UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf

Trabajo Fin de Máster

Máster Universitario en Ingeniería Mecatrónica

AUTOR/A: López Sanfeliciano, Antonio

Tutor/a: Armesto Ángel, Leopoldo

CURSO ACADÉMICO: 2023/2024

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf

Trabajo Fin de Máster

Máster Universitario en Ingeniería Mecatrónica

Documentación:

1. Memoria técnica
2. Planos
3. Pliego de condiciones
4. Presupuesto
5. Anexos

AUTOR/A: López Sanfeliciano, Antonio

Tutor/a: Armesto Ángel,Leopoldo

CURSO ACADÉMICO:2023/2024

Agradecimientos

Con este proyecto se cierra una etapa muy importante de mi vida, por ello querría agradecer a todos los que me han acompañado en este camino.

En primer lugar, a mis padres por brindarme la educación que tengo, siempre me han animado a realizar mis estudios y nunca me ha faltado de nada. Al igual que mi hermana, quien siempre está ahí cuando he necesitado algún consejo. Mi abuela fue otro de mis pilares, ella siempre quiso que fuera buena persona y que consiguiera todos aquellos objetivos que me propusiese. También, querría agradecer a Sara, por escucharme y animarme en cualquier momento, ya sean buenos o malos.

En este camino me junté con unos compañeros de clase, que ahora puedo dirigirme a ellos como amigos. Juntos nos hemos apoyado mutuamente para conseguir nuestros objetivos, además de pasar muchos buenos momentos que ya quedan grabados en mi memoria.

Por supuesto debo agradecer a mi tutor, Leopoldo Armesto, quién ha sido mi mentor en este apasionante proyecto. No puedo olvidarme tampoco del técnico Óscar Bodoque, quien me ayudó con diferentes tareas del trabajo. Por último, de un compañero llamado Hannes Deruytter, quién me ha acompañado en este proyecto.

Resumen

El presente TFM consiste en implementar técnicas de Optimización Bayesiana aplicadas a un brazo robótico localizado junto a una topografía impresa en 3D para el desarrollo de la actividad deportiva del minigolf. Ajustando diferentes parámetros, se consigue resolver de forma más eficiente las magnitudes angular y vectorial del robot OpenMANIPULATOR-X para depositar la bola en un hoyo objetivo.

Dicho robot, emplea la placa OpenRB-150, programada con el entorno Arduino y controlado mediante un puerto serie a través del software MATLAB. Además, se registra la trayectoria y la posición de la bola gracias a un sistema de visión artificial, empleando la librería de OpenCV.

Por último, el análisis concluye con la comparación de resultados entre el experimento realizado y una simulación en CoppeliaSim para ver las diferencias y semejanzas de ambas metodologías.

Palabras clave: Optimización Bayesiana, Robot manipulador, MATLAB, OpenCV, Visión artificial.

Resum

El present TFM consistix en implementar tècniques d'Optimització Bayesiana aplicades a un braç robòtic localitzat al costat d'una topografia impresa en 3D per al desenvolupament de l'activitat esportiva del minigolf. Ajustant diferents paràmetres, s'aconseguix resoldre de forma més eficient les magnituds angular i vectorial del robot OpenMANIPULATOR-X per a depositar la bola en un clot objectiu.

Este robot, empra la placa OpenRB-150, programada amb l'entorn Arduino i controlat mitjançant un port serie a través del programari MATLAB. A més, es registra la trajectòria i la posició de la bola gràcies a un sistema de visió artificial, emprantla la llibreria de OpenCV.

Finalment, l'anàlisi conclou amb la comparació de resultats entre l'experiment realitzat i una simulació en CoppeliaSim per a veure les diferències i semblances de ambdues metodologies

Paraules clau: Optimització Bayesiana, Robot manipulador, MATLAB, OpenCV, Visió artificial.

Abstract

This master thesis consists of implementing Bayesian Optimisation techniques applied to a robotic arm located next to a 3D printed topography for the development of the minigolf sport activity. By adjusting different parameters, the angular and vectorial magnitudes of the OpenMANIPULATOR-X robot are solved more efficiently to deposit the ball in a target hole.

This robot uses the OpenRB-150 board, programmed with the Arduino environment and controlled via a serial port through MATLAB software. In addition, the trajectory and position of the ball is recorded thanks to an artificial vision system, using the OpenCV library.

Finally, the analysis concludes with the comparison of results between the experiment and the simulation in CoppeliaSim to see the differences and similarities of both methodologies.

Key words: Bayesian optimization, Robot manipulator, MATLAB, OpenCV, Artifical vision.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf

Trabajo Fin de Máster

Máster Universitario en Ingeniería Mecatrónica

Primer documento:

Memoria técnica

AUTOR/A: López Sanfeliciano, Antonio

Tutor/a: Armesto Ángel, Leopoldo

CURSO ACADÉMICO:2023/2024

Índice

Índice de figuras

Índice de tablas

1 Objeto

La finalidad del presente proyecto es el diseño de experimentos para conseguir la optimización de parámetros en controladores, concretamente en un robot manipulador. Hay diferentes técnicas relacionadas con la optimización que se pueden emplear, entre ellas están: por gradiente, de enjambre, combinatoria, evolutiva, entre otras. Debido a la problemática que presenta el trabajo, donde es desconocida la solución y por sus características, que permiten evaluar de diferentes formas los experimentos, logrando conseguir una solución en pocas iteraciones. Se emplea, la herramienta de aprendizaje supervisado llamada Optimización Bayesiana, donde nos podemos adentrar en sus heurísticas para obtener una solución equilibrada respecto a la exploración y explotación. Para poder aplicarla, se necesita previamente haber montado y posteriormente configurado el OpenMANIPULATOR-X, del cual se requiere elaborar su cinemática (directa e inversa) y controlarlo con el *DynamixelShield,* comunicado a través de un puerto serie. Posteriormente se requiere localizar la trayectoria que realiza la pelota, se logra mediante la ayuda de una cámara y la librería de libre acceso llamada OpenCV, con la que se puede calibrar y aplicar las máscaras necesarias. Por último, se integra todo en un mismo entorno de programación, en este caso MATLAB, en las que nos ayudaremos de sus diferentes *add-ons* y scripts consiguiendo así optimizar los dos parámetros requeridos, la posición del robot y el tiempo de trayectoria. Por consiguiente, se cumple en este proyecto, los cinco ámbitos que definen a la ingeniería mecatrónica: control, informática, mecánica, electricidad y electrónica.

2. Antecedentes

En este apartado se valoran los orígenes de algunos de los conceptos empleados en este trabajo académico

El *Machine Learning* o aprendizaje automático ha sido toda una revolución desde sus inicios en los años cincuenta con el “Test de Turing”, en el que un ordenador debía intentar engañar a un humano haciéndole creer que se comunicaba con otro humano, unos pocos años después se creó el primer algoritmo en el que un ordenador era capaz de aprender a jugar a las damas y mejorar en cada partida. Se trata de una subrama de la inteligencia artificial cuyo objetivo es el de mediante un algoritmo, procesar unos datos y elaborar predicciones con un modelo generado. No es hasta principios del siglo veintiuno cuando estos conceptos se desarrollan de forma exponencial debido a las grandes mejoras de cómputo [1].

Por otra parte, la visión artificial empieza a finales de los años cincuenta, al igual que el *Machine Learning* es un campo de la inteligencia artificial en el que aparecen conceptos como el *deep learning* y la red neuronal convulocional (CNN), la cual permite el procesamiento de imágenes digitales, tales como vídeos en tiempo real. En 1974 se desarrolló el algoritmo de reconocimiento óptico de caracteres (OCR) este algoritmo que ha llevado a la detección de patrones, como la lectura de matrículas. Con el paso de los años es empleada en sectores industriales para enviar coordenadas a un robot, logística, control de calidad, entre otros. Se puede entrenar los diferentes patrones y al procesar las imágenes el ordenador será capaz de detectar

y reconocerlos, para poder clasificarlos [2]. Open CV es una librería de código abierto que tiene su primera aparición en enero de 1999, inicialmente fue desarrollado por Intel, contiene más de 500 funciones para el reconocimiento de objetos, calibración de cámaras… Además, es multiplataforma y su programación fue en C y C++, aunque en la actualidad ya se puede emplear en otros lenguajes como Python [3].

Respecto a los primeros manipuladores robóticos, George Devol y Joseph Engelberger, en la década de los cincuenta fueron los desarrolladores del primer brazo robótico industrial, el conocido Unimate, el cual se podía programar y controlar a través de un ordenador. Todo ello, unos pocos años más tarde, consiguieron darle al robot la primera tarea de soldadura en la industria, realizando labores repetitivas, siendo más exacto que un operario y aumentado notablemente la eficiencia y la calidad. A partir de entonces el uso de los robots industriales fue aumentando de forma exponencial y más con la llegada de PUMA en los años setenta, el primer robot con seis ejes, que permitió mayor aplicabilidad y una mejora notoria en la robótica moderna, en una sociedad de consumo cómo la actual, en la que gracias a estos avances se conseguía llegar a un mayor público que cada vez requiere de mayor demanda [4].

En cuanto a la optimización, Thomas Bayes fue un matemático inglés que utilizó la probabilidad de forma inductiva y estableció una base matemática para la inferencia de probabilidades, teniendo en cuenta la frecuencia con la que ocurría un evento para ensayos futuros. En los últimos años, la base del hallazgo de Bayes dio lugar a la optimización bayesiana, la cual data entorno los años setenta y ochenta, en el que mediante una serie de técnicas, que más adelante se detallarán, se pretende predecir una función objetivo desconocida, con coste computacional y monetario elevado. Se genera un modelo probabilístico en el que se intenta encontrar mediante un proceso generalmente gaussiano y una función de adquisición que decide cuál es el siguiente parámetro para experimentar y así, lograr alcanzar el óptimo con muchas menos iteraciones a nuestra función objetivo desconocida como si de una caja negra se tratase [5]. Por otro lado, también hay diversas metodologías como el *super vector machine* (SVM), que permiten clasificar datos con un gran nivel de ruido, el objetivo es generar un hiperplano que permita separar dos clases diferentes, siempre que el problema lineal, es otro método que no se valora en se emplea en este trabajo debido a la falta de tiempo [6].

3 Estudio de necesidades, factores a considerar: limitaciones y condicionantes

El empleo de estas técnicas, al igual que ocurre con la optimización bayesiana, puede resultar útil en algunos ámbitos industriales, como por ejemplo en el sector químico para la elaboración de productos o medicamentos. Además, si los materiales necesarios para su elaboración son costosos de adquirir, la optimización puede ayudar a predecir cual es la mejor combinación posible realizando el mínimo número de experimentos. De esta forma, se lograría ahorrar material apostando por unos procesos más sostenibles y económicamente ventajosos para la compañía u organización.

En el sector industrial, concretamente en el mundo de la robótica, la optimización es una técnica cada vez más recurrida. Se pretende conseguir que una máquina a cada iteración logre realizar un movimiento determinado para alcanzar la trayectoria óptima en el mejor tiempo posible. grandes compañías están dispuestas a remunerar estas condiciones dado que a largo plazo se alcanzan reducciones de costes relevantes.

Uno de los limitantes y condicionantes a tener en cuenta al realizar los experimentos, es el factor humano al posicionar la pelota en la localización correspondiente. El proyecto plantea un segundo robot para recoger la pelota y posicionarla de nuevo en el lugar que se requiera para el disparo, teniendo en cuenta que su zona de trabajo se encuentra limitada al no alcanzar todas las distancias. Para resolver esta combinación ambos manipuladores de la mejor forma se mide la distancia del robot a la madera donde reposa la pelota y comprueba que se encuentre en la misma distancia entre rango de -15 a15 grados, marcando también los ángulos en la madera. Por otro lado, por cada lanzamiento el programa imprimirá por el terminal el ángulo y la velocidad en el que el robot lanzará el siguiente experimento. Por tanto, se debe considerar un operario cuya labor de posicionar la pelota se complete al supervisar que la posición de ésta sea la correcta, de este modo se logra mayor precisión y tiempo.

4 Planteamiento de soluciones alternativas y justificación de la solución adoptada

El primer planteamiento del trabajo que se iba a abordar era el empleo de dos robots manipuladores, es decir, dos OpenMANIPULATOR-X, en el que ambos debían estar sobre una plataforma larga y estrecha, situados cada uno de estos en un extremo y establecer un control en el que la plataforma quedaría balanceada y a esta se le añadiría una pelota que tendría que mantenerse en equilibrio. Rápidamente esta propuesta fue desechada dado que los motores que lleva este robot no son lo suficientemente rápidos y se necesitaría una plataforma exageradamente larga.

Posteriormente se optó por realizar una optimización, concretamente la bayesiana y la tarea más laboriosa fue plantear que parte se podía optimizar mediante el robot manipulador anteriormente mencionado. Finalmente se llegó a la conclusión de que, si se podía realizar una serie de experimentos con este en el que, empujando una pelota sobre una superficie, sería un buen planteamiento por dónde empezar.

En un principio, la programación del OpenMANIPULATOR-X, fue llevada a cabo mediante el microcontrolador de Robotis, llamado Open-RB150. No obstante, tras varios problemas de uso dejó de funcionar correctamente, caso que no parece ser aislado puesto que diversos clientes del robot reportaron en el foro de la empresa que las placas estaban dando problemas hasta el punto de no reconocer los servomotores y, por tanto, impidiendo el movimiento del robot. Por todo lo expuesto, se decidió emplear un *shield* acoplado sobre un Arduino UNO, el cual recibe el nombre de DynamixelShield y se emplea para poder mover al robot, resolviendo la problemática anterior.

Para resolver la cuestión del factor humano comentado en el apartado anterior se plantearon varias alternativas. La primera era añadir un segundo robot que dispondría

de la cámara, pudiendo visualizar la pelota en todo momento para ir a recogerla después de cada experimento y así depositarla en el lugar preciso del siguiente lanzamiento. La otra opción planteada era poner un raíl al robot manipulador, de forma que cuando quisiera lanzar un experimento con un cierto ángulo, se desplazase de forma transversal para que la bola quedase dispuesta en el centro quedándose la pelota siempre en la misma posición. Estas técnicas podrían ser interesantes para completar el proyecto, no obstante, dada la magnitud de lo estudiado, no se requería para resolver los objetivos de trabajo aplicar dichas herramientas. Por ello, se consideró oportuno ejercer la función del operario que posiciona la bola para poder estudiar y comprender de primera mano el proceso.

La superficie en la que la pelota se iba a mover iba a ser impresa en 3D, pero al ser relativamente grande, se tendría que realizar mediante diferentes trozos y partirla. En consecuencia, se decide que lo mejor era mecanizarla, es decir, mediante una fresadora con un material tipo espuma consistente se consigue un resultado más que notorio. Por otro lado, en cuanto a la estructura que se diseña y monta para ubicar la superficie y la pelota, en principio la idea era imprimirla también en 3D o en corte laser. Pero al igual que ocurría con la superficie, era demasiado grande y si se optaba por realizar la cortadora láser, la estructura iba a ser lo suficientemente endeble para que el robot por accidente la golpease. En conclusión, la solución final por la que se optó fue comprar tableros de aglomerado, cortarlos y realizar el montaje de la estructura. El resultado fue una construcción muy resistente que une adecuadamente el conjunto por lo que al final la solución más tradicional ha resultado ser probablemente mejor de lo que hubiera sido llevar a cabo la impresión.

5 Componentes de la solución adoptada

En este apartado se detallan los elementos a emplear en el proyecto, donde cada uno tiene un papel significativo: algunos requieren cierta configuración previa, otros controlan el robot o lo alimentan y, por último, captan información para luego ser procesada.

5.1 OpenMANIPULATOR-X

Se trata de un robot redundante de cuatro grados de libertad fabricado por la empresa china ROBOTIS, el cual tiene un tamaño reducido y muy versátil. Está pensado para que se pueda programar mediante ROS o lenguajes como Arduino a través de alguna de sus placas.

El producto viene empaquetado y trae tanto las herramientas como el material necesario para construirlo, además de manual de instrucciones para su montaje [7].

El proceso de montaje dura alrededor de dos/tres horas, en las siguientes figuras se muestra su evolución:

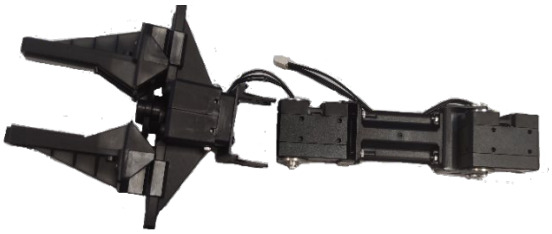


Figura 1 y 2: Procesos de montaje de las diferentes articulaciones junto con los eslabones y la pinza. (Fuente: Elaboración propia).

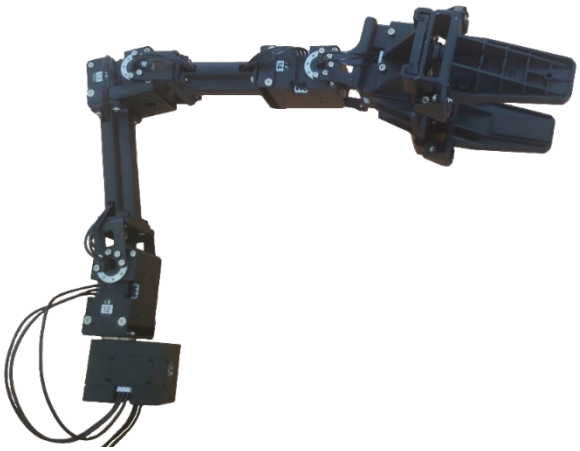


Figura 3: Robot OpenMANIPULATOR-X montado. (Fuente: Elaboración propia).

Este sería el aspecto del robot una vez realizado la fase de montaje, el siguiente paso sería configurarlo para asignar a cada servomotor un ID, y así saber que motor se está moviendo en cada momento, pero esta información se detalla en los anexos. Como se puede observar los motores van conectados entre sí de forma concadenada, es decir, sólo se alimenta a través del primero, por eso es tan importante asignarles correctamente su ID identificativo para que cuando se envíe una señal específica de movimiento, se mueva el que corresponda. La base que se imprime en 3D y posteriormente se acopla al primer servo del robot, es diseñada por un tercero [28].

5.1.1 Servomotores XM430-W350-T

Figura 4 y 5: Motor XM430-W350-T de la pinza. (Fuente: Elaboración propia).

Los servomotores que se emplean son los Dynamixel XM430-W350-T de ROBOTIS, en su *datasheet* se detallan sus características [8]. Estos se alimentan a 12 V y 2.3 A, tiene un par nominal de 4.1Nm y llevan un *encoder* de 12 bits con una resolución de 4096 pulsos por cada revolución. Ofrece diferentes modos de control, entre los cuales el que se emplea es el de control de posición por corriente, de este modo, se puede

limitar la corriente de cada motor dado que si tropezase con un obstáculo la intensidad aumentaría y correría el riesgo de que éste se dañase.

En cuanto al PID, ya viene de fábrica configurado y no necesita ningún reajuste ya que cumple sigue la señal de referencia sin ningún tipo de problema. Además, para mover varios motores en nuestro caso lo debemos de realizar de forma síncrona es decir a la vez ya que, tiene un cierto retraso entre el que más cerca está conectado a la placa y el siguiente a este.

5.2 DynamixelShield.

Figura 6: Dynamixel Shield y Arduino UNO. (Fuente: e-manual de ROBOTIS [9]).

Se trata de un *shield* que se acopla al famoso Arduino UNO, se programa a través de su plataforma, el Arduino IDE. Tiene varios interruptores, el grande, es para habilitar la potencia en el DynamixelShield, mientras que el pequeño selecciona la comunicación serie entre el Arduino o los motores de dynamixel, para poder subir un programa o controlar el robot. Además, se puede observar que tiene diferentes pines para conectar los servomotores, el que nos interesa es el TTL, en el caso de Dynamixel XM430-W350-T, solo se emplea uno de ellos ya que, como ya se ha explicado, los motores van conectados en serie. La etapa de potencia acepta hasta 12.6V y 3000 mA, en nuestro caso le hemos conectado una fuente que cumple con los requisitos. Por otro lado, se emplea un adaptador UART a TTL para poder comunicar el robot con el ordenador y pasarle los movimientos necesarios para realizar la optimización.

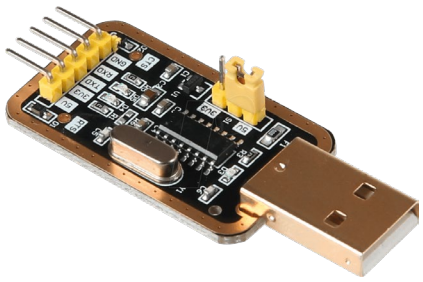


Figura 7: Dispositivo UART a TTL. (Fuente: Amazon [10]).

5.3 Fuente de alimentación.

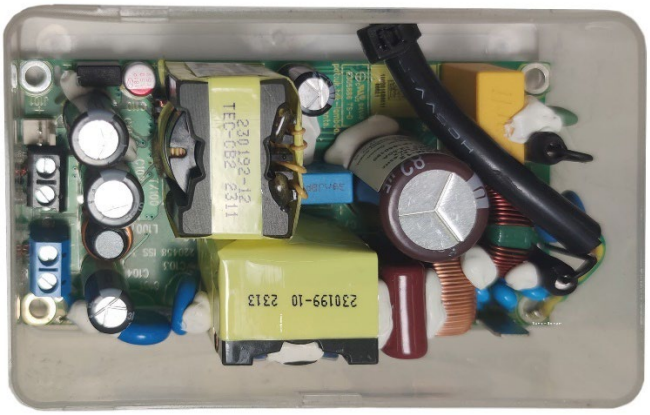


Figura 8: Fuente de alimentación 12V. (Fuente: Elaboración propia).

La fuente para poder alimentar a los servomotores dado que las especificaciones del *DynamixelShield* abarcan tensiones de hasta 24 V y corrientes de 10 A. En este caso, la fuente empleada, se eligió con respecto a la anterior placa empleada, la OpenRB-150, que admite voltajes de 12,6V y 3000 mA, por tanto, la fuente que se seleccionó fue de 12V y 3000mA, en la que ambas placas funcionan correctamente.

5.4 Webcam

Figura 9: Cámara web full HD de Aukey. (Fuente: PCcomponentes[11]).

La cámara web a empleada para la detección de la pelota es de la marca Aukey, tiene una resolución de 1920x1080 y dos megapíxeles. Para el proyecto que se desarrolla en el presente trabajo, una tasa de refresco de 30 imágenes por segundo es suficiente puesto que cumple con los requisitos mínimos: detectar objetos de un determinado color y realizar un tracking del movimiento que realiza la bola. Se conecta mediante un cable de USB 2.0 a cualquier conexión del ordenador. Además, no requiere de ningún software específico para la configuración por lo que la puesta a punto es instantánea. Por consiguiente, es versátil dado que tiene un soporte sencillo que se acopla y se sostiene a casi cualquier tipo de superficie, además admite ranura para que sea acoplada mediante un trípode.

5.5 Plataforma y estructura

Figura 10: Plataforma de material espumoso. (Fuente: Elaboración propia).

La plataforma por donde circula la pelota se ha diseñado con el software SOLIDWORKS. Además, contiene una cierta inclinación dificultando la entrada al agujero, el cual no se encuentra ubicado en el centro sino un tanto desplazado para que el disparo sea todavía más desafiante a la hora de acertar. El material empleado es una espuma suficientemente rígida como para permitir el desplazamiento de la bola sin mayor complicación, más concretamente de poliuretano. Esta ha sido lijada para que las líneas que ha dejado la fresadora sean mitigadas y se reduzca rozamiento.

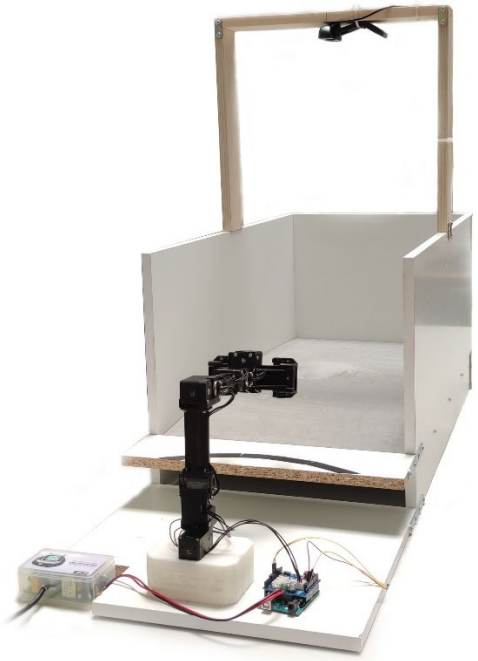


Figura 11: Estructura con la plataforma. (Fuente: Elaboración propia).

La estructura está formada por dos tablones de aglomerado idénticos de 80x39.5x1.6 cm que forman las paredes para evitar que la pelota se salga. La tapa del final, el reposa pelotas y donde está anclado el robot proceden de un tablón de aglomerado también el cual se ha cortado en diferentes medidas, de dimensiones 120x39.5x1.6 cm. Para la sujeción de la cámara se emplean listones de madera, llegan a una altura

de 40 cm respecto la pared, debido a que la cámara tiene una resolución baja (480p) se requiere mayor altura para poder visualizar toda la plataforma.

Figura 12: Pelota de frontón empleada para el desarrollo de los diferentes experimentos. (Fuente: Elaboración propia).

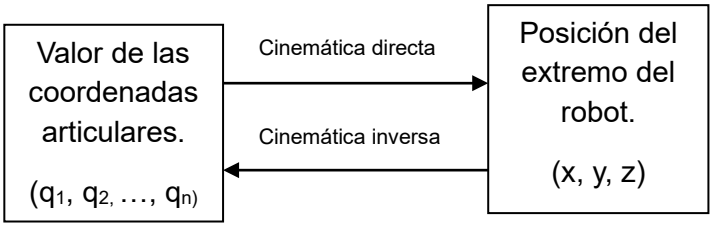
Por último, la pelota fue seleccionada a propósito, se eligió siguiendo un criterio funcional: la pelota más adecuada era aquella que mejores propiedades presentaba para el estudio del experimento. Una de tenis era demasiado grande y pesada (unos 57 gramos y 6,7 cm de diámetro) y, además, presentaba mayor rozamiento debido a su superficie con pelaje. Por otro lado, imprimir una pequeña pelota en 3D fue una opción descartada porque su peso sería demasiado bajo, dificultando adentrarse en el hoyo objetivo. Con respecto a una de tipo golf, también presentaba un tamaño demasiado pequeño con respecto al agujero. Por ello, la pelota empleada es de frontón, un término medio adecuado para el análisis por ser completamente liso, de aproximadamente 42 gramos y 6,4 cm diámetro.

6 Solución adoptada

En este apartado de la memoria técnica, se explican los procedimientos que se han seguido, así como las diferentes metodologías empleadas las cuales se combinan todas ellas para poder elaborar una serie de experimentos y lograr realizar una optimización real y precisa.

6.1 Cinemática

El problema de la cinemática es esencial en el campo de los robots, en este caso en un brazo manipulador, para hallar los parámetros necesarios y obtener bien sea las coordenadas cartesianas o la posición de las articulaciones. Se puede dividir en dos ramas: cinemática directa y cinemática inversa. En ambas lo que se busca es a partir de unos datos de entrada calcular mediante una serie de operaciones matemáticas, en las que se emplea la trigonometría, la relación que hay entre el espacio cartesiano y el de las propias articulaciones. En la directa los parámetros que se obtienen son del espacio cartesiano, por tanto, los datos de entrada son los ángulos de cada una de las cuatro articulaciones. No obstante, con la inversa se obtiene la configuración de las articulaciones a partir de las coordenadas dadas.



Antes de nada, para realizar la cinemática debemos tener en cuenta las longitudes de los diferentes eslabones, así como su nomenclatura. En la siguiente tabla se puede observar el valor de cada una.

Eslabones	Longitudes (m)
l_1	0.077
l_{21}	0.128
l_{22}	0.024
l_2	0.130
l_3	0.124
l_4	0.126

Tabla 1: Longitud de los eslabones. (Fuente: Robotis [7]).

Ambas cinemáticas serán implementadas con código Arduino, pero previamente se empleó dos *scripts* de MATLAB para comprobar su correcto funcionamiento y no tener un comportamiento inapropiado del robot con el que se dañen los motores, el microcontrolador e incluso a un operador.

6.1.1 Cinemática directa

Para realizar el cálculo de la cinemática directa existen dos métodos, el primero de ellos es mediante el convenio de Denavit-Hartenberg, el cual se basa en una serie de normas sistemáticas las cuales explicadas grosso modo consisten en: establecer un sistema de referencia fijo, normalmente en la base del robot, seguidamente se establecen los ejes de coordenadas de cada articulación de una determinada forma siguiendo unos criterios concretos, por último, localizar los giros y desplazamientos que se deben realizar en los ejes Z y X para determinar la diferentes matrices de transformación y así obtener la relación entren el efector final y el sistema de referencia de robot (su base).

En nuestro caso, el OpenMANIPULATOR-X tiene cuatro grados de libertad, ya que posee cuatro articulaciones de tipo revolución (RRRR) sin tener en cuenta la pinza, dada la simple geometría del brazo. Dado que tres de las articulaciones tienen los ejes paralelos entre sí, resulta más sencillo emplear el segundo procedimiento, el cual recibe el nombre de métodos geométricos. Consiste en realizar diferentes operaciones matemáticas que obtengan la relación que hemos mencionado entre las articulaciones y las coordenadas cartesianas.

Se han desarrollado una serie de ecuaciones que permiten resolver el problema de la cinemática directa a partir de los valores de las articulaciones. Empezaremos calculando la proyección de la distancia en el suelo desde la base del robot hasta el efector final.

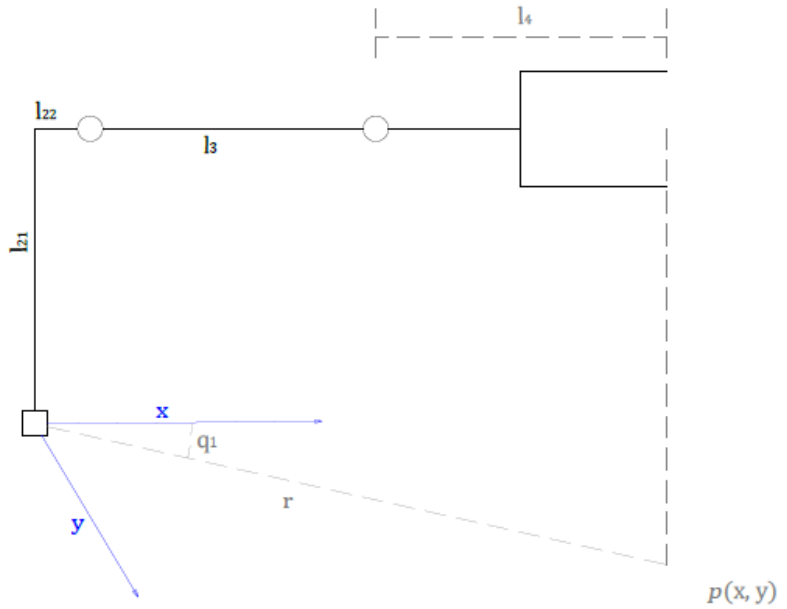


Figura 13: Esquema de la proyección r y las coordenadas X e Y. (Fuente elaboración propia).

$$q_{22} = \frac{\pi}{2} - q_2$$

$$r = l_{21} \cdot \sin q_{22} + l_{22} \cdot \cos q_{22} + l_3 \cdot \cos(-q_{22} + q_3) + l_4 \cdot \cos(-q_{22} + q_3 + q_4)$$

Primero de todo obtenemos en ángulo complementario de la segunda articulación q_{22} , seguidamente vamos calculando según cada tramo la proyección correspondiente, en el caso del eslabón dos se dividide en las expresiones que contienen las longitudes l_{21} y l_{22}. En cuanto al eslabón tres y cuatro están multiplicadas por las distancias de sus correspondientes eslabones y los cosenos con los ángulos de las articulaciones, restándole el efecto que tiene sobre ellos el ángulo complementario. Esta resta del ángulo complementario es debido a que si la segunda articulación es diferente de 90º, afecta a la inclinación,por consiguiente, se ha de incluir para el cálculo de los otros las articulaciones restantes.

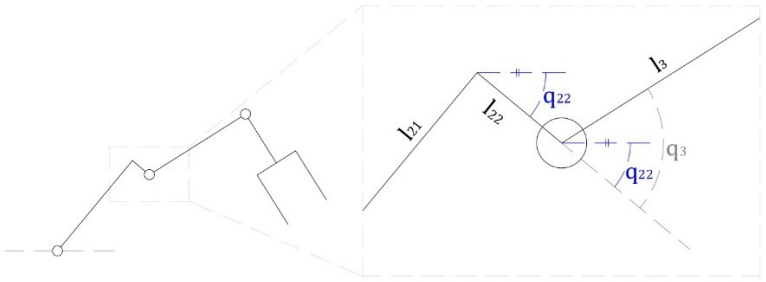


Figura 14: Impacto del ángulo complementario con la tercera y cuarta articulación. (Fuente: Elaboración propia).

