que alberga la gestión del juego, solicita al robot el servicio de mover la/s pieza/s correspondientes. Este, tras realizar su actividad informa a "Matlab" de que ha finalizado.

- Por otra parte, el bloque de "Matlab" ejerce como **servidor** ante bloque de la interfaz gráfica que hace de **cliente**. El GUI que recibe las instrucciones del usuario, solicitará al bloque de "Matlab" la gestión del movimiento o la llamada a uno de los motores para responder ante una jugada dada. Una vez realizado el movimiento solicitado por el usuario, o bien, una vez realizada la llamada a uno de los motores de ajedrez, "Matlab" informará a la interfaz gráfica de la nueva situación del tablero para que este la actualice y la muestre.

El resultado puede verse en la Figura 12.

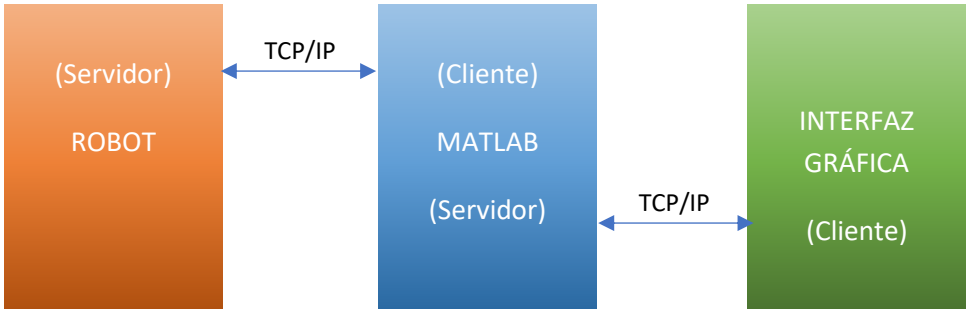



*Figura 12 Diagrama de comunicaciones*

## 3.4 Diseño 3D

Finalmente, otra de las partes importantes de este proyecto es el diseño asistido por ordenador. Cuando se establecieron las bases de este, se vio clara la necesidad de diseñar, por un lado, las piezas de ajedrez y, por otro lado, la garra del robot, de forma que los movimientos pudieran ser precisos y con el menor error posible ya que con la duración de una partida de ajedrez un pequeño error acumulado puede ser determinante.

Las piezas serán diseñadas con una estructura cuadrada lo suficientemente alta para que puedan ser atrapadas cómodamente y en su parte superior habrá espacio para el diseño libre del identificador de la pieza. Habrá que tener también presente para el diseño las diferentes alturas de las piezas en función de su jerarquía.

En cuanto a la pinza, se ha decidido que el diseño de la estructura base sea en forma de escuadra (Figura 13) de modo que la cogida de la pieza sea inequívoca. El problema de utilizar la pinza predeterminada que incorpora actualmente el robot ABB es que, al ser completamente plana, es posible que la cogida y dejada de la pieza vaya acumulando pequeños errores. Otra de las ventajas del diseño de la garra en forma de escuadra, es que la cogida se realizaría a través de la diagonal de la pieza con lo que se evitaría que el resto de las piezas adyacentes molestasen.

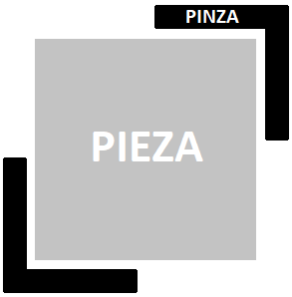



*Figura 13 Esquema Pieza-Pinza*

# CAPÍTULO 4. DESARROLLO DEL PROYECTO

En este capítulo se describe en detalle el contenido de cada una de las partes del proyecto. Corresponde a la etapa de ejecución, en la cual se lleva a cabo lo planificado y diseñado en la etapa anterior.

## 4.1 Gestión del ajedrez (Matlab)

El IDE de Matlab albergará principalmente dos módulos: por una parte, está el bloque de programación encargado de la gestión del juego y, por otra parte, el módulo de comunicación con los motores de ajedrez.

### 4.1.1 Gestión del juego

Gracias a la ayuda del libro anteriormente mencionado [26] que propone una solución sencilla basada en lenguaje C, se ha implementado una clase ("miAjedrez.m" consultar Anexo A: Código del proyecto) que es la encargada de administrar las propiedades básicas del juego de ajedrez. Para que no lleve a confusión, puntualizar que este módulo no se encarga de responder con un movimiento ante una jugada dada.

Entre las funcionalidades más destacadas que se encarga de llevar a cabo esta clase, están las siguientes:

- ✓ Permite seleccionar el modo de juego:
  - Humano vs Robot -> (Humano es blancas; robot es negras)
  - Robot vs Humano -> (Robot es blancas; humano es negras)
  - Robot vs Robot -> (Enfrenta a dos motores)
  - Humano vs Humano -> (Dos jugadores)

- ✓ Inicializa el tablero:
  - Crea una matriz 8x8 y asigna a cada casilla su color correspondiente.
  - Sitúa las piezas en su posición inicial.

- ✓ Determina los turnos, es decir, si juegan blancas o negras y si es turno del robot o del humano. En función de esto, se encargará, o bien, de pedir por teclado un movimiento al usuario, o bien, de realizar la llamada correspondiente al módulo de gestión de motores que se explica más adelante.

- ✓ Comprueba si el movimiento solicitado es legal o ilegal. Tiene una función específica para cada pieza que comprueba si el movimiento se adecúa a las normas del juego. Si el movimiento no se puede realizar porque la situación del juego no lo permite o porque el movimiento de la pieza no es correcto, informará de esto al usuario y volverá a pedir un nuevo movimiento.

- ✓ Se encarga de comprobar tras cada jugada si existe situación de jaque o jaque mate.

- ✓ Muestra una pequeña representación del tablero de una manera gráfica un cierto rudimentaria (esto sería remplazado por la interfaz gráfica que se ha desarrollado).

- ✓ Se encarga de actualizar el tablero tras cada movimiento.

- ✓ Se le ha incorporado los módulos de comunicación TCP/IP tanto con el robot como con la interfaz gráfica.

- ✓ Anota y muestra una lista de todos los movimientos que se han realizado tras cada jugada.

### 4.1.2 Gestión de los motores de ajedrez

Se ha implementado otra clase ("UCIEngine.m" consultar Anexo A: Código del proyecto) que es la encargada de comunicarse con los motores de ajedrez y que implementa el protocolo UCI.

Esta clase contiene una función "menú" que se encarga de mostrar los motores disponibles y permite seleccionar uno de ellos, o bien dos, en el caso de jugar en modo "Robot-Robot".

Este módulo es llamado desde la aplicación principal cada vez que la clase "miAjedrez.m" determina que es el turno del robot y realiza, por tanto, una petición. Implementa el protocolo UCI para comunicarse con el motor correspondiente y extraer una respuesta, que se traduce en un movimiento en el tablero. Emplea la "Notación FEN" para describir la situación del tablero tras cada jugada y comunicársela al motor (consultar el epígrafe Notación FEN unas líneas más abajo).

Además, tras cada respuesta del motor, se informa al usuario de las diferentes propiedades del resultado extraído, es decir:

- Profundidad de búsqueda en el árbol.
- Cantidad de nodos examinados.
- Nodos por segundo (kN/s).
- Ponderación de los nodos.
- Mejor movimiento.

Estos datos son bastante útiles ya que nos pueden dar una idea de la potencia de la máquina sobre la que se está ejecutando el algoritmo.

#### *Notación FEN*

Las siglas provienen de "Forsyth-Edwards Notation", y se trata de un sistema de notación propuesto en 1883, utilizada para describir una determinada posición en el tablero de ajedrez. Un registro FEN define una posición particular, en una sola línea de texto y usando un conjunto de caracteres ASCII.

La notación se hace de la siguiente manera:

- Se realiza por filas, empezando siempre por la octava y terminado por la primera.

- Las piezas blancas se anotan en mayúsculas y las negras en minúsculas.

- En cada fila se va anotando cada pieza de izquierda a derecha. En el caso de casillas sin pieza, se escribe un número correspondiente al número de casillas vacías que haya seguidas.

- Cada vez que termina una fila se escribe el carácter "/".

- Una vez terminada la descripción de las piezas, se indica a que color le corresponde el siguiente turno: "w" mueven blancas y "b" mueven negras.

- Si el enroque ya no es posible para ninguna de las partes, se coloca el carácter "-". De lo contrario, se escribirán una o más letras en función de lo que se describe a continuación:
  - "K": es posible el enroque del rey blanco en su flanco.
  - "Q": es posible el enroque del rey blanco en el flanco de su dama.
  - "k": es posible el enroque del rey negro en su flanco.
  - "q": es posible el enroque del rey negro en el flanco de su dama.

- Si no hay posibilidad de capturar un peón al paso se escribirá el carácter "-". Si un peón acaba de hacer una jugada de 2 casillas, se pondrá la casilla que está justo detrás del peón. Esta se deberá poner independientemente de si hay o no algún peón en condiciones de realizar una captura al paso.

- Se indica el número de medios movimientos desde la última captura o avance del peón. Esto se hace para determinar si unas tablas pueden ser reclamadas en virtud de la regla de los cincuenta movimientos.

- Y finalmente, se escribe el número total de movimientos de la partida. Tiene el valor de 1 para la posición inicial y se va incrementando con cada movimiento de las negras.

A continuación, se muestra un ejemplo de lo que sería la notación FEN para la posición inicial del tablero:

*"rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"*

Y otro ejemplo para el movimiento 1. E4:

*"rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq e3 0 1"*

[32] [33]

*Modularidad del software*

La modularidad, en programación orientada a objetos, hace referencia a la propiedad que permite subdividir una aplicación en partes más pequeñas, cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las partes restantes.

En este caso el software será diseñado de manera modular, de modo que no será necesaria la ejecución de los 3 nodos simultáneamente, si no que siendo "Matlab" el principal, podemos escoger entre las diferentes alternativas:

- Jugar al ajedrez exclusivamente en Matlab.
- Emplear Matlab junto con la interfaz gráfica.
- Utilizar Matlab y el robot.
- O bien, la opción más completa que sería Matlab con la interfaz gráfica y con el robot.

Esto también puede resultar muy útil a la hora de depurar el código, corregir errores o implementar mejoras, ya que no exige al programador disponer de todos los módulos del proyecto simultáneamente.

### 4.1.3 Script de lanzamiento

Para simplificar el lanzamiento de estos módulos se ha implementado un script "Ajedrez_TFM.m" que se encarga de inicializar todo y realizar las llamadas oportunas a cada una de las clases detalladas anteriormente.

Por una parte, el script permite al usuario determinar que módulos va a utilizar mediante dos variables de control (booleanas): "app_visual" y "robot_studio". A continuación, define las comunicaciones TCP/IP en función de que módulos se hayan activado y muestra por pantalla instrucciones al usuario de los pasos que tiene que seguir como, por ejemplo, asegurarse de que ha lanzado previamente el módulo de robot o cuando está listo para lanzar la aplicación visual.

Finalmente, se encarga de realizar las llamadas a las funciones de la clase "mi_Ajedrez.m" con los parámetros de comunicación previamente establecidos para que esta sea capaz de comunicarse con la aplicación visual o con el robot.

## 4.2 Gestión del robot (Robot Studio)

ABB comercializa la serie de robots articulados IRB de 6 ejes, capaces de manipular cargas desde 5 hasta 500 Kg. Esta serie de robots cubren las necesidades de la gran mayoría de aplicaciones de la industria actual, ya sea paletizado, pintura, sellado, manipulación y demás aplicaciones industriales.

El armario de control que precisa el IRB120 es un IRC5 (Figura 14). Este controlador contiene los elementos electrónicos necesarios para el control del manipulador y del equipo periférico. Esta diseñado específicamente para robots. El

sistema de control permite generar trayectorias punto a punto, movimientos lineales y movimientos circulares. También dispone de puertos de comunicación RS-232, puertos Ethernet y USB. El controlador tiene como opciones compatibles trabajar con otros módulos de comunicación tales como Profibus, DeviceNet o Interbus. Además, dispone de un módulo de entradas y salidas. Estas salidas se dividen en digitales 24Vdc y analógicas de tensión ±10 V y de corriente 4-20mA.

*Figura 14 Armario IRC5*

En la Figura 15 se muestra la FlexPendant, herramienta con la que se realizan las operaciones de programación y configuración a pie de línea. Su diseño y su funcionalidad hacen que sea una de las mejores del mercado. Esta programadora está diseñada ergonómicamente y se utiliza para la programación y para el control del robot.

*Figura 15 FlexPendant*

ABB ha desarrollado un software específico de simulación y de programación offline. Este software se llama RobotStudio (Figura 16). Nos permite la simulación y visualización de nuevas aplicaciones, realizar nuevos programas desde nuestra mesa de trabajo sin necesidad de ir a la celda del robot, ni de parar la producción. Con este

*Figura 16 Icono de RobotStudio*

software podemos simular y validar nuevas aplicaciones, realizar programas nuevos o incluso optimizar los programas que ya tenemos en producción. Esta herramienta está enfocada a maximizar el rendimiento de la inversión del sistema robot.

[34]

El lenguaje de programación de estos robots se denomina "RAPID" ("Roboics Application Programming Interactive Dialogue"). "RAPID" es un lenguaje de programación textual de alto nivel desarrollado por la empresa ABB. Una aplicación de este tipo consta de un programa y una serie de módulos del sistema (Figura 17).

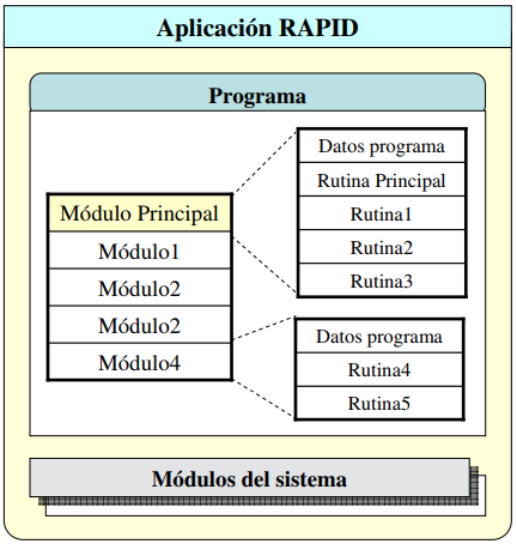



*Figura 17 Estructura de una aplicación en RAPID*

*El programa*

Es una secuencia de instrucciones que controlan el robot y en general consta de tres partes (Figura 18):

- **Una rutina principal (main):** es la rutina desde donde se inicia la ejecución del programa.
- **Un conjunto de subrutinas:** estas sirven para dividir el programa en partes más pequeñas para obtener un programa modular.
- **Los datos del programa:** aquí se definen variables globales, posiciones del TCP, planos de trabajo, herramientas, etc.

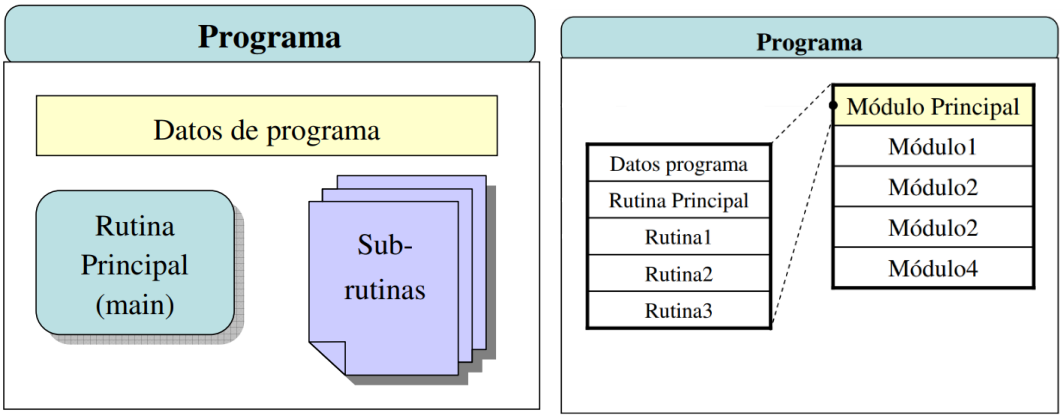



*Figura 18 Estructura del programa*

*Módulos*

Un conjunto de declaraciones de datos, seguido de un conjunto de rutinas. Los módulos pueden ser guardados, cargados y copiados en forma de archivos. Los módulos se dividen entre módulos de programa y módulos de sistema.

- **Módulos de programa:** Cada módulo puede estar formado por diferentes datos y rutinas. Uno de los módulos debe contener el procedimiento global de entrada, es decir, la rutina principal o "main", que será el punto de partida del programa. Además, estos módulos pueden ser cargados y descargados durante la ejecución.

- **Módulos de sistema:** se utilizan principalmente para datos y rutinas específicos del sistema, por ejemplo, un módulo de sistema de "ArcWare" que es común para todos los robots de soldadura en arco.

*Rutinas*

Una rutina es, normalmente, un conjunto de declaraciones de datos seguido de un conjunto de instrucciones utilizadas para implementar una tarea. Las rutinas pueden dividirse en tres categorías:

- Procedimientos: Un conjunto de instrucciones que no devuelve ningún valor.
- Funciones: Un conjunto de instrucciones que devuelve un valor.
- Rutinas TRAP: Un conjunto de instrucciones que es disparado por una interrupción.

*Estructura del proyecto*

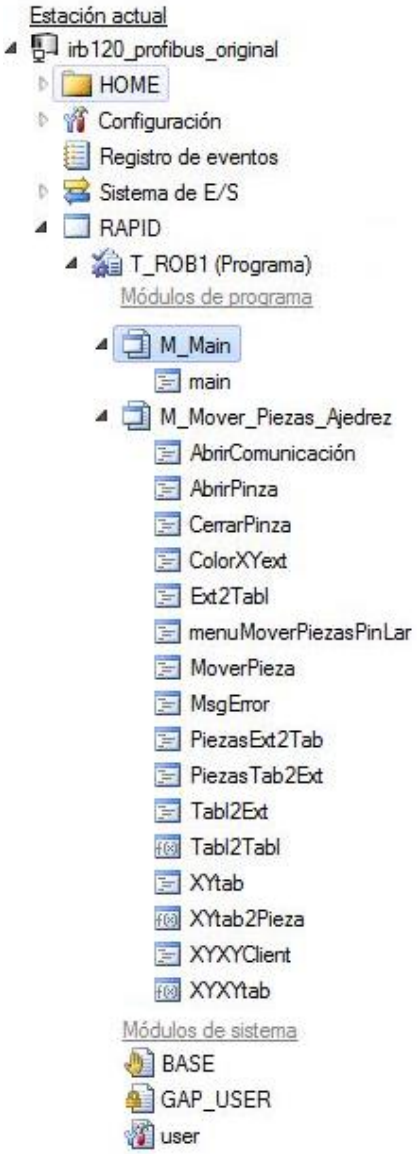



*Figura 19 Estrucutra de programa para el proyecto*

En el caso de este proyecto, la estructura del programa RAPID con sus módulos y rutinas es la que se puede ver en la Figura 19.

Como muestra el esquema, el programa principal consta de dos módulos:

- M_Main
- M_Mover_Piezas_Ajedrez

El módulo "M_Main" es el que contiene el procedimiento global de entrada o rutina principal (main). Desde esta rutina simplemente se realiza una llamada al segundo módulo "M_Mover_Piezas_Ajedrez". Se podrían haber fusionado ambos para mayor simplicidad, pero esto se hace así por comodidad a la hora de programar e introducir futuros cambios. Si más adelante se quisiera implementar otro módulo nuevo, bastaría con realizar una nueva llamada desde el "M_Main" sin tener que modificar nada en el módulo de ajedrez.