Las coordenadas ya se pueden obtener, las dos primeras (X, Y) mediante la proyección r calculada previamente junto con la primera articulación q_1. La correspondiente con la altura tiene una expresión similar a la de r, pero teniendo en cuenta la altura del primer eslabón y algún cambio que se puede apreciar, que se puede entender mejor observando la *Figura 15.*

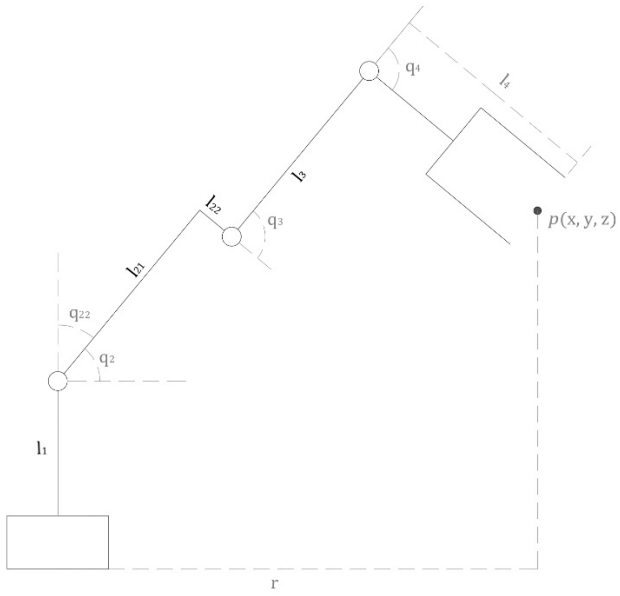


Figura 15: Ángulos de las articulaciones q2, q3 y q4. (Fuente elaboración propia).

Por tanto, las expresiones de la cinemática directa para obtener las coordenadas del efector final respecto a la base del robot serían la que se muestran a continuación:

$$p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} r \cdot \cos q_1 \\ r \cdot \sin q_1 \\ l_1 + l_{21} \cdot \sin q_2 + l_{22} \cdot \sin(-q_{22}) + l_3 \cdot \sin(-q_{22} + q_3) + l_4 \cdot \sin(-q_{22} + q_3 + q_4) \end{bmatrix}$$

6.1.2 Cinemática inversa

En la cinemática inversa, se debe obtener la posición de cada una de las articulaciones correspondientes para lograr alcanzar las coordenadas cartesianas especificadas, es posible en una misma coordenada haya varias configuraciones para alcanzarla. En otros robots, como los de seis grados de libertad, habría más

configuraciones conocidas como hombro adelante/atrás, codo arriba/abajo y muñeca arriba/abajo. Todo ello daría un total de ocho configuraciones diferentes para alcanzar una misma posición. En cuanto al robot que se va a emplear se tiene cuenta la particularidad del codo, debido a geometría del propio.

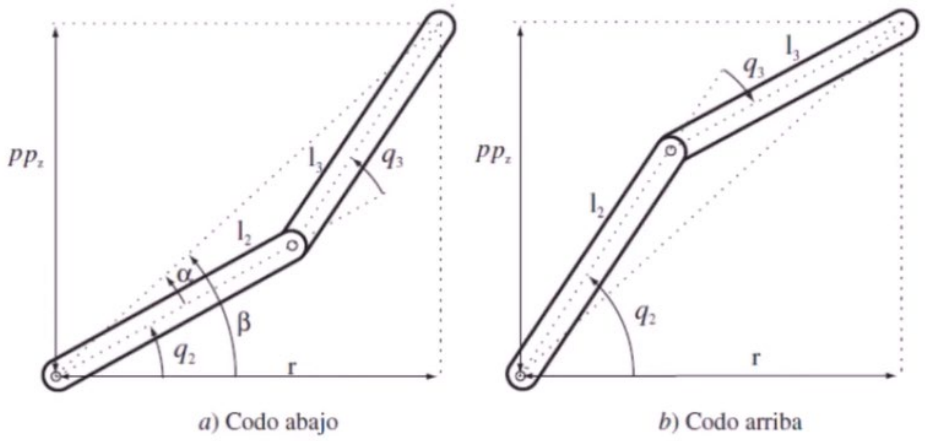


Figura 16: Configuración del codo. (Fuente: Apuntes de la asignatura de Sistemas Robotizados [12]).

Al igual que en la cinemática directa también hay otros métodos para obtener su inversa, porque depende mucho de lo complejo que sea su geometría, se emplea una u otra, la más conocida es también la de métodos geométricos o numéricos. El procedimiento empleado es el siguiente: antes de nada, se debe calcular el punto de muñeca. Para ello primero se obtiene el ángulo de la primera articulación, posteriormente mediante las coordenadas del efector final dadas y la geometría del robot se calcula donde se encuentra la articulación de la muñeca restándole la longitud del último eslabón, aplicado a cada una de las coordenadas ya que dependen del ángulo de la primera articulación y el del cuarto grado de libertad.

$$q_1 = \tan^{-1}\left(\frac{p_y}{p_x}\right)$$

Donde q_1 es el ángulo en radianes de la primera articulación que depende de p_x y p_y, las coordenadas de los ejes X e Y.

$$pm = \begin{bmatrix} pm_x \\ pm_y \\ pm_z \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} - l_4 \begin{bmatrix} \cos q_1 \cdot \cos q_t \\ \cos q_1 \cdot \sin q_t \\ \sin q_1 \end{bmatrix}$$

$$q_t = q_3 + q_4 - q_{22}$$

Donde pm es la matriz 3x1 que forma las coordenadas pm_x, pm_y y pm_z. Por otro lado q_t es el ángulo en radianes que corresponde al cuarto grado de libertad y que está formado por la suma de q_3, q_4 y la de q_{22} (es el complementario al ángulo q_2, que correspondería con la segunda articulación).

A continuación, calcularemos las distancias necesarias para posteriormente obtener los diferentes ángulos auxiliares con la finalidad de con ellos resolver las correspondientes articulaciones:

$$h = pm_z - l_1$$

$$r_1 = \sqrt{pm_x{}^2 + pm_y{}^2}$$

$$s = \sqrt{r_1{}^2 + h^2}$$

Se observa que h es la diferencia entre dos alturas, pm_z (correspondiente al punto de muñeca) y l_1(el primer eslabón). Seguidamente, obtenemos la distancia de la proyección con el suelo llamada desde la base del robot hasta el punto de muñeca, con las componentes restantes pm_x y pm_y.

$$\alpha_1 = \tan^{-1}\left(\frac{h}{r_1}\right)$$

$$\alpha_2 = \cos^{-1}\left(\frac{-l_3{}^2 + s^2 + l_2{}^2}{2 \cdot s \cdot l_2}\right)$$

$$\alpha_3 = \tan^{-1}\left(\frac{l_{22}}{l_{21}}\right)$$

El cálculo de los ángulos auxiliares α_1, α_2 y α_3 sirven para obtener la segunda articulación. En primer lugar, encontramos α_1, que se calcula a partir de dos de las distancias anteriores, más concretamente con la altura desde la segunda articulación al punto de muñeca h y la distancia proyectada en el suelo r_1. Cabe destacar que en los programas creados para el cálculo de la arcotangente se emplea la función *atan2(h,r₁),* para que sea válido en los cuatro cuadrantes. Respecto a α_2, se usa el teorema del coseno donde se necesitan las distancias de las aristas del triángulo formado, estas son l_2, el eslabón l_3 y la distancia s. Por último, α_3 es constante dado que está formado por dos distancias que componen el segundo eslabón, donde l_2 es la hipotenusa y los catetos son l_{21} y l_{22}, que como se puede comprobar forman 90°.

$$\beta_1 = \tan^{-1}\left(\frac{l_{21}}{l_{22}}\right)$$

$$\beta_2 = \cos^{-1}\left(\frac{-s^2 + l_2{}^2 + l_3{}^2}{2 \cdot l_2 \cdot l_3}\right)$$

Al igual que con los ángulos α_1, α_2 y α_3, se deben obtener los que forman la tercera articulación, reciben la denominación de β_1y β_2. El primero de ellos, β_1 es constante, al igual que α_3 debido a que se emplean los mismos valores que configuran el segundo eslabón ya mencionados previamente. El segundo, β_2 también se asemeja mucho a α_2 puesto que pertenecen al mismo triángulo, se emplea el teorema del coseno con las mismas aristas, pero cambia la disposición de las variables, ya que se trata de un ángulo diferente.

Una vez ya calculados todos los elementos necesarios se puede proceder al cálculo del resto de articulaciones según su configuración del codo:

- Configuración codo abajo:

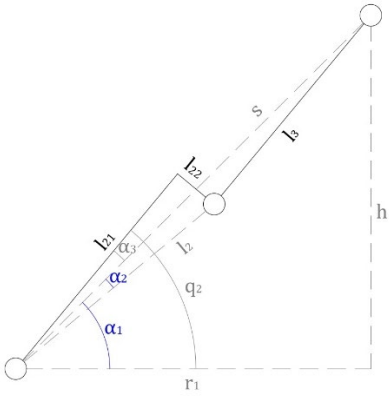


Figura 17: Ángulo auxiliar alpha2 contenido dentro de alpha1. (Fuente: Elaboración propia).

$$q_2 = \alpha_1 - \alpha_2 + \alpha_3$$

Observamos que a q_2, en esta configuración, se le resta el ángulo α_2, debido a que para alcanzarse α_1 ya contiene a este, por tanto, la relación correcta para alcanzar el ángulo correcto es haciendo la diferencia entre ambos.

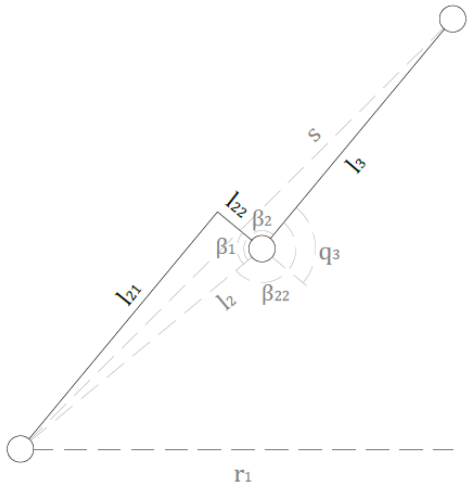


Figura 18: Esquema de los ángulos auxiliares de la tercera articulación en configuración codo abajo. (Fuente: Elaboración propia).

$$q_3 = ((2\pi - \beta_2) + \beta_1) - \pi)$$

En cuanto a q_3, se ven ciertas operaciones ya que cambia la forma del triángulo formado y se debe principalmente a la configuración que se muestra en la *figura 18,* tal y como se puede observar, el ángulo β_2 no ocupa la misma posición que cuando el codo se encuentra abajo.

- Configuración codo arriba:

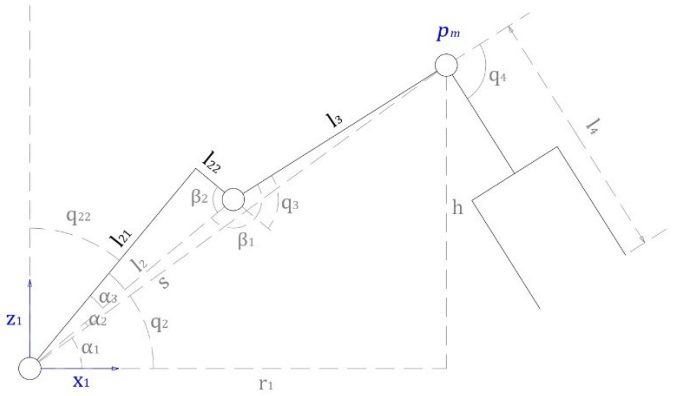


Figura 19: Configuración codo arriba OpenMANIPULATOR-X. (Fuente: Elaboración propia).

En esta configuración, podemos observar como la segunda articulación ya es la suma de los tres ángulos *alpha* y la tercera es la suma de ambas *betas* restadas a π radianes.

$$q_2 = \alpha_1 + \alpha_2 + \alpha_3$$

$$q_3 = (\beta_1 + \beta_2) - \pi)$$

Por último, pero no menos importante, queda la cuarta articulación q_4 que depende de q_t, q_2 y q_3, tal y como ya hemos visto en la cinemática directa. Con todo ello, ya se tiene el problema de la cinemática inversa resuelta y se le pueden pasar las coordenadas al robot para que pueda calcular la trayectoria y situarse en el punto objetivo.

$$q_4 = q_t + (\frac{\pi}{2} - q_2) - q_3$$

6.2 Funciones principales del Arduino

En este apartado, se van a explicar las funciones creadas para ser empleadas en el programa y evitar escribir el mismo código muchas veces seguidas. Como en todo programa en C, primero se declaran antes del *setup,* para que el programa sepa de su existencia antes de ser empleadas y que no de ningún error. Posteriormente después del bucle principal se constituyen sus diferentes líneas de código para realizar las diferentes tareas específicas. La configuración previa del microcontrolador que se ha de realizar, así como instalación de librerías queda detallada en el documento de *anexos.*

Se descomponen en las siguientes:

- **directKinematics**
  Como su propio nombre indica, realiza los cálculos necesarios de la cinemática directa, cuyas ecuaciones se encuentran en el apartado 6.1.1. En consecuencia, como entrada necesita los ángulos de las articulaciones deseadas y los datos necesarios como la medida de los eslabones de los robots. Ambas entradas son un *struct* de datos, al igual que la salida que se

devuelve que tiene la posición en coordenadas del mundo real con ángulo inclinación.

- **inverseKinematics**
  En esta función se mantiene como entrada los parámetros con las medidas de los eslabones y se invierte el de los ángulos de las articulaciones con el de la posición, es decir, este último ahora pasa a ser la salida y el otro un dato de entrada. Además, se le añade un dato extra a la entrada siendo este la posición del codo donde es una variable de tipo booleana que al ser verdadera toma el codo abajo y falso arriba. Con las ecuaciones desarrolladas previamente se calcula la cinemática inversa de este de una forma rápida y concisa.

- **calcula_qGoal**
  Lo que se intenta conseguir en esta función es transformar los datos de los ángulos de cada una de las articulaciones de radianes a un valor entre 0 y 4095, el cual es el rango que acepta el motor, para llegar a la posición deseada a excepción del quinto que corresponde a la pinza y que directamente se le pasa el valor correspondiente dentro de este rango según la queremos abierta o cerrada. Las transformaciones son las siguientes habiendo preestablecido que el cero de cada una de ellas está en una determinada posición como se muestra en los bocetos anteriores del robot:

$$qGoal_1 = q_1 \frac{4095}{2\pi}$$
$$qGoal_2 = \left(\frac{3\pi}{2} - q_2\right)\frac{4095}{2\pi}$$
$$qGoal_3 = (\pi - q_3)\frac{4095}{2\pi}$$
$$qGoal_4 = (\pi - q_4)\frac{4095}{2\pi}$$

- **leerPuertoSerie**
  Para comunicarse con MATLAB, se ha desarrollado esta función donde los parámetros de entrada son los siguientes: velocidad, aceleración, estado del codo, estado de la pinza y su salida es del tipo *struct* de posición. Recibe los datos de MATLAB y realiza los cálculos de verificación como lo es del *checksum*, si este no coincide envía al robot a la posición inicial. Los datos que contiene la trama se han especificado en el apartado que viene a continuación. Si son correctos los datos, guarda la posición a la que se quiere enviar al robot en coordenadas cartesianas y se devuelve por la salida. Se emplea el retorno de carro (\r) para saber cuándo una trama se ha terminado y que se pueda entender mejor que esta función ha realizado un flujograma que se muestra a continuación:

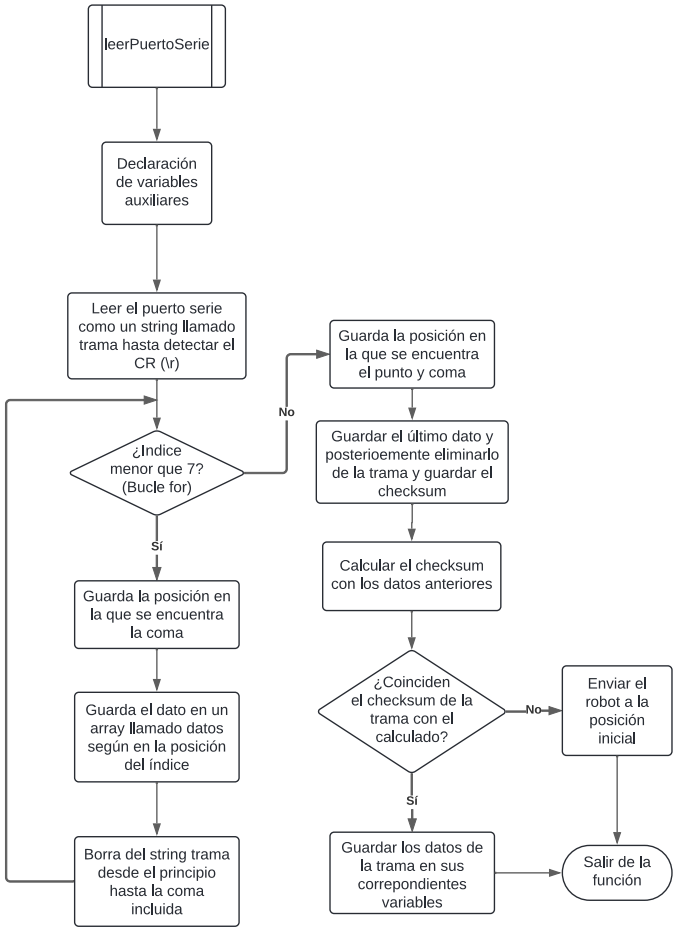


Figura 20: Flujograma de la función leerPuertoSerie. (Fuente: Elaboración propia).

En el bucle principal, se está esperando la entrada de alguna comunicación, se pasa estos datos a la cinemática inversa. Posteriormente, una vez transformados a ángulos de articulaciones, se envían a calcula_qGoal para poder mover los motores de forma síncrona. En consecuencia, el programa sigue el siguiente diagrama de flujo:

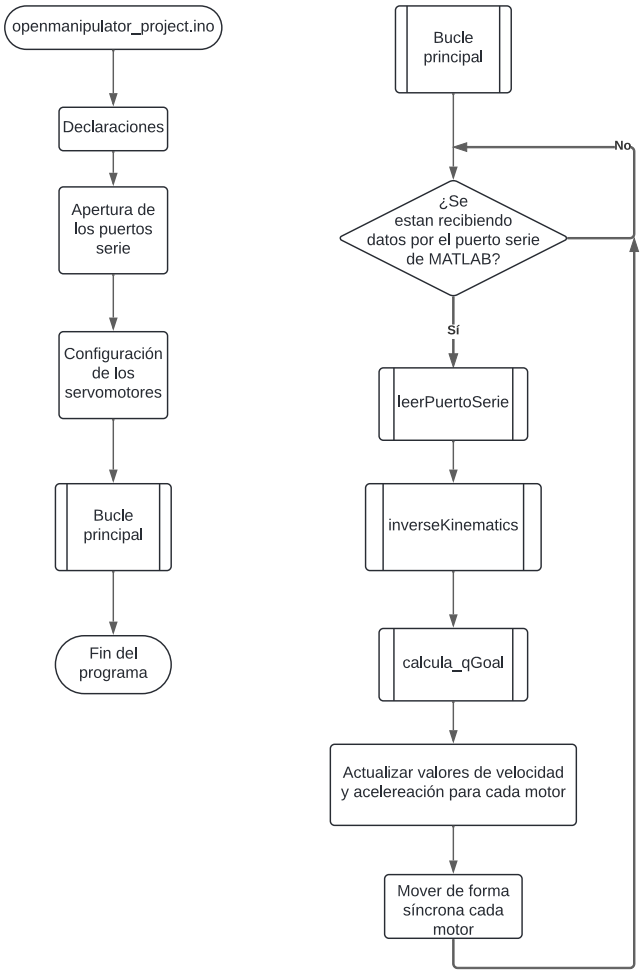


Figura 21: Flujograma del funcionamiento del programa principal. (Fuente: Elaboración propia).

6.3 Visión artificial

En este subapartado se explican los procesos que se han seguido para poder detectar la pelota y recolectar los datos de su trayectoria para posteriormente analizarlos y corroborar los experimentos mediante la optimización bayesiana.

El principal objetivo que se pretende es obtener la ubicación de un objeto en el plano de una imagen a coordenadas del mundo real, para ello se necesitan ciertos parámetros y conocer las distintas transformaciones que se realizan para obtenerlo.

Por tanto, primero conoceremos los parámetros y operaciones que se deben realizar, y luego los procedimientos a seguir para obtenerlos, para ello nos ayudaremos tanto de la librería de código abierto de *OpenCV* y de unos *Add-on* de MATLAB, en concreto: *Computer Vision Toolbox, Image Processing Toolbox* y *MATLAB Support Package for USB Webcams*.

6.3.1 Marco teórico

A continuación, se explica cómo se van a realizar las diferentes transformaciones necesarias y los datos que necesitamos para poder hacer el mejor seguimiento posible. En la *figura* que encontramos a continuación, se explican de forma visual

algunas de las características tales como: la disposición del eje de la cámara, la proyección al plano de la imagen, las coordenadas del mundo real, entre otros. Se trata de una representación de una cámara estenopeica (sin lente), es decir, no tendría ningún tipo de distorsión, en la realidad si tenemos y se puede calcular y corregir.

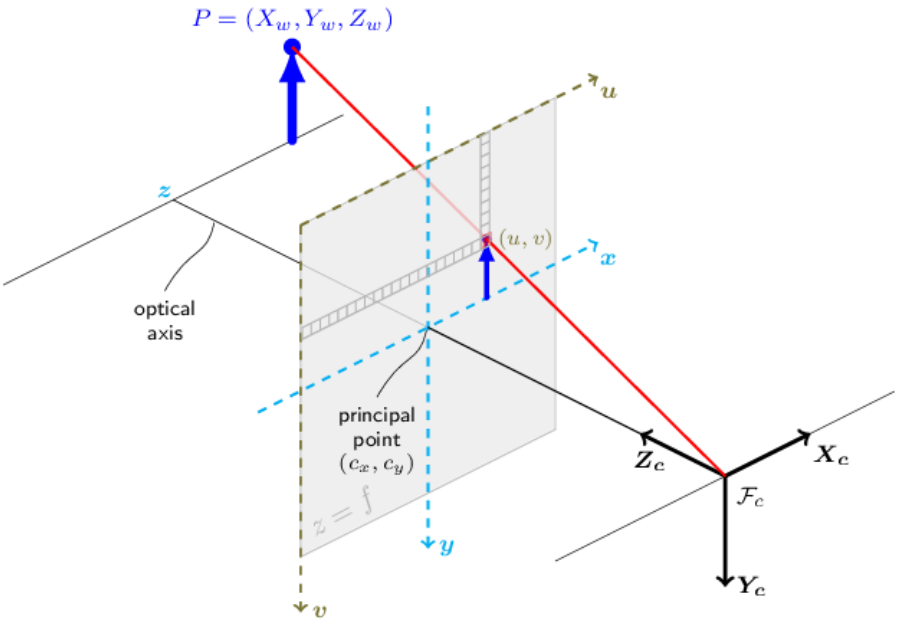


Figura 22: Esquema de una cámara estenopeica. (Fuente: OpenCV [15]).

La siguiente ecuación relaciona los parámetros de la figura anterior, la cual se empleará para obtener los puntos deseados.

$$s\,p = A\,[R|t]\,P_w = A(RP_w + t)$$

Donde s es el factor de escala que relaciona el objeto del mundo real con el proyectado en la cámara, p son las coordenadas en píxeles del plano de la imagen, A es la matriz intrínseca que contiene la información de la distancia focal y del punto principal que suele estar en el centro de la imagen, $[R|t]$ es la matriz extrínseca que está formada por una matriz de rotación y un vector de traslación y, por último, P_w son las coordenadas del mundo real.

Si desglosamos esta ecuación queda de la siguiente forma:

$$s\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}f_x & 0 & c_x\\0 & f_y & c_y\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}r_{11} & r_{12} & r_{13} & t_x\\r_{21} & r_{22} & r_{23} & t_y\\r_{31} & r_{32} & r_{33} & t_z\end{bmatrix}\begin{bmatrix}X_w\\Y_w\\Z_w\\1\end{bmatrix}$$

Para obtener las coordenadas reales necesitamos obtener cada uno de los datos de la ecuación. Para el factor de escala se debe obtener realizando ciertas operaciones una vez obtenidas las matrices como la intrínseca y extrínseca. Primero se ha de resolver el vector de coordenadas tridimensionales a partir de la proyectadas bidimensionalmente por el plano de la imagen.

$$Vtri = R^{-1}\left(A^{-1}\begin{bmatrix}u\\v\\1\end{bmatrix}\right)$$

$$Vbi = R^{-1}\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

El resultado de cada uno es un vector de 3x1, en el que la coordenada que nos interesa es la Z, dado que así sabremos según la distancia que hay desde la cámara al objeto, es decir, la relación que hay entre el objeto tridimensional y el del plano de la imagen. Obtenemos el factor de escala sabiendo que la altura es nula es decir $Z_w = 0$ está contenido todo en el mismo plano.

$$s = \frac{Vbi(3)}{Vtri(3)}$$

Aún siguen faltando datos que se deben obtener, como las coordenadas en píxeles *u* y *v* del objeto detectado, básicamente el procedimiento que se sigue es usando los comando y máscaras para la detección de los colores específicos.

La matriz intrínseca *A* se obtiene de la propia cámara, es decir, son parámetros que cada cámara tiene, datos como la distancia focal y el centro de la imagen se obtienen realizando una calibración con ayuda de un patrón, el conocido patrón del tablero de ajedrez.

En cuanto a la extrínseca, la obtendremos con varios comandos de la propia librería y estableciendo un punto de referencia aplicaremos los coeficientes de distorsión para corregir la imagen y se obtendrán los datos necesarios para la matriz extrínseca.

Por último, una vez conocidos todos los parámetros deberemos despejar las coordenadas del mundo real de la primera ecuación:

$$P_w = R^{-1}(s\,A^{-1}p\ - t)$$

6.3.2 Calibración de la cámara: Matriz intrínseca

Tal y como se ha mencionado anteriormente, necesitamos saber la distancia focal de la cámara(f_x,f_y) y el centro de la imagen (c_x,c_y). Todos ellos están contenidos en una misma matriz, llamada matriz intrínseca.

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

También esta matriz es comúnmente llamada K, como podemos observar tiene una dimensión de tres filas por tres columnas, por tanto, al ser cuadrada y su determinante es no nulo, es invertible.

Para la calibración se emplea la aplicación de una de las *toolbox* instaladas en MATLAB, llamada *Camera Calibrator*. Que, junto a un patrón de ajedrez de 30 mm de lado del cuadrado, podremos realizar capturando imágenes y posteriormente procesarlas para localizar los puntos internos del patrón.

Una vez abierta la aplicación tenemos dos opciones, o bien importar las fotos ya realizadas previamente con la webcam o bien realizarlas con la misma aplicación, tal y como se muestra en la siguiente *figura*:

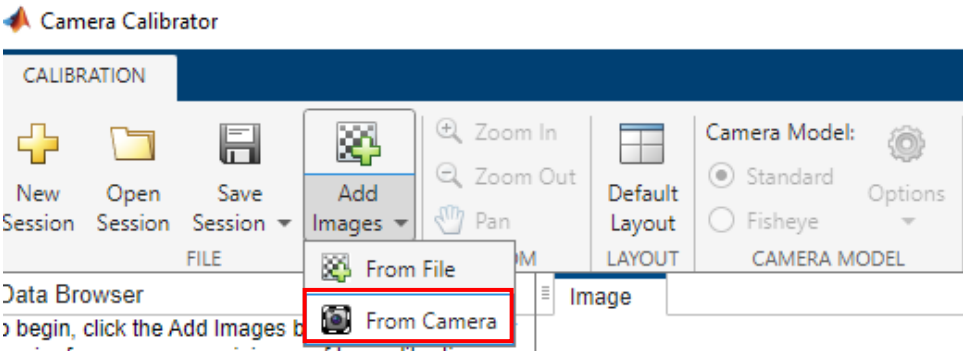


Figura 23: Barra de opciones de la aplicación Camera Calibrator. (Fuente: Elaboración propia).

En nuestro caso hemos optado por tomarlas directamente desde la cámara, programando que cada diez segundos capture una imagen para que dé tiempo a mover el patrón del tablero de ajedrez en diferentes posiciones. Capturando un total de diez imágenes ya se puede realizar una calibración bastante certera.

La aplicación una vez haya capturado las imágenes, nos saldrá una ventana *pop-up* en la que tenemos que seleccionar algunos parámetros como el patrón, las medidas del lado del cuadrado, junto con sus unidades y la distorsión que tiene.

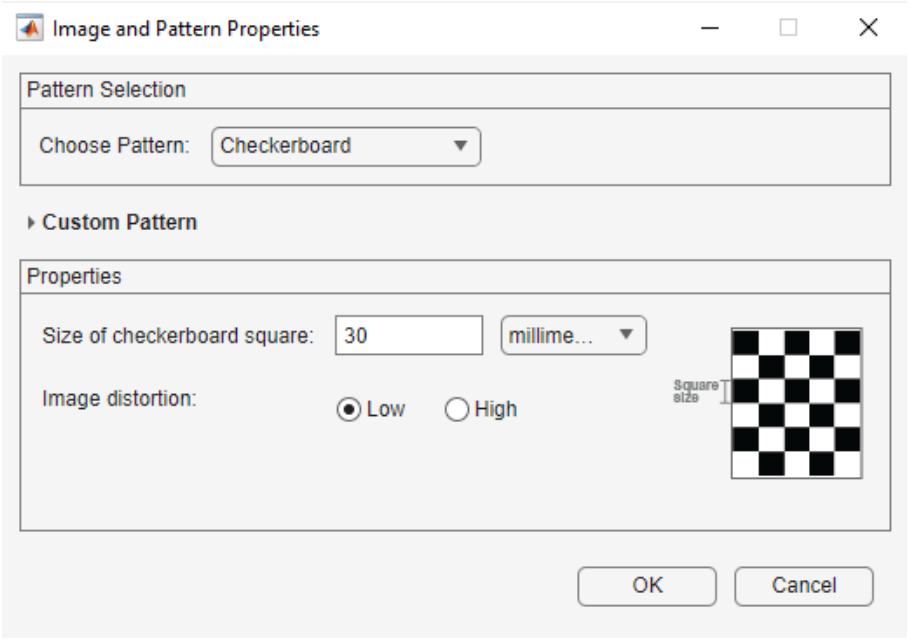


Figura 24: Configuración previa al calibrado en la aplicación de Camera Calibrator. (Fuente elaboración propia).

Ya habiendo aceptado los parámetros el programa localizará el patrón, cogerá los vértices internos de cada cuadrado, es decir, las filas y columnas más exteriores no los tiene en cuenta y establece en el de más abajo a la izquierda el punto de referencia. Ahora bien, como ya sabe lo que mide cada cuadro puede realizar las operaciones necesarias para obtener la matriz que necesitamos.

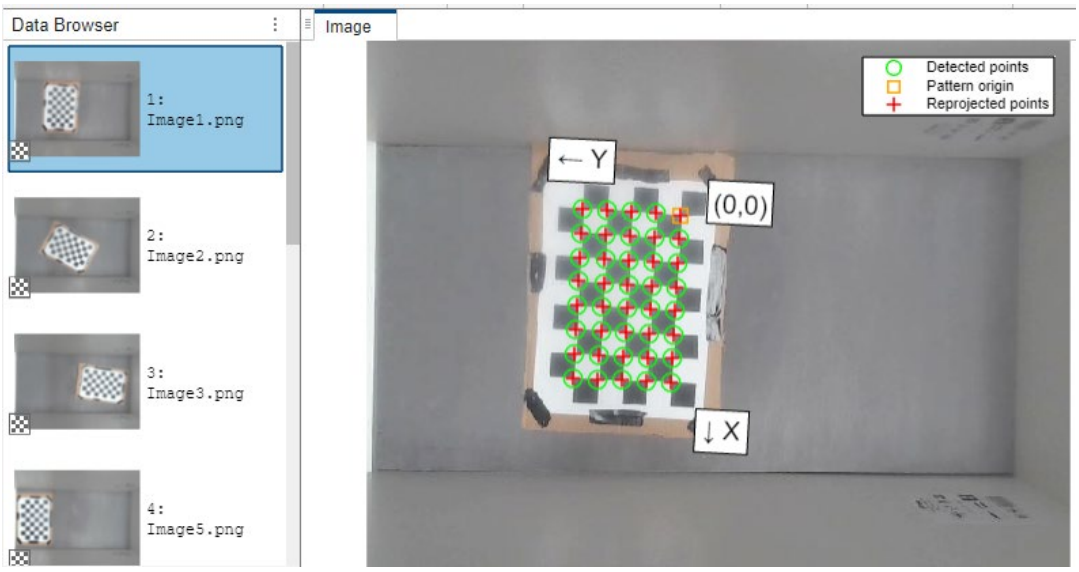


Figura 25: Imágenes cargadas en la aplicación, localizando el patrón en las imágenes. (Fuente: elaboración propia).

Una vez ya lo tenemos preparado, solo habría que pulsarle al botón de *Calibrate* para que empezase el procedimiento y así poder analizar algunos de los resultados, como la propia matriz intrínseca y su incertidumbre, así como el error en píxeles e incluso la posición de la cámara centrada y la representación de los planos con respecto a la cámara centrada. Empezando por los errores de reproyección [16], estos son producidos en cada imagen al calcular la distancia entre los puntos localizados en el patrón y su la correspondiente coordenada en mundo real. Se mide en píxeles y debe ser bajo, para obtener el mejor calibrado posible se tendría que descartar las imágenes con mayor error, añadir nuevas imágenes para el recalibrado. En este caso, se puede ver a continuación que el error medio es de 0.10 píxeles.

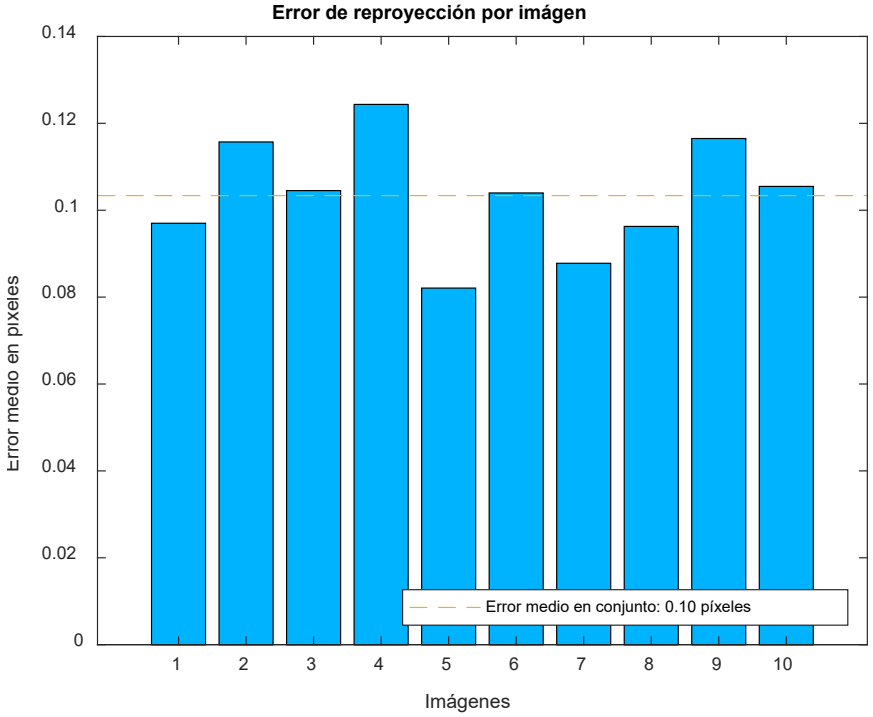


Figura 26: Error de reproyección por imagen en el proceso de calibración. (Fuente: Elaboración propia).

Además, se para comprobar que no hay errores en la calibración, se puede dibujar la cámara y la posición del patrón en cada imagen, esto se debe a que realiza el cálculo de la matriz extrínseca para cada una de las imágenes tomadas y para corroborar que no hay error basta con simplemente observar que no se dibuja ningún patrón detrás de la cámara o muy lejos de esta. En nuestro caso, corresponde perfectamente con la distancia que había entre la cámara y los patrones en diferentes posiciones, cada una representado con un color y enumerado con su correspondiente número de imagen.

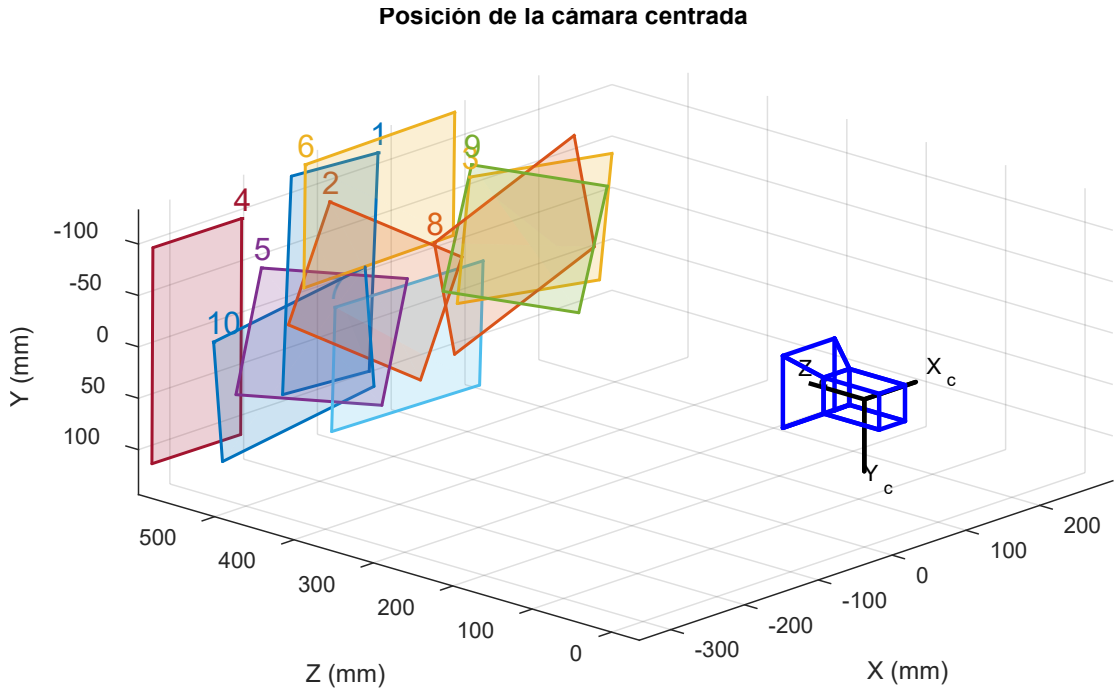


Figura 27: Posición de la cámara centrada y representación de los patrones de cada imagen. (Fuente: Elaboración propia).

Por otro lado, también representamos el caso en el que el patrón está centrado, es decir sólo habría uno representado y las diferentes diez cámaras posicionadas, según cada imagen, sería como la figura anterior, pero a la inversa, en el hipotético caso de que el patrón este fijo y la posición de la cámara no, el cual no es nuestro caso.

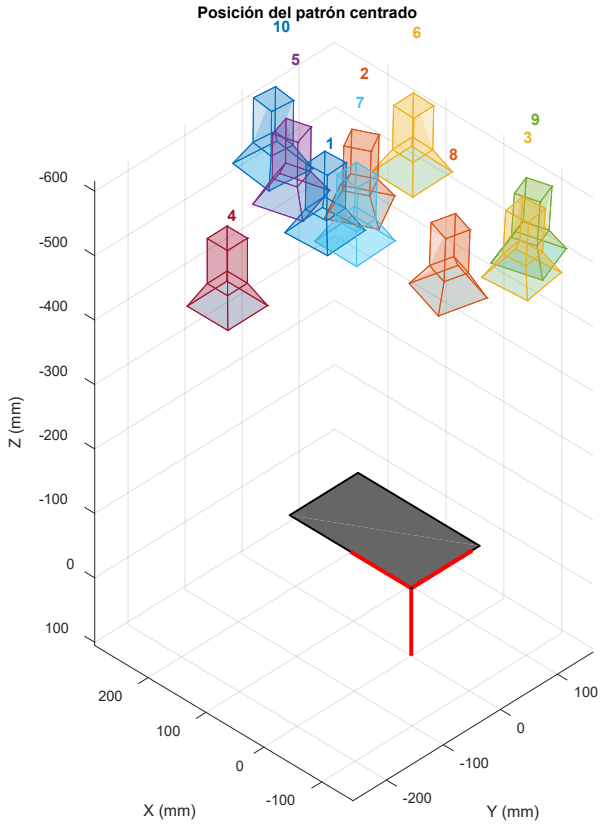


Figura 28: Posición del patrón centrado y representación de la posición de la cámara. (Fuente: Elaboración propia).

Antes de continuar, hay que tener en cuenta la distorsión causada por la lente de la cámara, esta puede ser tanto radial como tangencial. La primera de ellas puede ser negativa (forma de almohada) o positiva (forma de barril), esta característica viene dada por el primer de los coeficientes, denominado k_1, según su signo determinará un tipo de distorsión u otra. emplearemos dos coeficientes para la distorsión, se pueden obtener hasta tres de ellos, pero en casos donde la distorsión es muy elevada.

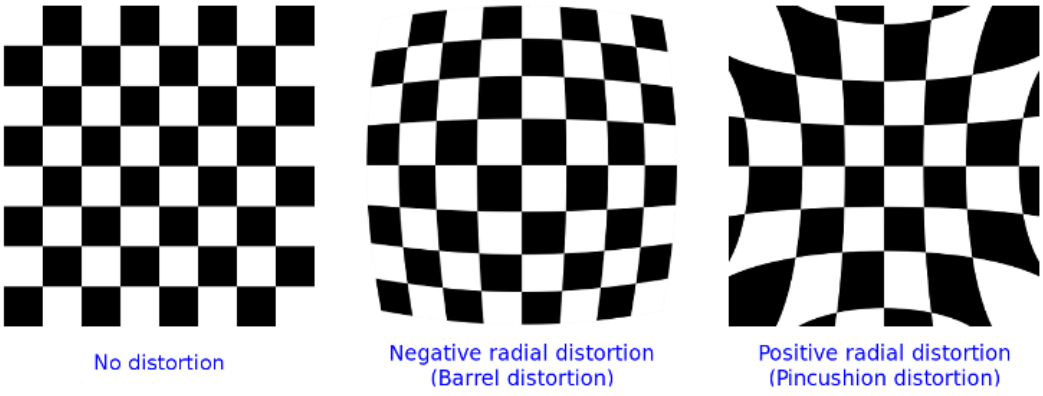


Figura 29: Ejemplos de una imagen con el patrón de ajedrez sin distorsión, con distorsión negativa y con distorsión positiva. (Fuente: OpenCV[15]).

Los polinomios que se emplean para el cálculo de la distorsión son los siguientes:

$$x' = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y' = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$r^2 = x^2 + y^2$$

Donde x, y son las coordenadas normalizadas. En nuestro caso tras realizar la calibración con sólo dos parámetros, se obtiene el siguiente resultado, donde la distorsión radial es positiva dado que el primer parámetro lo es:

$$\begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix} = \begin{bmatrix} 0.0104 \ +/- \ 0.1055 \\ -0.0010 \ +/- \ 0.0194 \\ 0 \end{bmatrix}$$

Por otro lado, también existe la distorsión tangencial que es causada cuando la lente y el sensor no son paralelos. Está formado por un vector de dos parámetros p_1 y p_2 . La ubicación de estos píxeles no distorsionados se muestra con el origen en el centro óptico [17].

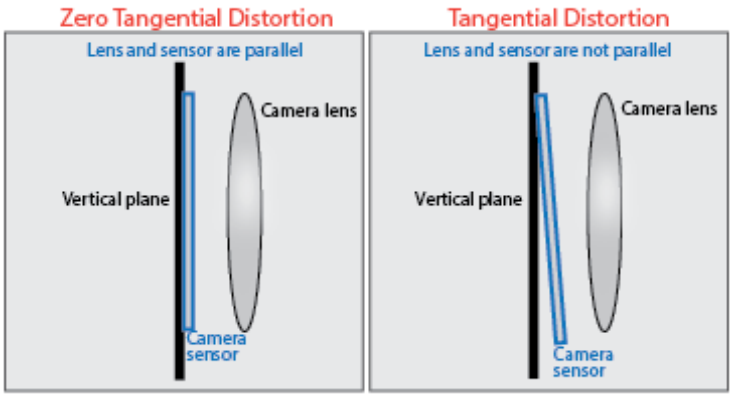


Figura 30: Representación de la distorsión tangencial. (Fuente: Mathworks [17]).

Las ecuaciones correspondientes a este tipo de distorsión son las que se muestran a continuación:

$$x'' = x' + [2 \cdot p_1 \cdot x \cdot y + p_2 \cdot (r^2 + 2 \cdot x^2)]$$

$$y'' = y' + [p_1 \cdot (r^2 + 2 \cdot y^2) + 2 \cdot p_2 \cdot x \cdot y]$$

$$r^2 = x^2 + y^2$$

Donde x, y son parámetros no distorsionados. Y el valor de los parámetros de la distorsión tangencial con su incertidumbre son los siguientes:

$$\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 0.0002 \ +/- \ 0.0011 \\ -0.0012 \ +/- \ 0.0059 \end{bmatrix}$$

Para poder entender como se aplica la distorsión, hay que realizar una breve explicación de las ecuaciones explicadas en el marco teórico, separándolas en dos partes: transformación de las coordenadas del mundo a coordenadas de la cámara y la proyección a coordenadas de la imagen.

$$s\, p = A\, [R|t]\, P_w = A(RP_w + t)$$

Primero de todo se obtiene la posición de la cámara

$$P_{camera} = \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = R \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + t$$

Posteriormente las coordenadas normalizadas, que son las empleadas en las ecuaciones de distorsión:

$$x = \frac{X_c}{Z_c} \qquad y = \frac{Y_c}{Z_c}$$

Al sustituir la función quedaría de la siguiente forma:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Pero al aplicar la distorsión debemos sustituir las obtenidas previamente:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix}$$

Esto se realiza internamente en la función que se emplea para la matriz extrínseca, explicada en el siguiente subapartado.

Por último, de la calibración obtenemos los resultados de los parámetros característicos de la matriz intrínseca, la distancia focal y el centro de la imagen, datos imprescindibles para realizar un seguimiento de nuestra pelota.

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 420.0350 & 0 & 312.8125 \\ 0 & 419.9953 & 245.4877 \\ 0 & 0 & 1 \end{bmatrix}$$

6.3.3 Matriz extrínseca

Hasta ahora, el procedimiento seguido no se ha empleado la librería OpenCV. Se ha ejecutado la calibración con una aplicación de un paquete de MATLAB, pero para obtener los datos extrínsecos y realizar las operaciones de captura de vídeo en tiempo real, es necesario utilizar este tipo de librería instalada mediante la ayuda de un compilador externo, para ello se instala el mexOpenCV [18] con la ayuda de un vídeo tutorial [19].

El primer paso que tenemos que hacer es detectar un objeto. Para ello nos ayudaremos del color de éste para que nos sea más fácil. Aplicando una máscara se puede localizar el color que deseamos, pero para ello primero hay que entender que es el *HSV, d*e sus siglas en el inglés *Hue(*tonalidad), *Saturation*(saturación) y *Value*(valor). Estos tres parámetros en OpenCV van de 0 a 180 para H y desde el valor 0 al 255 para S y V. Permiten discernir entre el rango de tonalidades de los diferentes colores para discernir los rangos de cada uno de los colores.

El *frame* capturado en tiempo real, debe pasarse de RGB (*Red, Green, Blue*), son los valores que toma cada píxel para de representarse en una pantalla, a HSV, con el comando:

```
frameHSV = cv.cvtColor(frame, 'RGB2HSV');
```

Una vez ajustado esta imagen debemos crear la máscara para que el programa sea capaz de diferenciar qué color queremos detectar. Para ello lo primero que hay que hacer es elegir los parámetros necesarios para crear el rango que necesitamos.

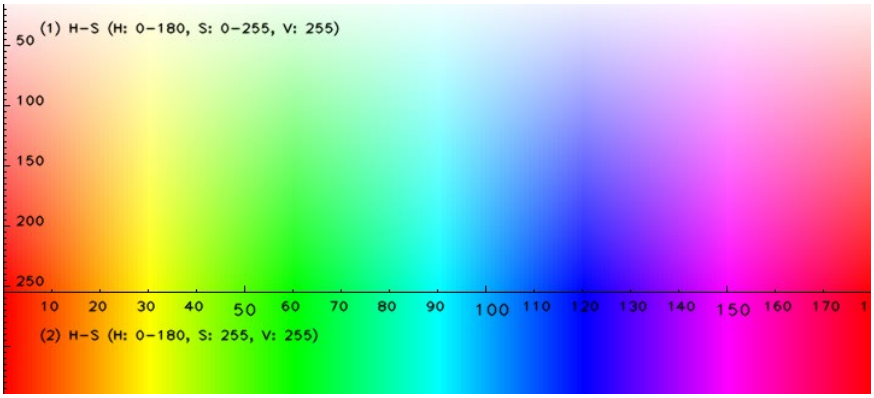


Figura 31: Rango de los parámetros HSV. (Fuente: Gabriela Solano [20]).

Por ejemplo, si queremos asignar un rango para el color amarillo observamos que para el rango de tonalidad (H), se encuentra entre 20 a 30 y para el S y V, van desde el 100 hasta el 255. Una vez detectado, para que sea más visual, se dibuja el contorno del objeto y se localiza su centroide, el cual a partir de éste obtener sus coordenadas en píxeles, es decir *u* y *v*, datos también son necesarios para la transformación que requerida.

El rango empleado de HSV para detectar el color amarillo es el siguiente:

$$yellowBajo = [20 \quad 100 \quad 100]$$

$$yellowAlto = [30 \quad 255 \quad 255]$$

Ahora que ya tenemos los dos rangos se debe aplicar el comando `cv.inRange para`, finalmente aplicar la máscara con la función `cv.bitwise_and`, que consiste en emplear el frame con el operando AND utilizando la máscara que contiene el rango generado, por tanto la salida sería un array que detectaría las tonalidades del color deseado. El siguiente paso consistiría en encontrar los contornos de los objetos de este color. Por ello, se debe emplear en la función `cv.findContours` habiendo pasado el resultado de la detección de colores previamente a escala de grises de la misma forma que lo hemos hecho antes con el frame a HSV con la propiedad en la función `cv.cvtColor` de `'RGB2GRAY'`. Por consiguiente, dispondríamos de todos los contornos de este color, incluso los más pequeños, para hacer un filtro y que no nos detecta "ruido" debemos calcular el área de cada uno de ellos y así sólo tener en cuenta los correspondientes con el objeto que queremos, en este caso la pelota. Así pues, empleamos `cv.contourArea` y conseguimos un valor equivalente al área del contorno, el cuál si es mayor de cierto valor, se procede a calcular su centroide con *cv.moments*, ya que devuelve un resultado con unos parámetros en formatos *struct*. Para el cálculo del centroide en píxeles se realizan las siguientes operaciones:

$$u = \frac{M.m10}{M.m00}$$

$$v = \frac{M.m01}{M.m00}$$

Donde M es el *struct* de datos tras aplicar la función de momentos. Cabe destacar que previamente a estos cálculos se comprueba que el valor $M.m00$ (representa el área del contorno) no sea cero, si es así, se actualiza a uno para evitar indeterminaciones dado que se encuentra en la parte del divisor. Respecto a $M.m10$, es el momento espacial en la dirección x, $M.m01$en la dirección y.

Posteriormente, se procede a poner el contorno, y su centroide junto con las coordenadas reales. Sin embargo, debemos antes obtener la matriz extrínseca motivo por el cual empelamos la siguiente función de OpenCV:

```
[rvec, tvec, success] = cv.solvePnP(centrosMreal,centrosPvirtual,
          camIntrinsicsK,'DistCoeffs',coefDistorsion)
```

SolvePnP calcula la posición del mundo 3D al 2D. Solamente se necesitaría, localizar cuatro objetos de la imagen y saber su posición en la realidad. Los siguientes parámetros de entrada son los siguientes:[21]

- ***centrosMreal***: Es un vector que contiene cuatro puntos del mundo real XYZ, donde Z=0 dado que la altura es nula y no todos los objetos están en el mismo plano.
- ***centrosPvirtual***: Es un vector también de cuatro puntos, pero esta vez son los centroides de los objetos localizados, es decir en píxeles (u, v).
- ***coefDistorsion***: Son los coeficientes de distorsión obtenidos de la calibración, tanto radiales como tangenciales, presentados en el siguiente formato: [k_1; k_2; p_1; p_2]

En cuanto a lo que devuelve la función son los siguientes parámetros:

- ***rvec***: Es un vector de rotación, que con la función *cv.Rodrigues* podemos transformar en matriz de rotación.
- ***tvec***: Es el vector de traslación, que junto con la matriz de rotación obtendríamos la extrínseca.
- ***success***: Devuelve un uno lógico si la transformación se ha realizado correctamente.

Una vez entendidas las funciones principales utilizadas, procedemos a aplicarlas en el proyecto.

Para poder emplear esta función deberemos detectar cuatro objetos, en este caso se ha optado por cuatro colores diferentes y se ha medido su posición real tomando como referencia uno de ellos en coordenadas de mundo real, es decir, nuestro (0,0,0). Los otros tres puntos se han medido y se han dispuesto en un determinado orden midiendo las distancias para saber las coordenadas reales que tienen. Los colores que se han empleado son: naranja, verde, cian y rosa. Se realiza de la misma forma que hemos hecho con el color amarillo para las máscaras.

Figura 32: Representación de las coordenadas y los círculos para la calibración en el mundo real. (Fuente: Elaboración propia).

Por otro lado, ya dispondríamos de las coordenadas de cada uno de los círculos que emplearemos para la calibración en la siguiente tabla:

Objetos	Posición XYZ (mm)
Círculo naranja	(780,385,0)
Círculo verde	(17,379,0)
Círculo cian	(785,16,0)
Círculo rosa	(17,16,0)

Tabla 2: Posición en el mundo real de cada círculo.

Habrá que ejecutar el script de MATLAB que se encuentra en los anexos, llamado *calibrarReferencia.m* y al detectar los cuatro círculos nos devolverá tanto el vector de rotación como el de traslación, es decir *rvec* y *tvec* respectivamente. Estos son los resultados:

$$rvec = \begin{bmatrix} -0.0202 \\ 3.1399 \\ -0.0023 \end{bmatrix}$$

$$tvec = \begin{bmatrix} 396.5078 \\ -193.6990 \\ 160.1134 \end{bmatrix}$$

Seguidamente, los añadimos a los scripts llamados *camera_detectaAmarillo.m* y *visionCapture.m,* donde se le pasan estos datos a la función *calculaCoordenadasMreal.m* estas funciones quedan detalladas en los anexos, la cual internamente calcula la matriz de rotación con *cv.Rodrigues* se obtendría. Por tanto, la matriz extrínseca la cual es la siguiente:

$$[R|t] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} = \begin{bmatrix} -0.9999 & -0.0129 & 0.0016 & 396.5078 \\ -0.0129 & 0.9999 & -0.0015 & -193.6990 \\ -0.0016 & -0.0015 & -1 & 160.1134 \end{bmatrix}$$

Así pues, ya concluidos todos los datos necesarios se puede hacer un seguimiento del recorrido de la pelota, con los datos mencionados y los parámetros del robot, está todo listo para proceder con la optimización

Con todo esto ya puede localizar la pelota ubicando el color amarillo tal, así como las coordenadas de su centroide, que pertenecen a la del hoyo objetivo [0.365,0.150] como se muestra a continuación:

Figura 33: Detección de la pelota de frontón en el agujero con coordenadas reales. (Fuente: Elaboración propia).

6.4 Optimización Bayesiana

El objetivo que se pretende con este tipo de optimización es encontrar el mínimo de una función que a priori es desconocida, como si de una caja negra se tratase, a partir de una serie de predicciones utilizando procesos gaussianos. Lo que se quiere conseguir es minimizar el número de experimentos hasta encontrar los mejores parámetros. Antes de continuar debemos conocer cómo funciona internamente dicha optimización y los pasos que siguen:

1 Introducir datos de experimentos iniciales (Requisito no obligatorio).
2 Decidir cuál o cuáles son los siguientes parámetros, x_k.
3 Observar el resultado obtenido en la función$(x_k, y_k = f(x_k) + e_k)$.
4 Actualizar el modelo del proceso gaussiano y guardarlo los datos obtenidos.
5 Comprobar si se ha cumplido el objetivo para finalizar la optimización. Volver al segundo paso si no es así.

6.4.1 Proceso gaussiano

Mediante el uso de la distribución de probabilidad, se es capaz de predecir de la función objetivo no solo los parámetros de entrada que se necesitan sino además la incertidumbre del ruido que puede tener dicha aproximación. Para un número finito de puntos $\{x^{(1)}, \dots, x^{(m)}\}$, y las evaluaciones de la función asociados $\{y^{(1)}, \dots, y^{(m)}\}$, se encuentran distribuidos de acuerdo con [22]:

$$\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \sim N\left(\begin{bmatrix} m(x^{(1)}) \\ \vdots \\ m(x^{(m)}) \end{bmatrix} , \begin{bmatrix} k(x^{(1)}, x^{(1)}) & \cdots & k(x^{(1)}, x^{(m)}) \\ \vdots & \ddots & \vdots \\ k(x^{(m)}, x^{(1)}) & \cdots & k(x^{(m)}, x^{(m)}) \end{bmatrix} \right)$$

Donde $m(x)$ es la media de la función y $k(x, x')$ es la función de covarianza o kernel. Y la media predicha $\hat{\mu}(x)$, se puede expresar en función de x:

$$\hat{\mu}(x) = m(x) + K(x, X)K(X, X)^{-1}(y - m(X))$$

Por tanto, la varianza de la media predicha también se puede expresar en función de x:

$$\hat{\sigma}(x) = K(x, x) - K(x, X)K(X, X)^{-1}K(X, x)$$

Donde se tienen en cuenta los puntos que se están evaluando x y los que ya se han evaluado a priori X.

6.4.2 Función de covarianza

La covarianza o kernel es una ecuación que se emplea en el aprendizaje supervisado, como lo es la optimización bayesiana. Intenta predecir el resultado de puntos cercanos para valores similares de entrada espera valores similares en la respuesta. Una de la ecuaciones más empleadas y conocidas es la exponencial cuadrática.

$$K(x, x') = \exp\left(-\frac{(x - x')^2}{2\ell^2}\right)$$

Donde ℓ corresponde con la escala de longitud la cual, a grosso modo, correspondería con que hay que desplazarse hasta que la función de objetivo cambie significativamente.

En este caso, se va a emplear el *Matern 5/2* que es el que incorpora la *Toolbox* de *bayesopt* de MATLAB predefinido dado que internamente usa la función de *fitrgp.* Esta es la encargada de realizar el proceso gaussiano y no es modificable dentro del programa.

$$K(x, x') = {\sigma_f}^2\left(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2}\right)\exp\left(-\frac{\sqrt{5}r}{\ell}\right)$$

Donde σ_f es la desviación estándar, r la distancia euclídea entre (x, x') y ℓ es la escala de longitud. [23].

6.4.3 Funciones de adquisición

Las funciones de adquisición establecen el criterio de decisión para elegir el siguiente punto a explorar y posteriormente se realiza el nuevo experimento en la función

objetivo. Estas son esenciales en la optimización bayesiana, ya que existen diversas de ellas y dependiendo del problema unas funcionan mejores que otras. A continuación, se van a detallar las más empleadas y las que se tratan en este proyecto.

- Lower confidence bound

Se establece una incertidumbre en la función objetivo, donde los valores seguramente se encuentran cerca de estos límites. Posteriormente, el punto elegido para el siguiente experimento es el mínimo del límite inferior. La fórmula que sigue esta función de adquisición es la siguiente:

$$LCB(x) = \hat{\mu}(x) - \hat{\sigma}(x)\alpha$$

Donde, $\hat{\mu}(x)$ es la media y $\hat{\sigma}(x)$ es la desviación típica. Por ejemplo, si queremos asegurarnos según la disposición del normal entorno a un 95% deberemos establecer un rango de dos veces la desviación típica 2σ, lo que equivale a un valor de α de 1,96.

En la figura que se muestra a continuación, se puede apreciar un ejemplo del empleo de la función de adquisición y su funcionamiento para un 95%, en una función objetivo conocida en la que sabemos cuál es el mínimo.

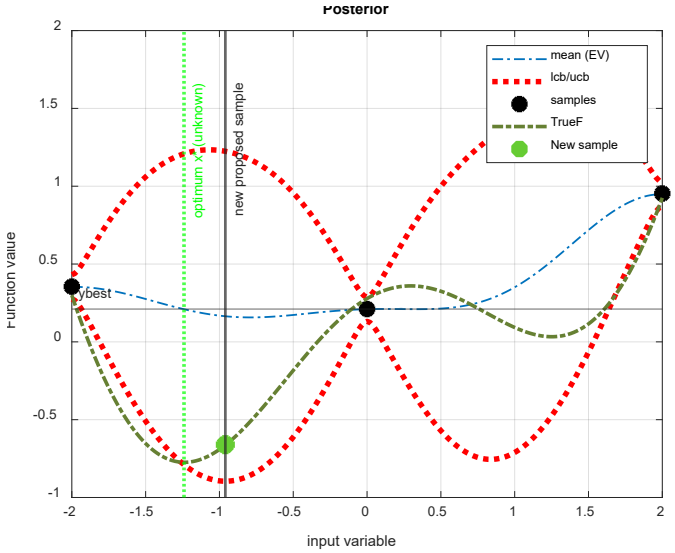


Figura 34: Ejemplo tercera iteración de LCB en una función objetivo predefinida conocida. (Fuente: Dr. Antonio Sala-Piqueras [24]).

En color azul se puede ver la media de la función objetivo que se va obteniendo con cada iteración. Como se puede apreciar las líneas en rojo indican el límite donde pasa el 95% de posibilidades de que la función objetivo se encuentre en ese intervalo. Además, se observa que en el siguiente experimento probará el mínimo que se está a la izquierda para determinar si el mínimo absoluto de la función objetivo se encuentra en esa posición o no.

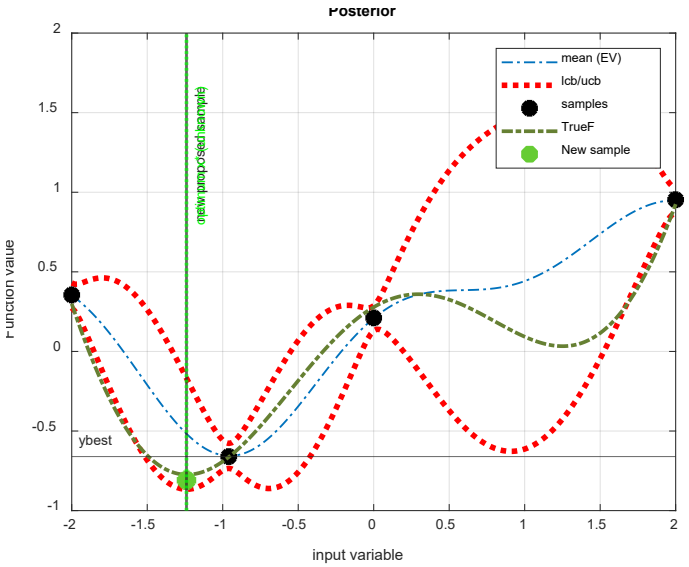


Figura 35: Ejemplo cuarta iteración de LCB en una función objetivo predefinida conocida. (Fuente: Dr. Antonio Sala-Piqueras [24]).

Al realizar la iteración, se visualiza cómo se actualiza la función objetivo y los límites. Se dispone a probar un nuevo experimento con el nuevo mínimo, aunque se puede percibir que ya se ha acercado al mínimo de la función verdadera.

- Pobability of improvement

La probabilidad de mejora es otra función de adquisición que elige los mejores puntos para maximizar mejor opción con mayor probabilidad de encontrar el mínimo de la función objetivo.

$$PI(x) = \Phi\left(\frac{y_{min} - \hat{\mu}(x)}{\sqrt{\hat{\sigma}(x)}}\right)$$

Donde Φ es la función normal estándar acumulativa, y_{min} es de los resultados obtenidos, es decir el valor más pequeño, cada vez que la función objetivo devuelva un valor todavía más pequeño este se actualiza, $\hat{\mu}(x)$ le media y $\hat{\sigma}(x)$ la varianza, estas dos últimas obtenidas del proceso gaussiano y la función de covarianza. A continuación, se muestra un ejemplo en el que también se presenta la misma función verdadera predefinida, y el PI máximo que se tiene que como se puede ver, va desde 0 a 1 como mucho. Donde es máximo este valor es donde hay mayor probabilidad de encontrar el mínimo de la función objetivo.

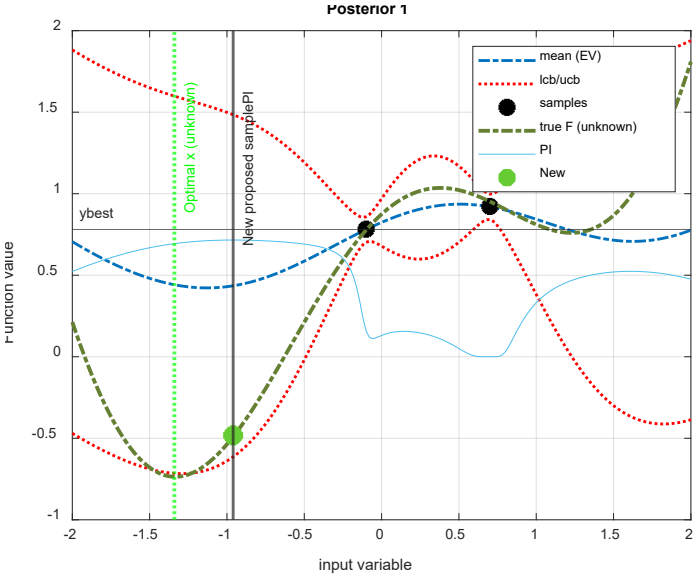


Figura 36: Ejemplo tercera iteración de PI en una función objetivo predefinida conocida. (Fuente: Dr. Antonio Sala-Piqueras [24]).

Si realizamos la siguiente iteración, se observa que las probabilidades de encontrar el mínimo han cambiado y que cada vez se va reduciendo esta probabilidad cuando más cerca se encuentra del mínimo absoluto.