En cuanto al módulo "Mover_Piezas_Ajedrez", está formado por varios procedimientos y funciones que se explicarán con detalle a continuación.

[35]

[36]

*M_Mover_Piezas_Ajedrez*

Constituye el software principal del robot que se ha desarrollado íntegramente en este proyecto. En este módulo se integran todas las rutinas necesarias para realizar la comunicación con Matlab y mover las piezas del tablero.

### Declaración de datos

Aquí se definen los puntos, planos, herramientas y variables que van a ser necesarios para el resto de las rutinas. En este caso, se ha tenido que crear y declarar una nueva herramienta (pinza modificada) que como se verá posteriormente ha sido diseñada específicamente para este proyecto.

```
LOCAL PERS tooldata Pinza_Neumatica_Lag
```

También, se han definido tres puntos o "robtarget" importantes. Un "robtarget" define la posición y orientación deseada para el extremo del robot incluyendo los datos de configuración necesarios para que no existan ambigüedades. También especifica la posición deseada para los ejes externos, si éstos existieran. Los "robtaget" definidos son los siguientes:

- Origen tablero: este punto define el origen del tablero, más concretamente establece la pose (posición y orientación) de la primera casilla del tablero "A1". A partir de este punto se crean el resto de las posiciones del tablero como puntos relativos, de modo que si en algún momento se desea modificar el tablero de posición bastaría con modificar el punto del origen.

```
local CONST robtarget robOrigTabl
```

- Origen blancas: Define la posición donde se encuentran situadas las piezas blancas fuera del tablero. Al igual que antes, a partir de este punto se definen los siguientes de manera relativa. Se define este punto porque una de las rutinas de este módulo permite que el robot coloque las piezas en el interior del tablero desde una posición exterior y viceversa, y también, porque cuando una de las piezas es comida, necesita un punto de dejada en el exterior del tablero.

```
local CONST robtarget robOrigBlancaExt
```

- Origen negras: Exactamente idéntico al anterior, pero en este caso para las piezas de color negro.

```
local CONST robtarget robOrigNegraExt
```

También se ha definido un "jointtarget" que establece la posición de reposo del robot. Un "jointtarget" especifica una posición deseada para los ejes (articulaciones) del robot. Se utiliza cuando se quiere gobernar el robot directamente a través de sus articulaciones y no especificando posiciones y orientaciones.

```
LOCAL VAR jointtarget jinit
```

[37]

Por otra parte, se declaran diferentes variables con una importancia relevante, como son:

- Distancia entre piezas: muy útil ya que si cambiamos el tamaño del tablero bastaría con modificar este parámetro.

```
LOCAL CONST num mTab:=50
```

- Posiciones de aproximación: se trata de un vector de tres posiciones que define diferentes alturas para realizar una aproximación correcta hacia las piezas.

```
LOCAL CONST num zPos{3}:=[220,100,45]
```

- Velocidades de movimiento y aproximación, se establecen diferentes velocidades para los movimientos amplios y para los movimientos precisos.

```
LOCAL CONST speeddata vel{2}:=[v20,v300]
```

- Un vector de piezas negras y blancas para tener todas localizadas: es importante que este módulo conozca donde están las piezas, ya que, si recibe una orden de movimiento hacia una casilla que está ocupada por una pieza, lo primero que debe hacer es retirarla antes de colocar la siguiente, si no el robot se estrellaría.

```
local pers num PiezaBlanca{8,2}
local pers num PiezaNegra{8,2}
```

- TCP/IP: se trata simplemente de una variable booleana que se emplea para determinar si se quiere utilizar comunicación TCP/IP con Matlab o no. En caso de que no, podríamos mover las piezas, pero no recibiríamos contestación de ningún motor de ajedrez.

```
LOCAL PERS bool TCPIP:=TRUE
```

- IP: determina la dirección IP en caso de que se vaya a comunicar.

```
LOCAL CONST string IPdir:="127.0.0.1"
```

- Puerto: establece el puerto de comunicación.

```
LOCAL CONST num IPport:=30000
```

- Sockets: declaración de los sockets servidor y cliente.

```
local VAR socketdev server_socket
local VAR socketdev client_socket
```

### Declaración de rutinas

El procedimiento de entrada a este módulo es “Menú mover piezas” que es llamado desde el módulo principal “Main”. A continuación, se describen con detalle cada una de las rutinas que lo componen, indicando en cada caso si se trata de un procedimiento o una función.

- Menú mover piezas: es el procedimiento encargado de mostrar un menú por la interfaz del robot o “FlexPendant”. Inicialmente se muestran las siguientes alternativas:

- Partida: permite comenzar o continuar una partida ya empezada. Si la opción “TCP/IP” está activada, se encarga de llamar al procedimiento “Abrir comunicación” y si está desactivada comenzaría una partida en modo manual.

- Movimiento de conjuntos: abre un nuevo menú que permite elegir entre las siguientes opciones:

  - Mover todas las piezas del exterior al tablero.
  - Mover todas las piezas del tablero al exterior.
  - Salir

- Movimientos individuales: sirve para mover piezas de manera individual. De nuevo, se abre un menú que permite elegir entre:

  - Mover una pieza del exterior al tablero
  - Mover una pieza del tablero al exterior
  - Mover pieza en el tablero

- Red: se encarga de setear la variable booleana “TCP/IP” para decidir si se quiere emplear la comunicación con Matlab o no.

- Salir

- Mover pieza: procedimiento encargado de mover una pieza de un origen a un destino. Tiene como argumentos de entrada dos “robtarget” y cuatro variables numéricas que son:
  - Robtarget: la posición de la casilla origen y la posición casilla destino
  - V. numéricas: “X” e ”Y” del origen y “X” e “Y” destino.

  De esta forma, este procedimiento se encarga de calcular las trayectorias definiendo puntos relativos de cogida y dejada de las piezas y pasando por el punto de reposo. También establece la velocidad de cada trayectoria según fueron definidas en la variable “speeddata vel [ ]”.

- Ext2Tabl: procedimiento encargado de mover una pieza de manera individual del exterior a su posición inicial en el tablero. Emplea la rutina “ColorXYext” para determinar la pieza (explicada más abajo).

- Piezas Ext2Tabl: procedimiento que gestiona la colocación de todas las piezas en el tablero desde el exterior.

- Tabl2Ext: similar a “Ext2Tabl” es el encargado de mover una pieza del tablero a su posición exterior. Emplea la rutina “XYtab2Pieza” explicada más adelante.

- Piezas Tab2Ext: procedimiento que mueve todas las piezas del tablero al exterior

- Tabl2Tabl: Uno de los procedimientos más importantes. Se encarga de gestionar los movimientos dentro del tablero. Por una parte, comprueba que en la casilla origen haya pieza y, por otra parte, verifica que en la casilla destino no haya pieza. Esto es fundamental ya que, si la casilla destino está ocupada, se debe retirar antes esa pieza al exterior del tablero antes de realizar el movimiento solicitado.

- ColorXYext: procedimiento encargado de pedir la pieza exterior que se desea tomar. Cada pieza está definida con tres dígitos que establecen respectivamente Color – Pos Columna – Pos Fila. Muestra una información de ayuda en la pantalla para el usuario y espera a que este introduzca un valor por pantalla.

- XYtab2Pieza: función que calcula la pieza que está en una posición del tablero. Devuelve la pieza en su formato Color-Xpieza-Ypieza. Por ejemplo, la torre blanca del flanco del rey sería 111.

- XYtab: procedimiento que solicita por pantalla al usuario una casilla del tablero en formato XY.

- Abrir comunicación: procedimiento de una importancia relevante ya que es el encargado de gestionar la comunicación por TCP/IP. Primero cierra posibles sockets que hayan quedado abiertos para evitar problemas. Después indica por pantalla la IP del robot y el puerto por el que se debe establecer la comunicación para informar al usuario. Finalmente, se encarga de abrir el socket servidor y queda a la espera de que se conecte un cliente.

- XYXYClient: se encarga de comunicarse con Matlab y espera una respuesta del socket cliente.

- XYXYtabl: función encargada de pedir un movimiento al usuario. Determina si esta información debe llegarle por socket o por pantalla, en función de si se está trabajando en modo red o no. Devuelve un “1” si todo ha sido correcto.

- Msg error: pequeño procedimiento útil para publicar errores por pantalla

Y finalmente, dos procedimientos habituales que realmente no tienen que ver con los anteriores pero que son necesarios para el correcto funcionamiento de la aplicación como son “Abrir_Pinza” y “Cerrar_Pinza”.

Para más información consultar el Anexo A: Código del proyecto

## 4.3 Aplicación Gráfica (Visual Studio)

El objetivo de esta parte del proyecto es diseñar una interfaz plenamente operativa que sea capaz de comunicarse con el módulo de Matlab para mostrar gráficamente toda la gestión del juego y poder interactuar directamente con el usuario e indirectamente con el robot.

Tras el estudio de alternativas realizado se determinó emplear la herramienta de “Visual Studio” para diseñar esta aplicación gráfica. Además, el lenguaje empleado para desarrollarla será “C#”.

La aplicación se ha dividido en diferentes pantallas o “Windows Forms” según la funcionalidad de cada una de ellas, y son detalladas a continuación.

### 4.3.1 Pantalla 1: Conexión

Esta es la pantalla inicial, es el punto de partida de la aplicación. Su función principal es la de establecer conexión con Matlab. Si no se logra este hito no se podrá pasar a la siguiente ventana.

Esta ventana tiene tres botones y una etiqueta de información como puede verse en la Figura 20.


*Figura 20 Pantalla 1*

- El primer botón es el de “Conectar”. Al hacer clic sobre él se gestiona el proceso de establecer comunicación con Matlab. Para ello, el servidor de Matlab debe estar activo, si no es así, se muestra un mensaje de error en la etiqueta informando al usuario de lo que debe hacer.

- El segundo botón es el de “Iniciar”. Este permanece inactivo mientras no se haya establecido la comunicación. Una vez logrado este hito se activa y permite pasar a la siguiente pantalla.

- El tercer botón es el de “cerrar” que se encarga de cerrar los procesos activos, cortar la comunicación con Matlab si es que se ha llegado a establecer y finalizar la aplicación.

### 4.3.2 Pantalla 2: Modo de juego

Esta segunda pantalla tiene como objetivo principal solicitar los datos al usuario del modo de juego que desea mediante un desplegable o lo que también se denomina en programación un “ComboBox”, como puede verse en la Figura 21.


*Figura 21 Pantalla 2*

En función de que opción sea elegida emergerán en el formulario otros desplegables para completar la información, además de unos cuadros de texto o “TextBox” para solicitar el nombre del jugador. Hasta que no esté completada toda la información necesaria para comenzar el juego no se habilitará el botón “Jugar” que dará acceso a la tercera pantalla.

A continuación, en la Figura 22 se puede ver un ejemplo para el caso de escoger el modo Humano-Robot:

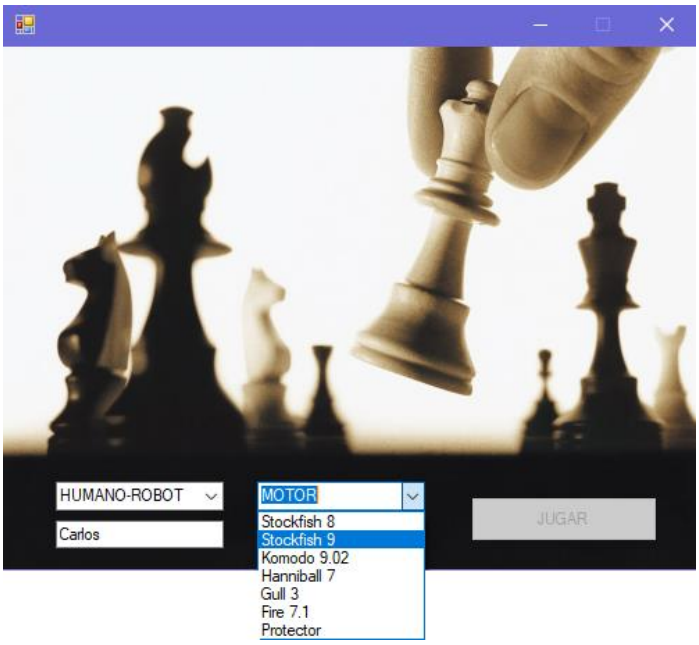



*Figura 22 Ejemplo pantalla 2, modo de juego Humano-Robot*

Como puede apreciarse, se muestra un desplegable para escoger un motor de ajedrez entre los disponibles y un cuadro de texto para introducir el nombre del jugador humano. Una vez completados ambos, se habilitará el botón "Jugar".

### 4.3.3 Pantalla 3: Tablero

Esta ventana se corresponde ya con el propio tablero de juego y este es el aspecto inicial que presenta (Figura 23):


*Figura 23 Pantalla 3*

El principal objetivo de esta pantalla es mostrar gráficamente el tablero de ajedrez, actualizando en todo momento los movimientos que se produzcan. Además,

permite interactuar al usuario con el sistema a través de dos cuadros de texto por los que se introduce la información del movimiento deseado (Figura 24).

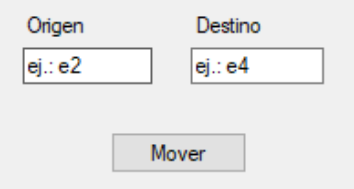



*Figura 24 Cuadro de interacción con el usuario*

Este cuadro se habilitará en el espacio correspondiente al jugador humano mientras que en el lado robot aparecerá un botón. Al clicar sobre él, se realiza una petición de respuesta al motor elegido (recordad que toda esta gestión la realiza Matlab).

Además, en esta pantalla se han implementado diferentes funcionalidades:

- ✓ Se indica el nombre del usuario y el nombre del motor.
- ✓ Comprueba que la sintaxis del movimiento introducido es correcta.
- ✓ Muestra un listado con todos los movimientos.
- ✓ Indica si hay jaque o jaque mate.
- ✓ Muestra las piezas comidas.
- ✓ Destaca la casilla de destino del último movimiento en color.

## 4.4 Diseño 3D

Siguiendo los pasos que se llevaron a cabo en el diseño previo, esta etapa del proyecto tendrá como objetivos diseñar y fabricar las piezas de ajedrez y la garra del robot.

### 4.4.1 Piezas de ajedrez

Como ya se explicó en el diseño previo, todas las piezas consistirán en una base y estructura cuadrada de modo que puedan ser cogidas adecuadamente con la herramienta del robot. Además, este cuerpo geométrico rectangular variará su altura en función de las piezas en el siguiente orden ascendente: peón, torre, caballo, alfil, dama y rey.

La parte superior queda reservada para el diseño de un distintivo que identifique que tipo de pieza es. A continuación, puede verse una captura del diseño CAD realizado de cada una de ellas:

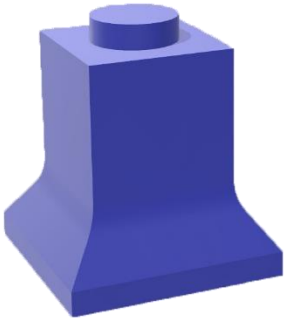


*Figura 25 Peón*

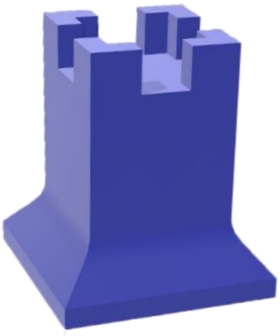


*Figura 26 Torre*

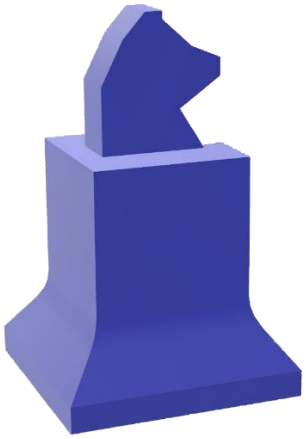


*Figura 27 Caballo*

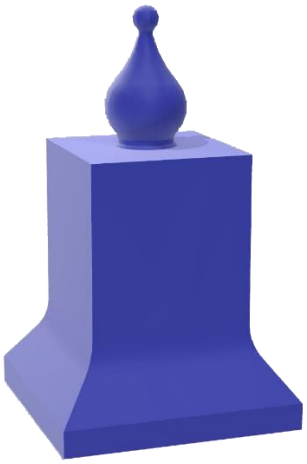


*Figura 28 Alfil*

*Figura 29 Dama*

*Figura 30 Rey*

Finalmente, se muestra dos capturas (Figura 31 y Figura 32) de todas las piezas juntas donde se puede apreciar mejor el tamaño relativo entre ellas.

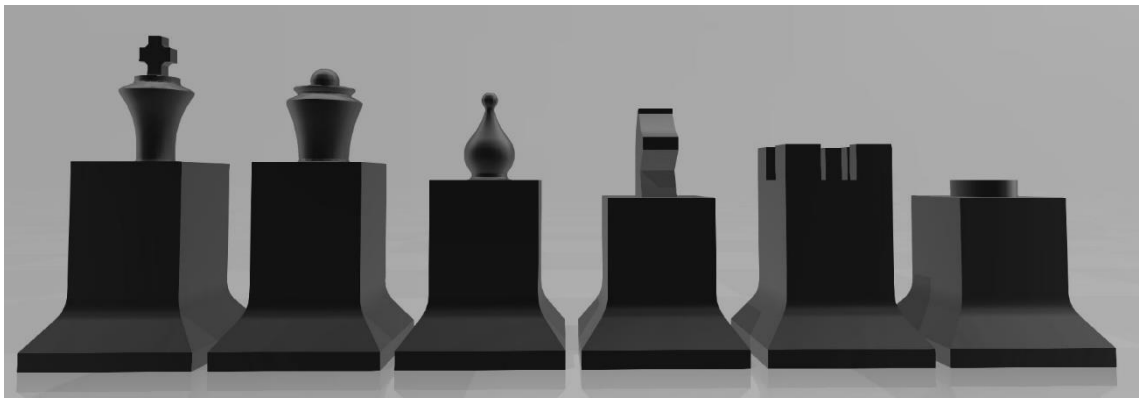


*Figura 31 Comparación de alturas de las piezas*

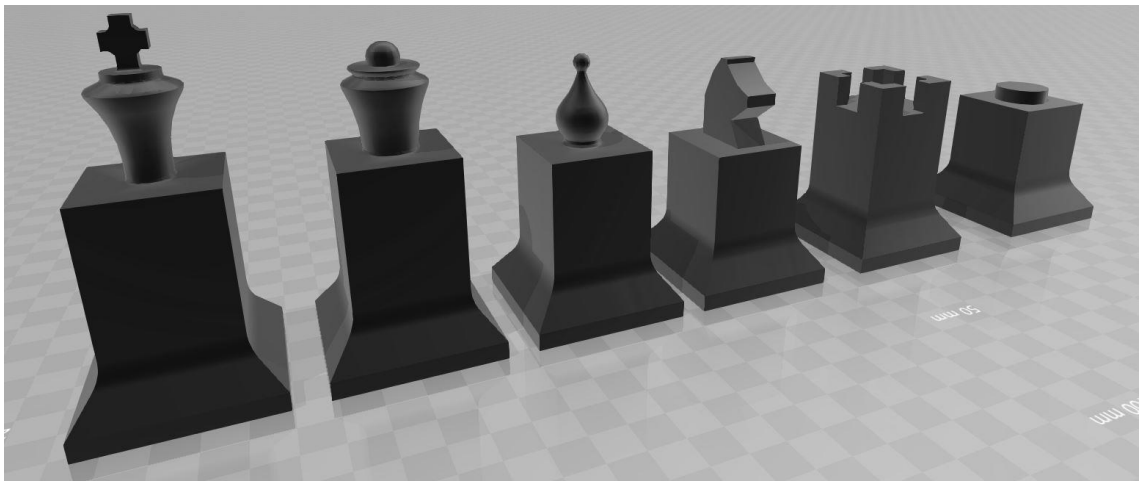


*Figura 32 Perspectiva de las piezas*

### 4.4.2 Garra del robot

Este es uno de los puntos más críticos de todo el proyecto. La herramienta con la que cuenta el robot por defecto es una pinza neumática de acero inoxidable con las palas planas como se puede ver en la Figura 33 y Figura 34.

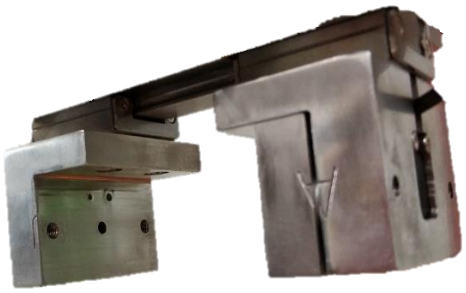


*Figura 33 Pinza con la que cuenta el robot*

Esto conlleva tres problemas principalmente:

- Al ser tan corta es imposible acceder a coger una pieza sin golpear a las adyacentes
- Por otra parte, al ser tan gruesa, no se podría realizar la acción de apertura y cierre con facilidad sin golpear a las piezas colindantes.
- Y lo que es más importante, al ser una pinza plana, no se puede asegurar que no haya error en la cogida y dejada de la pieza. Un pequeño desvío acumulativo de este tipo puede llevar a errores muy grandes a lo largo de la partida.

*Figura 34 Detalle de la pinza actual*

Por estas razones, se vio la necesidad de diseñar un modelo CAD de una herramienta adecuada para cumplir con estos requisitos. Además, como se desea mantener este mecanismo de pinza, ya que es la mejor manera de coger una pieza, se debe realizar un diseño adaptado que encaje en la herramienta actual, es decir, una extensión de la pinza.

Asimismo, como ya se justificó en la introducción, la cogida de la pieza se realizará a lo largo de la diagonal de su base, como se ve en la Figura 35.

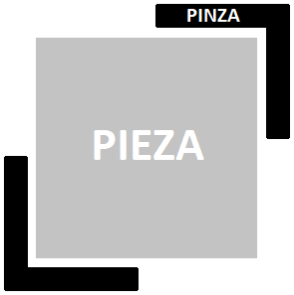



*Figura 35 Cogida de pieza*

Tras desarrollar el modelo con estas premisas, el CAD preliminar de la herramienta es el siguiente (Figura 36 y Figura 37):

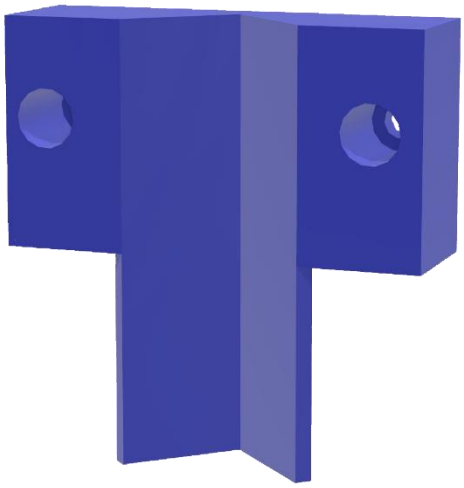


*Figura 36 Extensión de la pinza (1)*

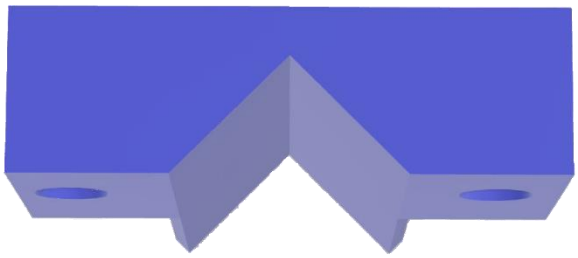


*Figura 37 Extensión de la pinza (2)*

Además, se ha modelado también la herramienta actual del robot para poder simular un ensamblaje y corroborar que encaja correctamente antes de fabricarla. En la Figura 38 puede verse como ensamblan la garra del robot con la extensión de la pinza diseñada mientras que en la Figura 39 se ha simulado incluso la cogida de una pieza (el rey porque es la más alta) para evitar posibles colisiones en la acción de cerrar pinza.

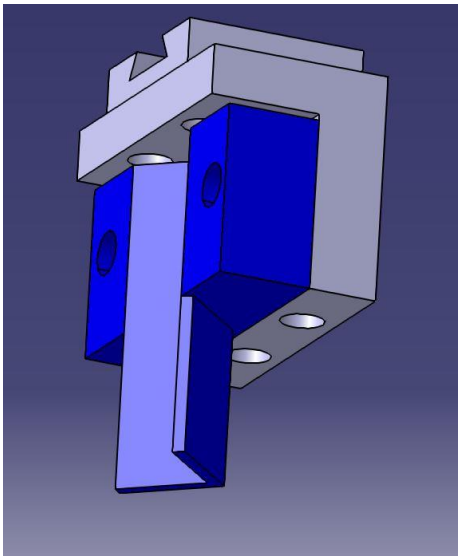


*Figura 38 Ensamblaje garra y extensión*

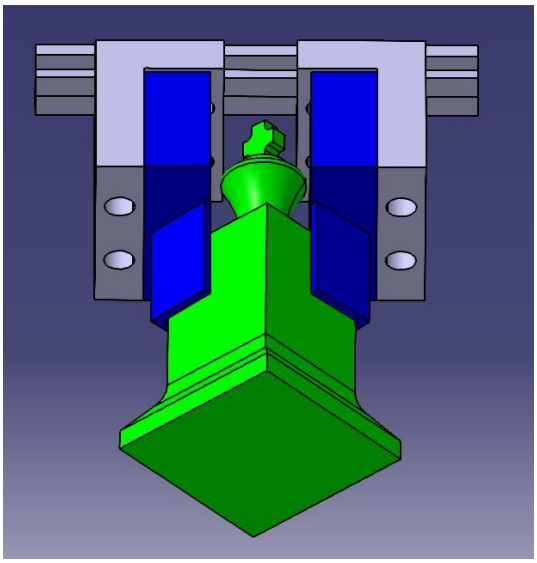


*Figura 39 Simulación de cogida de pieza*

### 4.4.3 Soporte piezas

Por último, se ha diseñado también un soporte para depositar las piezas fuera del tablero. Tiene un diseño individual y modular que permite encajar los bloques de formas muy diversas para adaptarse a cualquier entorno de trabajo. Pueden estar colocados por ejemplo en una sola fila, en dos filas, en un cuadrado o de formas muy diversas según sea el espacio disponible.

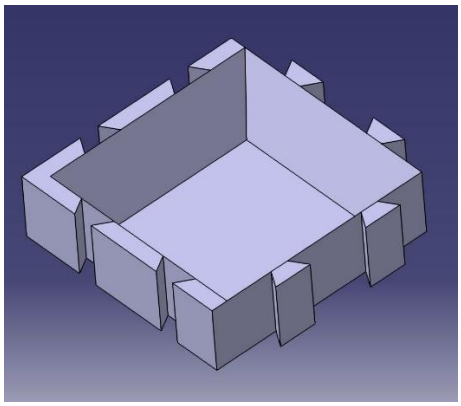


*Figura 40 Soporte de pieza individual*

Además, se ha ideado una pequeña rampa por los cuatro costados que conduce a la pieza a la base con una posición inequívoca. De este modo, no es necesario que la pieza sea dejada ni con mucha precisión ni velocidad reducida. En la Figura 41 puede verse una simulación realizada en Catia del ensamblaje entre diferentes soportes individuales y las piezas depositadas sobre ellos.

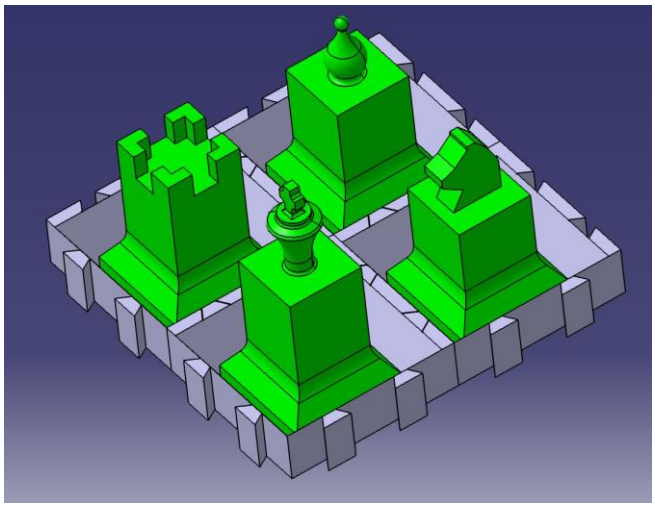


*Figura 41 Simulación del ensamblaje entre varios soportes con piezas*

# CAPÍTULO 5. RESULTADOS

Es un capítulo muy importante, ya que deja constancia de que se han alcanzado los objetivos iniciales. Aquí se van a describir los experimentos y pruebas reales que se han llevado a cabo para determinar el correcto funcionamiento de la plataforma.

Se van a seguir dos líneas básicas de actuación. Por un lado, se testeará toda la parte software que incluye la gestión del juego en Matlab y la interfaz gráfica y, por otro lado, se verificará la parte mecánica que incluye el sistema robótico y el modelado de piezas 3D y su fabricación.

## 5.1 Gestión de juego e Interfaz gráfica

*Matlab*

Comenzando por el software central del proyecto, en Matlab se ha logrado desarrollar el código que gestiona el juego. Simplemente, con ejecutar el "script" de lanzamiento "Ajedrez_TFM.m" se lanzan todos los módulos necesarios para el correcto funcionamiento de la aplicación. Se ha implementado una pequeña interfaz gráfica que permite elegir el modo de juego y visualizar el tablero de forma esquemática (Figura 42).

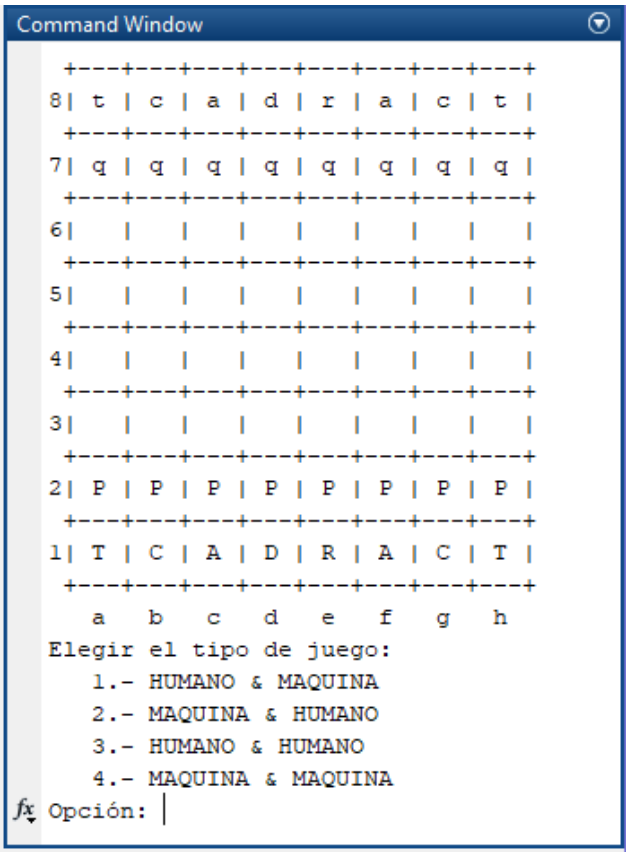



*Figura 42 Captura de la aplicación en Matlab*

En el caso de elegir un modo de juego en el que participe el robot como jugador, puede apreciarse como la aplicación logra comunicarse con los motores de ajedrez y ofrecer un movimiento de respuesta. En la Figura 43 puede verse un ejemplo del modo de juego Humano-Robot.

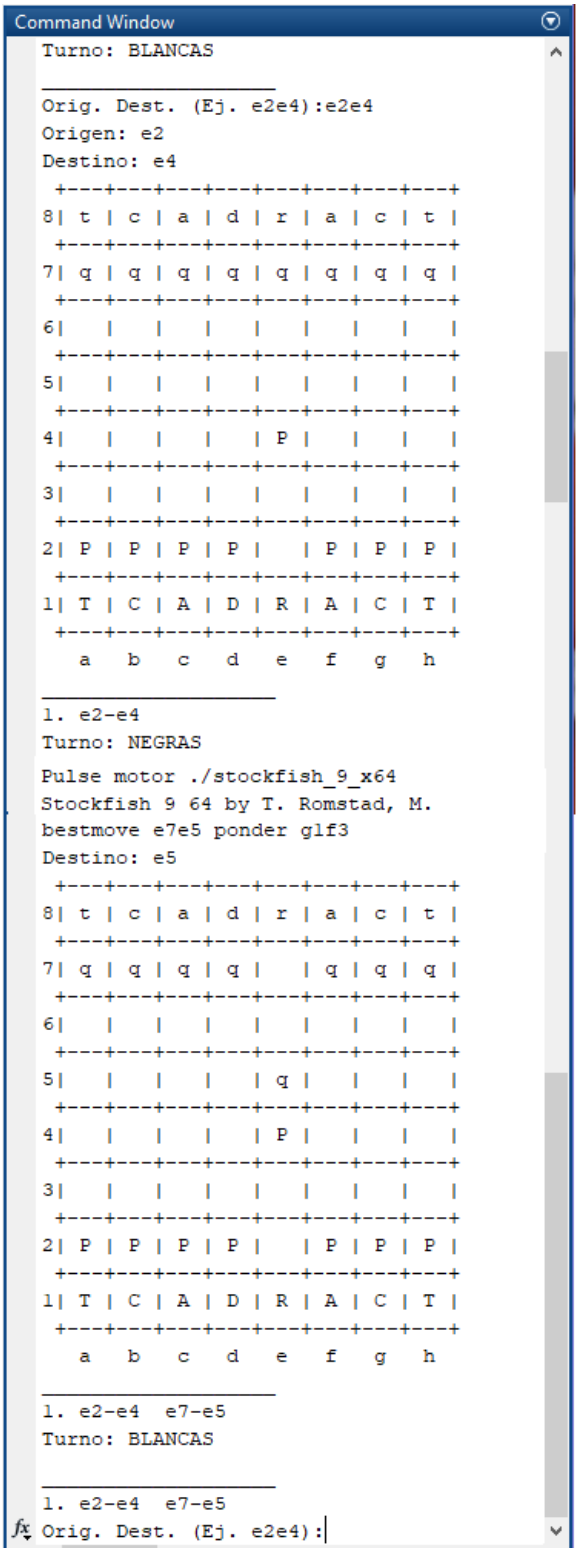



*Figura 43 Ejemplo Humano-Robot*

Tal y como se había definido en los objetivos del proyecto, se ha conseguido que la aplicación de Matlab gestione los aspectos del juego tales como, modos de juego, comunicación con los motores, turnos, movimientos, jaques y estado del tablero.

### *Visual Studio*

En el apartado 4.3 Aplicación Gráfica (Visual Studio), ya se mostró algunas capturas del desarrollo de la interfaz gráfica. No obstante, lo que se quiere demostrar aquí es el correcto funcionamiento de esta. En la Figura 44, por ejemplo, puede verse una imagen del aspecto que presenta la aplicación en una partida Humano-Robot.

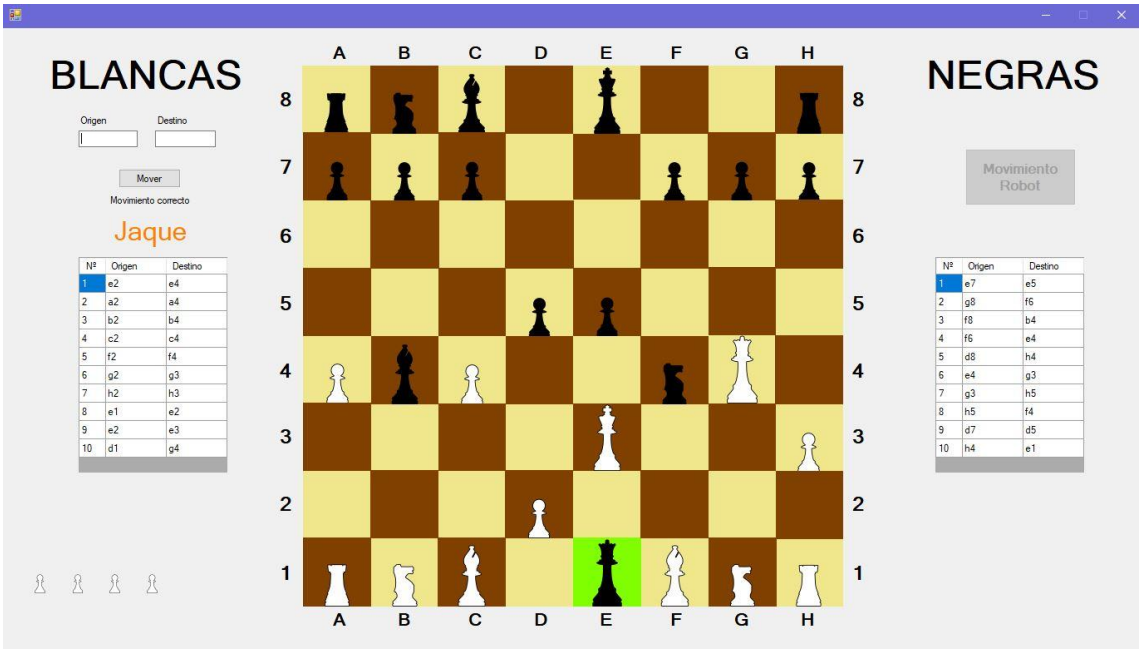



*Figura 44 Ejemplo partida Humano-Robot en la aplicación visual*

En este caso, se aprecia como en la parte izquierda, correspondiente al jugador humano, aparecen dos casillas para introducir los valores de las casillas de origen y destino del movimiento deseado mientras que, en la parte derecha, correspondiente al robot, hay un botón para ejecutar el movimiento. Además, en esta jugada puede verse la etiqueta que informa de que el rey blanco está en jaque. Por otra parte, aparecen las tablas que van registrando todos los movimientos de la partida. Asimismo, se colorea en verde la casilla destino del último movimiento ejecutado.

Otro ejemplo, se muestra en la Figura 45, en este caso hace referencia a una partida en modo Robot-Robot.