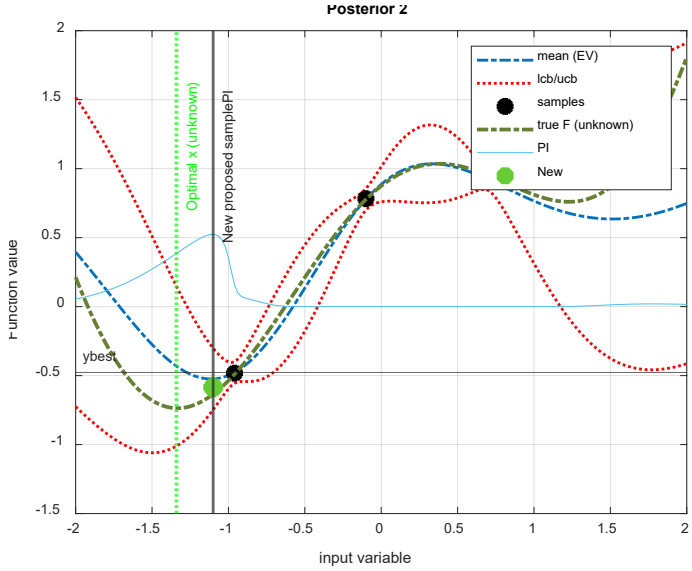


Figura 37: Ejemplo cuarta iteración de PI en una función objetivo predefinida conocida. (Fuente: Dr. Antonio Sala-Piqueras [24]).

El problema del uso de este tipo de función de adquisición es que cuando la probabilidad de mejora es muy baja, puede que intente explorar por otras zonas que a priori son relativamente erróneas y se puede perder tiempo en la realización de estos experimentos.

- Expected improvement

Esta función de adquisición calcula la cantidad de mejora que tiene la función objetivo, donde ignora los valores que causan un incremento en la función objetivo.

$$EI(x) = \Phi\big(y_{min} - \hat{\mu}(x)\big) + \big(y_{min} - \hat{\mu}(x)\big) \cdot N(y_{min} - \hat{\mu}(x))$$

Se emplea la función normal acumulativa al igual que en PI, pero además también la función de densidad de distribución normal. Del resultado se obtiene el máximo y los parámetros de entrada que correspondan a ese valor son los que se emplean en la siguiente iteración para proseguir con el experimento.

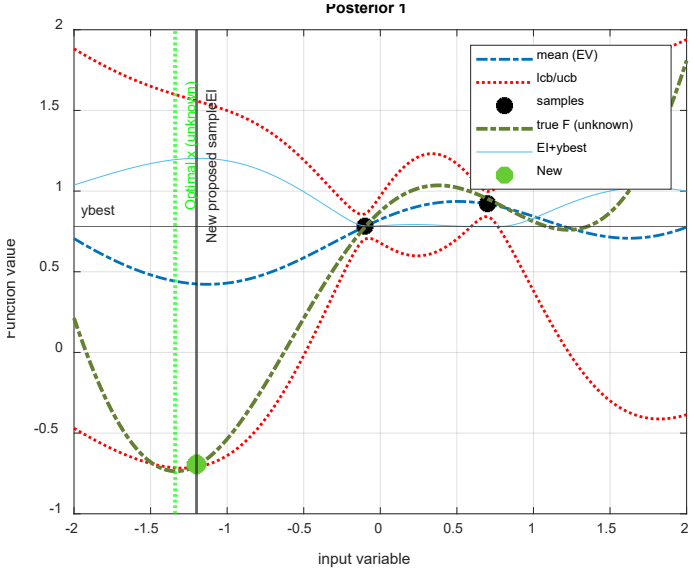


Figura 38: Ejemplo tercera iteración de EI en una función objetivo predefinida conocida. (Fuente: Dr. Antonio Sala-Piqueras [24]).

Se puede observar que el comportamiento es parecido al del PI, tal y como se puede ver en la gráfica, valor del EI es mayor cuanto más cerca se encuentre del mínimo.

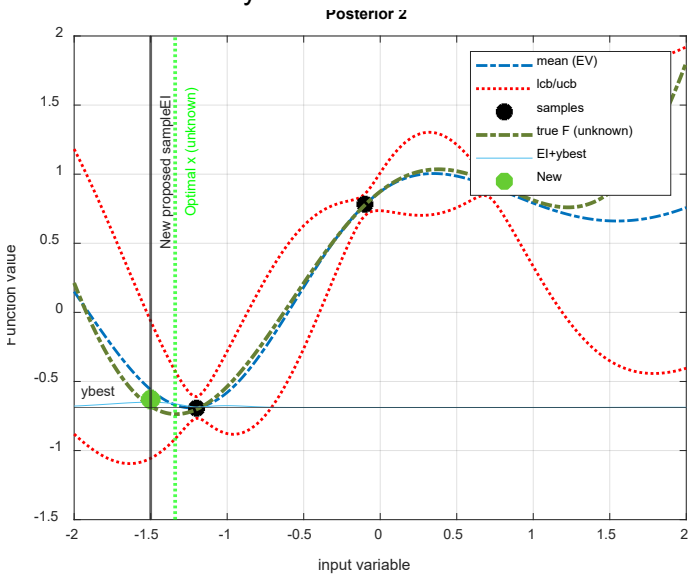


Figura 39: Ejemplo cuarta iteración de EI en una función objetivo predefinida conocida. (Fuente: Dr. Antonio Sala-Piqueras [24]).

La mejora esperada se ha reducido de forma significativa debido a que la anterior iteración es muy cercana al mínimo de la función verdadera, se irá acercando cada vez más según las próximas iteraciones.

Por último, en la función de MATLAB, *bayesopt* permite emplear una mejora que denomina *plus* en la que sólo actúa en el caso de la función de *Expected Improvement* (EI). Lo que se consigue es que si una zona está siendo sobrexplotada y detecta que es posible que se encuentra en un mínimo local y no en el absoluto de la función

objetivo. Para mayor detalle de su funcionamiento queda detallado en su documentación [27].

6.5 Funciones de MATLAB

Respecto al funcionamiento del programa, hay diversos scripts creados que componen el funcionamiento de los tres pilares fundamentales: robótica, visión y optimización.

- lanzamiento
  Como su propio nombre indica, esta función se encarga de comunicarse con el robot para enviarle los lanzamientos. En concreto controla dos posiciones, la primera vista es la previa a lanzar y la segunda es la del brazo completamente estirado, pasándole los ángulos deseados a la función de cinemática directa del robot, además, ejecuta otra función llamada *visionCapture.m.* Así pues, hace que por el propio terminal de MATLAB aparezca en qué ángulo y tiempo de trayectoria va a lanzar. Por otra parte, pausa el programa para que le introduzcas un 1 para realizar el lanzamiento y un 0 para volver a posicionarlo, y también una vez realizado el lanzamiento si se ha posicionado de forma errónea la pelota vuelve a preguntar si el lanzamiento ha sido correcto o no. Los datos que devuelve son los puntos de la trayectoria que ha leído la cámara.

Figura 40 y 41: Brazo replegado previo al lanzamiento y desplegado una vez a lanzado la pelota. (Fuente: Elaboración propia).

- visionCapture
  Como se ha mencionado, esta función devuelve los datos de la trayectoria en coordenadas X e Y. Se le ha establecido el rango de HSV para detectar el color de la pelota de frontón. El funcionamiento que sigue es el explicado en apartados anteriores, donde tras detectar con una máscara el color amarillo establece un contorno y se localiza su centroide. Con sus coordenadas en píxeles se pasa a las del mundo real, cada cincuenta milisegundos capturan un punto y guarda un total de cincuenta. Una vez recolectados termina su ejecución.

- RobotyVision
  Esta función emplea la de *lanzamiento.m* que a su vez contiene a *visionCapture.m.* Se utiliza para la realización de pruebas como mapeados, ensayos de repetibilidad y el correcto funcionamiento de las funciones, en esta no se realiza ningún tipo de optimización. Se calcula el valor de la función objetivo con ayuda de *calculaNuevaJ.m*.
- bayesoptfunction
  Por último, la de la optimización, contiene la función *bayesopt* a la cual se le pasa la función *lanzamiento2.m. Ésta* es idéntica a la de *lanzamiento.m* pero calculando con *calculaNuevaJ.m* el valor de la función objetivo, por lo cual devuelve directamente el valor de J y no el de las coordenadas de los puntos. Se le define previamente el rango de las variables a optimizar, en este caso el ángulo de la primera articulación y el tiempo de la trayectoria de velocidad, de --15 a 15 grados y de 650 a 750 ms.

7 Resultados

7.1 Validación de los resultados previos

Previamente a la realización de los experimentos, se requiere la validación de los resultados obtenidos en la simulación. Cuyo autor es Hannes Deruytter, quién ha realizado su tesina sobre la simulación previa del experimento que se realiza en el presente proyecto. Se agrega un resumen de su trabajo adjuntado en el documento de anexos. La realidad difiere de la simulación, los principales problemas que se encuentran son dos. El primero de ellos, es el hecho de que muchos de los disparos que se realizan en el mallado llegan a pasar muy cerca del hoyo objetivo, pero no terminan de quedarse dentro. Esto se debe a que no hay suficiente profundidad, por ende, la pelota acaba saliendo. El segundo problema viene dado por la propia función de coste (J) empleada en la simulación, en la que se penaliza en exceso el punto final de la trayectoria y por consiguiente no se tiene en cuenta si el disparo ha sido cercano al objetivo, puntuando de forma muy negativa y dificultando el uso de la optimización bayesiana.

Para ser conocedores de estos dos problemas a los que nos estamos enfrentando, se realiza un mapeado. Para ello, se establece el rango del tiempo que tarda el robot en lanzar la pelota entre 650 a 750 ms, siendo el primero de ellos el más rápido y el del ángulo de la primera articulación del robot que va de -15 a 15 grados. Una vez analizaremos las diferentes trayectorias.

Se procede a mostrar los disparos que han acertado y los que se han fallado en la siguiente figura:

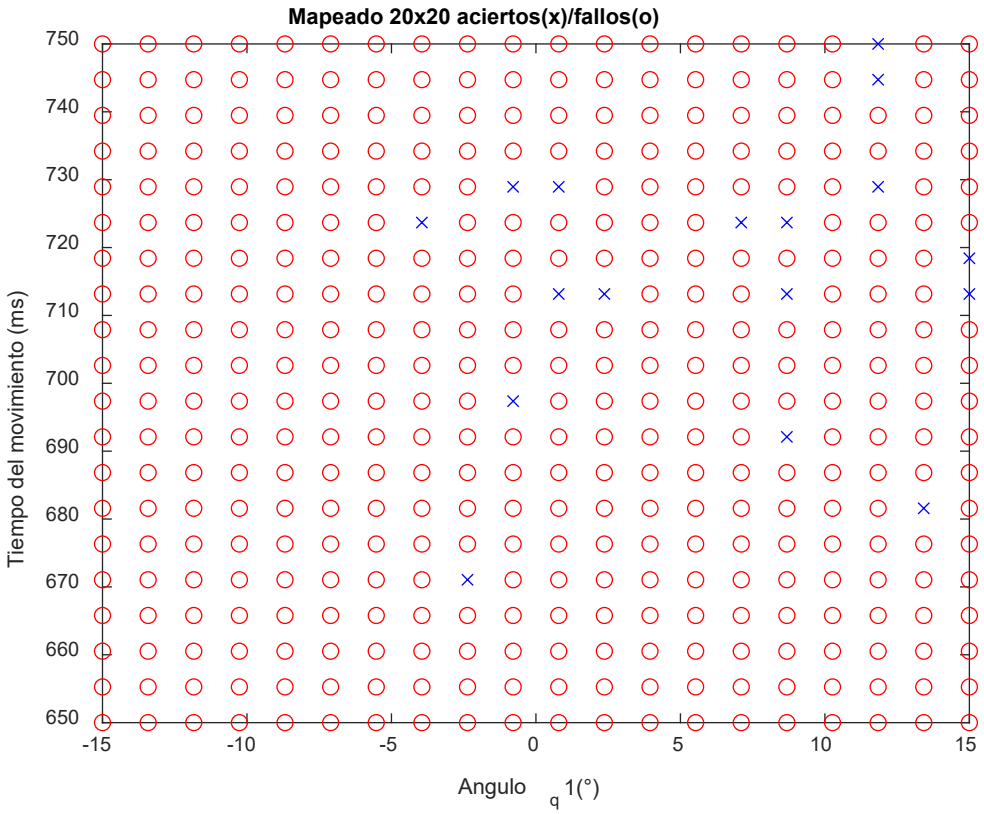


Figura 42: Mapeado de aciertos y fallos. (Fuente: Elaboración propia).

Como vemos, se han cogido los valores que como máximo tienen una tolerancia de 5 cm del agujero, para comprobar la trayectoria que realiza la pelota y que de verdad estos puntos corresponden a un acierto:

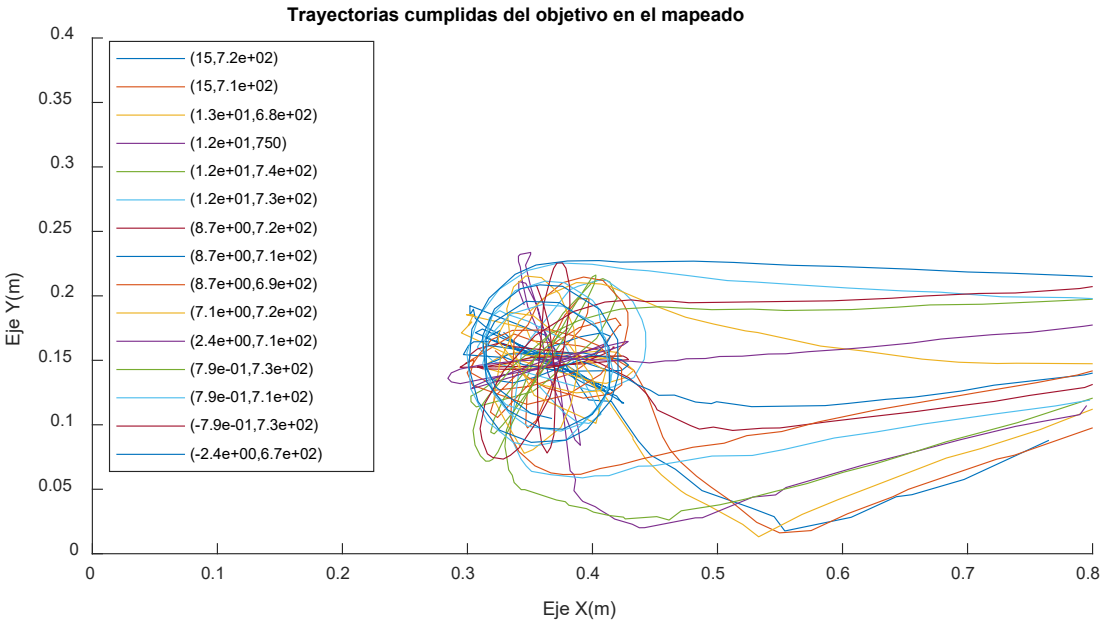


Figura 43: Trayectorias de los aciertos. (Fuente: Elaboración propia).

Se puede observar cómo la pelota puede entrar de diferentes formas, entre ellas, desde ángulos cercanos al cero en el que empieza a oscilar. También puede ser mediante un disparo directo sin oscilación, otro tiro con oscilaciones desde ángulos cercanos a nueve o diez grados, por último, mediante rebote con la pared.

Ahora bien, si observamos los tiros que han pasado muy cerca del objetivo, es decir a menos de 9 cm del hoyo objetivo (radio del agujero), muchos de estos han fallado. Por lo que se les daría una puntuación elevada con la función de coste de la simulación, debido a que se penaliza mucho el no acertar.

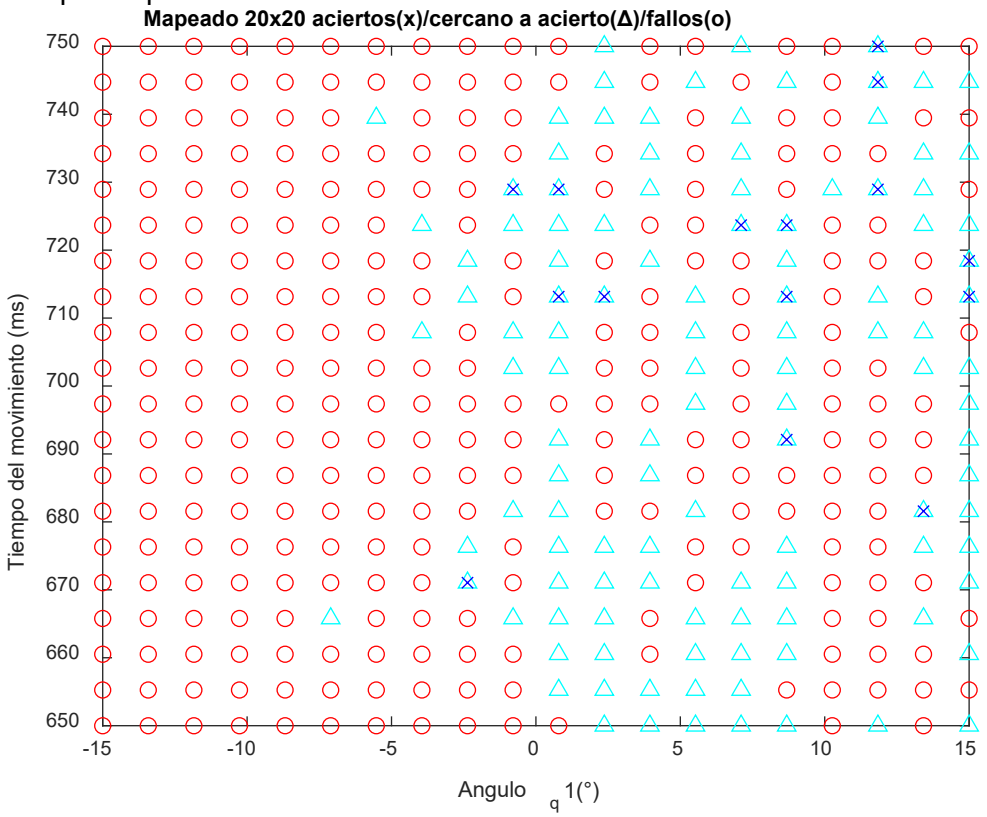


Figura 44: Mapeado de aciertos, cercano a aciertos y fallos. (Fuente: Elaboración propia).

Se puede apreciar que muchos de estos puntos están cerca de llegar a lograr el objetivo. Si se representan sus trayectorias se puede comprobar de forma más clara:

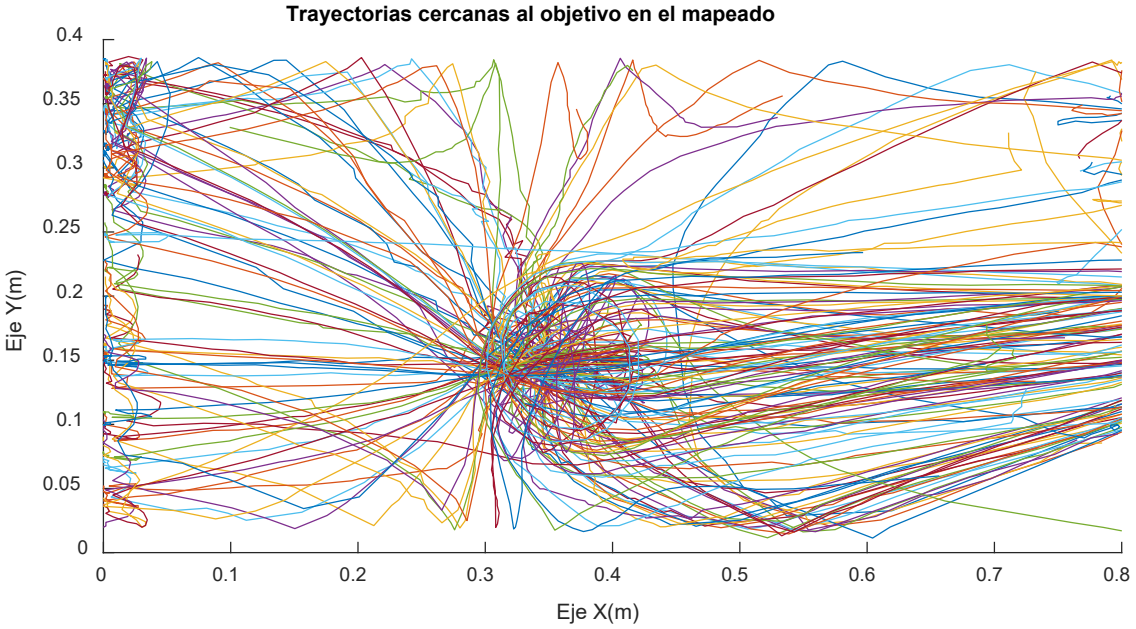


Figura 45: Trayectorias de los casi aciertos. (Fuente: Elaboración propia).

El mapeado, se ha realizado de forma correcta, pero el modelo de la función no es la esperada, por tanto, se procede a modificar la función de coste que se tenía planteada

en la simulación, dado que es cierto que la posición final es lo más importante, no se debería castigar tanto si el disparo ha sido cercano a la solución. Es decir, el problema reside en la parte de la función de coste donde se tienen en cuenta todas las distancias de la trayectoria con respecto al punto final. Para que resida tanto peso en esto se procede a tener muy en cuenta **la distancia mínima al hoyo** y se le añade otro elemento en el que se calcula la longitud de la trayectoria normalizada, es decir todo son distancias al igual que la anterior, además, se prescinde del elemento cuadrático porque hace que los intervalos sean mayores cuando hay una diferencia notoria y así se consigue suavizar la función. También, para disminuir el efecto de oscilación cuando la pelota está entrando dentro del hoyo los últimos cinco puntos se realiza una media.

$$J_{Nueva} = \alpha \cdot \overline{dist}_{fin} + \beta \cdot \min\left(dist_{tray}\right) + \Omega \cdot N(long_{tray})$$

Donde α recibe un valor de quince y es la variable que asigna un peso al valor de la media $\overline{dist}_{fin}$la cual es donde la pelota se ha parado, mientras que β tiene un valor de cincuenta y le da el peso a la distancia mínima de la trayectoria $dist_{tray}$, por último, Ω recibe un valor de cinco y le da el peso a la longitud de la media normalizada, donde la longitud máxima y mínima se ha obtenido de los puntos del propio mallado. Nótese que la nueva función de coste emplea distancias y longitudes. Todas las unidades son en metros, pero al tener diferente peso unas de otras y las medidas son diferentes, se considera a la función adimensional.

A continuación, se observa el mapeado visto desde la parte superior empleando tanto la función objetivo de la simulación como la presentada en el presente trabajo. Se detalla por diferentes colores las zonas, en color amarillo se reflejan las zonas con una mayor puntuación, mientras que las azules oscuras indican una puntuación baja. Para el empleo de la optimización bayesiana se requiere que haya una abundancia de color azul, indicando que se está cerca del hoyo e incluso que es un acierto. En la primera de las figuras, se ve de forma clara que no hay una zona en concreta donde se haya la solución, se encuentran picos en zonas donde la pelota ha pasado cerca y no refleja la realidad, por lo que se obtendría una mala optimización con mucho error. No obstante, en la segunda si se observa ya una diferencia entre los ángulos de -15º a 0º (zona de no acierto) y la de 0º a 15º (zona de acierto). Generando un valle alrededor de los 3º grados indicando que hay una gran probabilidad de acertar a en esos ángulos y otras zonas de disparo indirecto que se reflejan como rebote con la pared. A continuación, se muestran ambas figuras para que la diferencia pueda ser apreciada:

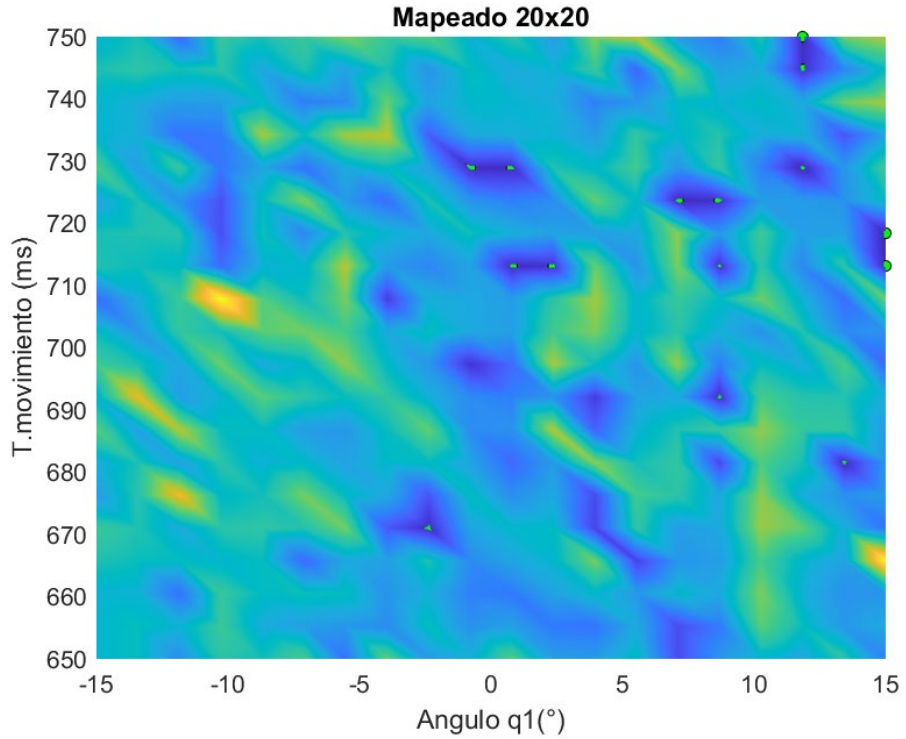


Figura 46: Mapeado de 20x20 vista desde arriba con la función de coste de la simulación. (Fuente: Elaboración propia).

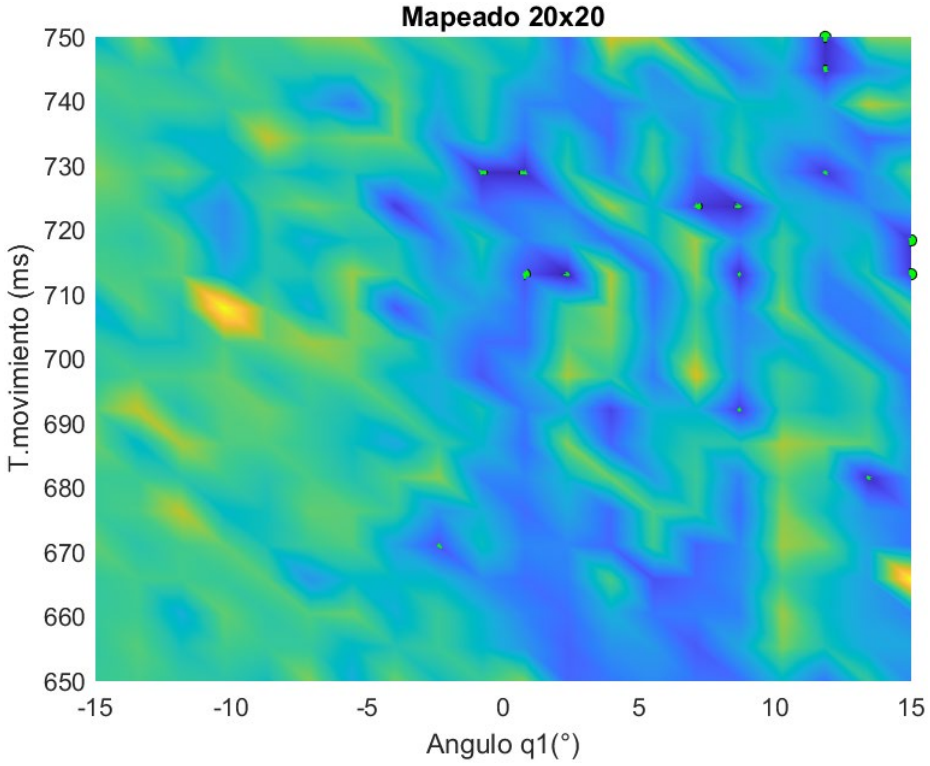


Figura 47: Mapeado de 20x20 vista desde arriba con la función de coste nueva. (Fuente: Elaboración propia).

Para que la comparación quede más detallada, se representan las mismas figuras en diferente perspectiva:

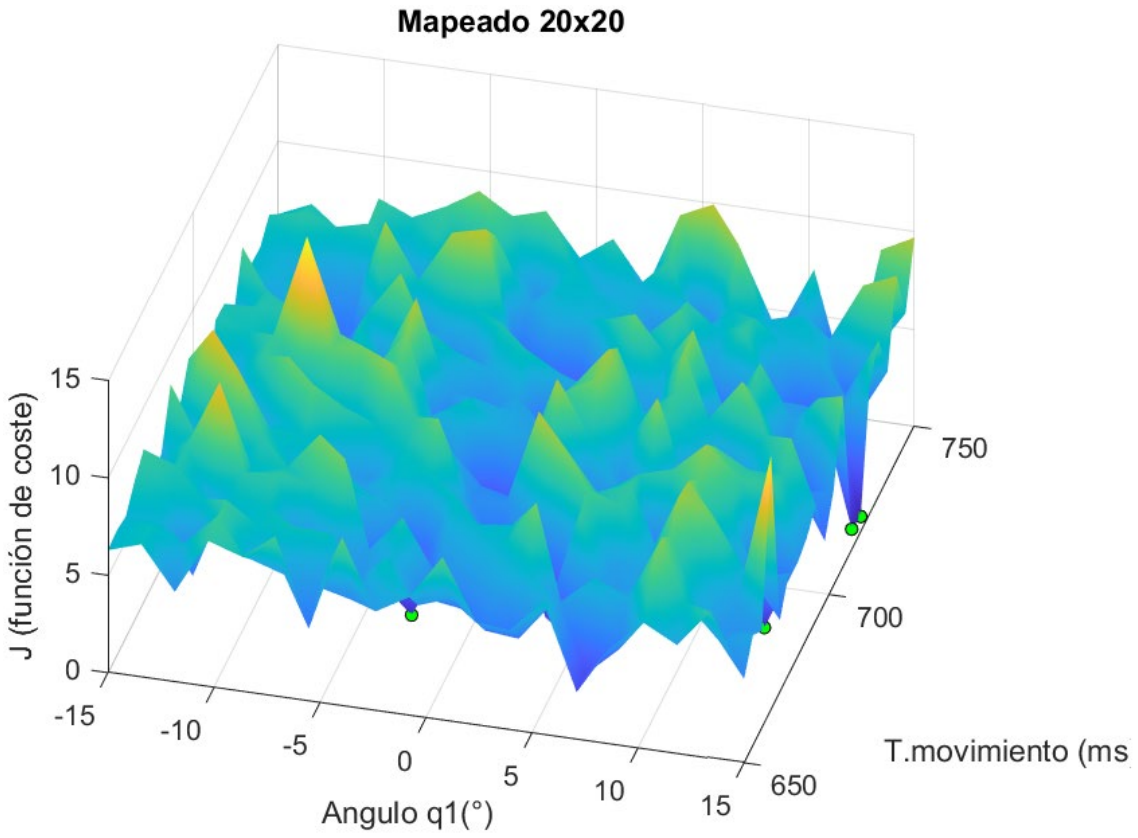


Figura 48: Mapeado de 20x20 con la función de coste de la simulación. (Fuente: Elaboración propia).

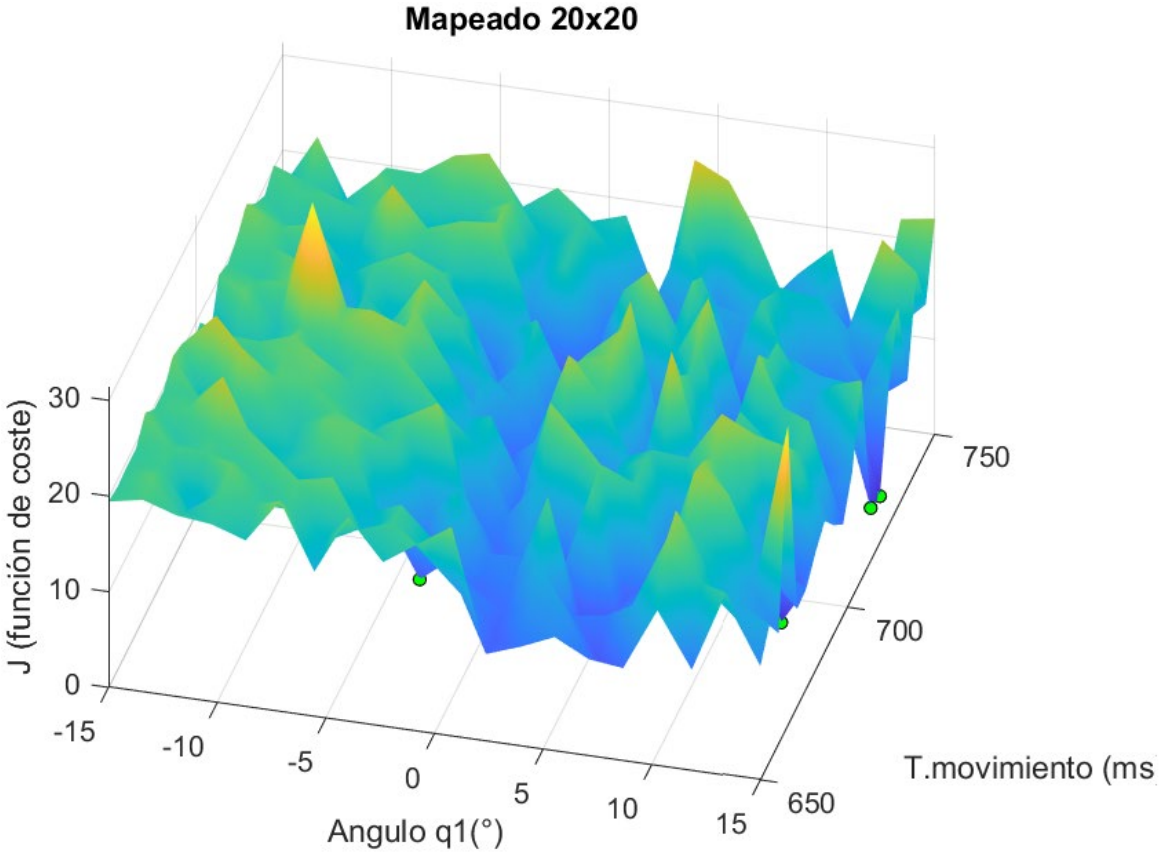


Figura 49: Mapeado de 20x20, con la función de coste nueva. (Fuente: Elaboración propia).