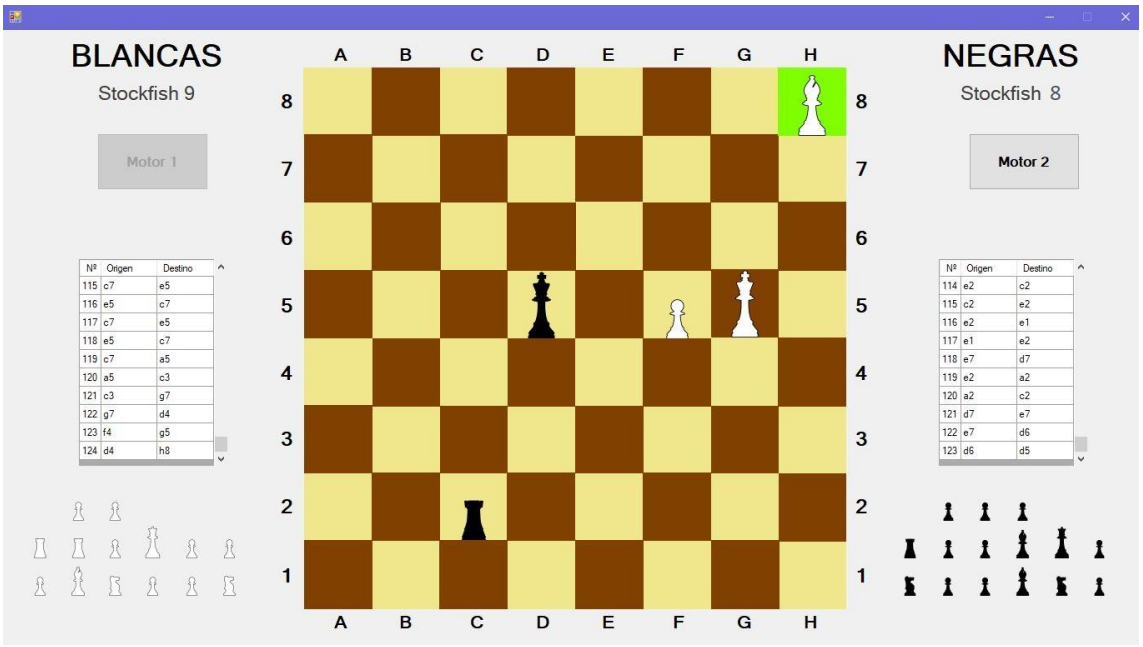



*Figura 45 Ejemplo partida Robot-Robot en la aplicación visual*

Aquí se aprecia mejor como se muestran en la parte inferior las piezas comidas de cada uno de los jugadores. En este caso, al ser la partida Robot-Robot, se cuenta con dos botones para ejecutar los movimientos uno a cada lado que se irán habilitando y deshabilitando automáticamente en función de los turnos.

## 5.2 Sistema robótico y diseño 3D

En este apartado se van a mostrar tanto las simulaciones del entorno de trabajo que se han llevado a cabo en el sofware de "RobotStudio" como imágenes reales del resultado final de la plataforma.

### 5.2.1 Simulación

Los resultados aquí obtenidos han permitido llevar a la estación real todo lo programado y diseñado en el proyecto. La etapa de simulación es una de las más importantes en la mayoría de los proyectos y, mucho más aún, en el campo de la ingeniería. Gracias al software de "RobotStudio" se ha podido modelar el entorno de trabajo incluidas las piezas y la garra del robot para poder definir las trayectorias con seguridad y evitar posibles colisiones.

En la siguiente imagen (Figura 46) aparece simulado en el entorno gráfico de "RobotStudio" el tablero junto con las piezas diseñadas.

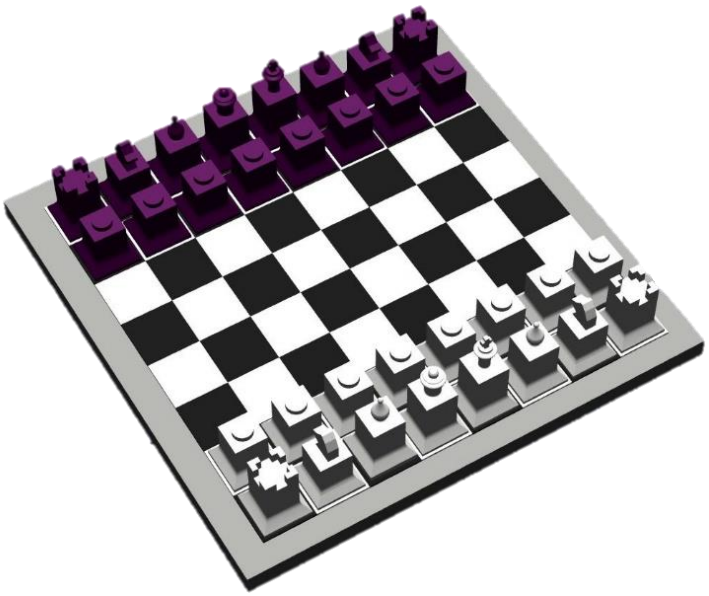


*Figura 46 Tablero con piezas simulado*

En la Figura 47 puede verse el aspecto que tiene la estación completa simulada en "RobotStudio". Apréciese que se ha incluido la pinza nueva a la herramienta del robot, para poder simular las cogidas y movimientos de las piezas.

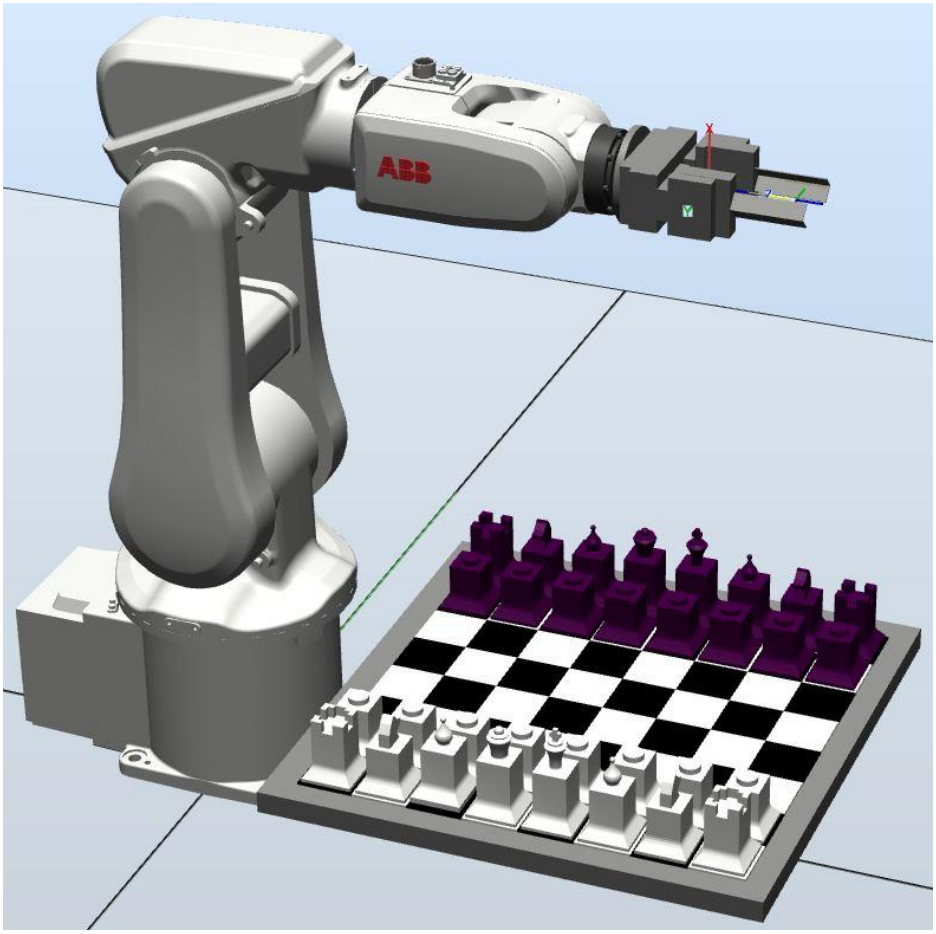



*Figura 47 Aspecto de la estación simulada en RobotStudio*

A continuación, se muestra una captura de como quedaría las piezas colocadas fuera del tablero.

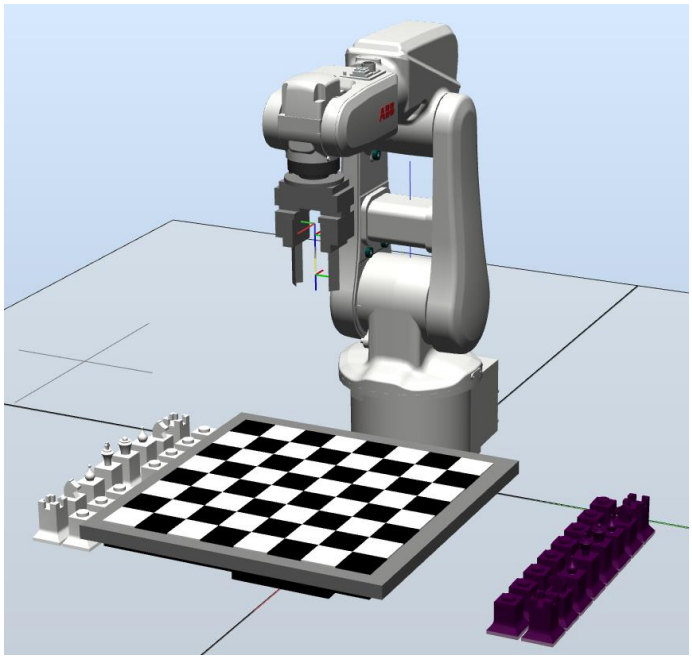


*Figura 48 Captura de la estación simulada con las piezas en el exterior*

Las siguientes tres imágenes (Figura 49, Figura 50, Figura 51) muestran la simulación de una de las trayectorias, para demostrar que se han comprobado todas y cada una de ellas para corroborar que no se va a producir ninguna colisión.

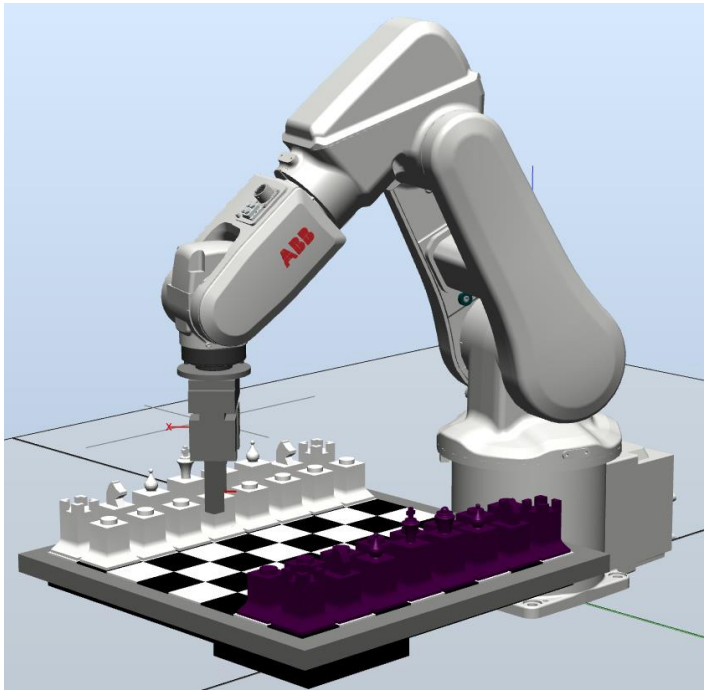



*Figura 49 Cogida de pieza*

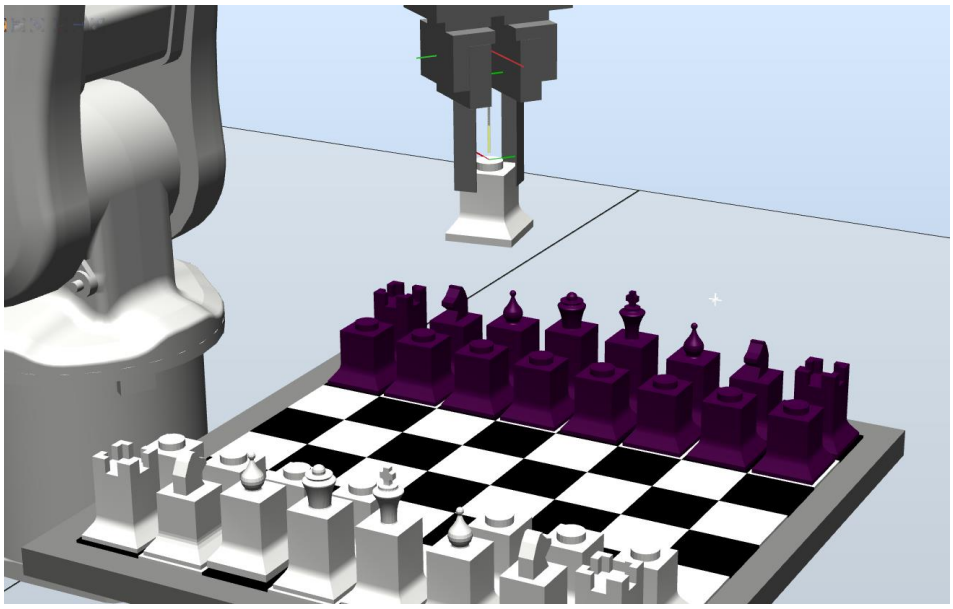


*Figura 50 Transporte de pieza*

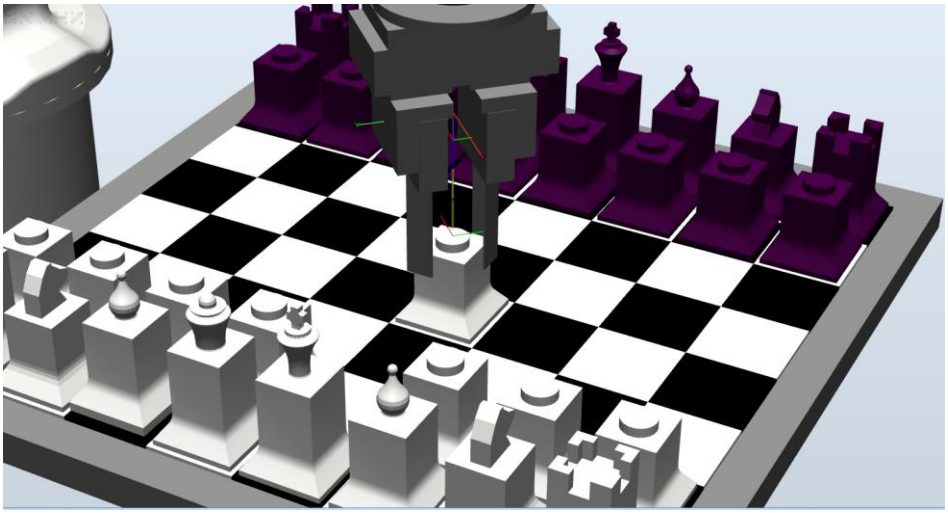


*Figura 51 Dejada de pieza*

### 5.2.2 Real

Una vez comprobados los resultados en la simulación, el siguiente paso es realizar pruebas sobre la estación real. Primero se van a mostrar los resultados conseguidos en la fabricación de las piezas y de la pinza y, posteriormente, cómo realiza el robot su actividad sobre la plataforma.

#### *Piezas*

Las piezas han sido fabricadas por una empresa externa a la UVa especialista en el sector de la impresión 3D y que se llama "Ecoespacio 3D Print". Las especificaciones de fabricación que se exigieron fueron las siguientes:

- Altura de capa de 0,2 mm.
- Material de impresión PLA blanco y negro.
- Patrón de relleno diagonal.

- Densidad del relleno 30%. Con esto se consigue que las piezas no sean muy ligeras.
- Paredes exteriores de 4 capas, para ofrecer más consistencia a las piezas.

Los resultados obtenidos se pueden ver en las siguientes imágenes (Figura 52 y Figura 52).

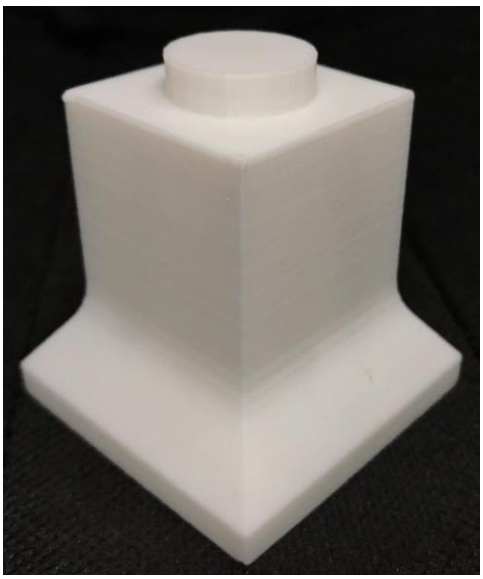


*Figura 52 Peón real (1)*

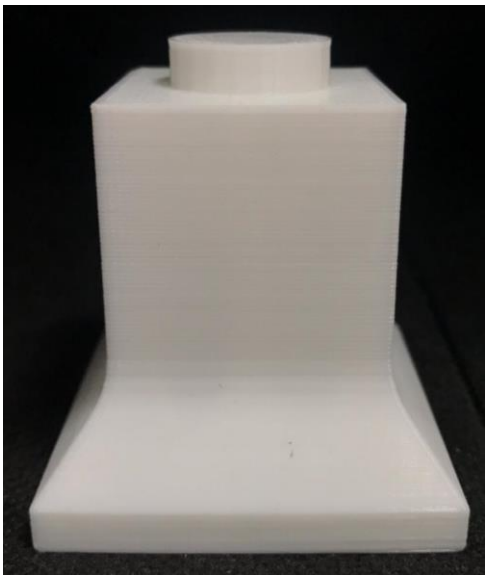


*Figura 53 Peón real (2)*

Las siguientes (Figura 54 Figura 54) imágenes muestran las piezas dispuestas sobre el tablero en una partida real.

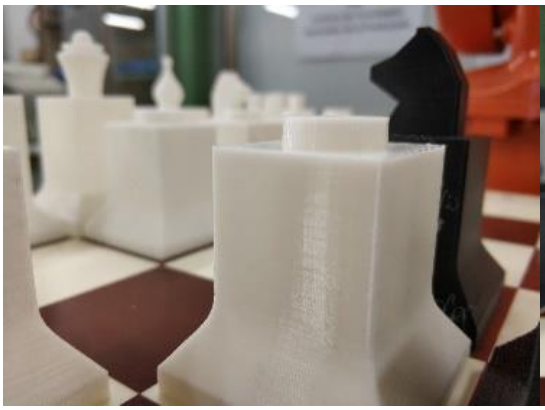


*Figura 54 Figuras sobre tablero (1)*

*Figura 55 Figuras sobre tablero (2)*

Por otro lado, se imprimió en la facultad una prueba de los soportes diseñados para depositar las piezas en el exterior del tablero. La diferencia de calidad es notable, pero simplemente se trataba un prototipo.

*Figura 56 Resultado de soporte de piezas*

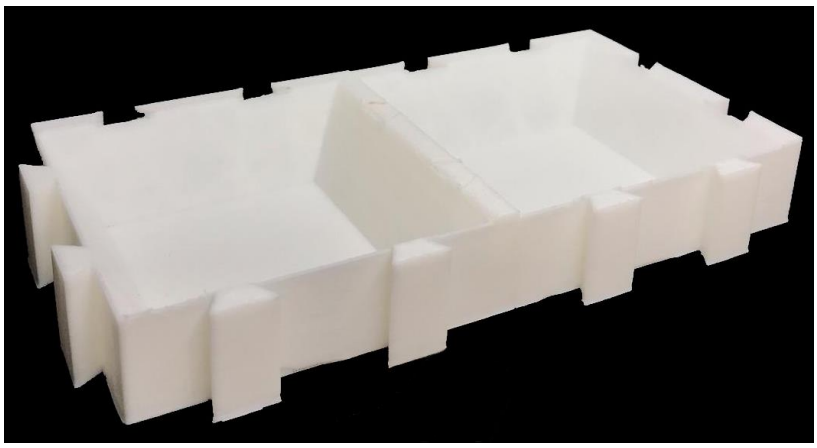


*Figura 57 Resultado del ensamblaje soporte de piezas*

### *Pinza*

La idea principal era fabricar la pinza al igual que las piezas mediante tecnología de impresión 3D. No obstante, se optó por fabricarla de forma artesanal con materiales más robustos como el acero. Esto aporta una mayor resistencia y robustez a la hora de trabajar sobre la estación real. En la Figura 58 y Figura 59 puede verse el resultado.

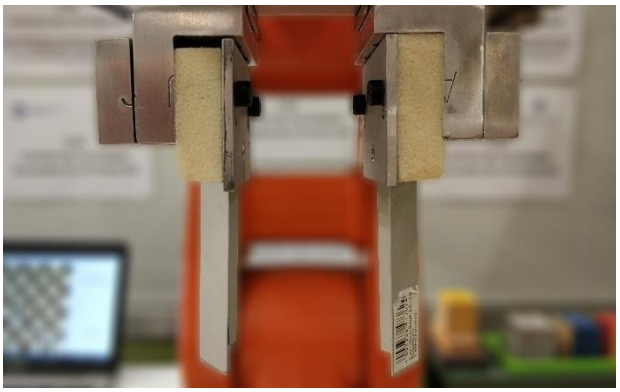


*Figura 58 Pinza fabricada (1)*

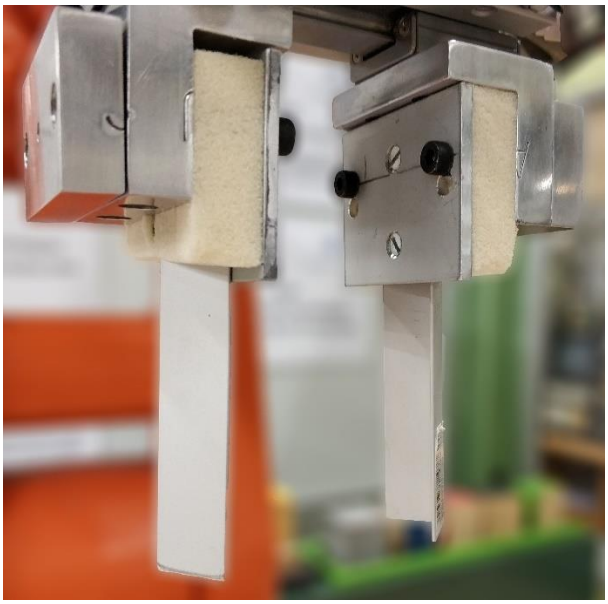


*Figura 59 Pinza fabricada (2)*

*Estación*

Para comenzar, se muestra dos fotografías del aspecto que presenta la estación real:

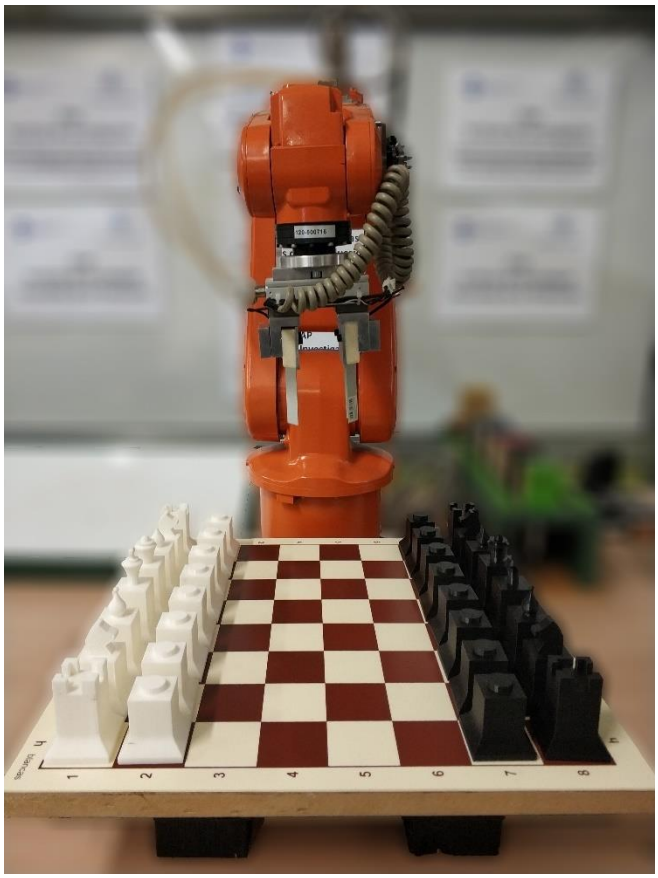


*Figura 60 Fotografía de la estación real (1)*


*Figura 61 Fotografía de la estación real (2)*

El aspecto más importante de estas pruebas es la cogida y dejada de piezas con la pinza. En la Figura 62 puede verse al robot ejecutando una de las trayectorias con una de las piezas cogidas al igual que se hizo en la simulación. Los resultados de cogida y dejada son muy precisos, lo cual supone que tanto el diseño de piezas y pinza como la programación de trayectorias ha sido óptima.

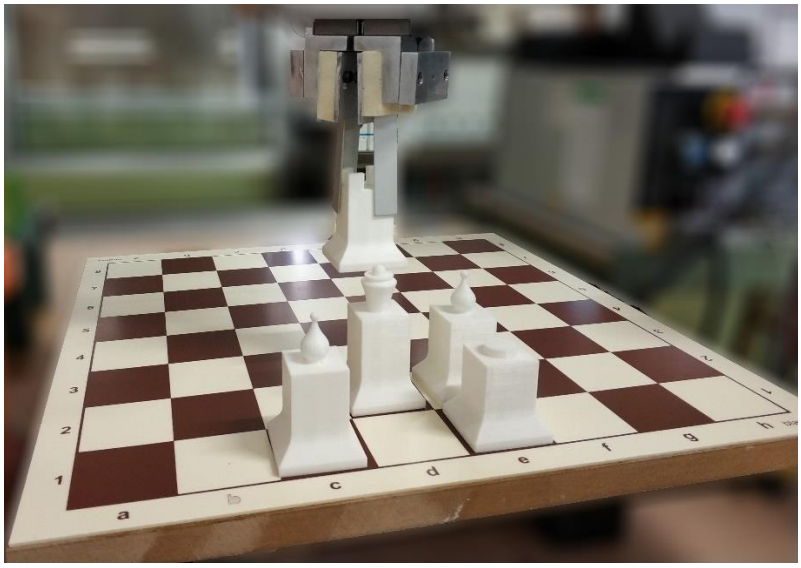


*Figura 62 Cogida de pieza*

Finalmente, este es el aspecto que presenta la plataforma en su conjunto con el PC, el HMI (que podría estar en otra pantalla si se desea) y la estación robótica en mitad de una partida real (Figura 63).

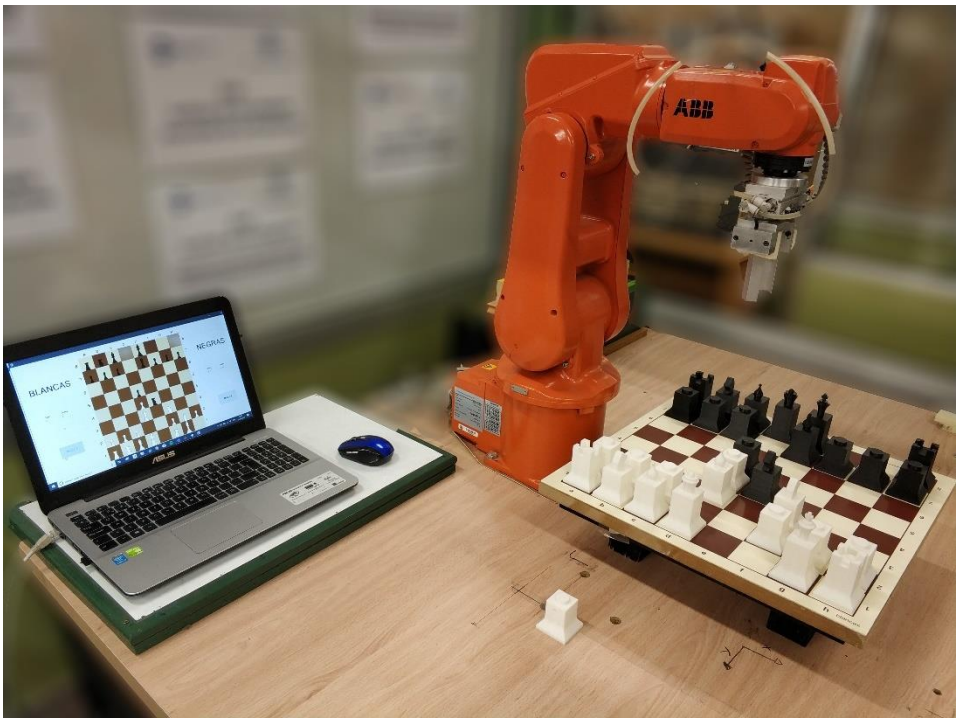



*Figura 63 Aspecto que presenta la estación en su conjunto*

# CAPÍTULO 6. MANUAL DE USUARIO

Se dispone de un comprimido "AJEDREZ_TFM.rar" que contiene los siguientes archivos:

- **"miAjedrez.exe"**: Se trata de la interfaz gráfica hombre-máquina.
- **"Ajedrez_TFM.m"**: Script de lanzamiento para Matlab.
- **"miAjedrez.m"**: Clase de Ajedrez que gestiona el juego en Matlab.
- **"UCIEngine.m"**: Clase encargada de gestionar los motores de ajedrez
- **"Estacion_Abril_2019"**: Esta es la estación de RobotStudio donde se encuentra modelada la plataforma en su conjunto, con tablero, piezas y pinza. Además, contiene la programación en RAPID del robot.
- **"[motores].exe"**: Aparecen los ejecutables de todos los motores de ajedrez.

*PASOS A SEGUIR:*

1. Abrir "Ajedrez_TFM.m" en Matlab. Hay dos variables a nivel de programación ("app_visual" y "robot_studio") que pueden ser modificadas para elegir qué módulos de la aplicación van a ser utilizados:

| MÓDULOS A UTILIZAR | "app_visual" | "robot_studio" |
|---|---|---|
| a) Matlab | 0 | 0 |
| b) Matlab y robot | 0 | 1 |
| c) Matlab y HMI | 1 | 0 |
| d) Matlab, HMI y robot | 1 | 1 |

2. Si has elegido la opción b), ejecuta "Ajedrez_TFM.m". Te pedirá que ejecutes el programa de RobotStudio y cliques en "Partida" con la opción de "RED=on". Una vez esté listo el socket del robot, pulsar "Enter" en Matlab. Listo.

3. Si has elegido la opción c), ejecuta "Ajedrez_TFM.m" en Matlab y lanza la aplicación "miAjedrez.exe" haciendo doble clic sobre ella. Seguir las instrucciones que se indican en la propia aplicación.

4. Y en el caso de haber escogido la opción d), ejecuta "Ajedrez_TFM.m". Primero te pedirá lanzar RobotStudio y después la aplicación de Visual. Seguir los pasos 2 y 3.

# CAPÍTULO 7. ESTUDIO ECONÓMICO

En este capítulo se llevará a cabo una estimación del coste del proyecto al mismo tiempo que una enumeración de los materiales empleados. Se agruparán los costes según el origen y serán divididos en costes directos e indirectos. Finalmente, se detallará el presupuesto final.

## 7.1 Costes directos

Los costes directos son aquellos relacionados con un producto o un servicio, por lo que en este apartado se incluyen los costes de material, construcción, recursos y herramientas empleadas.

### 7.1.1 Costes asociados al equipo

Dentro de este apartado se distinguen: costes de material y equipo, de herramientas y de software.

*Costes de Material y equipo*

Los costes del material y equipo necesario para la construcción del robot pueden verse en la Tabla 2.

*Tabla 2 Costes de Material y equipo*

| Costes de Material y equipo | | | | |
|---|---|---|---|---|
| **Concepto** | Procedencia | Coste unitario (€) | Unidades | Coste total (€) |
| **Robot ABB IRB120 junto con el software Software RobotStudio Armario ABB IRC5** | ABB | 15.000,00 (precio educación) | 1 | 15.000,00 |
| **Piezas Ajedrez** | Ecoespacio 3D Print | ~9 € / pieza | 32 | 288 |
| | | | Total | 15.288,00 |

*Costes de herramientas*

En este apartado se incluyen los costes derivados del uso de herramientas necesario para el desarrollo del proyecto. Se detallan en la Tabla 3.

*Tabla 3 Costes de herramientas y equipo*

| Coste de herramientas | | | |
|---|---|---|---|
| **Concepto** | Coste unitario (€/h) | Unidades (horas) | Coste total (€) |
| **Banco de trabajo (herramientas varias)** | 2,32 | 10 | 23,20 |
| **Material de laboratorio** | 0,75 | 20 | 15,00 |
| **Ordenador** | 0,29 | 900 | 261,00 |
| | | Total | 299,20 |

### *Costes de software*

En este apartado se detallan los costes de software utilizados a lo largo del proceso. No obstante, cabe destacar que, debido al ámbito educativo en el que se desarrolla el proyecto, la mayoría de los programas informáticos empleados no han tenido coste alguno ya que la Uva cuenta con licencias para estudiantes, por tanto, el coste total será cero. Se puede ver una descripción más detallada en Tabla 4, además de un número aproximado de horas de utilización de cada uno de ellos.

*Tabla 4 Costes de software*

| Coste del software empleado | | | |
|---|---|---|---|
| **Concepto** | Coste Real licencia(€/año) | Coste Alumno UVa licencia | Coste total (€) |
| **MATLAB** | 500 | 0,00 | 0,00 |
| **RobotStudio (incluido en el equipo)** | - | - | - |
| **Visual Studio** | 641 | 0,00 | 0,00 |
| | | Total | 0,00 |

### *Resumen de costes asociados al equipo*

En la Tabla 5 pueden verse los costes generales asociados a cada apartado y su suma total.

*Tabla 5 Suma de costes asociados al equipo*

| Resumen de costes asociados al equipo | | |
|---|---|---|
| **Concepto** | | Importe (€) |
| **Coste material y equipo** | | 15.288,00 |
| **Coste de herramientas** | | 299,20 |
| **Coste del software empleado** | | 0,00 |
| | Total | 15.587,20 |

### 7.1.2 Coste asociados a la mano de obra

En este apartado se van a estimar los costes asociados a la mano de obra necesarios para la programación y puesta en marcha del robot. Los costes del montaje de los diferentes componentes mecánicos, eléctricos y electrónicos pueden ser abaratados si este trabajo lo realiza un titulado en formación profesional, cuya mano de obra es más barata que la de un ingeniero.

En primer lugar, se calcularán las horas de trabajo que hay en un año y, de esta manera se podrá estimar el suelo por horas y ajustar el pago al tiempo que ha sido necesario para realizar cada parte del trabajo (Tabla 6).

*Tabla 6 Cálculo de las horas laborables en un año*

| Coste de la mano de obra por tiempo empleado | | |
|---|---|---|
| **Concepto** | | Tiempo (días) |
| **Año medio** | | 365 |
| **Vacaciones** | | 22 |
| **Días festivos** | | 14 |
| **Días perdidos (aprox.)** | | 5 |
| **Fines de semana** | | 104 |
| | Total | 220 |
| | x 8 horas/día | 1760 h laborables/año |

Una vez calculadas las horas laborables por un año se tasará el sueldo que cobra un ingeniero a la hora, teniendo en cuenta el sueldo medio de estos profesionales en España con un nivel medio de experiencia. Este cálculo puede verse en la Tabla 7.

*Tabla 7 Cálculo del sueldo por horas de un ingeniero*

| Cálculo del sueldo de un ingeniero por hora | | |
|---|---|---|
| **Concepto** | | Importe (€) |
| **Sueldo bruto e incentivos** | | 27.600,00 |
| **Sueldo neto e incentivos** | | 25.806,00 |
| **S.S. a cargo de la empresa** | | 8.197,00 |
| | Sueldo Total | 35.797,00 |
| | Sueldo por hora | 20,34 |

Una vez conocido el sueldo por horas se estimarán las horas necesarias que ha de realizar el ingeniero y con ello se estimarán los costes totales de personal.

El tiempo de desarrollo del proyecto ha sido de seis meses. Antes se ha calculado que en un año las horas laborables son 1760, por lo que las horas laborables en 6 meses trabajando 8 horas diarias serán 880 repartidas como muestra la Tabla 8.

*Tabla 8 Costes de personal*

| Costes de personal | | | |
|---|---|---|---|
| **Concepto** | Coste por hora (€/h) | Unidades | Coste Total (€) |
| **Software MATLAB** | 20.34 | 350 | 7.119,00 |
| **Programación Robot ABB** | 20.34 | 280 | 5.695,20 |
| **Interfaz Gráfica** | 20.34 | 200 | 4.068,00 |
| **Diseño 3D** | 20.34 | 50 | 1.017,00 |
| | Total | 880 | 17.899,20 |

### *Total costes directos*

Puede verse un resumen de los gastos que suponen los costes totales en la Tabla 9.

*Tabla 9 Total de costes directos*

| Total de costes directos | | |
|---|---|---|
| **Concepto** | | Importe (€) |
| **Costes asociados al equipo** | | 15.587,20 |
| **Coste de personal directo** | | 17.899,20 |
| | Total | 33.486,40 |

## 7.2 Costes indirectos

Los costes indirectos son aquellos que corresponden a recursos consumidos durante la realización del proyecto pero que no pueden asociarse directamente a la realización de este. Se incluyen, por ejemplo, los gastos de electricidad, agua, climatización, etc. En la Tabla 10 se muestra un cálculo estimado.

*Tabla 10 Total de costes indirectos*

| Costes Indirectos | | |
|---|---|---|
| **Concepto** | | Importe (€) |
| **Consumo eléctrico de herramientas y equipos** | | 45,00 |
| **Otros consumos eléctricos (calefacción, iluminación, etc.)** | | 75,00 |
| | Total | 120,00 |

## 7.3 Coste total del proyecto

Finalmente, en la Tabla 11 se realiza un resumen del total de costes que supone la realización del proyecto, tanto directos como indirectos.

*Tabla 11 Coste total del proyecto*

| Coste total del proyecto | |
|---|---|
| **Concepto** | Importe (€) |
| **Costes directos** | 33.486,40 |
| **Costes indirectos** | 120,00 |
| Total | 33.606,40 |

# CAPÍTULO 8. CONCLUSIONES

Este capítulo contiene las conclusiones que se han obtenido tras la realización del proyecto, además de una serie de líneas de mejora, que puedan ser útiles en proyectos futuros sobre esta plataforma.

Lo primero y más importante, es que se han logrado superar con éxito todos y cada uno de los objetivos que se marcaron inicialmente. Esto es algo que resulta gratificante, teniendo en cuenta el alcance del proyecto y la cantidad de técnicas y conocimientos que ha sido necesario aplicar.

Se ha llevado a cabo desarrollo y la implantación en un PC de un software capaz de gestionar una partida de ajedrez y comunicarse con motores mediante el protocolo UCI.

En la estación robótica se han implementado las trayectorias y algoritmos que permiten mover las piezas de ajedrez sobre un tablero real. Este programa desarrolla la capacidad de saber si se trata de un movimiento de captura para que, en caso afirmativo, retire primero la pieza que se encuentra en la casilla de destino.

Se ha diseñado una interfaz gráfica que desempeña la funcionalidad de HMI (Human-Machine Interface). No solo permite visualizar el tablero de ajedrez en tiempo real, si no también, interactuar con las piezas y el robot.

La comunicación mediante sockets empleando el protocolo TCP/IP ha permitido la integración de las diferentes partes del proyecto. Esta característica modular del software hace posible ejecutar cada una de las partes en diferentes máquinas que, incluso, no tienen por qué estar presentes en el mismo lugar físico.

Se ha realizado el diseño asistido por ordenador de todas las piezas de ajedrez y de la pinza del robot con el objetivo de lograr altas precisiones en las trayectorias y, sobre todo, en las acciones de cogida y dejada de piezas.