Cabe destacar que la finalidad del mallado es para hacerse una idea del problema al que nos enfrentamos, en un proceso que se deba optimizar no se realiza este tipo de mapeados, el objetivo del proyecto es lograr una solución con muchas menos iteraciones y corroborando el potencial de la optimización bayesiana frente a la fuerza bruta.

Otro problema observado era que para un mismo ángulo y tiempo de trayectoria se observa que tan solo había un 60% de repetibilidad cuando se intenta encestar en el hoyo, por lo que se acentúa más esa discontinuidad cerca de la solución. Para solucionarlo se aumenta ligeramente la profundidad del hoyo. Se puede observar en la siguiente tabla, dos ensayos de repetibilidad, el primero es previo al realizar el hoyo más hondo. Los resultados en **negrita** son cuando la pelota ha acertado, para un ángulo de 0,98 y 730 ms de tiempo de trayectoria:

Disparo	J ensayo repetibilidad 1	J ensayo repetibilidad 2
1	9.19	**6.03**
2	**5.41**	**5.23**
3	16.14	**4.13**
4	**4.58**	14.28
5	10.70	**3.64**
6	11.22	**3.56**
7	**4.66**	10.48
8	**5.49**	**5.20**
9	**4.87**	**7.23**
10	**3.72**	**2.65**

Tabla 3: Ensayos de repetibilidad para ángulo de 0.98 º y 730 ms de tiempo de trayectoria.

Se aprecia que aumenta de un 60% a un 80%, consiguiendo una media de 6.24 de resultado respecto al 7.60 del ensayo anterior sin lijar el hoyo.

Una vez ajustado el entorno se continua con la realización de los experimentos de optimización bayesiana.

7.2 Experimentos de optimización bayesiana

7.2.1 Relación entre exploración y explotación

A continuación, se determinan una serie de experimentos en los que se varía el ratio de exploración, para averiguar si es mejor explorar en busca de mejores soluciones o explotar más una zona hasta ir mejorando los experimentos. Por este motivo se usan tres ratios diferentes los cuales son 0.1,0.3 y 0.5, en los que se emplean 30 iteraciones para cada uno de ellos.

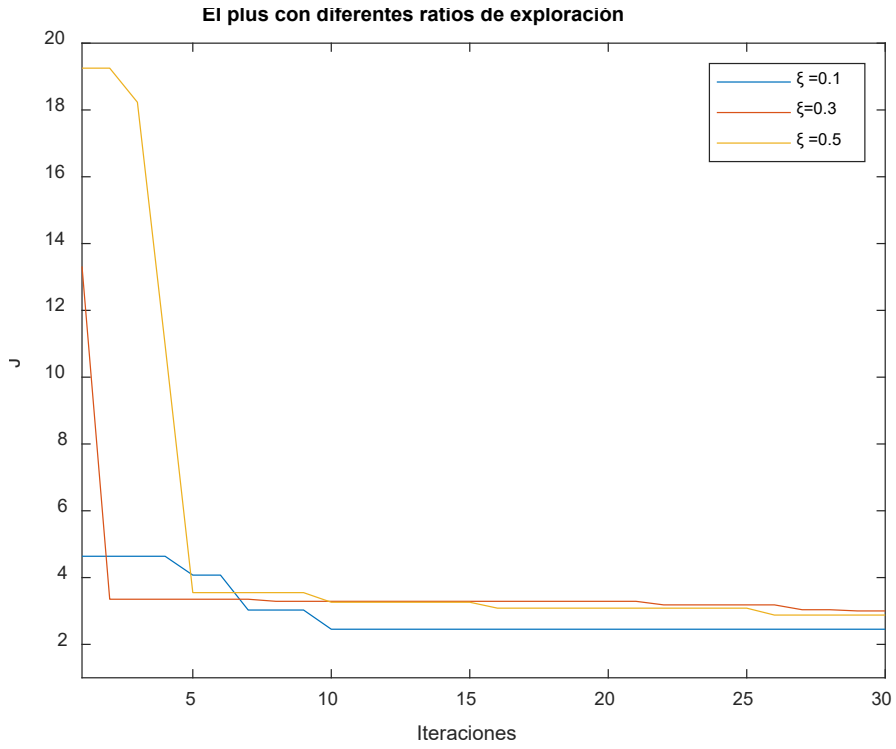


Figura 50: Comparación de los resultados obtenidos con diferentes ratios de exploración, empleando la función de adquisición Elplus. (Fuente: Elaboración propia).

A simple vista se pueden apreciar diferentes aspectos interesantes, como por ejemplo que para un ratio de exploración de 0.1, desde el primer experimento consigue un valor muy pequeño debido a que el algoritmo ha decidido probar por esa zona y que en muy pocas iteraciones consigue explotar esa zona y conseguir una puntuación mínima de 2.46, resultado calificado como muy apto. Cabe destacar que con la función objetivo que se emplea con respecto a la simulación. Por otro lado, los otros dos experimentos van más a la par en cuanto a mejora, e incluso acaban obteniendo mejores resultados el de 0.5, donde con 16 iteraciones consigue un experimento menor de 3.2 mientras que el de 0.3 tarda 22. En la siguiente tabla resumen se puede observar los ángulos y tiempo de trayectoria empleados hasta alcanzar un valor menor de 3.2.

ξ	Iteraciones hasta J menor de 3.2	Ángulo (º)	t.velocidad (ms)	Solución J
0.1	7	1.143	722.038	3.028
0.3	22	2.049	724	3.182
0.5	16	3.048	748.75	3.08

Tabla 4: Resumen de experimento con diferentes ratios de exploración.

Para que se pueda apreciar con mayor detenimiento como varían los intentos según su ratio de exploración, se representa en la siguiente *figura*. A mayor ratio, los intervalos de los parámetros son más dispares, se observa en el de 0.3 y aún más en el de 0.5, mientras que a menor ratio cuando en encuentra el mínimo los cambios son mucho menores, aunque es cierto que en las primeras seis iteraciones se centra en explorar descartar otras soluciones.

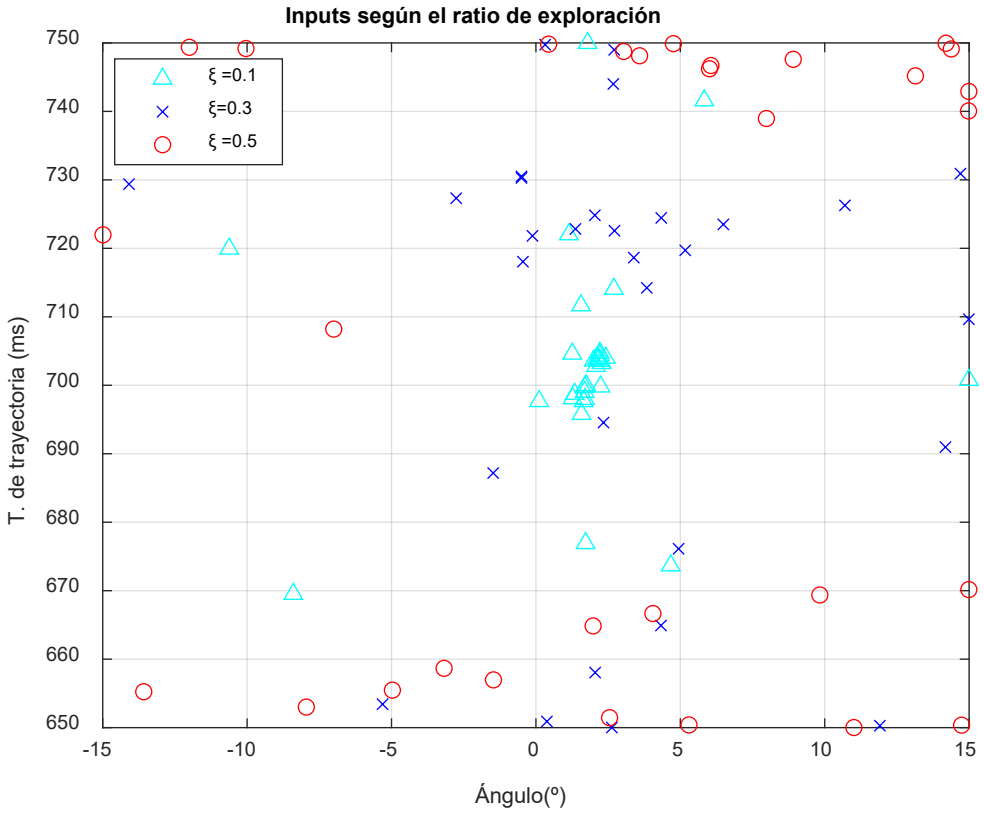


Figura 51: Inputs según el ratio de exploración. (Fuente: Elaboración propia).

Por tanto, al igual que en la simulación, el mejor ratio de exploración es el de 0.1. Si tenemos en cuenta que para hacer el mapeado, hay que realizar 400 experimentos y con tan sólo 10 disparos empleando la optimización ya se consigue una solución muy buena. Por lo cual, en la realización de los experimentos que se estudian a continuación se realizarán únicamente 15 iteraciones.

A continuación, se realizan los experimentos para cada una de las funciones de adquisición planteadas anteriormente y las posibles combinaciones de las soluciones que se consiguen alcanzar.

7.2.2 Elplus

Como se ha mencionado previamente, esta función de adquisición actúa de forma similar al *Expected Improvement* pero con el añadido de la detección de un punto sobrexplotado, está directamente relacionado con el ratio de exploración, tiene una condición que si se cumple detectaría que se encuentra en un mínimo local de la función objetivo y pasaría a buscar en otra zona. El modelo de la función obtenida se representa a continuación, se aprecia un valle como en el mapeado realizado previamente por donde se encuentra la solución.

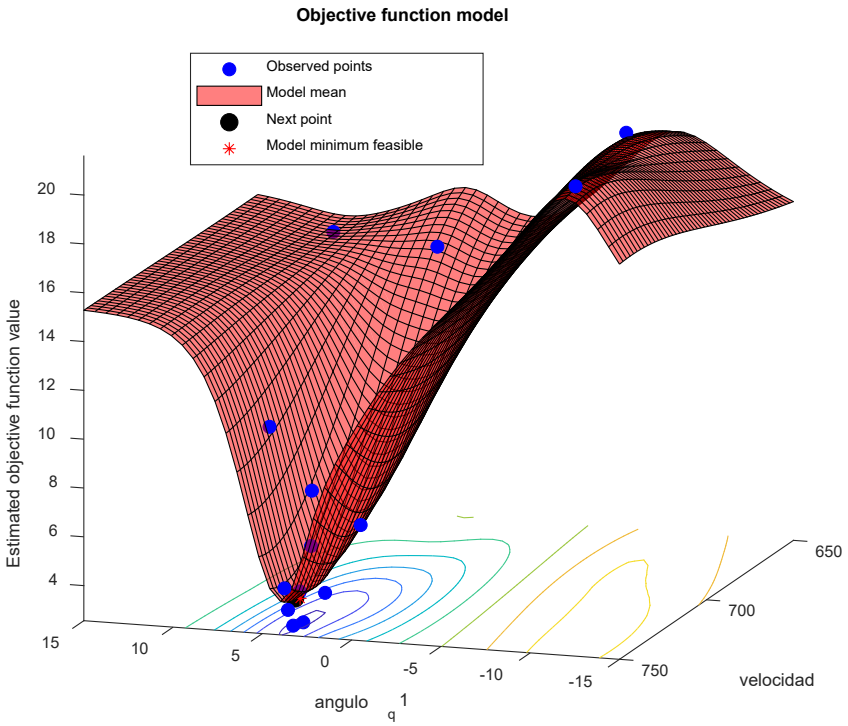


Figura 52: Modelo de la función objetivo empleando Elplus. (Fuente: Elaboración propia).

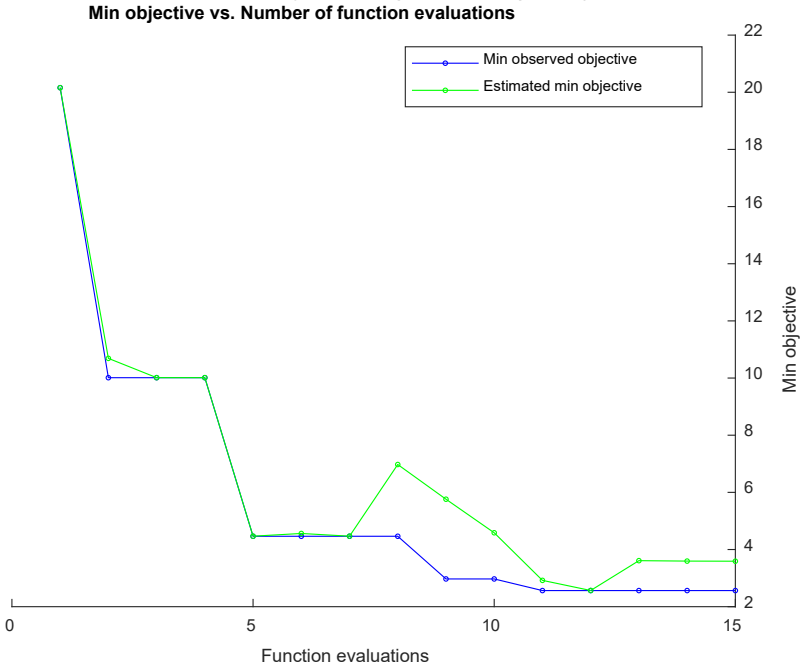


Figura 53: Comparativa mínimo observado y mínimo estimado con Elplus. (Fuente: Elaboración propia).

En cuanto a la comparativa del mínimo observado y el estimado, se puede observar que ambos van a la par. Por consiguiente, la optimización bayesiana consigue predecir qué valor aproximado se logra en la función de objetivo definida. Así pues, con esta configuración la solución abordada es más que correcta.

En la siguiente tabla se muestra cada una de las configuraciones que el robot ha ido tomando y la puntuación obtenida en cada uno de ellos. Está resaltada la iteración 11 en la que la puntuación es de 2.56 y la configuración del ángulo y de la trayectoria de velocidad son de 3.75º y 739.20 ms.

Disparo	Ángulo (º)	T. tray (ms)	J
1	-8.15	675.87	20.15
2	6.48	730.65	10.01
3	9.73	660.56	14.40
4	-12.15	746.02	21.59
5	3.79	749.68	4.46
6	1.37	730.69	6.25
7	3.10	668.81	14.55
8	2.23	749.93	8.56
9	3.28	749.85	2.97
10	3.57	749.93	3.60
11	**3.75**	**739.20**	**2.56**
12	3.44	729.90	3.33
13	3.22	740.20	5.76
14	4.07	738.17	3.75
15	4.04	738.32	3.76

Tabla 5: Configuraciones y resultado de cada uno de los disparos para EIplus. (Fuente: Elaboración propia).

7.2.3 EI

El resultado obtenido en éste es similar al de la versión plus, puesto que sólo tienen la diferencia mencionada previamente, el modelo es muy parecido, centrado en otro ángulo y velocidad el cuál ha ido explotando y obteniendo mejores resultados.

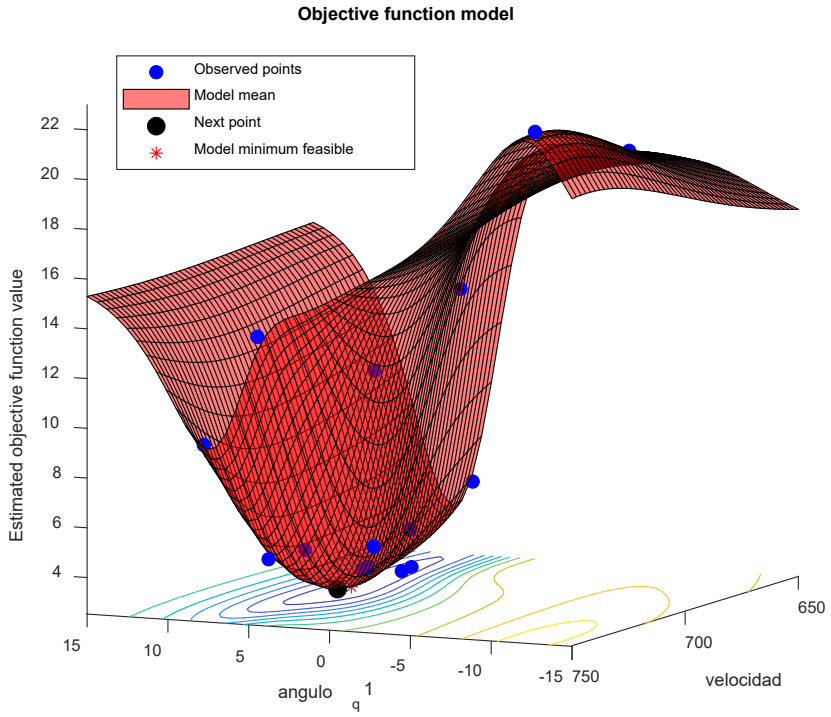


Figura 54: Modelo de la función objetivo empleando EI. (Fuente: Elaboración propia).

La estimación también es correcta tal y como se observa en la gráfica de abajo y va mejorando con cada disparo efectuado hasta incluso la iteración 13.

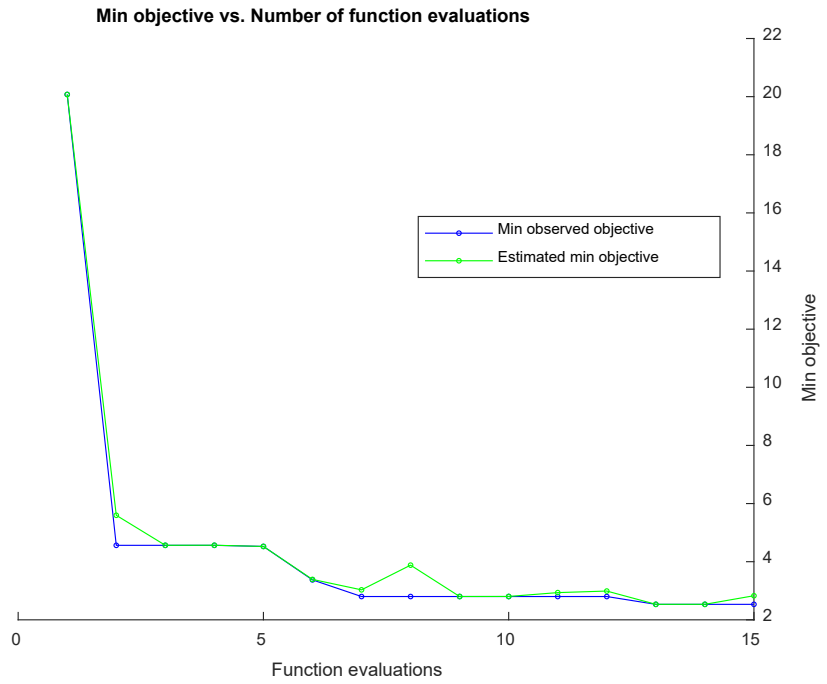


Figura 55: Comparativa mínimo observado y mínimo estimado con EI. (Fuente: Elaboración propia).

El mejor de los resultado se ha obtenido para un ángulo de 6.17º y 669.63 ms de tiempo para la trayectoria del OpenMANIPULATOR-X.

Disparo	Ángulo (º)	T. trayectoria (ms)	J observado
1	-8.15	675.87	20.07
2	6.48	730.65	4.56
3	9.73	660.56	10.01
4	-12.15	746.02	22.98
5	3.36	706.48	4.52
6	4.92	699.96	3.37
7	4.91	682.98	2.80
8	5.14	650.02	5.46
9	4.47	749.97	14.14
10	7.50	707.238	4.22
11	2.62	673.43	13.97
12	6.15	688.87	3.07
13	**6.17**	**669.63**	**2.53**
14	8.96	741.55	9.35
15	5.60	674.37	4.21

Tabla 6: Configuraciones y resultado de cada uno de los disparos para EI. (Fuente: Elaboración propia).

7.2.4 PI

Lo siguiente a analizar, es la probabilidad de mejora. Algunos de los problemas que puede albergar esta metodología es la exploración debida a un punto sobrexplotado, es decir, si analiza un punto pongamos que se trata del mínimo absoluto de nuestra función objetivo y posteriormente busca una configuración donde claramente no se encuentra este mínimo se debe particularmente a que las probabilidades de mejora ya son tan bajas que la mejor opción es buscar en otro sitio, a pesar del ratio de

exploración bajo. Sin embargo, como se muestra a continuación, para 15 iteraciones esto no ocurre y se plantea un buen modelo de la función de adquisición.

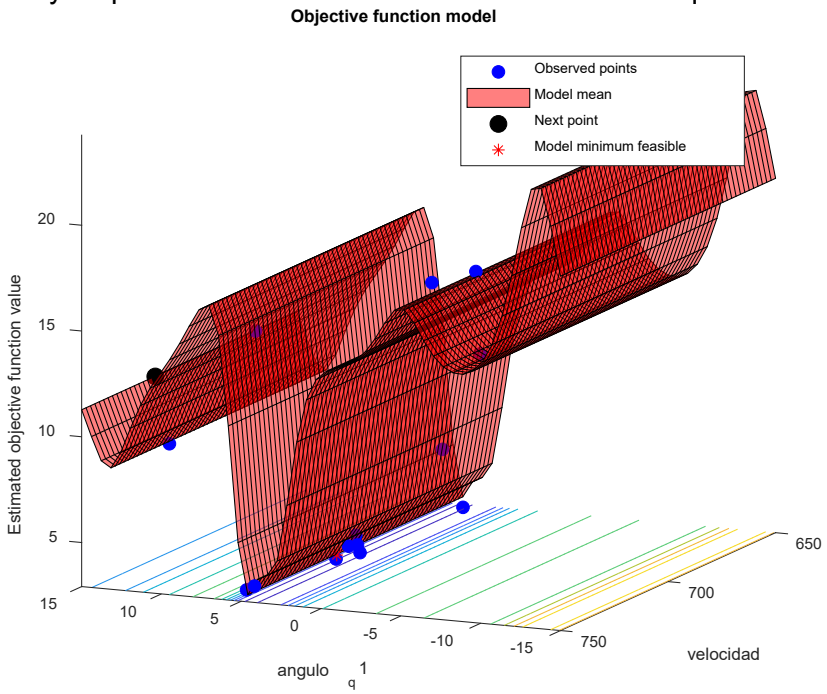


Figura 56: Modelo de la función objetivo empleando PI. (Fuente: Elaboración propia).

El modelo obtenido muestra que la solución que ha determinado se encuentra alrededor de los 5º, a velocidades bajas y medias sobre todo, en cuanto al modelo de estimación es bastante acertado en junto con el mínimo obtenido, a la séptima iteración ya no consigue mejorar más la trayectoria, obteniendo una puntuación de 2.91 para un ángulo de 4.19º y 652.85 ms de tiempo de trayectoria.

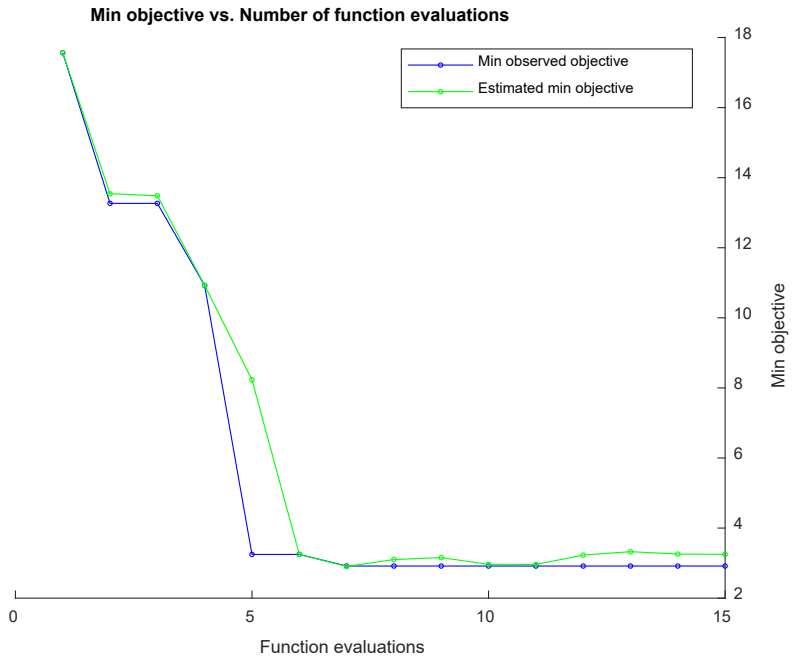


Figura 57: Comparativa mínimo observado y mínimo estimado con PI. (Fuente: Elaboración propia).

En la tabla se puede apreciar la diferencia que hay entre esta función de adquisición y las dos tablas anteriores previamente comentadas, en las que los parámetros eran

más similares de una iteración a otra mientras que estos tienen mayores diferencias entre ellos.

Disparo	Ángulo (º)	T. trayectoria (ms)	J observado
1	-4.95	714.78	17.56
2	10.13	705.29	13.26
3	-13.49	682.40	24.25
4	1.49	665.05	10.92
5	3.56	705.38	3.24
6	4.93	748.01	3.36
7	**4.19**	**652.85**	**2.91**
8	4.14	706.45	3.56
9	6.04	653.60	13.44
10	4.58	709.02	3.04
11	3.09	710.11	3.86
12	4.58	699.81	3.73
13	3.74	665.76	6.28
14	4.20	749.75	3.67
15	13.32	721.62	8.47

Tabla 7: Configuraciones y resultado de cada uno de los disparos para PI. (Fuente: Elaboración propia).

7.2.5 LCB

Por último, pero no por ello menos importante, se encuentra el *lower confidence bound* al igual que en el caso anterior, éste tiende también a explorar más que con el EI, debido a que en cada momento a que el mínimo del 95% de incertidumbre de nuestra función objetivo difiere. Al igual que en resto de experimentos el modelo de la función objetivo muestra una similitud con el valle obtenido en el mapeado. En este caso se encuentra en torno a los 5º al igual que con el *probability of improvement*.

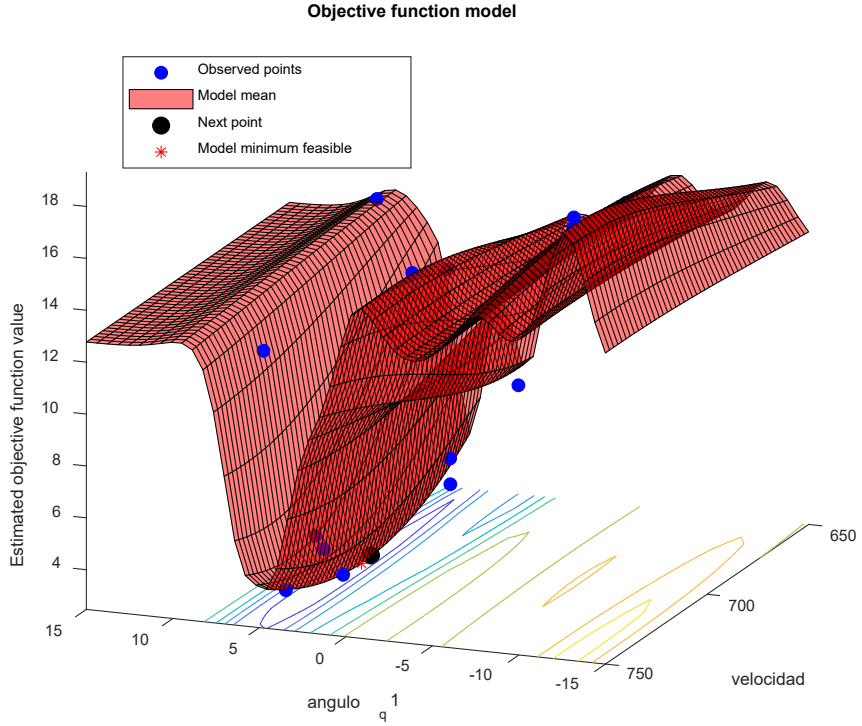


Figura 58: Modelo de la función objetivo empleando LCB. (Fuente: Elaboración propia).

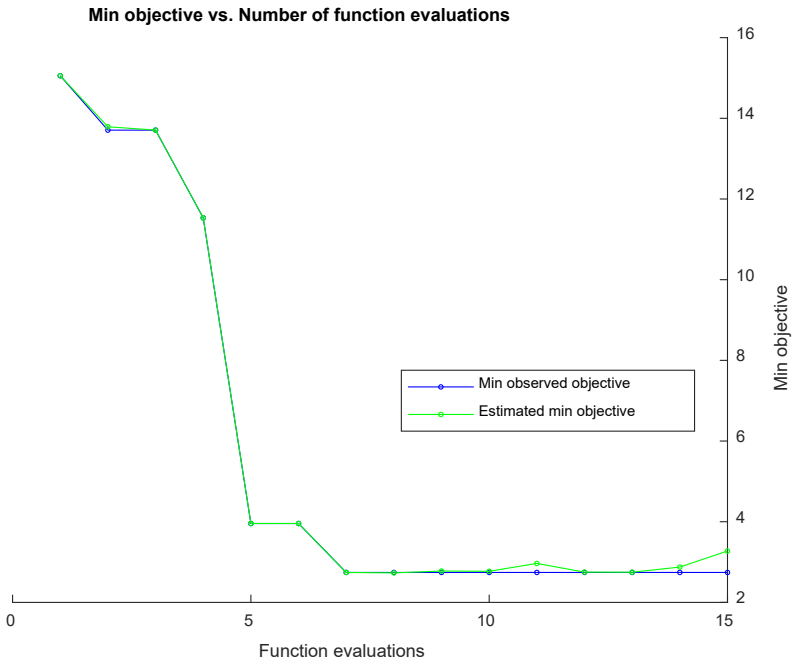


Figura 59: Comparativa mínimo observado y mínimo estimado con LCB. (Fuente: Elaboración propia).

La mejor iteración es la séptima, con un ángulo prácticamente idéntico al de *Probability of Improvement,* la única diferencia se encuentra en la velocidad donde aquí se produce un poco más lento, pero con mejor puntuación de la función objetivo. Esto es debido probablemente a cómo entra la pelota dentro del hoyo, debido a que si va a mayor velocidad hace el intento de salir de éste, mientras que a menor cae directamente dentro por lo que se capturan más puntos en la localización correcta, en consecuencia, menor es la puntuación.

La tabla con las 15 iteraciones de LCB, se muestra a continuación:

Disparo	Ángulo (º)	T. trayectoria (ms)	J observado
1	-6.20	664.89	15.05
2	9.11	657.22	13.70
3	-12.70	746.01	19.30
4	7.83	723.86	11.53
5	6.07	713.33	3.95
6	-7.64	703.21	16.26
7	**4.14**	**716.09**	**2.74**
8	1.30	654.02	6.90
9	4.71	658.44	3.08
10	4.77	738.90	3.32
11	3.02	657.65	9.75
12	5.20	716.50	3.67
13	-0.35	720.31	14.91
14	4.37	661.46	4.27
15	5.83	650.21	10.83

Tabla 8: Configuraciones y resultado de cada uno de los disparos para LCB. (Fuente: Elaboración propia).

7.2.6 Comparativa de las distintas funciones de adquisición

Si comparamos el mínimo observado de cada una de las funciones, se puede apreciar que todas ellas tienen un comportamiento muy similar donde al principio empiezan con un resultado de la función de coste bastante elevado, a partir de la segunda o tercera iteración se reduce su valor de forma drástica. Debido a que entre los 0 y 15 grados es más probable acercarse al hoyo. Todos los experimentos acaban con un valor menor de 3 de puntuación en menos de 15 iteraciones. Las puntuaciones más bajas que son muy similares son los de EI y EIplus. En cuanto a PI y LCB, van también bastante ligados como se puede apreciar a continuación:

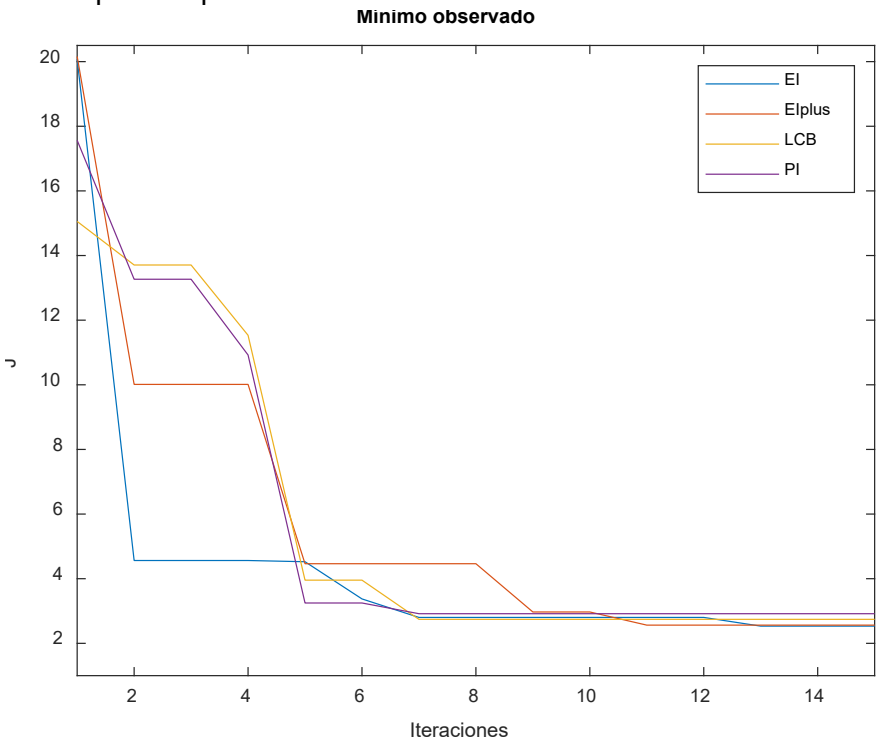


Figura 60: Comparativa de los resultados de los experimentos. (Fuente: Elaboración propia).