En el ámbito de la simulación, se ha modelado el entorno de trabajo en el software de ABB “RobotStudio”. Esto ha permitido verificar las trayectorias programadas antes de probarlo en la estación real para evitar colisiones. También, se han simulado en Catia los ensamblajes de la pinza con la garra actual y con las piezas.

Para concluir, se introducen, a continuación, una serie de líneas de mejora para que puedan ser tenidas en cuenta en futuros proyectos de este tipo.

En la parte software del ajedrez sería interesante incluir sistemas de “Deep Learning” dentro de la Inteligencia Artificial (AI). Como se ha explicado en este documento, están apareciendo motores de ajedrez que se basan en redes neuronales como “Alpha Zero” de “Deep Mind”.

En la aplicación gráfica las posibilidades que caben son muy amplias. Se podrían introducir algunas mejoras tales como:

- ✓ Emplear el “clic” del ratón para seleccionar los movimientos de las piezas en lugar de utilizar cuadros de texto como actualmente.

- ✓ Dotar a la aplicación de la capacidad de guardar los datos de las partidas y poder reproducirlas posteriormente.

- ✓ Mostrar información y estadísticas de los jugadores y/o motores de ajedrez.

- ✓ Poder crear perfiles y registrarse.

Incluso, se podría plantear la posibilidad de desarrollar la interfaz gráfica para que se ejecute sobre un "smartphone" y, así, poder interactuar con el sistema desde el móvil.

Por otra parte, una de las grandes líneas de mejora de este proyecto sería dotar al sistema de la capacidad de ser extensible a cualquier robot industrial. Esto se podría hacer mediante "RoboDK" un software de simulación y programación que contiene en sus librerías multitud de robots de las principales marcas de la industria (ABB, KUKA, FANUC, UR…). La idea sería que el modelo de robot fuera totalmente transparente para el software que lleva a cabo la gestión del ajedrez. "RoboDK" actuaría como nexo entre Matlab y el robot.

En cuestión de entretenimiento, una idea interesante sería generar "puzzles" de ajedrez por niveles, es decir, jugadas muy determinadas que se proponen al usuario para resolver. El robot se encargaría de colocar las piezas necesarias desde el exterior al interior del tablero y una vez preparado el "puzzle" comenzaría el juego.

Finalmente, en la parte robótica del proyecto uno de los avances más interesantes sería dotar a la plataforma de un sistema de visión artificial 2D, que sea capaz de visualizar e interpretar el estado del tablero para informar al PC. De este modo, se tendría una supervisión continua mediante la cámara lo que evitaría problemas como:

- Mal posicionamiento de alguna de las piezas sobre la casilla.
- Factores externos que puedan mover las piezas y descolocarlas en el tablero.
- Movimientos que se han ordenado ejecutar desde el controlador pero que no se han llevado a cabo por motivos diversos como, por ejemplo, fallos de comunicación.

Incluso, en el caso de contar con un robot colaborativo, el usuario no tendría por qué depender del HMI si no que podría actuar él mismo sobre las piezas en el tablero real. En definitiva, con esta mejora pasaríamos de tener un sistema en lazo abierto (LA) a un sistema supervisado en lazo cerrado (LC).

# BIBLIOGRAFÍA


[1] S. J. M. Á. Mato, «Simulación, control cinemático y dinámico de robots comerciales usando la herramienta de Matlab, "Robotic Toolbox".,» Valladolid, 2014.

[2] J. A. Á. Herrero, «Modelado de una célula robótica con fines educativos usando el programa "RobotStudio",» Valladolid, 2015.

[3] Á. G. d. l. Santos, «Control remoto de una célula robotizada con fines educativos mediante tecnología WIFI,» Valladolid, 2016.

[4] «Ajedrez,» [En línea]. Available: https://es.wikipedia.org/wiki/Ajedrez.

[5] «Historia del ajedrez,» [En línea]. Available: https://es.wikipedia.org/wiki/Historia_del_ajedrez.

[6] «Historia del origen del ajedrez,» [En línea]. Available: https://users.dcc.uchile.cl/~jegger/ajedrez/HistoriaAjedrez.htm.

[7] R. A. D.-P. Arenas, Ajedrez, Ajedrecistas y la vida, CHILE: Atelí Ltda, 1995.

[8] «Eslovaquia News,» [En línea]. Available: https://eslovaquianews.wordpress.com/2015/07/07/21-inventos-eslovacos/el-turco/.

[9] «El Turco,» [En línea]. Available: https://es.wikipedia.org/wiki/El_Turco.

[10] «El Ajedrecista,» [En línea]. Available: https://es.wikipedia.org/wiki/El_Ajedrecista.

[11] «ecured,» [En línea]. Available: https://www.ecured.cu/Notaci%C3%B3n_algebraica.

[12] «ichess,» Julio 2018. [En línea]. Available: https://www.ichess.es/blog/notacion-ajedrecistica-guia-principiantes/.

[13] «Notación algebraica,» [En línea]. Available: https://es.wikipedia.org/wiki/Notaci%C3%B3n_algebraica.

[14] «Minimax,» [En línea]. Available: https://es.wikipedia.org/wiki/Minimax.

[15] «Juego de suma cero,» [En línea]. Available: https://es.wikipedia.org/wiki/Juego_de_suma_cero.

[16] «Poda alfa-beta,» [En línea]. Available: https://es.wikipedia.org/wiki/Poda_alfa-beta.

[17] «Negamax,» [En línea]. Available: https://es.wikipedia.org/wiki/Negamax.

[18] «Negascout,» [En línea]. Available: https://es.wikipedia.org/wiki/Negascout.

[19] «blogspot,» [En línea]. Available: http://ajedrezga.blogspot.com/2015/04/explicacion-de-como-funciona-un-motor.html.

[20] S. G. d. l. Torre. [En línea]. Available: http://nowinchess.com/2015/01/23/como-utilizar-un-motor-de-analisis-1/.

[21] NETFLIX, *AlphaGO.*

[22] R. Rodriguez, «La Vanguardia,» [En línea]. Available: https://www.lavanguardia.com/deportes/otros-deportes/20171214/433624379301/alpha-zero-deep-mind-gary-kasparov-ajedrez-inteligencia-artificial.html.

[23] «AlphaZero,» [En línea]. Available: https://es.wikipedia.org/wiki/AlphaZero.

[24] «Motor de Ajedrez,» [En línea]. Available: https://es.wikipedia.org/wiki/Motor_de_ajedrez.

[25] «Planeta Ajedrez,» [En línea]. Available: http://www.planetajedrez.com/p/blog-page_19.html.

[26] A. M. Vozmediano, Ajedrez en C: Cómo programar un juego de ajedrez en lenguaje C... ¡y que funcione! (Programación nº 1), http:// ensegundapersona.es.

[27] «Modelo TCP/IP,» [En línea]. Available: https://es.wikipedia.org/wiki/Modelo_TCP/IP.

[28] J. P. P. y. M. Merino, «Definicion.de,» 2008. [En línea]. Available: https://definicion.de/tcp-ip/.

[29] Comdat4, «slidesharecdn,» [En línea]. Available: https://image.slidesharecdn.com/nps107-tmp-100421180022-phpapp02/95/protocolo-tcpip-16-728.jpg?cb=1271872829.

[30] «Redes locales y globales,» [En línea]. Available: https://sites.google.com/site/redeslocalesyglobales/6-arquitecturas-de-redes/6-arquitectura-tcp-ip/2-funciones-de-las-capas-de-tcp-ip.

[31] «Herramientas Web,» [En línea]. Available: http://neo.lcc.uma.es/evirtual/cdd/tutorial/aplicacion/cliente-servidor.html.

[32] «fen,» [En línea]. Available: http://www.edicionesma40.com/blog/la-notacion-fen.htm.

[33] «ajedrez de ataque,» 2004. [En línea]. Available: http://www.ajedrezdeataque.com/10%20Miscelanea/FEN/Notacion.htm.

[34] UPC, «UPC,» [En línea]. Available: https://upcommons.upc.edu/bitstream/handle/2099.1/5543/Mem%C3%B2ria.pdf?sequence=4&isAllowed=y.

[35] ABB, Manual del operador. RobotStudio, 2008-2011.

[36] jhorrillo, «UC3M,» [En línea]. Available: http://personal.biada.org/~jhorrillo/INTRODUCCIO%20RAPID.pdf.

[37] «umh,» [En línea]. Available: https://isa.umh.es/asignaturas/rvc/ejemplo%20rapid.pdf.

[38] «Deep Blue,» [En línea]. Available: https://es.wikipedia.org/wiki/Deep_Blue_(computadora).

[39] J. Pastor, «Xakata,» [En línea]. Available: https://www.xataka.com/robotica-e-ia/de-ajedrez-maquinas-y-humanos.

[40] «Ajedrez por computadora,» [En línea]. Available: https://es.wikipedia.org/wiki/Ajedrez_por_computadora.

[41] «Explicación motor,» [En línea]. Available: http://ajedrezga.blogspot.com/2015/04/explicacion-de-como-funciona-un-motor.html.

# ANEXOS

Aquí se incluye información que, por su extensión o especificidad, no se ha incluido en los otros capítulos.

## Anexo A: Código del proyecto

Para realizar este proyecto ha sido necesario el empleo de varios lenguajes de programación, como son, "RAPID" para la programación del robot ABB; Lenguaje propio de Matlab (lenguaje M); y "C#" (C Sharp) para la interfaz gráfica desarrollada en Visual Studio.

Debido a la extensión del código, en estos anexos se va a mostrar un pequeño fragmento de cada uno de ellos como, por ejemplo, algún procedimiento o función destacable. Si fuese necesario acceder al código completo de alguna de las partes, se podrían consultar los archivos en formato virtual de los cuales dispondrá el tutor en todo momento.

### Script de lanzamiento

Aquí se presenta el código completo del Script de lanzamiento que lleva por nombre "Ajedrez_TFM.m".

```
    %% AJEDREZ TFM

close all
clear all
clc

%% VARIABLES DE CONTROL
app_visual=0;       %Si queremos utilizar la aplicaci□r6ca
robot_studio=0;     %Si queremos utilizar el simulador de ABB


%% Defino las comunicaciones TCP/IP

%Socket con el robot
if(robot_studio)
    %Simulador
        socket_robot=tcpip('127.0.0.1', 30000, 'NetworkRole',
'client');
    %Red de la escuela
        %socket_robot=tcpip('157.88.201.130', 30000, 'NetworkRole',
'client');
    %Conexion directa con el robot
        %socket_robot=tcpip('192.168.125.1', 30000, 'NetworkRole',
'client');
    disp('Ejecute RobotStudio y comience una partida')
    disp('Pulse cualquier tecla')
    pause;
    disp('Esperando conexion robot...')
```

```
    fopen(socket_robot);
    disp('Conectado con Robot')
else
    socket_robot=0;
end

%Socket con la aplicaci□r6ca
if(app_visual)
    socket_visual=tcpip('127.0.0.1', 2000, 'NetworkRole', 'server');

    disp('Esperando conexi□on Visual...')
    fopen(socket_visual);
    disp('Conectado con Visual')

else
    socket_visual=0;
end

%%
this=miAjedrez();

if(app_visual)
    %%Recibir datos
    while true
        %Esperando datos
        if socket_visual.BytesAvailable~=0
            data=fread(socket_visual, socket_visual.BytesAvailable);
            switch data(1)
                case 1
                    this.modo=data(2)+1; %Ajuste con el ice de
Visual
                    this.motor1=data(3)+1;
                    this.motor2=data(4)+1;
                    this.iniciar()
                case 2
                    this.jugar(socket_visual,socket_robot);
            end
        end
    end

else
    this.modo=0;
    this.motor1=0;
    this.motor2=0;
    this.iniciar()
    this.jugar(socket_visual,socket_robot);
end
```

### Clase miAjedrez

Pequeño fragmento de código de esta clase que muestra las funciones "iniciar", "jugar" y "realizar movimiento".

```
function iniciar(this)

            this.valores_iniciales();
            % iniciar nodo para apuntar jugadas
            this.nodo=[];
            this.refrescar_pantalla();
            this.menu();
end % function


function jugar(this,s1,s2)
            if s1==0 %Sin Visual
                this.visual=false;
            else
                this.visual=true;
                this.socket_visual=s1;
            end

            if s2==0 %Sin RobotStudio
                this.robot=false;
            else
                this.robot=true;
                this.socket_robot=s2;
            end

            this.refrescar_pantalla();
            while this.estado.terminar == this.FALSO

              if(this.visual)
            %Si estamos en modo robot-robot, esperamos a la orden de
            %visual para ejecutar un movimiento
                if this.modo==4

                   while this.socket_visual.BytesAvailable==0
                     %Bucle de espera a confirmacion de visual
                   end
                  conf=fread(this.socket_visual,
this.socket_visual.BytesAvailable);
                end
              end

                this.realizar_movimiento();
                this.estado.movimientos= this.estado.movimientos+ 1;
                if (this.estado.turno == this.BLANCAS)
                    this.estado.turno = this.NEGRAS;
                else
                    this.estado.turno = this.BLANCAS;
                end
            end
end
```

```
function realizar_movimiento(this)
    % // Determina a que jugador le toca mover y realiza el
movimiento alterando el tablero

    % // Mostramos estado de la partida
    if (this.estado.turno == this.BLANCAS)
      escribir('Turno: BLANCAS'); %// mostramos de quien es el turno
        else
            escribir('Turno: NEGRAS');
        end
        this.mostrar_movimientos(); %// Mostramos ltimos movimientos

        if (this.comprobar_jaque() == this.VERDADERO)
           if (this.comprobar_jaque_mate() == this.VERDADERO)  %//
Hay jaque mate
                    escribir('JAQUE MATE');
                    %socket visual informa jaque mate
                    if(this.visual)
                        fwrite(this.socket_visual,'1');
                    end
                    this.estado.terminar = this.VERDADERO;
                else            %// Hay jaque
                    escribir('JAQUE');
                    %socket visual informa jaque
                    if(this.visual)
                        fwrite(this.socket_visual,'2');
                    end
                end
            else
                %socket visual informa que no hay jaque
                %disp('No hay jaque')
                if(this.visual)
                    fwrite(this.socket_visual,'0');
                end
            end

            %// Realizamos el movimiento
            if (this.estado.turno == this.BLANCAS)
                if (this.estado.jug_blanco == this.HUMANO)
                    this.movimiento_humano();
                end
                if (this.estado.jug_blanco == this.MAQUINA)
                    this.movimiento_maquina();
                end
            else    %// turno == NEGRAS
                if (this.estado.jug_negro == this.HUMANO)
                    this.movimiento_humano();
                end
                if (this.estado.jug_negro == this.MAQUINA)
                   this.movimiento_maquina();
                end
            end
            this.refrescar_pantalla();
            this.mostrar_movimientos(); %// Mostramos ultimos
movimientos
end
```

## Clase UCI_Engine

Pequeño fragmento de código de esta clase que muestra el constructor

```
function this= UCIEngine(Engine)

          if Engine==0
              fprintf('Motores: \n');
              fprintf('    1.- stockfish_8_x64\n');
              fprintf('    2.- stockfish_9_x64\n');
              fprintf('    3.- komodo-9.02-64bit\n');
              fprintf('    4.- Hannibal1.7x64\n');
              fprintf('    5.- Gull 3 x64\n');
              fprintf('    6.- Fire_7.1_x64\n');
              fprintf('    7.- Protector_Win64\n');
              Engine= input('Motor: ');
          end

          switch (Engine)
              case 1
                  Engine= './stockfish_8_x64';
              case 2
                  Engine= './stockfish_9_x64';
              case 3
                  Engine= './komodo-9.02-64bit';
              case 4
                  Engine= './Hannibal1.7x64';
              case 5
                  Engine= './Gull 3 x64';
              case 6
                  Engine= './Fire_7.1_x64';
              case 7
                  Engine= './Protector_Win64';
              otherwise
                  error('Motor no v□do');
          end


          p = java.lang.ProcessBuilder(Engine).start();
          this.Engine= Engine;
          % Connect to engine's stdout
          iStream = p.getInputStream();
          iStreamReader = java.io.InputStreamReader(iStream);
          this.in = java.io.BufferedReader(iStreamReader);

          % Connect to engine's stdin
          oStream = p.getOutputStream();
          this.out = java.io.PrintWriter(oStream,true);
          %this.out = java.io.PrintWriter(oStream);
          read(this);
      end
```

### Interfaz gráfica (HMI)

Ejemplo de un fragmento de la aplicación de Visual Studio (HMI), que muestra la función “mov_humano”.

```
void mov_humano()

  {
   if (flag_jaque)
    {
          jaque(humano);
          if (modo!=2)
          {
              flag_jaque = false;
          }

    }

    if (turno)
     {
      info_b.Text = "Esperando respuesta";
      info_b.Refresh();
      Thread.Sleep(200);
      cadena = System.Text.Encoding.UTF8.GetBytes(orig_b + dest_b);
      Socket_Matlab.Send(cadena);
      }

    else
     {
      info_n.Text = "Esperando respuesta";
      info_n.Refresh();
      Thread.Sleep(200);
      cadena = System.Text.Encoding.UTF8.GetBytes(orig_n + dest_n);
      Socket_Matlab.Send(cadena);
     }

          Thread.Sleep(200);
          byte[] rec = new byte[1];
          Socket_Matlab.Receive(rec);

          switch (System.Text.Encoding.UTF8.GetString(rec))
          {
              case "1":          //Comando no válido
                  if (turno)
                  {
                      info_b.Text = "Comando no válido";
                  }
                  else
                  {
                      info_n.Text = "Comando no válido";
                  }

                  break;
```

```
case "2":         //Movimiento origen ilegal
   if (turno){
         info_b.Text = "Movimiento origen ilegal";
   }
   else{
         info_n.Text = "Movimiento origen ilegal";
   }

 break;

 case "3":        //Movimiento destino ilegal
    if (turno)
    {
          info_b.Text = "Movimiento destino ilegal";
    }
    else
    {
          info_n.Text = "Movimiento destino ilegal";
    }
  break;

  case "4":         //Correcto

     if (turno){
          info_b.Text = "Movimiento correcto";
          if(!enroque()) //Si no hay enroque anota el movimiento
             actualiza_tablero(orig_b, dest_b, humano);
             textBox_OrigenB.Text = "";
             textBox_DestB.Text = "";
      }
      else
      {
          info_n.Text = "Movimiento correcto";
          if (!enroque()) //Si no hay enroque anota el movimiento
              actualiza_tablero(orig_n, dest_n, humano);
              textBox_OrigenN.Text = "";
              textBox_DestN.Text = "";
          }
          if (modo!=2){
              jaque(robot);
              button_MoverB.Enabled = false;
              button_MoverN.Enabled = false;
              button3.Enabled = true;
           }
           else            //Modo humano-humano
           {
              if(turno)    //Turno blancas{
                 button_MoverB.Enabled = false;
                 button_MoverN.Enabled = true;
              }
                    else    //Turno negras{
                        button_MoverB.Enabled = true;
                        button_MoverN.Enabled = false;
                    }
                    turno = !turno;
                 }
                 break;
          }
      }
```

## RAPID

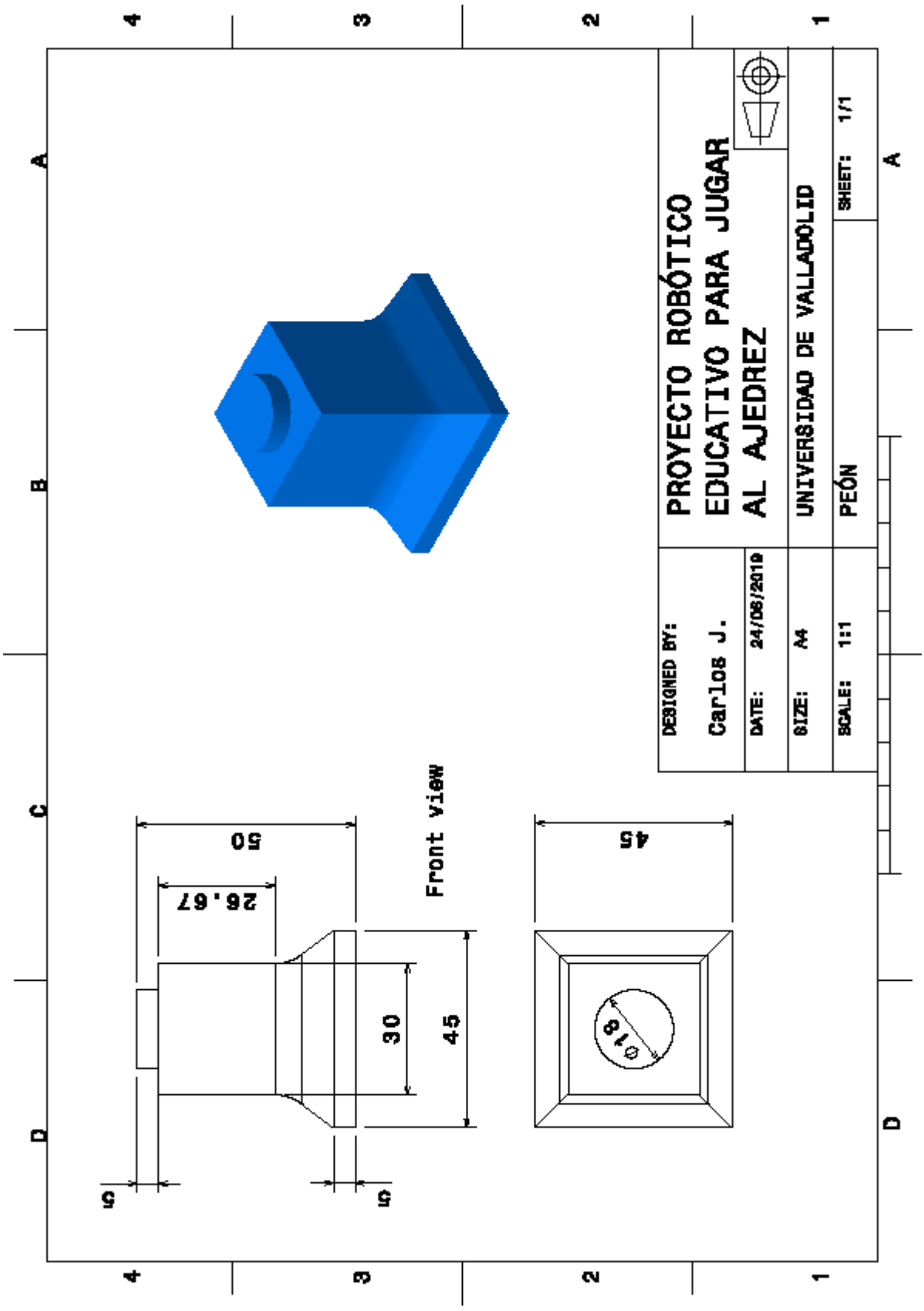
Ejemplo de una de las funciones del programa robot en RAPID, en concreto, “menuMoverPiezas ( )”

```
PROC menuMoverPiezas()
        VAR num opcion;
        VAR bool ok1;
        VAR bool ok2;

        ok1:=FALSE;
        AbrirPinza;
        WHILE not ok1 DO
            !            TPReadFK
opcion,"Mover:","Ext2Tab","Tab2Ext","Tab2Tab","Init","End";
            TPErase;
            TPwrite "Menú movimientos ajedrez:";
            TPwrite "   1.- Partida: Pos. Piezas Tablero";
            TPwrite "   4.- Movim. Conjuntos:";
            TPwrite "   3.- Movim. Individuales:";
            TPwrite "   4.- Red:";
            TPwrite "   5.- Salir:";
            !TPReadNum opcion,"Opción:";
            TPReadFK opcion,"Opción:","Partida","Mov. Conj.","Mov.
Ind.","Red","Salir";
            TEST opcion
            CASE 1:
                ok2:=FALSE;
                TPWrite "Nueva/Continuar Partida:";
                TPReadFK opcion,"     Borra
Memoria?","Si","","","","No";
                IF opcion=1 THEN
                    ! Reset de la posición de las piezas. Todas
están en su sitio.

PiezaBlanca:=[[11,12],[21,22],[31,32],[41,42],[51,52],[61,62],[71,72
],[81,82]];

PiezaNegra:=[[18,17],[28,27],[38,37],[48,47],[58,57],[68,67],[78,77]
,[88,87]];
                ENDIF

                IF TCPIP THEN
                    AbrirComunicación;
                ENDIF
                WHILE not ok2 DO
                    IF Tabl2Tabl()=0 THEN
                        ok2:=TRUE;
                    ENDIF
                ENDWHILE
            CASE 2:
                ! Movimientos conjuntos
                ok2:=FALSE;
                TPErase;
                WHILE not ok2 DO
                    TPwrite "   1.- Mover Piezas de Ext.a Tab.:";
                    TPwrite "   2.- Mover Piezas de Tab.a Ext.:";
                    TPwrite "   4.- Salir:";
                    TPReadFK
```

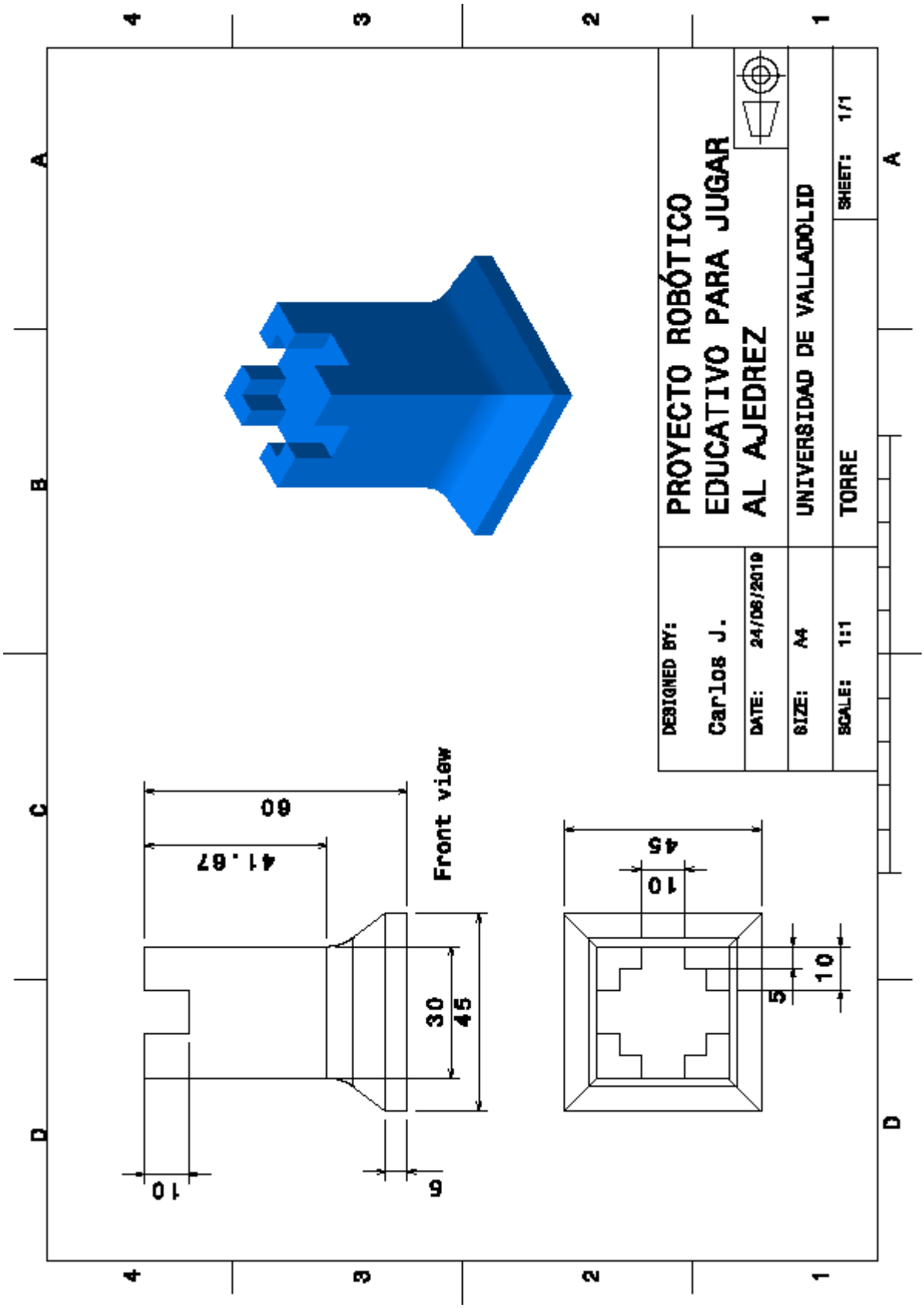
```
opcion,"Opción:","Ext2Tab","Tab2Ext","","","Salir";
                    ! TPReadNum opcion,"Opción:";
                    TEST opcion
                    CASE 1:
                        ! Reset de la posición de las piezas. Todas
están en el exterior

PiezaBlanca:=[[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0]];

PiezaNegra:=[[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0]];
                        PiezasExt2Tab;
                    CASE 2:
                        PiezasTab2Ext;
                    DEFAULT:
                        ok2:=TRUE;
                    ENDTEST
                ENDWHILE
            case 3:
                ! Movimientos individuales
                ok2:=FALSE;
                TPErase;
                WHILE not ok2 DO
                    TPwrite "   1.- Mover Pieza de Ext.a Tab.:";
                    TPwrite "   2.- Mover Pieza de Tab.a Ext.:";
                    TPwrite "   3.- Mover Pieza en Tab.:";
                    TPwrite "   4.- Salir:";
                    TPReadFK
opcion,"Opción:","Ext2Tab","Tab2Ext","Tab2Tab","","Salir";
                    ! TPReadNum opcion,"Opción:";
                    TEST opcion
                    CASE 1:
                        Ext2Tabl;
                    CASE 2:
                        Tabl2Ext;
                    CASE 3:
                        IF Tabl2Tabl()=0 THEN
                            ok2:=TRUE;
                        ENDIF
                    DEFAULT:
                        ok2:=TRUE;
                    ENDTEST
                ENDWHILE
            CASE 4:
                ! Conextar a red
                IF TCPIP THEN
                    TCPIP:=FALSE;
                    TPWrite "No Red";
                ELSE
                    TCPIP:=True;
                    TPWrite "Red";
                ENDIF
                WaitTime twait;
            DEFAULT:
                ok1:=TRUE;
            ENDTEST
        ENDWHILE
ENDPROC
```

## Anexo B: Planos de las piezas y la garra del robot.

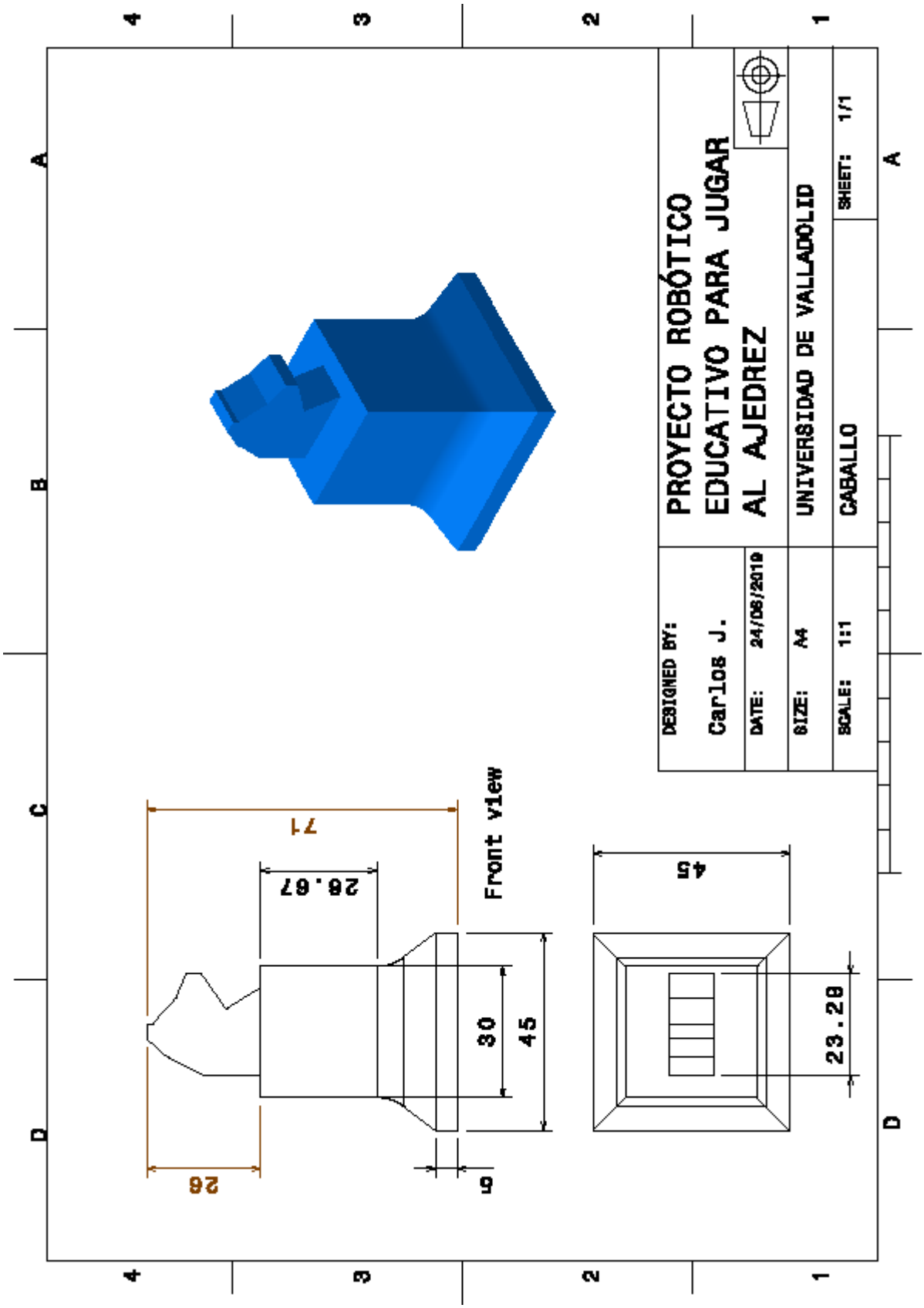
### Peón



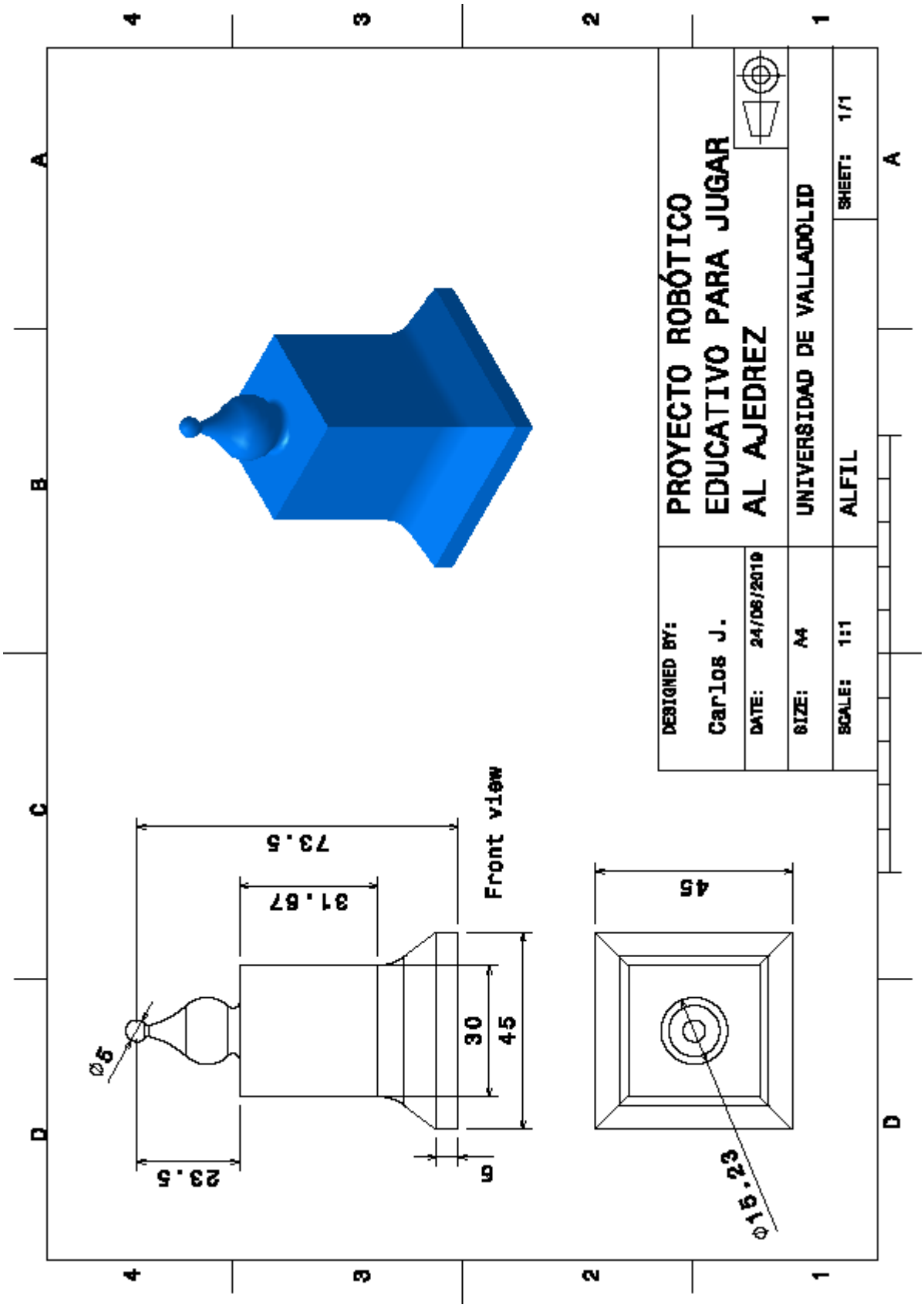
## Torre


PROYECTO ROBÓTICO EDUCATIVO PARA JUGAR AL AJEDREZ
UNIVERSIDAD DE VALLADOLID
TORRE
DESIGNED BY: Carlos J.
DATE: 24/06/2010
SIZE: A4
SCALE: 1:1
SHEET: 1/1
Front view
60
41.87
30
45
10
6
45
10
5
10