En la siguiente tabla resumen, como se comenta cada función de adquisición ha encontrado un resultado óptimo, en el rango que cabía esperar y con resultados muy similares. En definitiva, las soluciones al ser tan similares, se podría decir que con cualquiera de ellas se resolvería, aunque el de *Expected Improvement* tiene una tendencia a lograrlo rápidamente y con la mejor trayectoria.

Func. Adquisición	Mejor Iteración J	Mejor Ángulo (º)	Mejor t.velocidad (ms)	Mejor J
EI+	11	**3.75**	**739.20**	**2.56**
EI	13	**6.17**	**669.63**	**2.53**
PI	11	12.61	705.31	2.74
LCB	10	12.07	723.47	2.76

Tabla 9: Resumen de las soluciones obtenidas. (Fuente: Elaboración propia).

7.2.7 Resultados con obstáculo

Como bien se han mostrado en las pruebas anteriores, los disparos han sido todos muy similares y se ha acertado con pocos intentos. Para ir un paso más allá se le añade un obstáculo, evitando así el tiro directo y que el robot busque el disparo con rebote en la pared. En la siguiente *figura* se muestra el escenario con el obstáculo dispuesto, se trata de un listón pequeño de madera sobrante de la construcción de la estructura, pegado con cinta adhesiva a la plataforma.

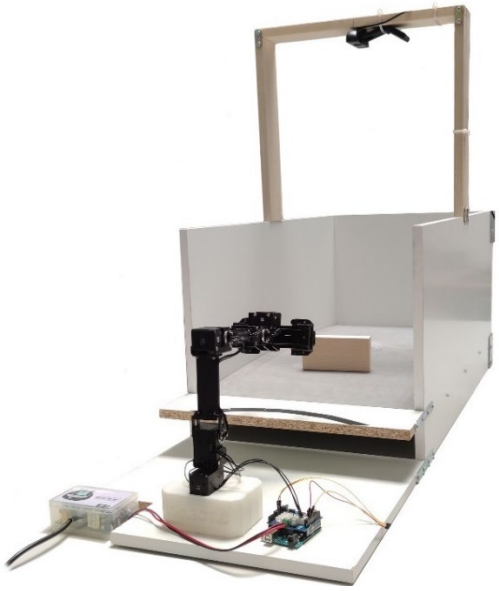


Figura 61: Estructura y plataforma con obstáculo incorporado. (Fuente: Elaboración propia).

Si comprobamos el mínimo obtenido de todas las funciones de adquisición anteriores, en una sola gráfica. Se aprecia claramente la dificultad añadida para acertar.

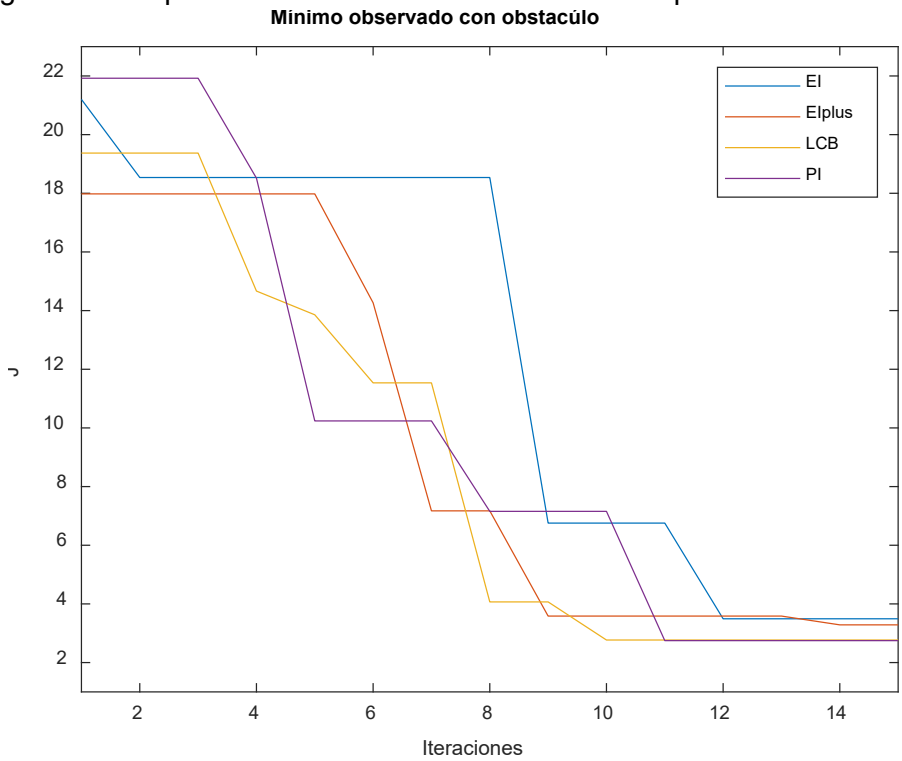


Figura 62: Comparativa del mínimo observado con las diferentes funciones de adquisición con un obstáculo. (Fuente: Elaboración propia).

A simple vista, se puede ver que en este caso los que mejor puntuación obtienen son *probability of improvement* y *lower confidence bound*, ambos con un resultado muy similar, en el que el primero de ellos obtiene una puntuación de 2.74 mientras que el segundo de 2.76. El ángulo como cabía esperar para todos los casos se encuentran a 13º y el tiempo del perfil de trayectoria de la velocidad es de 700 ms.

Func. Adquisición	Mejor Iteración J	Mejor Ángulo (º)	Mejor t.velocidad (ms)	Mejor J
EI+	14	13.13	692.97	3.28
EI	12	13.79	671.67	3.49
PI	11	12.61	705.31	2.74
LCB	10	12.07	723.47	2.76

Tabla 10: Resumen de las soluciones obtenidas con obstáculo.

El modelo para el mejor de los resultados se representa a continuación, se observa perfectamente como el valle donde se encuentra la solución ya no se encuentra sobre los 5º como en los experimentos sin obstáculo, el modelo es adecuado y el robot mediante aprendizaje supervisado es capaz de encontrar la solución mediante el rebote con la pared.

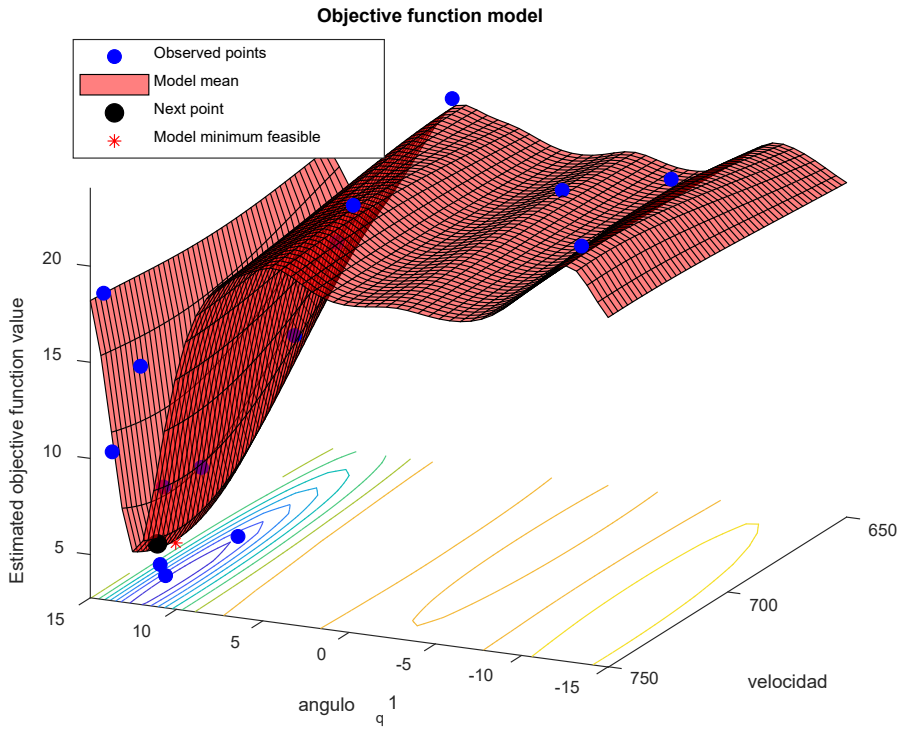


Figura 63: Modelo de la función objetivo para PI con obstáculo. (Fuente: Elaboración propia).

8 Objetivos de desarrollo sostenible

La agenda 2030 tiene unas metas denominadas Objetivos de Desarrollo Sostenible (ODS), los cuales abarcan diferentes ámbitos en los que se pretende que todas las personas tengan las mismas oportunidades, además, que el mundo vaya progresivamente responsabilizándose con los aspectos más sostenibles del sector, es decir, se intenta frenar el calentamiento global.

Figura 64: Los 17 objetivos de desarrollo sostenible. (Fuente: Pacto mundial ONU [26]).

En el proyecto desarrollado hay algunos de los objetivos que se consiguen abordar, en este apartado se específica que uso se le puede dar a la optimización bayesiana para lograr una mejora sustancial. De los diecisiete objetivos, hay tres que concuerdan con el presente proyecto:

- ODS 3: Salud y bienestar

  La optimización bayesiana puede ser aplicada en el desarrollo de nuevos medicamentos, algunos de estos muy importantes como el desarrollo de una vacuna como lo hubiera podido ser para la pandemia del COVID-19, consiguiendo desarrollarla en menos tiempo y salvando un mayor número de vidas, esto se logra gracias a la optimización de los diferentes ensayos clínicos.

- ODS 7: Energía asequible y no contaminante
  En este objetivo, la optimización puede realizar varias funciones. El primero de ellos sería el lograr mayor eficiencia en los parques eólicos cambiando la orientación de los molinos según la dirección del viento, del mismo modo se puede aplicar a las placas solares, cambiando su orientación según la hora del día, y la posición del sol logrando mayor rendimiento energético. También se podría aplicar en estos ámbitos, para predecir un mantenimiento preventivo, para realizarlo lo antes posible y no perder eficiencia, sobre todo con las placas solares que sólo son efectivas durante el día. Esto reduciría los costes y se estaría aplicando en energías limpias y sostenibles, las cuales si se consiguen emplear en mayor proporción acabarían sustituyendo a las tradicionales que funcionan con combustibles fósiles y generan gases de efecto invernadero.

- ODS 9: Industria, innovación e infraestructura
  En la industria es otro campo en el que se pueden aplicar estas metodologías. El desarrollo de productos de manufacturación es un claro ejemplo debido a que se reducen tiempos de producción, se consigue mayor eficiencia de los recursos que se emplean y todo ello acaba generando mayores beneficios en la industria debido al aumento del empleo que acaba afectando indirectamente a la economía y llegando a un mayor número de personas.

  En cuanto a la innovación, se aplica a predecir y evaluar el diseño de un cierto producto empleando los parámetros más adecuados, obteniendo asimismo los mismos beneficios explicados anteriormente.

  Y, por último, con la infraestructura, se podría emplear en las ciudades en una planta de reciclaje para clasificar de mejor forma los residuos generados y obtener un menor impacto ambiental, en los anexos se detalla una tabla con su grado de aportación para este proyecto, más concretamente en el último apartado.

9 Conclusiones

El proyecto abarca tres temáticas, las cuales son la robótica, la visión artificial y el *machine learning* que combinadas entre ellas hacen realidad el presente proyecto. Los resultados y comparativas que se obtienen hacen tener una idea bastante certera del funcionamiento de la optimización bayesiana y sus grandes ventajas a la hora de buscar un resultado, sin a priori, saber que aspecto tiene el sistema al que se enfrenta, dado que es difícil de calcular. Si la simulación no hubiera diferido tanto de la realidad por los problemas que se presentan se hubieran podido utilizar los datos recolectados en esta para pasárselos al algoritmo de la realidad. Estos datos reciben el nombre de *priormean*, donde el algoritmo directamente habría probado por donde se encuentra la solución y no habría tenido que realizar una exploración previa hasta encontrar un mínimo e ir mejorando por cada iteración la trayectoria siendo así más eficiente.

Además, se incorpora un grado más de dificultad, debido a que la plataforma tal y cómo está construida no es muy irregular, mediante el obstáculo se agrega esa dificultad extra que termina de corroborar que el sistema desarrollado es capaz de llegar a la solución en muy pocas iteraciones, en comparación así lo hubiéramos hecho a fuerza bruta como se realiza en el mapeado. Cabe destacar que esto se realiza para entender mejor el problema y corregir como es el caso la función objetivo y aumentar la repetibilidad para que se pudiera realizar la optimización. Si los experimentos hubieran mostrado un resultado erróneo, se habría enfocado el proyecto hacía otros sistemas en los que clasifican los experimentos y aunque se falle tienen mayor tolerancia como lo pueden ser el algoritmo de *Super Vector Machine (SVM).*

Otro aspecto destacado es el del factor humano cuando de disponer la pelota, ya que el algoritmo sí es capaz de predecir un ángulo muy exacto. Pero es cierto que el operario por muy metódico que sea al posicionar la pelota siempre hay un margen de error que puede afectar al resultado de las pruebas.

Por último, el experimento realizado muestra un ejemplo claro de como emplear estas herramientas tan útiles en estos tiempos en los que el *machine learning* y la inteligencia artificial están dando pasos agigantados hacia un mundo cada vez más concienciado con el medio ambiente y la eficiencia.

10 Bibliografía

[1] Nalda, V. (2021, 17 septiembre). *Machine Learning: Los orígenes y la evolución*. Future Space S.A. https://www.futurespace.es/machine-learning-los-origenes-y-la-evolucion/

[2] *¿Qué es la visión artificial? | IBM*. (s. f.). https://www.ibm.com/mx-es/topics/computer-vision

[3] EcuRed. (s. f.). *OpenCV - ECURed*. https://www.ecured.cu/OpenCV

[4] Del Carmen Sarmiento Vera, E. (s. f.). *Unimate fue el primer robot industrial*. Scribd. https://es.scribd.com/document/414781868/Unimate-Fue-El-Primer-Robot-Industrial

[5] Del Carmen Sarmiento Vera, M. (s. f.). *Biografía de Thomas Bayes*. platzi. https://platzi.com/tutoriales/1835-programacion-estocastica/7690-quien-fue-thomas-bayes-por-que-es-tan-relevante-este-matematico-biografia-de-thomas-bayes

[6] *Support Vector Machine (SVM)*. (s. f.). MATLAB & Simulink. https://es.mathworks.com/discovery/support-vector-machine.html

[7] ROBOTIS. (s. f.). *ROBOTIS e-Manual OpenMANIPULATOR-X*. https://emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/

[8] ROBOTIS. (s. f.). *ROBOTIS e-Manual XM430-W350* https://emanual.robotis.com/docs/en/dxl/x/xm430-w350/

[9] ROBOTIS. (s. f.). *ROBOTIS e-Manual DynamixelShield* https://emanual.robotis.com/docs/en/parts/interface/dynamixel_shield/

[10] Amazon. (s. f.). Convertidor de USB a TTL UART CH340G. https://www.amazon.es/dp/B07DJ3ZQM4/ref=sspa_dk_detail_3?psc=1&pd_rd_i=B07DJ3ZQM4&pd_rd_w=sWgXQ&content-id=amzn1.sym.d9fd07ad-95b5-4079-8602-de55e6918bc7&pf_rd_p=d9fd07ad-95b5-4079-8602-de55e6918bc7&pf_rd_r=0V4EPZK6D8EG7P9Y911N&pd_rd_wg=fSrW6&pd_rd_r=910b3f30-2fe8-4559-b6fa-969652ee333c&s=electronics&sp_csd=d2lkZ2V0TmFtZT1zcF9kZXRhaWw

[11] PcComponentes. (s. f.). *Aukey webcamfullhd USB.* https://www.pccomponentes.com/aukey-webcam-fullhd-usb

[12] Zotovic Stanisic, Ranko. (s. f.). *Apuntes de la asignatura de Sistemas Robotizados.* https://poliformat.upv.es/

[13] ROBOTIS OpenSourceTeam. (2022, 29 agosto). *DYNAMIXEL Quick Start Guide with DYNAMIXEL Wizard 2.0* [Vídeo]. YouTube. https://www.youtube.com/watch?v=JRRZW_l1V-U

[14] Leopoldo Armesto. (2022, 29 noviembre). *Control del multiples servos Dynamixel (asíncrono y síncrono)* [Vídeo]. YouTube. https://www.youtube.com/watch?v=_Q3Zc4sYiWM

[15] *OpenCV: Camera Calibration and 3D Reconstruction*. (s. f.). https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html

[16] *Evaluating the Accuracy of Single Camera Calibration - MATLAB & Simulink - MathWorks España*. (s. f.). https://es.mathworks.com/help/vision/ug/evaluating-the-accuracy-of-single-camera-calibration.html

[17] *Object for storing intrinsic camera parameters - MATLAB - MathWorks España*. (s. f.). https://es.mathworks.com/help/vision/ref/cameraintrinsics.html?searchHighlight=distortion%20camera&s_tid=srchtitle_support_results_3_distortion%20camera

[18] *mexopencv*. (s. f.). https://kyamagu.github.io/mexopencv/

[19] Universitat Politècnica de València - UPV. (2019, 27 noviembre). *Instalación de OpenCV en Matlab | | UPV* [Vídeo]. YouTube. https://www.youtube.com/watch?v=6YsOlfu60y0

[20] Solano, Gabriela. (2019, 2 marzo). *DETECCIÓN DE COLORES en OPENCV [en 4 Pasos] - Parte1* [Vídeo]. YouTube. https://www.youtube.com/watch?v=giwtDYcIXKA

[21] Gluón. (2020, 28 febrero). Estimación de posición con openCV y Python. *Lab. Gluón*. https://www.laboratoriogluon.com/estimacion-de-posicion-con-opencv-y-python/

[22] Kochenderfer, M. J., & Wheeler, T. A. (2019). *Algorithms for Optimization*. MIT Press.

[23] *Kernel (Covariance) Function Options - MATLAB & Simulink - MathWorks España*. (s. f.). https://es.mathworks.com/help/stats/kernel-covariance-function-options.html

[24] Sala-Piqueras, Antonio, *Bayesian optimization: PI, EI, LCB detailed example (Matlab)*. https://personales.upv.es/asala/DocenciaOnline/Video/boloop2EN.html

[25] Deruytter, H. (2024). *Exploring Bayesian optimization methods on a ball pushing robotic application on a golf-like terrain.* Universitat Politècnica de València. http://hdl.handle.net/10251/205291

[26] *ODS Objetivos de Desarrollo Sostenible | Pacto Mundial ONU*. (2023, 7 septiembre). Pacto Mundial. https://www.pactomundial.org/que-puedes-hacer-tu/ods/?gad_source=1&gclid=Cj0KCQjwj9-zBhDyARIsAERjds1Go5qiO3Je9ooYFF1ulKeUIEzrq_DXMKUOXlHWJYpa_lFZGiW-FOkaAkwzEALw_wcB

[27] *Bayesian Optimization Algorithm - MATLAB & Simulink - MathWorks España*. (s. f.). https://es.mathworks.com/help/stats/bayesian-optimization-algorithm.html#bvaz8tr-1

[28] Thingiverse.com. (s. f.). *OpenManipulator SARA by ROBOTIS*. Thingiverse. https://www.thingiverse.com/thing:3455668/files

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf

Trabajo Fin de Máster

Máster Universitario en Ingeniería Mecatrónica

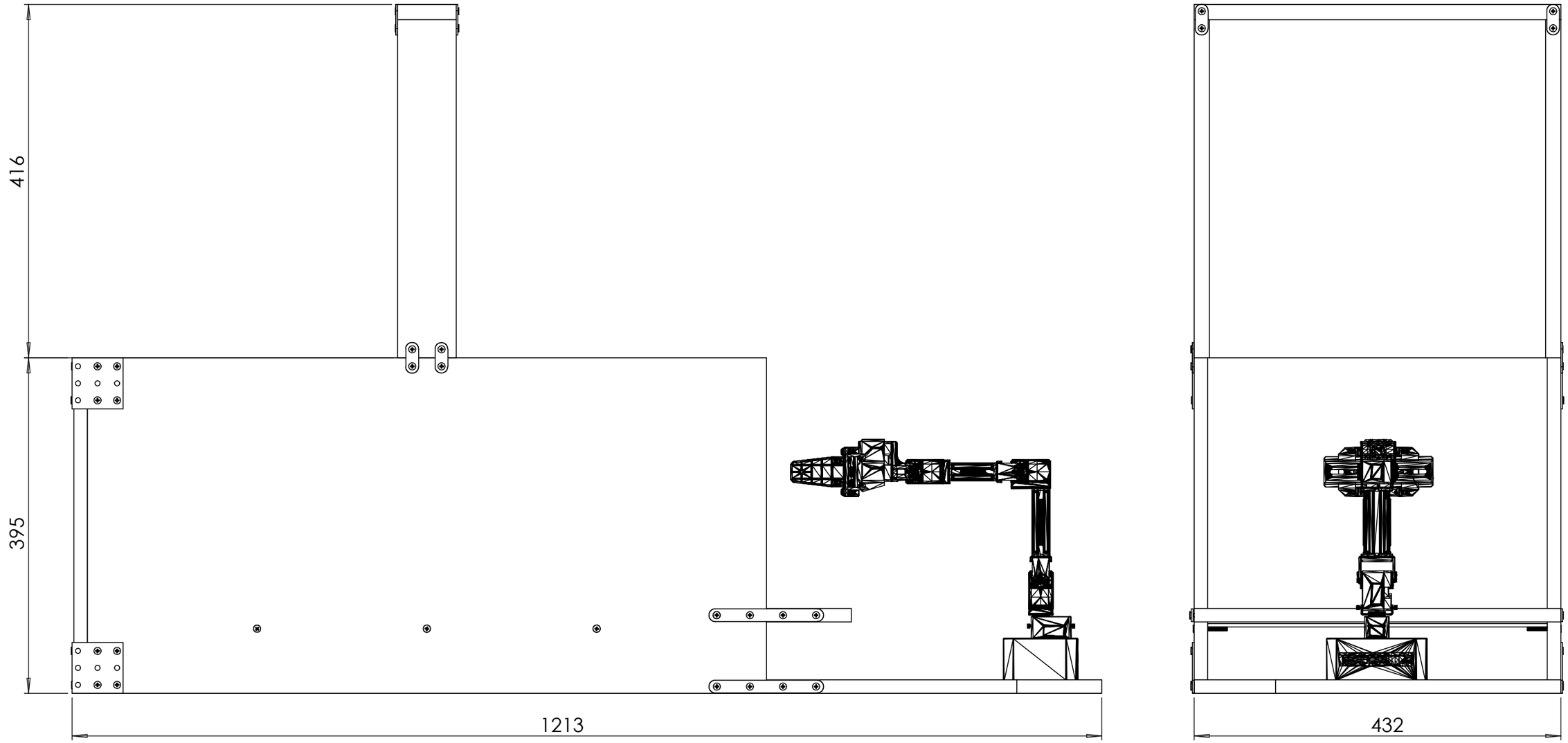
Segundo documento:

Planos

AUTOR/A: López Sanfeliciano, Antonio

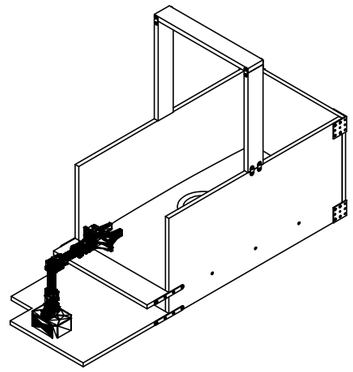
Tutor/a: Armesto Ángel, Leopoldo

CURSO ACADÉMICO:2023/2024



Proyecto: Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf Universitat Politècnica de València(UPV) Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial(ETSIADI)		Fecha: 11/6/2024 Escala: **1:5**
Autor: Antonio López Sanfeliciano	Plano: **Plano de conjunto**	Plano Nº: **00**

Producto SOLIDWORKS Educational. Solo para uso en la enseñanza.



N.º	NOMBRE	CANT.
1	Plataforma	1
2	Listón abeto lateral	2
3	Madera aglomerado posterior	1
4	Escuadra de acero	4
5	Madera aglomerado derecha	1
6	ISO 7046-1 - M4 x 16 - Z - 16S	64
7	Madera aglomerado sujeta pelota	1
8	Placa unión acero cincado larga	4
9	Madera aglomerado sujeta robot	1
10	Base del robot	1
11	OpenMANIPULATOR-X	1
12	ISO 7046-1 - M4 x 40 - Z - 40S	6
13	Madera aflomerado izquierda	1
14	Placa unión bricomada	8
15	Listón abeto superior	1

Proyecto: Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf

Universitat Politècnica de València(UPV)
Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial(ETSIADI)

Fecha: 11/6/2024

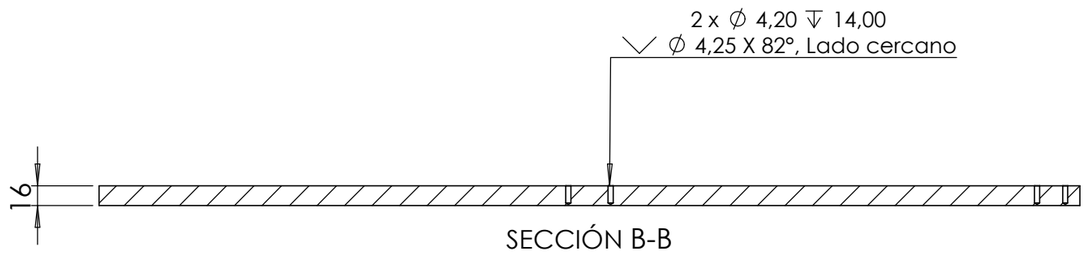
Escala: **1:5**

Autor: Antonio López Sanfeliciano

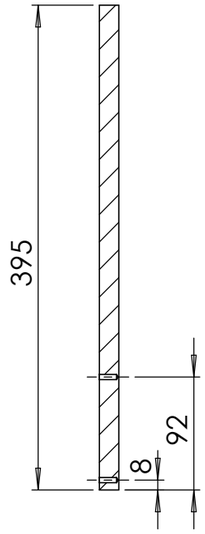
Plano: **Axonometría explosionada**
Despiece del material

Plano Nº: **01**

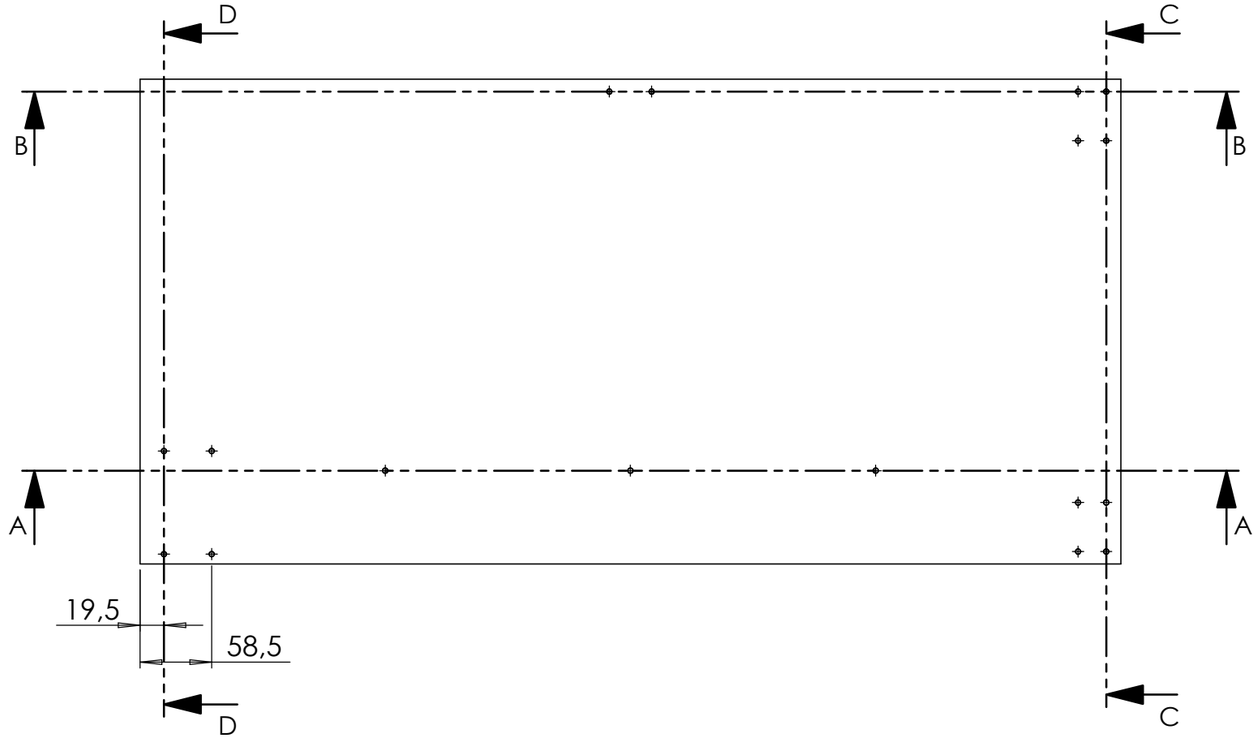
Producto SOLIDWORKS Educational. Solo para uso en la enseñanza.

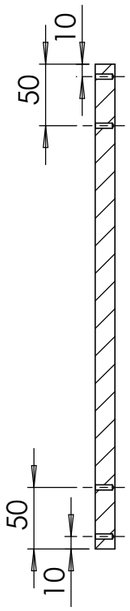


SECCIÓN B-B

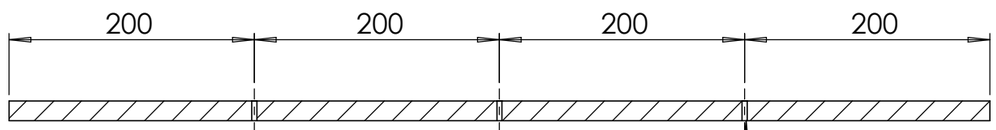


SECCIÓN D-D



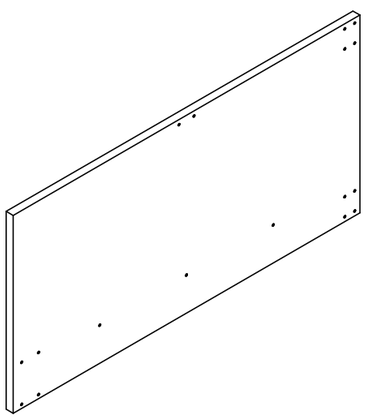


SECCIÓN C-C



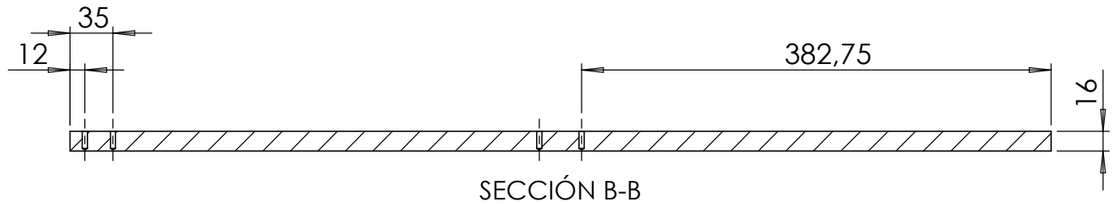
SECCIÓN A-A

3 x ∅ 4,20 ↧ 16,00
⌵ ∅ 4,25 X 82°, Lado cercano

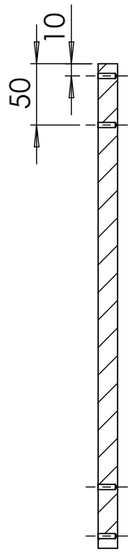


Proyecto: Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf Universitat Politècnica de València(UPV) Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial(ETSIADI)		Fecha: 11/6/2024 Escala: **1:5**
Autor: Antonio López Sanfeliciano	Plano: **Madera aglomerada derecha**	Plano Nº: **02**

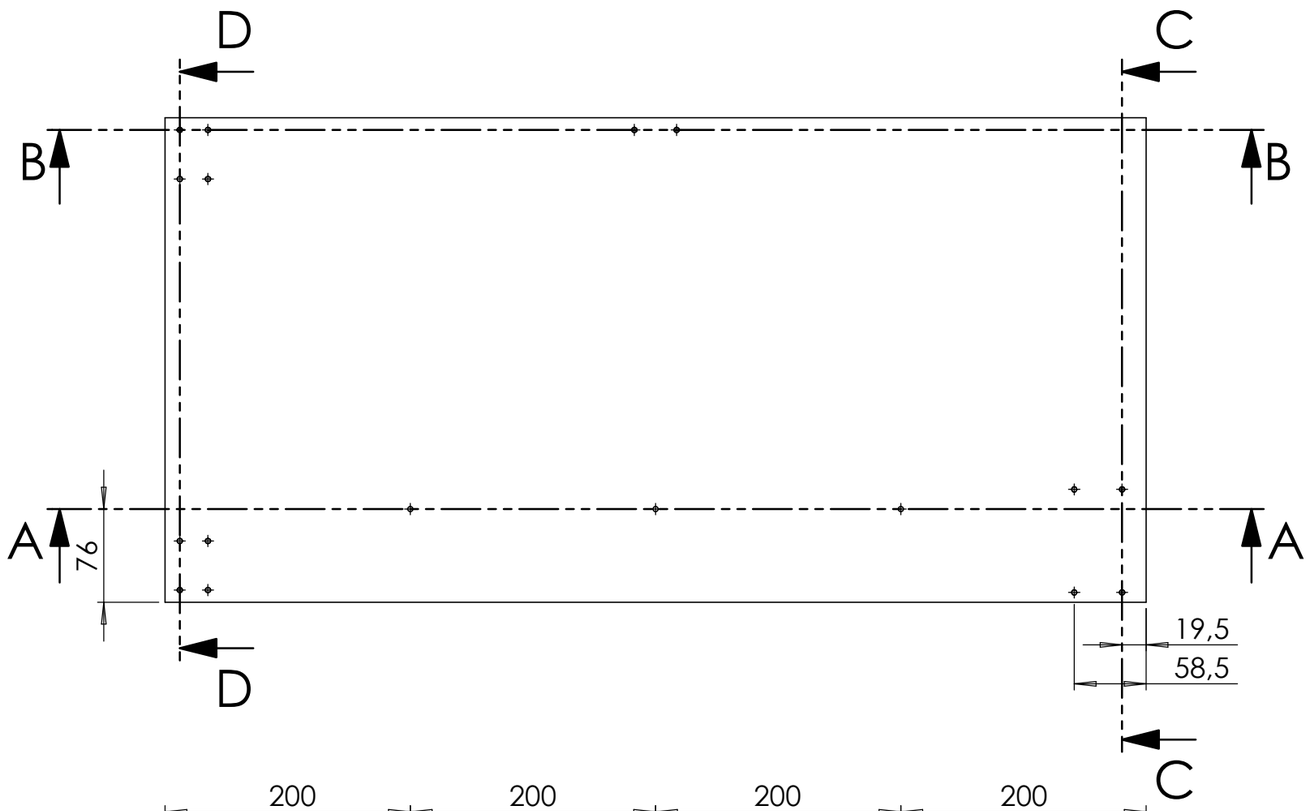
Producto SOLIDWORKS Educational. Solo para uso en la enseñanza.