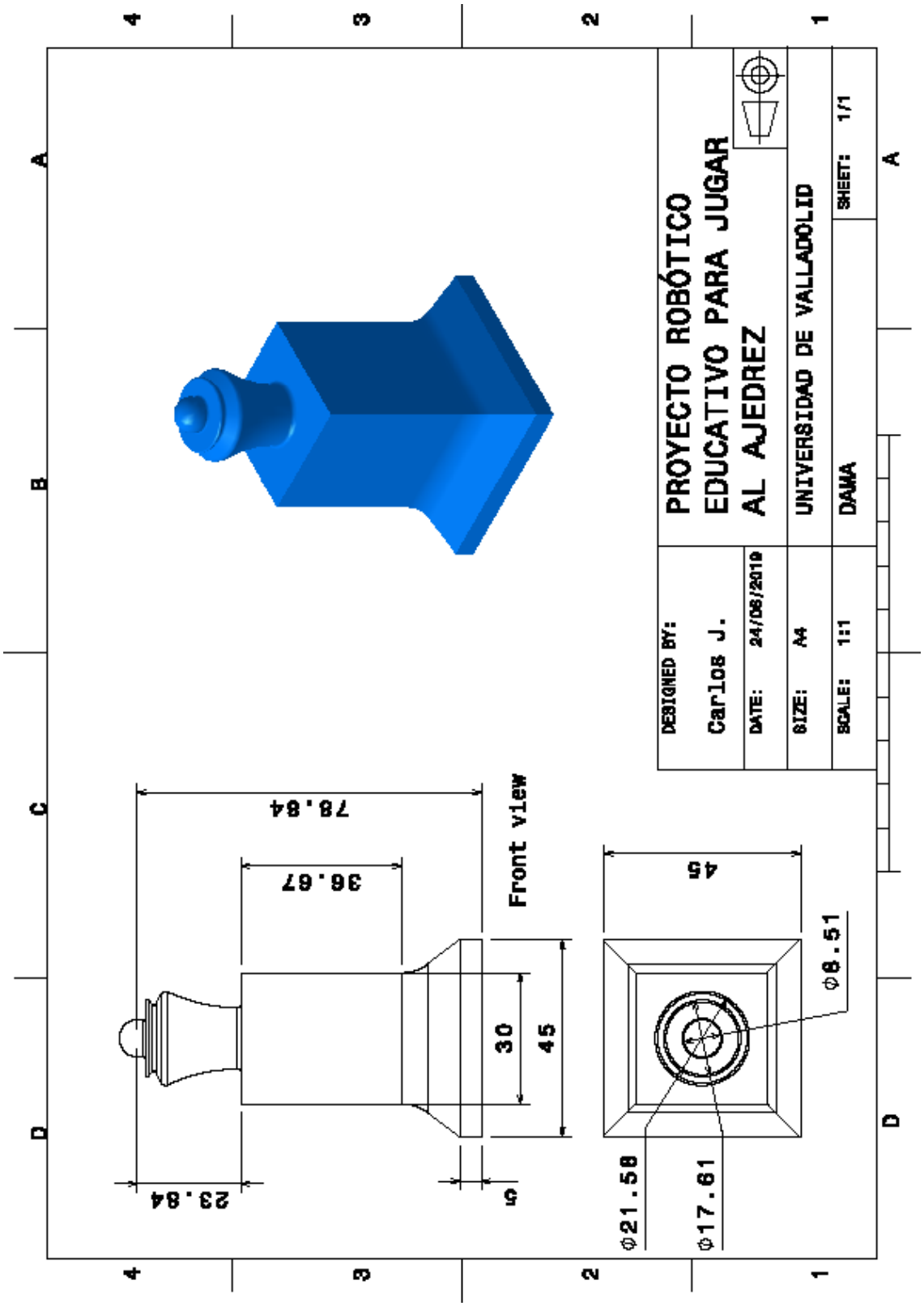
## Caballo


PROYECTO ROBÓTICO
EDUCATIVO PARA JUGAR
AL AJEDREZ
UNIVERSIDAD DE VALLADOLID
CABALLO
DESIGNED BY:
Carlos J.
DATE: 24/06/2010
SIZE: A4
SCALE: 1:1
SHEET: 1/1
Front view
71
26.67
30
45
26
5
45
23.28

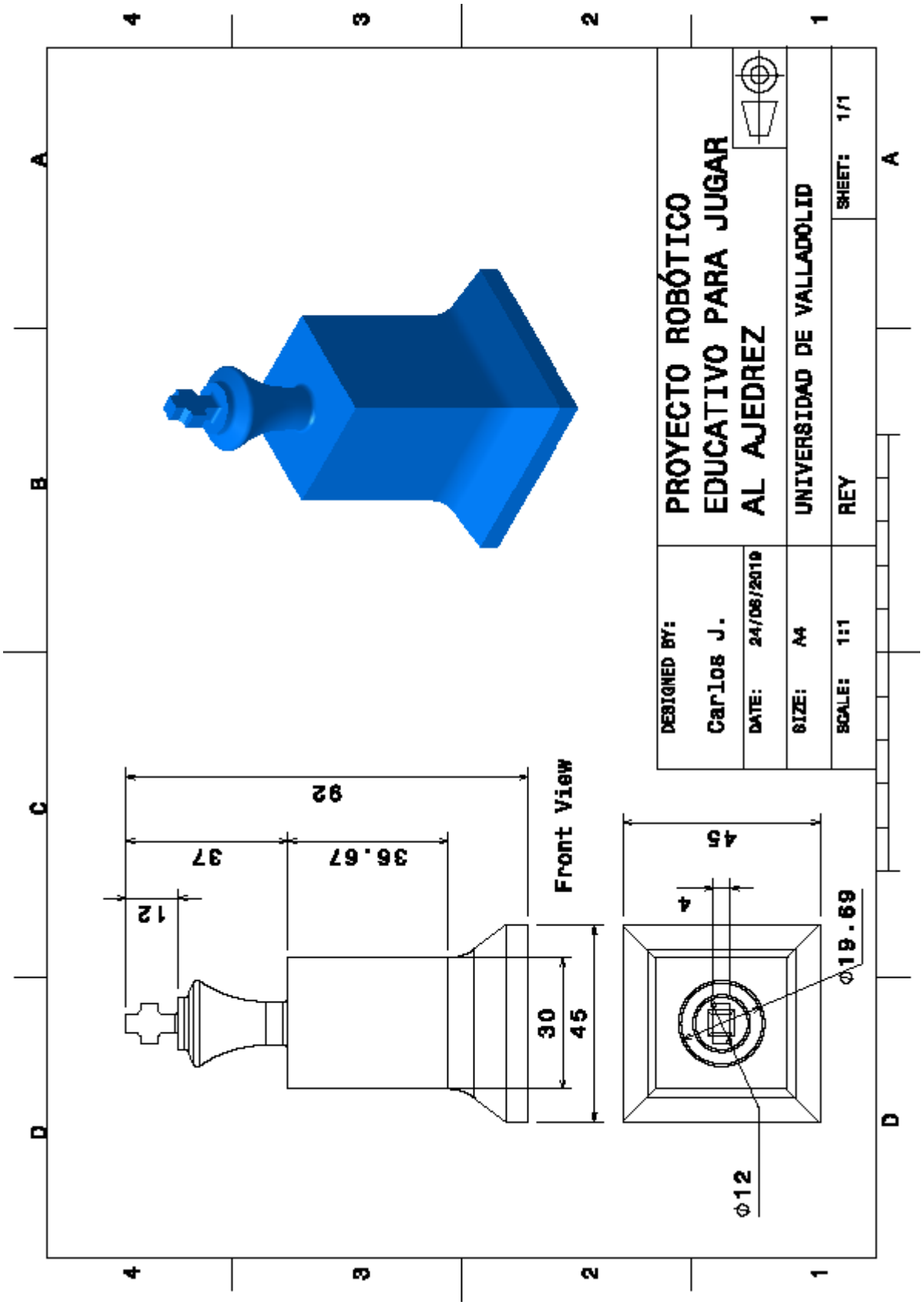
## Alfil


PROYECTO ROBÓTICO
EDUCATIVO PARA JUGAR
AL AJEDREZ
UNIVERSIDAD DE VALLADOLID
ALFIL
SHEET: 1/1
DESIGNED BY: Carlos J.
DATE: 24/06/2019
SIZE: A4
SCALE: 1:1
Front view
73.5
31.67
23.5
30
45
5
Ø5
45
Ø15.23

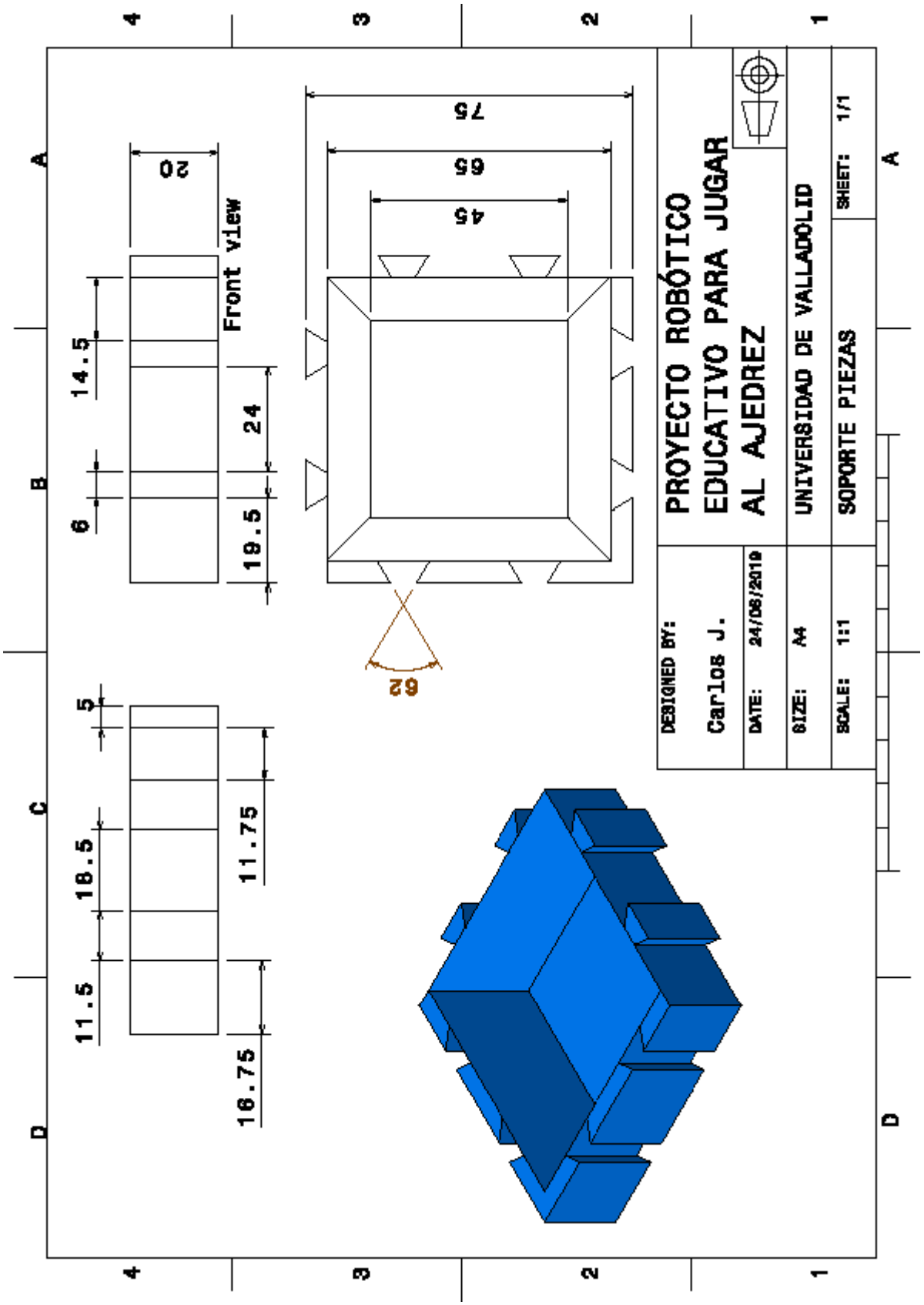
## Dama


PROYECTO ROBÓTICO
EDUCATIVO PARA JUGAR
AL AJEDREZ
UNIVERSIDAD DE VALLADOLID
DAMA
DESIGNED BY:
Carlos J.
DATE: 24/06/2019
SIZE: A4
SCALE: 1:1
SHEET: 1/1
Front view
78.84
36.67
23.84
30
45
5
45
Φ8.51
Φ21.58
Φ17.61

## Rey


PROYECTO ROBÓTICO
EDUCATIVO PARA JUGAR
AL AJEDREZ
UNIVERSIDAD DE VALLADOLID
REY
DESIGNED BY:
Carlos J.
DATE: 24/06/2019
SIZE: A4
SCALE: 1:1
SHEET: 1/1
Front View
92
37
36.67
12
30
45
45
4
Φ19.69
Φ12

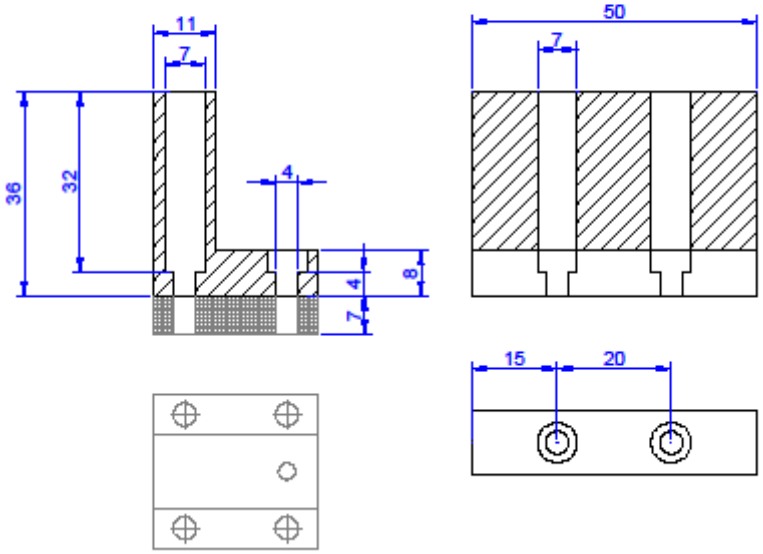
## Soporte de piezas



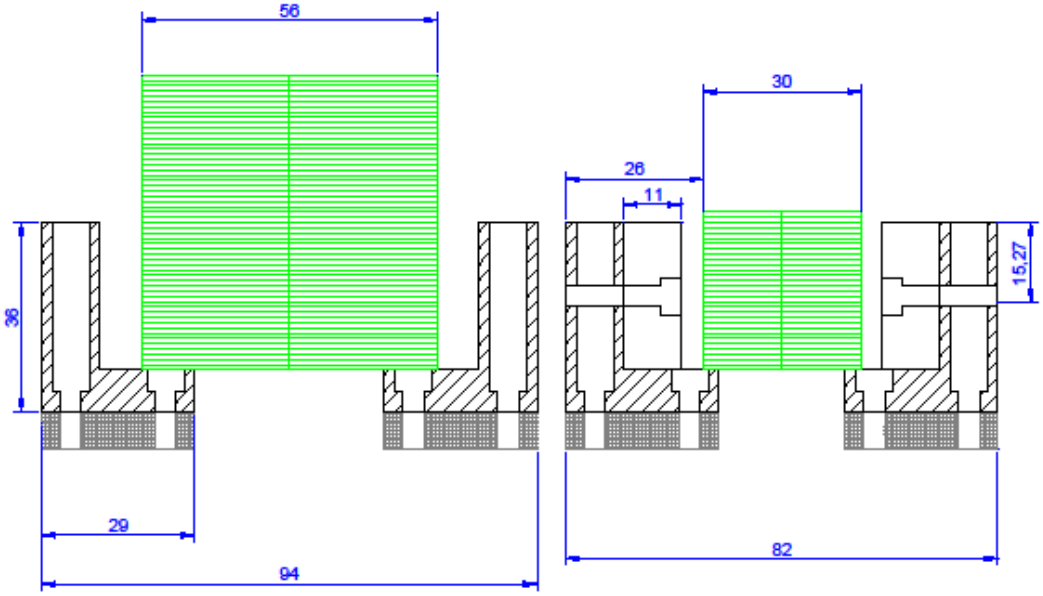
## Pinza

### *Pinza original*

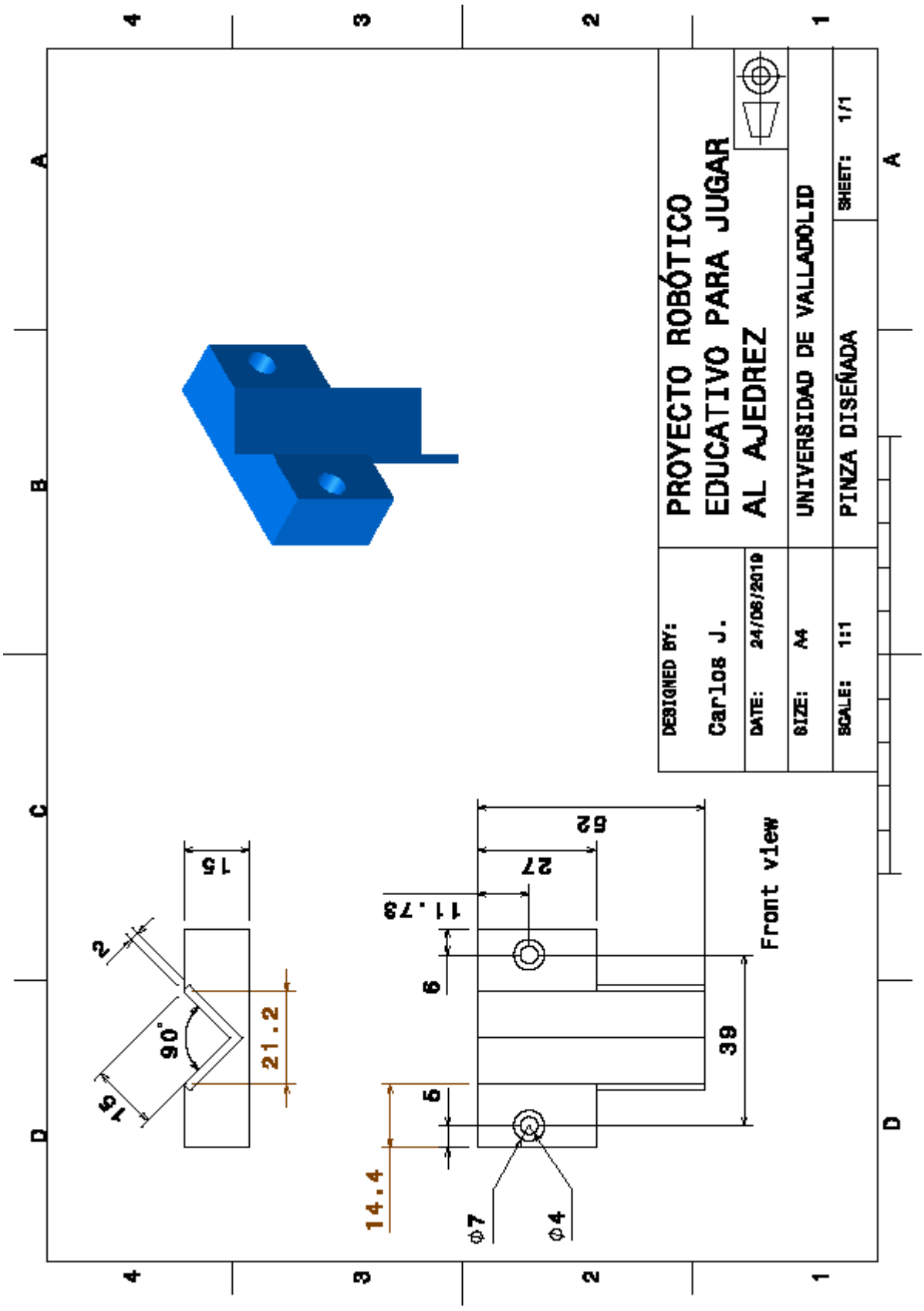
Detalle Pinza Neumática




Adaptación para los distintos objetos (vista general)



*Pinza diseñada*



*Pinza fabricada*