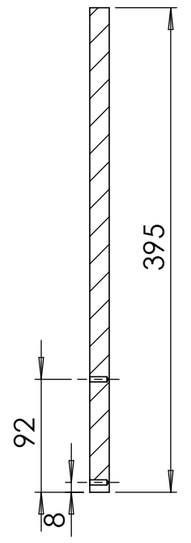


SECCIÓN B-B

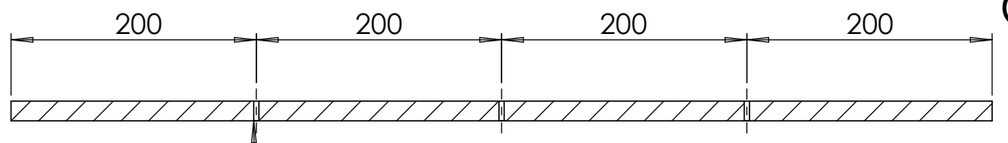


SECCIÓN D-D

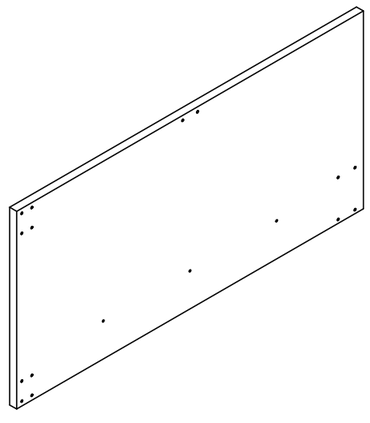




SECCIÓN C-C



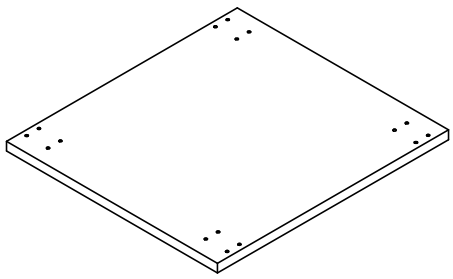
SECCIÓN A-A

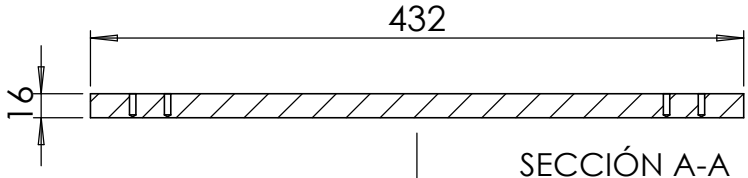


3 x ⌀ 4,20 ↧ 16,00
⌵ ⌀ 4,25 X 82°, Lado cercano

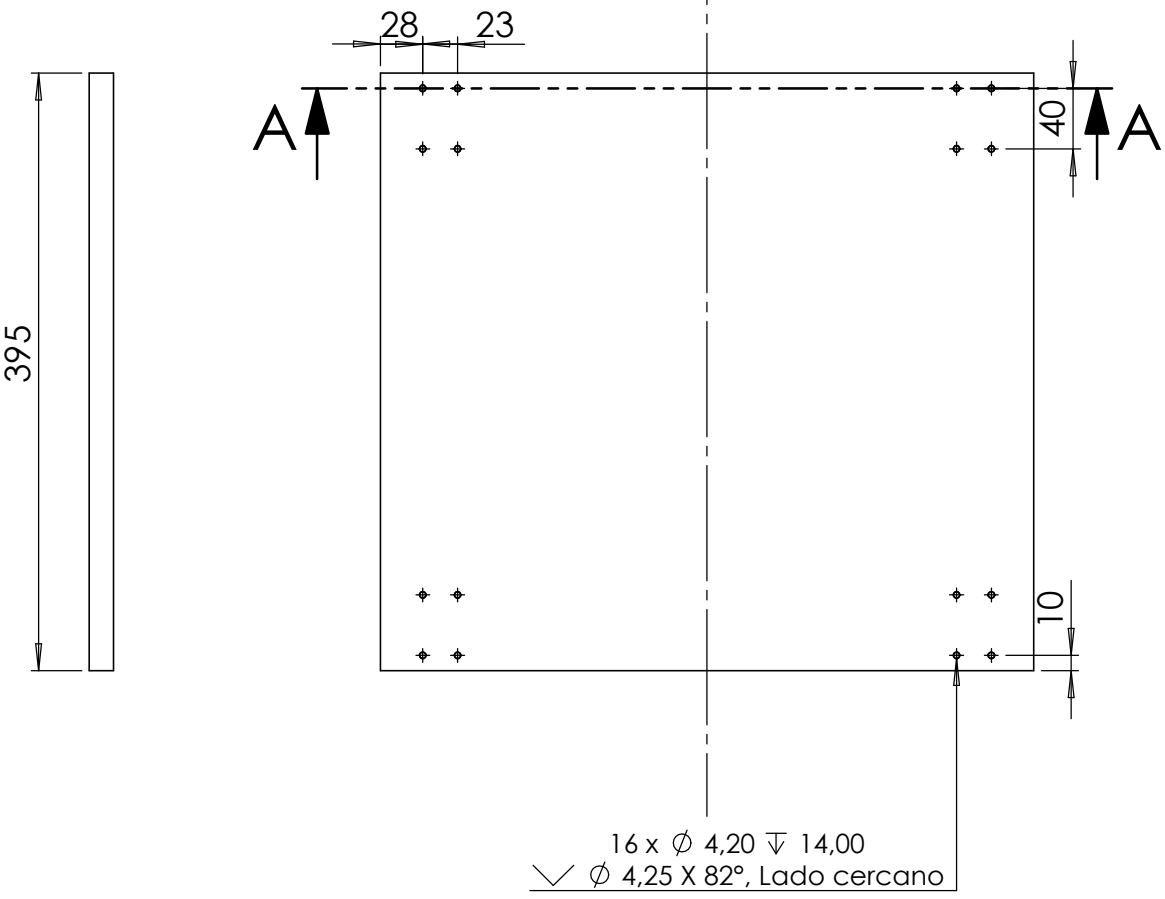
Proyecto: Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf Universitat Politècnica de València(UPV) Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial(ETSIADI)		Fecha: 11/6/2024 Escala: **1:5**
Autor: Antonio López Sanfeliciano	Plano: **Madera aglomerada izquierda**	Plano Nº: **03**

Producto SOLIDWORKS Educational. Solo para uso en la enseñanza.



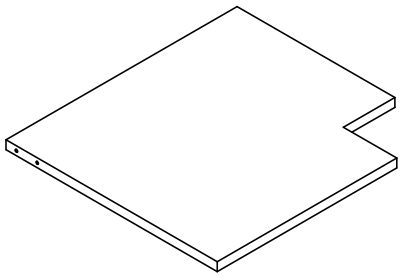


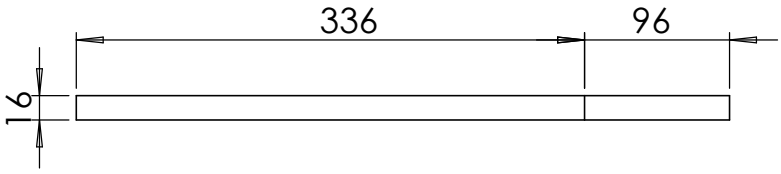
SECCIÓN A-A

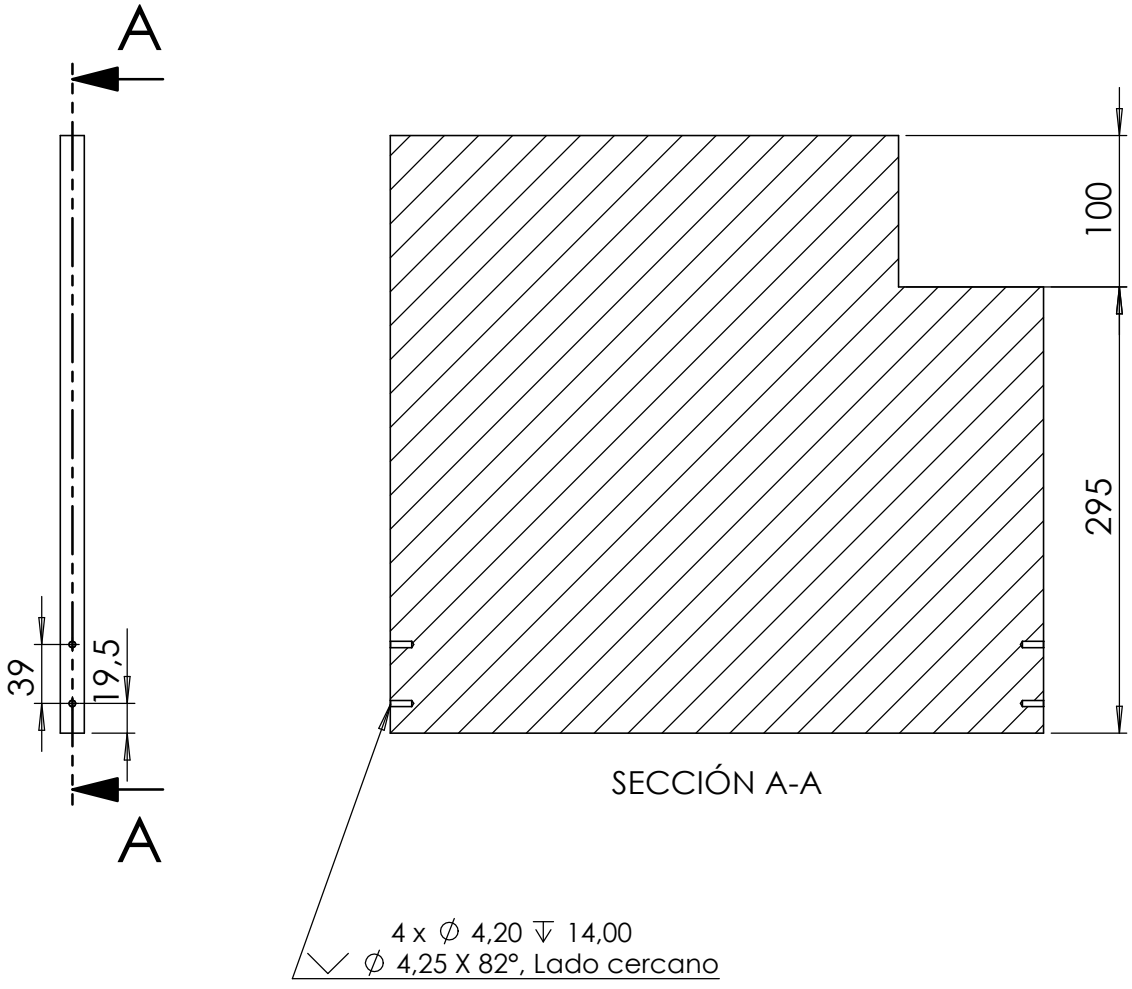


Proyecto: Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf Universitat Politècnica de València(UPV) Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial(ETSIADI)		Fecha: 11/6/2024 Escala: **1:5**
Autor: Antonio López Sanfeliciano	Plano: **Madera de aglomerada posterior**	Plano Nº: **04**

Producto SOLIDWORKS Educational. Solo para uso en la enseñanza.

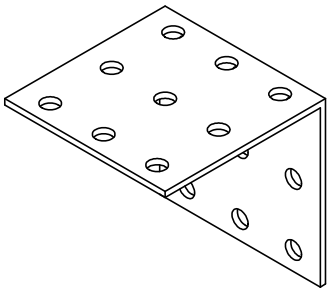


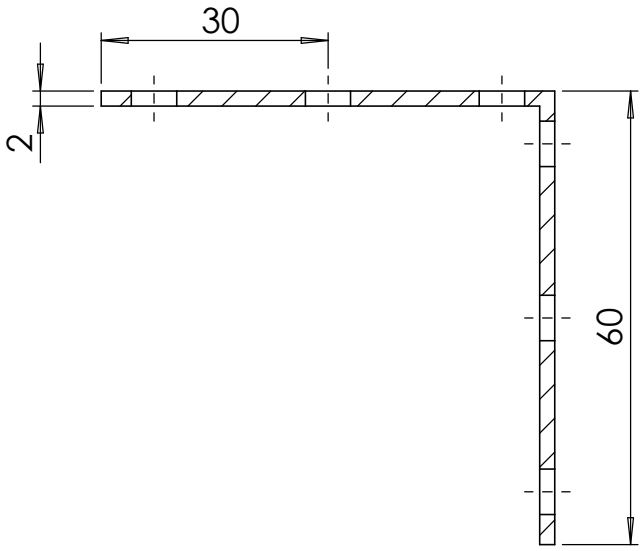




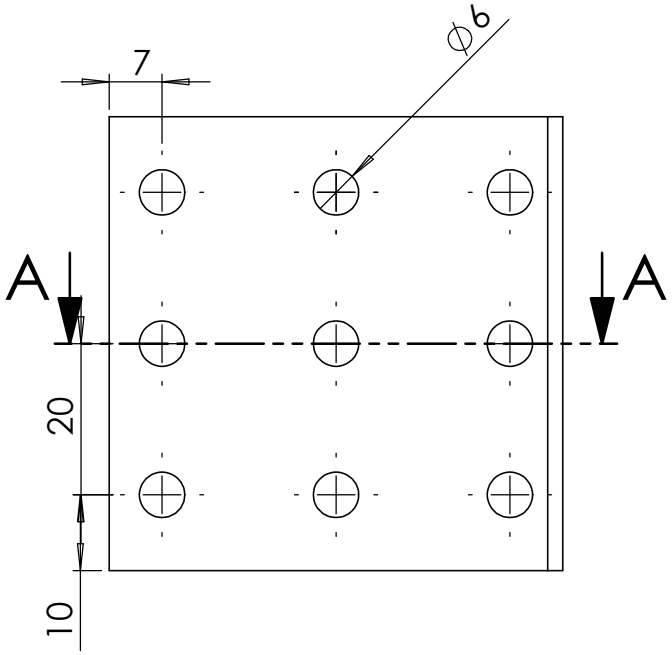
Proyecto: Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf Universitat Politècnica de València(UPV) Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial(ETSIADI)		Fecha: 11/6/2024 Escala: **1:5**
Autor: Antonio López Sanfeliciano	Plano: **Madera aglomerada sujeta robot**	Plano Nº: **05**

Producto SOLIDWORKS Educational. Solo para uso en la enseñanza.



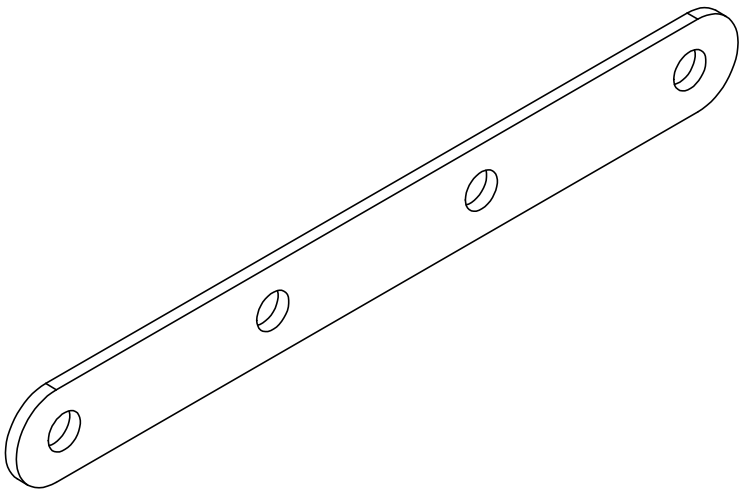


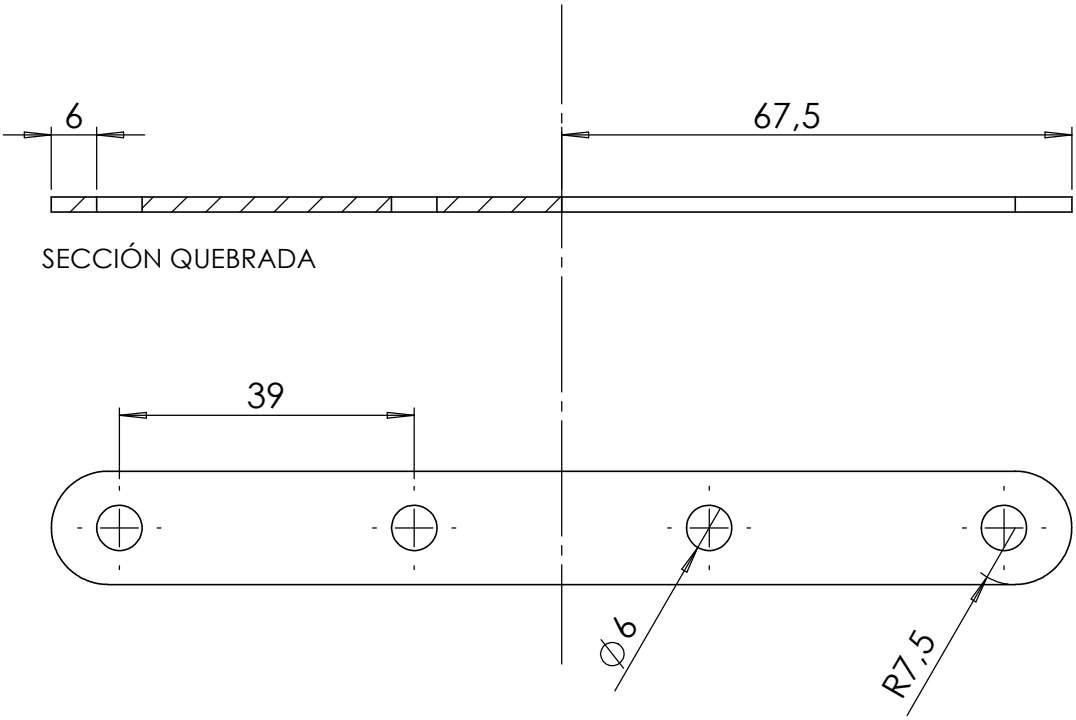
SECCIÓN A-A



Proyecto: Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf Universitat Politècnica de València(UPV) Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial(ETSIADI)		Fecha: 11/6/2024 Escala: **1:1**
Autor: Antonio López Sanfeliciano	Plano: **Escuadra de acero**	Plano Nº: **06**

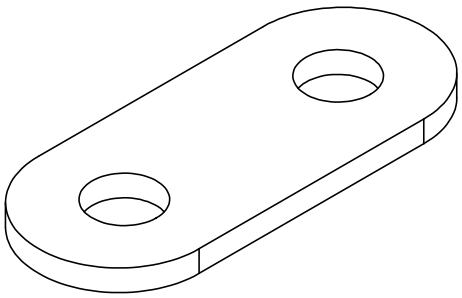
Producto SOLIDWORKS Educational. Solo para uso en la enseñanza.

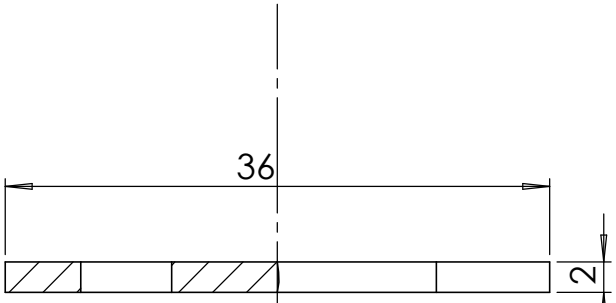




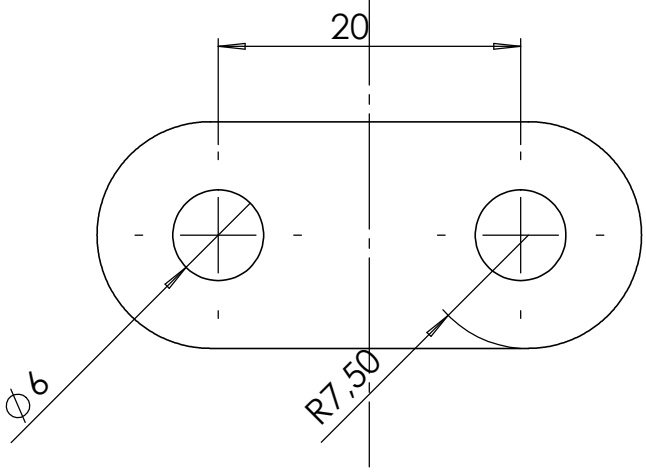
Proyecto: Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf Universitat Politècnica de València(UPV) Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial(ETSIADI)		Fecha: 11/6/2024 Escala: **1:1**
Autor: Antonio López Sanfeliciano	Plano: **Placa unión acero cincado larga**	Plano Nº: **07**

Producto SOLIDWORKS Educational. Solo para uso en la enseñanza.



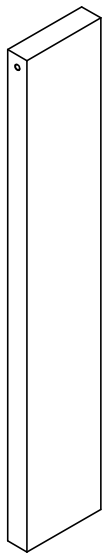


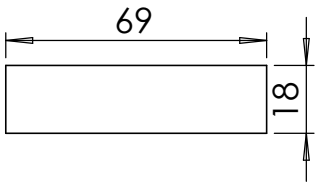
SECCIÓN QUEBRADA

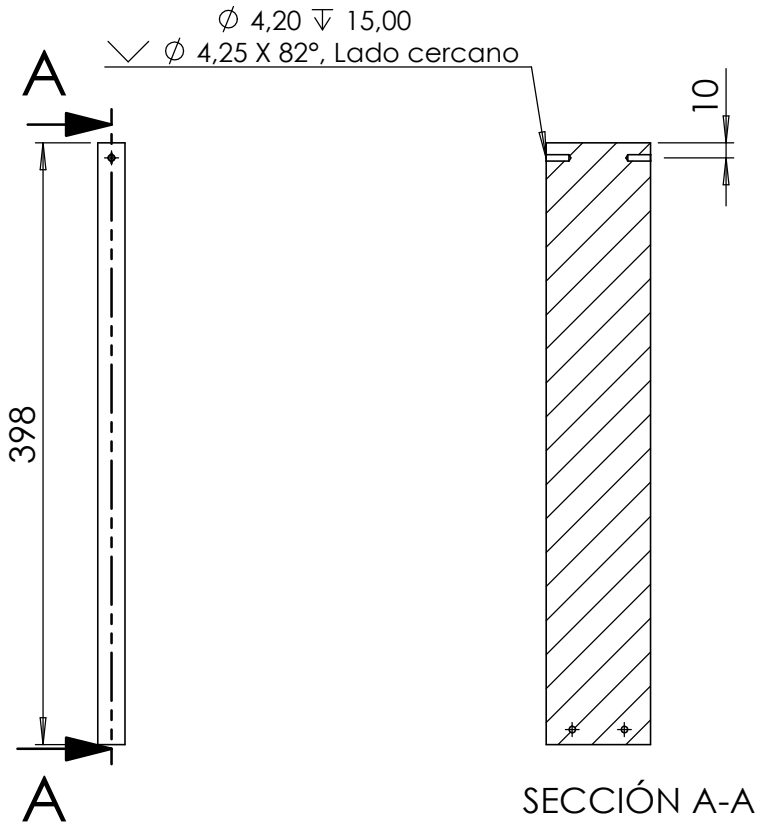


Proyecto: Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf Universitat Politècnica de València(UPV) Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial(ETSIADI)		Fecha: 11/6/2024 Escala: **1:1**
Autor: Antonio López Sanfeliciano	Plano: **Placa unión bricomatada**	Plano Nº: **08**

Producto SOLIDWORKS Educational. Solo para uso en la enseñanza.



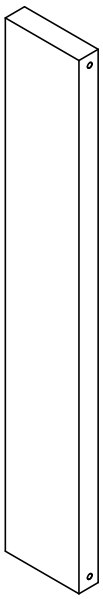


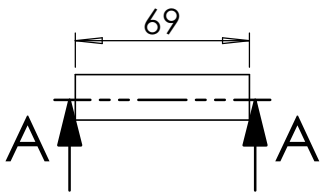


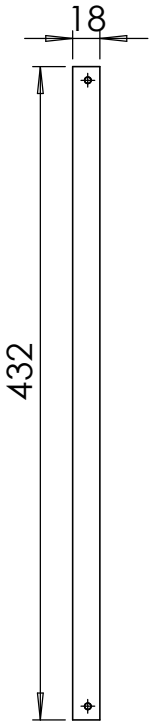
SECCIÓN A-A

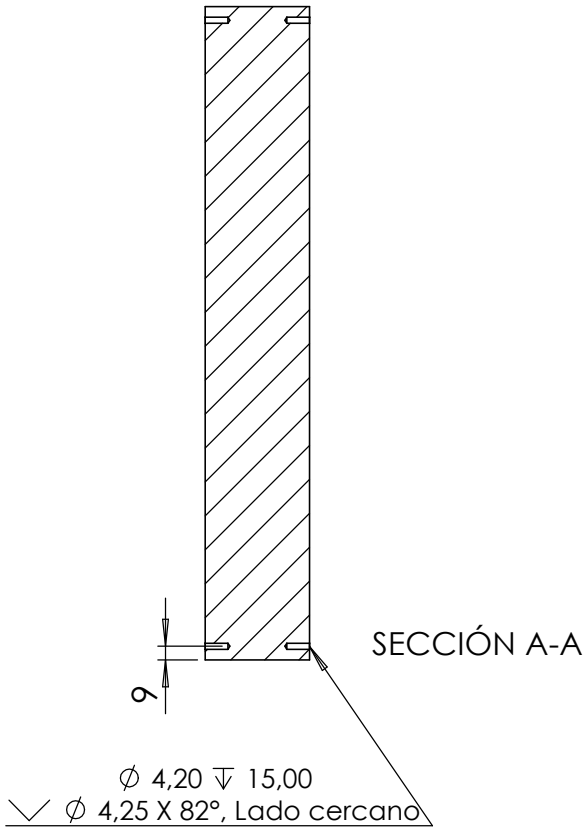
Proyecto: Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf Universitat Politècnica de València(UPV) Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial(ETSIADI)		Fecha: 11/6/2024 Escala: **1:5** SECCIÓN A-A
Autor: Antonio López Sanfeliciano	Plano: **Listón de madera lateral**	Plano Nº: **09**

Producto SOLIDWORKS Educational. Solo para uso en la enseñanza.



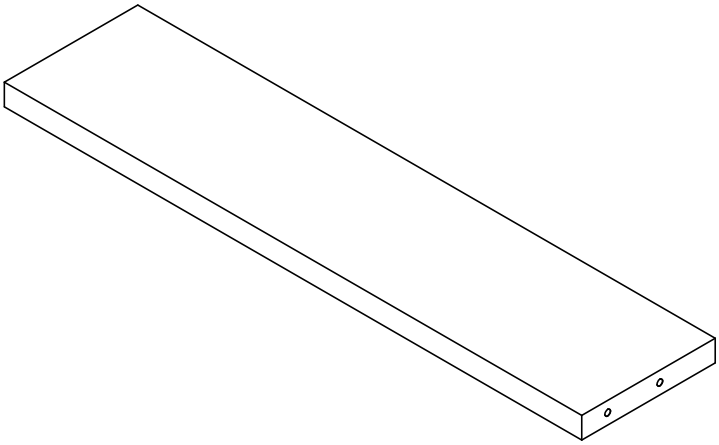


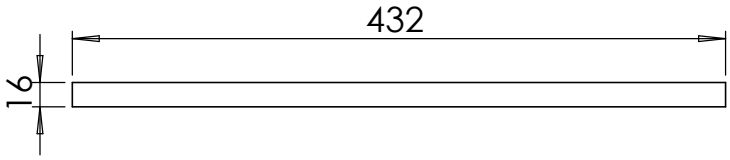




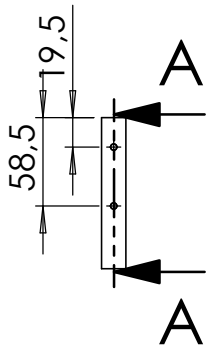
Proyecto: Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf Universitat Politècnica de València(UPV) Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial(ETSIADI)		Fecha: 11/6/2024 Escala: **1:5**
Autor: Antonio López Sanfeliciano	Plano: **Listón abeto superior**	Plano Nº: **10**

Producto SOLIDWORKS Educational. Solo para uso en la enseñanza.





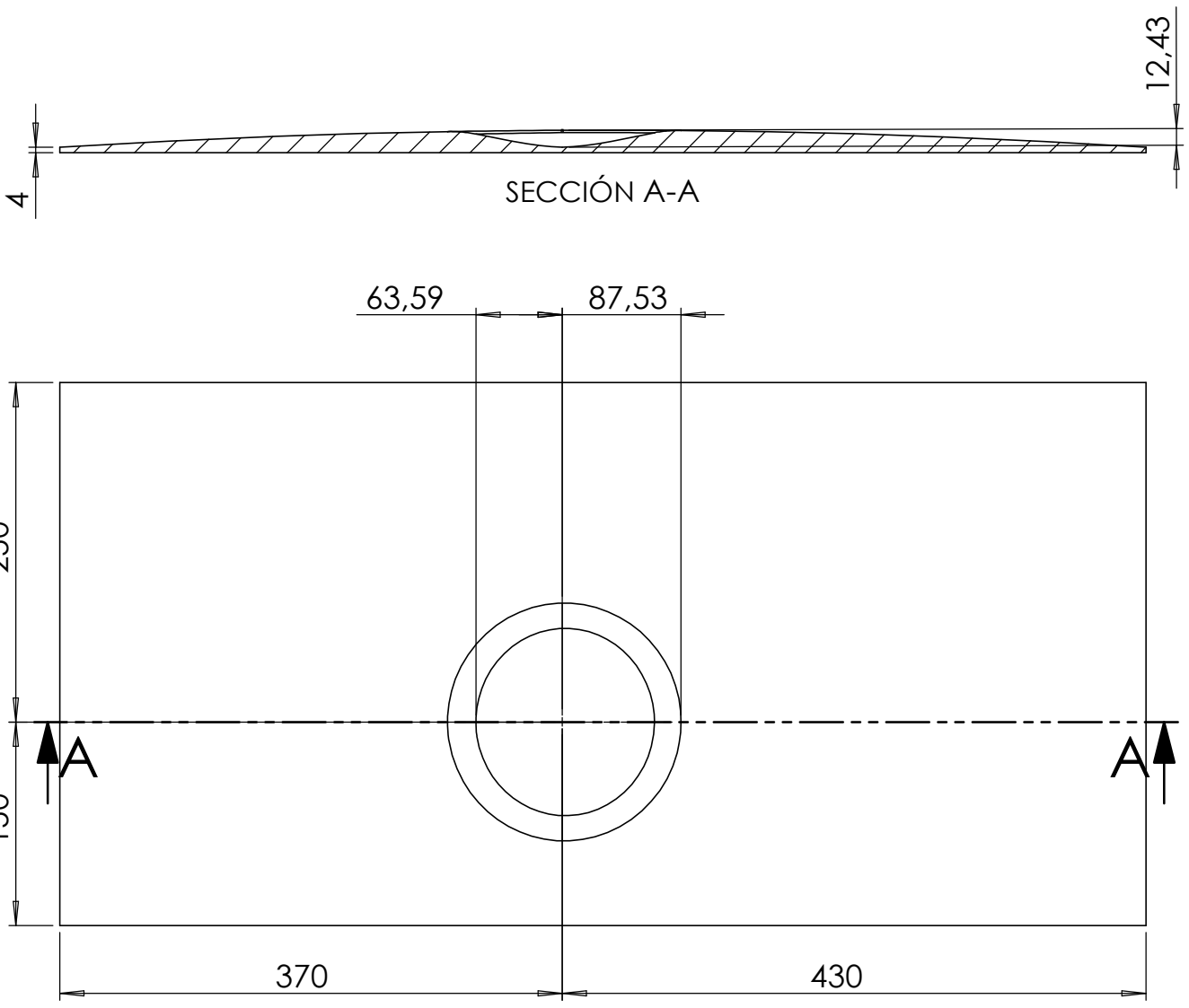
SECCIÓN A-A



2 x ∅ 4,20 ↧ 14,00
⌵ ∅ 4,25 X 82°, Lado cercano

Proyecto: Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf Universitat Politècnica de València(UPV) Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial(ETSIADI)		Fecha: 11/6/2024 Escala: **1:5**
Autor: Antonio López Sanfeliciano	Plano: **Madera aglomerada sujeta pelota**	Plano Nº: **11**

Producto SOLIDWORKS Educational. Solo para uso en la enseñanza.



Proyecto: Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf Universitat Politècnica de València(UPV) Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial(ETSIADI)		Fecha: 11/6/2024 Escala: **1:5**
Autor: Antonio López Sanfeliciano	Plano: **Plataforma**	Plano Nº: **12**

Producto SOLIDWORKS Educational. Solo para uso en la enseñanza.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf

Trabajo Fin de Máster

Máster Universitario en Ingeniería Mecatrónica

Tercer documento:

Pliego de condiciones

AUTOR/A: López Sanfeliciano, Antonio

Tutor/a: Armesto Ángel,Leopoldo

CURSO ACADÉMICO:2023/2024

Índice pliego de condiciones

1. Objeto

La presente especificación técnica refiere a la implementación y programación de una estructura destinada a la realización de experimentos de optimización bayesiana en los que un robot manipulador, en concreto el OpenMANIPULATOR-X, aprende a orientarse y ejecutar el movimiento con en un periodo de tiempo determinado para depositar una pelota de frontón en un hoyo objetivo. Por otra parte, hay un sistema de visión que localiza la posición de dicha pelota donde todo ello es procesado, posteriormente controlado a través de un ordenador mediante un software específico, más concretamente el MATLAB.

2. Normativa

Al trabajar en baja tensión se hará hincapié en las ITC (Instrucciones Técnicas Complementarias) establecidas en el Real Decreto 842/2002, que establece el reglamento electrotécnico de baja tensión, donde se aplicarán de forma total o parcial las normas UNE establecidas. El robot al no ser de carácter industrial sino de uso doméstico, no requiere de normativa, por tanto únicamente se ha de tener precaución con el mismo.

- ITC-BT-05: Verificaciones e inspecciones.
- ITC-BT-23: Instalaciones interiores o receptoras. Protección contra sobretensiones.
- ITC-BT-51: Instalaciones de sistemas de automatización, gestión técnica de la energía y seguridad para viviendas y edificios.

Además, se tiene en cuenta la protección de la persona que realice los experimentos frente a riegos eléctricos:

- Real Decreto 614/2001, de 8 de junio, sobre disposiciones mínimas para la protección de la salud y seguridad de los trabajadores frente al riesgo eléctrico.

3. Materiales y software

En este apartado se detallan tanto los materiales como el software requerido en el presente proyecto de optimización bayesiana.

3.1 Sistema de robótica

Este sistema describe los materiales empleados referente al robot manipulador empleado y su base de sujeción.

3.1.1 OpenMANIPULATOR-X

Robot encargado de realizar los movimientos correspondientes para empujar la pelota sobre la superficie donde se encuentra la pelota. Contiene los servomotores encargados de realizar estos movimientos, en concreto los XM430-W350-T, con 4.1 Nm de par nominal a 12 V 2.3 A y dispone de 12 bits de resolución del *encoder*. Se encuentran conectados desde el primer servomotor hasta el quinto en serie, mediante un cable TTL, en los cuáles se comunica desde el primero hasta el último.

3.1.2 Base de sujeción

El robot manipulador necesita una base en la que estar sujeto, dado que al efectuar de realizar los movimientos necesarios este si no se atornilla a la base impresa en 3D no podría producir los lanzamientos de los diferentes experimentos elaborados. El material que se emplea para la realización de la pieza es el PLA, el cuál es lo suficientemente resistente para aguantar tanto el peso como los movimientos del robot.

3.2 Sistema de visión artificial

Es el encargado de capturar las imágenes para su posterior procesado y localización de la pelota de frontón.

3.2.1 Aukey FullHD

La cámara, alimentada mediante conector USB 2.0 con el ordenador, es de dos megapíxeles con una resolución de 1080p y 30 FPS. Posteriormente, se le baja la resolución a 480p para que el procesamiento de imagen se puedan conseguir de forma estable la tasa de imágenes por segundo especificada y así recolectar una las cincuenta coordenadas en el momento que la pelota está recorriendo.

3.3 Sistema de la estructura.

Para la construcción de la estructura que sujeta la plataforma, contiene la cámara, por consiguiente, también la pelota de frontón se requiere de:

- Dos maderas de aglomerado 80x39.5x1.6 cm
- Una madera de aglomerado de 120x39.5x1.6 cm
- 64 tornillos de diámetro cuatro y longitud 16 mm
- 6 tornillos de diámetro cuatro t longitud 40 mm
- 4 placas de acero de longitud 13,5. cm
- 8 placas bricomatadas de longitud 3.6 cm
- Dos listones de abeto de 90x6.9x1.8 cm
- Plataforma mecanizada de poliuretano
- Pelota de frontón

3.4 Sistema de control

Especificación del hardware en el que se controla el funcionamiento de todos los sistemas y se coordinan entre ellos para lograr el correcto funcionamiento del algoritmo de la optimización bayesiana.

3.4.1 Portátil

Es el núcleo que coordina todos los sistemas de visión, y robótica. Se comunica con el OpenMANIPULATOR-X y con la cámara. Tras cada experimento es el encargado de hacer los diferentes cálculos y clasificar que ángulos y tiempo de trayectoria son los más adecuados para lograr el objetivo. Las especificaciones del portátil empleado son las mencionadas a continuación:

- Procesador: Intel i7-8550 1.8GHz
- RAM:12Gb de ram gddr3
- Memoria: 1 Tb SSD
- Gráfica: Intel(R) UHD Graphics 620

3.4.2 Arduino UNO

Se trata de una placa basada en el microcontrolador ATmega328P, el cual tiene 14 pines digitales, operan a 5V y pueden ser empleados como entradas o salidas, de los cuales seis pueden emplearse como salidas PWM. Tiene seis entradas analógicas y una frecuencia de reloj de 16 MHz.

3.4.3 Dynamixel Shield

Se acopla directamente al Arduino UNO, es alimentado entre 5 y 24 V, admite hasta 10 A como máximo, tiene tres conectores diferentes dependiendo de los motores, los TTL, TTL(XL-329) y RS485. Además de dos interruptores, uno para encender o apagara el *shield* y otro para especificar si el puerto serie lo emplea el Arduino para subir el programa o para comunicarse con el DynamixelShield. Por otra parte, el dispositivo UART a TTL que se comunica se conecta el RX/TX a los pines 7 y 8.

3.4.4 Dispositivo UART a TTL

Se encarga de transmitir desde el ordenador con el DynamixelShield, se puede conectar a 5 V o a 3.3V, en este proyecto se emplea a 3.3V del DynamixelShield, por tanto se ha de posicionar el jumper para que funcione a este voltaje. Posteriormente como se ha especificado se conecta al 7 y al 8, en este pero al revés que en DynamixelShield, es decir, en el 7 el de trasmisión (TX) y en el 8 el de recepción (RX). Es muy importante que se conecte de esta forma ya que sino no se comunicarán el Arduino con el portátil, por tanto no se podrían realizar los experimentos.

3.4.5 Fuente de alimentación

Este elemento es necesario para que el robot pueda moverse, se conecta por un lado a la red eléctrica alterna y transforma en continua a 12 V con un máximo de 3 A. La salida que ofrece va a los conectores de potencia del DynamixelShield.

3.5 Software

A continuación, se presentan los programas empleados para realización del presente proyecto.

3.5.1 Arduino IDE

Es un software gratuito desarrollado para las placas Arduino, entorno de programación en C/C++, contiene un monitor serie y un serial plotter para observar las lecturas de los sensores o actuadores que se emplean. Contiene un amplía gama de librerías para el uso de funciones específicas, que facilitan el empleo de los componentes, también realiza la compilación y subida del programa. Contiene una amplia gama de ejemplos y documentación, en la cual hay una gran comunidad de usuarios.

Requisitos mínimos:

- 1 GB de almacenamiento
- 256 MB de memoria RAM
- Procesador Pentium 4 o superior
- Multiplataforma

3.5.2 MATLAB

Programa desarrollado por MathWorks, en el que se emplea un lenguaje de alto nivel, contiene librerías o *toolbox* que permiten una mayor eficiencia con el desarrollo de algoritmos. Contiene una extensión llamada Simulink que permite la simulación de sistemas dinámicos, controles… Se requiere de licencia para su uso, tiene una plataforma de ayuda para el entender las funciones, incluyendo además varios ejemplos. Es el encargado de gestionar los diferentes scripts empleados para el desarrollo del proyecto.

Requisitos mínimos MATLAB 2023b:

- 3.8 GB de almacenamiento
- 8 GB de memoria RAM
- Procesador de dos núcleos o superior
- Windows 10 o superior, compatible con Linux o MAC

3.5.3 OpenCV

El OpenCV no es exactamente un program, sino una librería de OpenSource que se decidide añadir a este apartado debido a su gran relevancia en el proyecto. Contiene un gran número de funciones para el empleo de visión artificial. Su entorno de programación es en C/C++ aunque se puede emplear también en lenguajes de más alto nivel como lo es Python. Proporciona una gran cantidad de documentación para el correcto empleo de sus funciones. Se instala en MATLAB con ayuda de un compilador y el mexopencv.

3.5.4 Wizard 2.0

Desarrollado por ROBOTIS, para la configuración de sus servomotores mediante el uso de la placa OpenRB150. Se puede actualizar el firmware de estos, cambiar el PID de los motores, fijar un ID, visualización de posicionamiento, entre otros aspectos. En el proyecto es empleado para poder configurar los cinco motores que contiene el OpenMANIPULATOR-X.

3.5.5 SolidWorks

Es el software de diseño CAD en 3D, empleado para el crear los elementos necesarios en el desarrollo del proyecto, así como la obtención de los planos para su montaje. Permite la elaborar archivos desde cero e incluso la importación de otros ya existentes como los STL, empleados en la impresión 3D. Los requerimientos recomendados para el uso de SolidWorks 2022 son:

- 20 GB de almacenamiento
- 16 GB de memoria RAM
- Procesador de 3.3GHz
- Windows 10 o superior, compatible con Linux o MAC

4. Condiciones de ejecución

En este apartado se explica, los pasos a seguir para empezar a realizar los experimentos.

4.1 Montaje

El primer paso, es cortar con precaución las diferentes partes de la madera tal y como se indica en los planos, cogiendo únicamente el tablón de madera aglomerada que mide 120 cm de largo y obteniendo de ahí la madera que sujeta el robot, la madera posterior y la sujeta pelota. Posteriormente, los listones se deben realizar los cortes necesarios también como se indica en los planos. Seguidamente con la ayuda de un taladro se hacen las perforaciones necesarias para evitar que al atornillar se parta la madera. Cabe destacar que hay un total de seis perforaciones que deben atravesar la madera, estos se encuentran en los tableros 80 cm de largo, tres en cada una de ellas. Por último, se fija la cámara en el listón superior, teniendo en cuenta que es capaz de observar toda la superficie. Finalmente se deja caer la plataforma mecanizada sobre los tornillos que han atravesado las dos maderas laterales, con esto el montaje quedaría realizado y se acopla el robot a su base, la cual hay que atornar con 4 tornillos de longitud 40 mm.

4.2 Cableado

La salida de la fuente de alimentación se conecta al *DynamixelShield*, el que es de color azul es el que va al positivo y el negro al negativo. Una vez realizado esto se acopla el shield con sus pines al Arduino UNO. Procedemos a conectar el OpenMANIPULATOR-X a uno de los pines TTL. A continuación, debemos acoplar el adaptador USB a TTL, el pin de 3.3V (poner el jumper en 3.3V) tiene que ir a este mismo voltaje en el *DynamixelShield*, y el de negativo al GND. En cuanto al pin TX del adaptador va al pin 7 y el RX al 8, con esto finalizaríamos el cableado del hardware que controla el robot, sólo habría que conectar el USB al PC. Por último, se debe conectar el USB de la cámara al portátil.

4.3 Subida del programa

El primer paso que se debe hacer para subir el programa al Arduino UNO es el poner el switch del DynamixelShield en modo UART, ya que emplearemos ese puerto serie para subirlo. Posteriormente, se conecta el cable de USB tipo A y B. Abrimos el programa con Arduino IDE, selecciona el puerto COM en el que se encuentra conectado, por último, subimos el programa, primero compilará y posteriormente se vuelve a poner el switch en la posición que controla los servomotores de Dynamixel.

4.4 Calibración de la cámara

Una vez dispuesta la cámara se ha de realizar la calibración, para ello abrimos la aplicación de MATLAB que se emplea en el proyecto, se toman diez fotos con la cámara a una hoja impresa con el tablero de ajedrez (se debe saber que mide el lado del cuadrado) y se le da al botón de *calibrate*. Si alguna foto tiene mucho error se descarta y se toma otra. Luego si es correcto exportamos la calibración a un script a parte y al ejecutarlo en la parte del *workspace* hay que buscar la matriz intrínseca de la cámara con el nombre K. Para obtener la matriz extrínseca debemos ejecutar el script de *calibrarReferencia.m*, asegurándose que se abre una figura y se observe que es la cámara que hemos montado la que se visualiza, sino habría que cambiar el número de cv.Capture(X), donde X es una valor que

normalmente puede empezar en 0. Se disponen las etiquetas en las esquinas de la plataforma, con un orden determinado como se especifica en la memoria, si se desea cambiar el color de la etiqueta se deben alterar los rangos establecidos al correspondiente en HSV. Una vez detecte los cuatro colores, podemos dejar de ejecutar el programa y obtendríamos los vectores necesarios de rotación y traslación que emplearemos para calcular la matriz extrínseca. Finalmente obtenidas estas dos matrices, la calibración estaría completada.

4.5 Ejecución programa principal

Si se ha modificado en *calibrarReferencia.m* el número de la cámara, hay que actualizar el valor en la función *visionCapture.m junto también con los datos de la calibración*, además, comprobar en *Administrador de Dispositivos* en que puerto COM se encuentra el adaptador TTL y modificar el valor en las funciones *lanzamiento.m* y *lanzamiento2.m.* Una vez ajustados todos los parámetros, únicamente quedaría poner que función de adquisición se va a emplear y ejecutar el script de *bayesoptfunction.m.*

5. Prueba de servicio

La forma más eficaz de comprobar que todo funciona de forma correcta es ejecutar el script de *RobotyVision.m,* previamente habiendo configurado el ángulo y tiempo de trayectoria deseado para la ejecución. Si se quiere cambiar alguno de los ángulos en los que realiza el movimiento del robot, se deberá hacer dentro de la función de *lanzamiento.m*.

El programa repite diez veces el mismo disparo para realizar pruebas de repetibilidad, comprobar que funciona correctamente, sino lijar el agujero con un gramaje no muy grande, hasta que se pueda ver que la pelota se pueda depositar de forma más consistente. Otra prueba que se ha de realizar es probar en los extremos de los rangos para que todo funcione correctamente, es decir, por ejemplo, para 15º probar las trayectorias de 750 y 650 ms, luego para 0º, por último en -15º.

El último paso que se realiza en la prueba de servicio es ejecutar *bayesoptfunction.m*, previamente habiendo configurado la función de adquisición que se quiere emplear, así como el ratio de exploración y el número de iteraciones. Si el robot consigue encontrar una solución y repetirla, la puesta en marcha habrá sido ejecutada de forma correcta.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf

Trabajo Fin de Máster

Máster Universitario en Ingeniería Mecatrónica

Cuarto documento:

Presupuesto

AUTOR/A: López Sanfeliciano, Antonio

Tutor/a: Armesto Ángel, Leopoldo

CURSO ACADÉMICO:2023/2024

4.-Presupuesto

Índice de tablas del presupuesto

El presupuesto presenta las diferentes tablas que contienen los materiales, precios descompuestos y valoración. Algunos precios se calculan a partir del precio medio de la luz en España (0,12€ kWh) en junio de 2024. Se tiene en cuenta los costes que son difíciles de cuantificar en un 10%, llamados medios auxiliares sobre costes directos.

1. Cuadro de precios elementales			
Referencia	**Unidad**	**Descripción**	**Precio(€)**
Materiales			
m1	ud.	OpenMANIPULATOR-X	1489,00
m2	ud.	Arduino UNO	29,04
m3	ud.	DynamixelShield	24,27
m4	ud.	Aukey Webcam FullHD USB	25,47
m5	ud.	UART-TTL USB adaptador	6,29
m6	ud.	Cable USB2.0 TIPO macho A a TIPO B macho	3,96
m7	ud.	Fuente de alimentación	10,09
m8	ud.	Tablero blanco aglomerado 80x39.5x1.6 cm	8,79
m9	ud.	Tablero blanco aglomerado 120x39.5x1.6 cm	11,99
m10	ud.	Listón cepillado abeto 1.8x6.9x90 cm	5,69
m11	ud.	Tornillo plano acero cincado 4x16 mm	0,02
m12	ud.	Tornillo plano acero cincado 4x20 mm	0,09
m13	ud.	Escuadra perforada acero 60x60x2 mm	1,99
m14	ud.	Placa unión acero cincado 135mm	1,39
m15	ud.	Placa unión bricomada 36 mm	0,18
m16	m.	Cinta adhesiva americana de tela	0,23
m17	kg.	Filamento PLA	15,99
m18	ud.	Espuma rígida de poliuretano 50x100x4 cm	19,99
Software			
sw1	licencia mensual	Software de programación MATLAB	75
sw2	licencia mensual	Software de programación Arduino IDE	0
sw3	licencia mensual	Software de programación Wizard 2.0	0
sw4	licencia mensual	Software de documentación Microsoft Office	5,75
sw5	licencia mensual	Software de dibujo SolidWorks	130,83
Equipo			
e1	h.	Portátil HP periféricos incluidos	0,32
e2	h.	Impresora 3D	0,14
e3	h.	Sierra eléctrica	0,09
e4	h.	Fresadora industrial	1,2
e5	h.	Taladro	0,10
M.O.D			
h1	h.	Ingeniero desarrollador	35,00
h2	h.	Técnico	24,00

Tabla 1: Precios elementales.

2.Cuadro de precios descompuestos I					
Referencia	**Unidad**	**Descripción**	**Precio(€)**	**Cantidad**	**Parcial(€)**
d1	ud.	Subsistema mecánico y electrónico en el que se incluyen los materiales empleados para realizar la estructura donde se realizan los experimentos y mano de obra incluida.			
Materiales					
m1	ud.	OpenMANIPULATOR-X	1489	1	1489
m2	ud.	Arduino UNO	29,04	1	29,04
m3	ud.	DynamixelShield	24,27	1	24,27
m4	ud.	Aukey Webcam FullHD USB	25,47	1	25,47
m5	ud.	UART-TTL USB adaptador	6,29	1	6,29
m6	ud.	Cable USB2.0 TIPO macho A a TIPO B macho	3,96	1	3,96
m7	ud.	Fuente de alimentación	10,09	1	10,09
m8	ud.	Tablero blanco aglomerado 80x39.5x1.6 cm	8,79	2	17,58
m9	ud.	Tablero blanco aglomerado 120x39.5x1.6 cm	11,99	1	11,99
m10	ud.	Listón cepillado abeto 1.8x6.9x90 cm	5,69	2	11,38
m11	ud.	Tornillo plano acero cincado 4x16 mm	0,02	64	1,28
m12	ud.	Tornillo plano acero cincado 4x20 mm	0,09	10	0,9
m13	ud.	Escuadra perforada acero 60x60x2 mm	1,99	4	7,96
m14	ud.	Placa unión acero cincado 135mm	1,39	4	5,56
m15	ud.	Placa unión bricomada 36 mm	0,18	8	1,44
m16	m.	Cinta adhesiva americana de tela	0,23	0,5	0,12
Equipo					
e3	h.	Sierra eléctrica	0,09	0,5	0,04
e5	h.	Taladro	0,10	1	0,10
M.O.D					
h1	h.	Ingeniero desarrollador	35	15	525
h2	h.	Técnico	24	0,5	12
Medios auxiliares					
	%	Medios auxiliares sobre costes directos	10	2183,46	218,35
				Precio de ejecución material	**2401,81**

Tabla 2: Precios descompuestos I.

2.Cuadro de precios descompuestos II					
Referencia	**Unidad**	**Descripción**	**Precio(€)**	**Cantidad**	**Parcial(€)**
d2	ud.	Subsistema de dibujo (CAD) en el que se desarrolla los planos de la estructura, así como la fabricación e impresión de las piezas.			
Materiales					
m17	kg.	Filamento PLA	15,99	0,2	3,20
m18	ud.	Espuma rígida de poliuretano 50x100x4 cm	19,99	1	19,99
Software					
sw5	licencia mensual	Software de dibujo SolidWorks	130,83	1	130,83
Equipo					
e1	h.	Portátil HP periféricos incluidos	0,32	30	9,6
e2	h.	Impresora 3D	0,14	7	0,98
e4	h.	Fresadora industrial	1,2	10	12
M.O.D					
h1	h.	Ingeniero desarrollador	35	30	1050
Medios auxiliares					
	%	Medios auxiliares sobre costes directos	10	1226,60	122,66
				Precio de ejecución de material	**1349,26**

Tabla 3: Precios descompuestos II.

2.Cuadro de precios descompuestos III					
Referencia	**Unidad**	**Descripción**	**Precio(€)**	**Cantidad**	**Parcial(€)**
d3	ud.	Programación tanto de Arduino UNO, aplicación realizada con MATLAB y redacción de la documentación del proyecto.			
Software					
sw1	licencia mensual	Software de programación MATLAB	75	5	375
sw2	licencia mensual	Software de programación Arduino IDE	0	2	0
sw3	licencia mensual	Software de programación Wizard 2.0	0	1	0
sw4	licencia mensual	Software de documentación Microsoft Office	5,75	2	11,5
Equipo					
e1	h.	Portátil HP periféricos incluidos	0,32	275	88
M.O.D					
h1	h.	Ingeniero desarrollador	35	275	9625
Medios auxiliares					
	%	Medios auxiliares sobre costes directos	10	10099,5	1009,95
				Precio de ejecución de material	**11109,45**

Tabla 4: Precios descompuestos III.

3.Valoración					
Referencia	**Unidad**	**Descripción**	**Precio(€)**	**Cantidad**	**Parcial(€)**
d1	ud.	Subsistema mecánico y electrónico en el que se incluyen los materiales empleados para realizar la estructura donde se realizan los experimentos y mano de obra incluida.	2401,81	1	2401,81
d2	ud.	Subsistema de dibujo (CAD) en el que se desarrolla los planos de la estructura, así como la fabricación e impresión de las piezas.	1349,26	1	1349,26
d3	ud.	Programación tanto de Arduino UNO, aplicación realizada con MATLAB y redacción de la documentación del proyecto.	11109,45	1	11109,45
				Total presupuesto ejecución de material sin impuestos	14860,52
Impuestos					
	%	IVA	21	14860,52	3120,71
				Total presupuesto ejecución	**17981,23**

Tabla 5: Valoración.

El coste total del proyecto asciende a un total de **diecisiete mil novecientos ochenta y un euros con diecinueve céntimos.**

Julio 2024

Antonio López Sanfeliciano

4.-Presupuesto

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Análisis de las técnicas de Optimización Bayesiana en un robot manipulador aplicado a la tarea del minigolf

Trabajo Fin de Máster

Máster Universitario en Ingeniería Mecatrónica

Quinto documento:

Anexos

AUTOR/A: López Sanfeliciano, Antonio

Tutor/a: Armesto Ángel, Leopoldo

CURSO ACADÉMICO:2023/2024

5.-Anexos

Índice de anexos

Índice de figuras

Índice de tablas

1. Programación Arduino UNO

El microcontrolador Arduino UNO es el encargado de ejecutar las funciones del robot, se programa mediante el Arduino IDE, que está basado en C, permite que seamos capaces de elabora funciones personalizadas para realizar operaciones o enviar órdenes y comunicarnos con otros programas como MATLAB.

Primero de todo se han de añadir las librerías correspondientes para emplear este complemento, tanto para comunicarse con el *DynamixelShield* como para poder mover el robot con las funciones correspondientes. Con el IDE abierto, se accede a *Herramientas/Administrar Bibliotecas* en el gestor de librerías, buscamos las dos que se necesitan, para mover los servomotores, en este caso Dynamixel2Arduino, la cual contiene las funciones que se emplean para el protocolo de comunicación del propio robot.

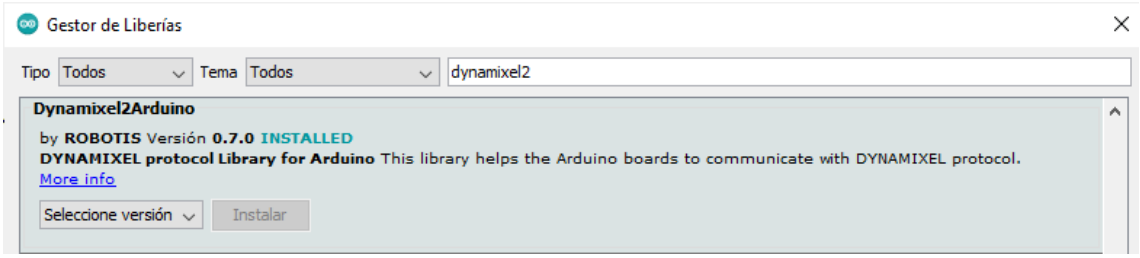


Figura 1: Instalación de la librería Dinamyxel2Arduino. (Fuente: Elaboración propia).

Y para comunicarse con la placa *DynamixelShield*, agregamos su librería, tal y como específica en su hoja de datos.

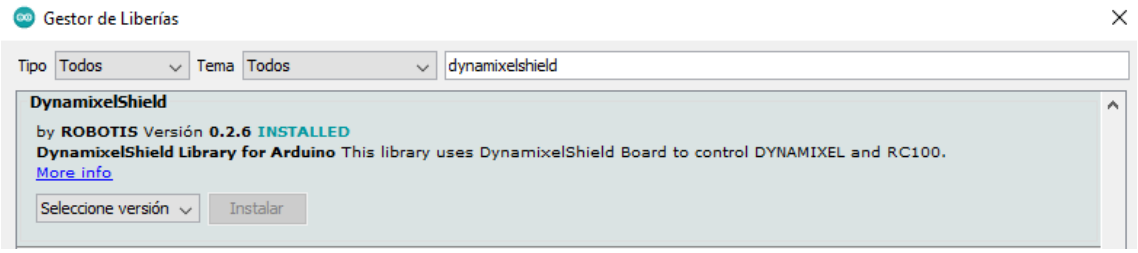


Figura 2: Instalación de la librería DinamyxellShield. (Fuente: Elaboración propia).

1.2 Configuración de los servomotores Dynamixel

La configuración se realizó con la placa OpenRB150 antes de que dejara de funcionar. El primer paso, una vez el robot está montado, es la configuración de cada uno de sus motores con la ayuda de la combinación entre un código de que hay de la librería *Dynamixel2Arduino* aplicado al OpenRB150 llamado *"usb_to_dynamixel.ino"* y el software de *DYNAMIXEL Wizard 2.0,* para establecer los IDs identificativos de cada servo, los cuales van del 11 al 15.

Para empezar, se debe subir el archivo mencionado al microcontrolador, una vez subido se van siguiendo los pasos que nos indica el fabricante [13]. Como el software no detectaba ninguno de los servomotores, hubo que actualizar el firmware de cada uno de ellos, conectando únicamente uno al OpenRB150, posteriormente se selecciona el modelo correspondiente y cuando lo vamos a actualizar el programa indica tal que debemos desconectar el servo y volverlo a conectar. Si se ha realizado correctamente se subirá la actualización y nos dejará ver los parámetros del motor.

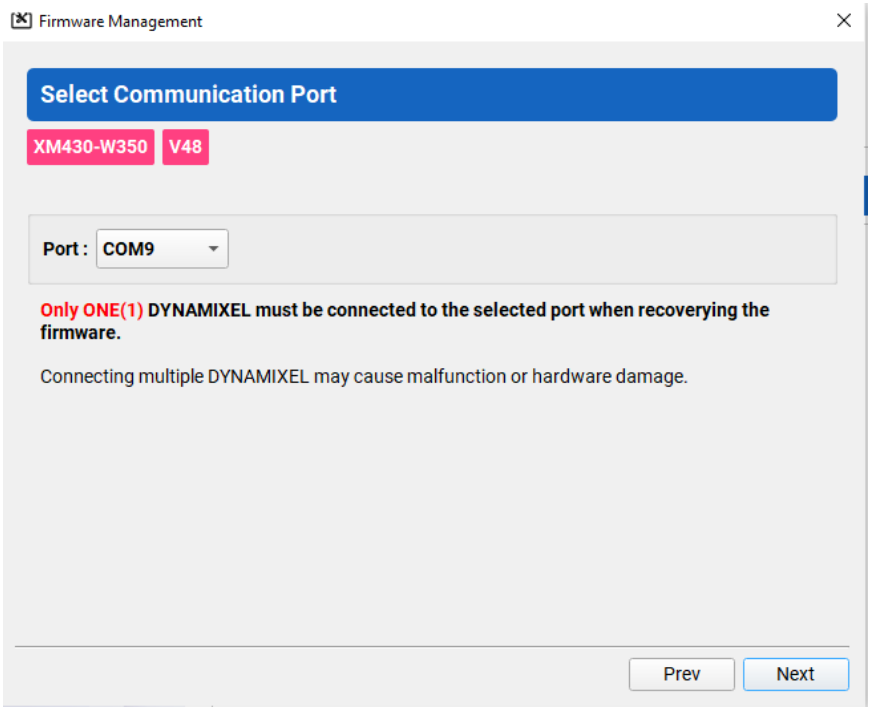


Figura 3: Actualización del firmware de uno de los servormotores a través de Wizard 2.0. (Fuente: Elaboración propia).

Los motores de Dynamixel tienen parámetros que se guardan en la memoria EEPROM, es decir, al quitar la alimentación se queda guardado, como por ejemplo el ID o el modo de control, como se mencionaba en anteriormente se selecciona el control de posición por corriente para no perjudicar al motor si el motor se bloquea debido a un obstáculo y hay una subida en la corriente intentando vencerlo. Otros en cambio, se guardan en la RAM, tales como la velocidad y la aceleración, estos se pueden modificar teniendo el par activo del propio servo.

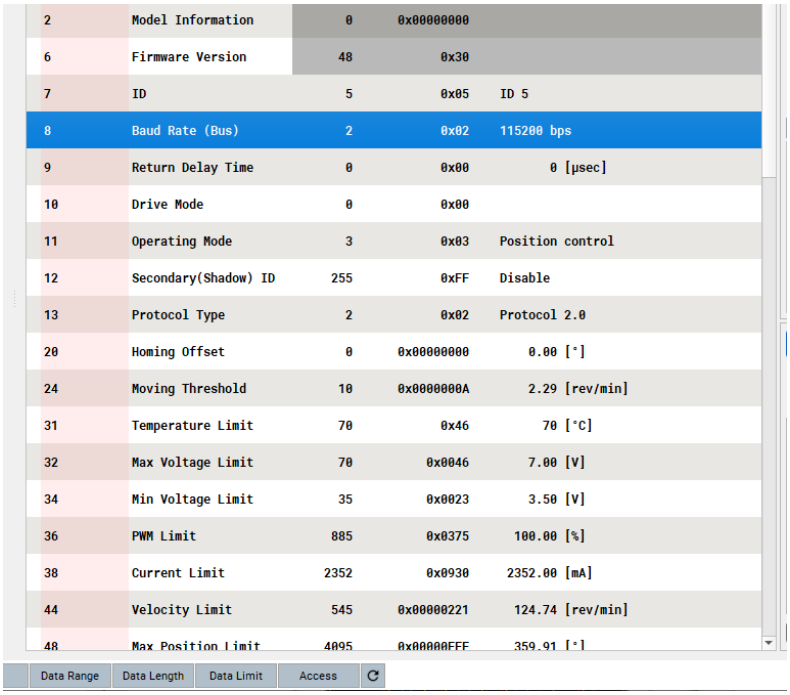


Figura 4: Parámetros de uno de los servormotores a través de Wizard 2.0. (Fuente: Elaboración propia).

Otras configuraciones que se realizan son por ejemplo el perfil de velocidad y aceleración basado en tiempo (ms), esto se le asigna configurándolo desde el propio programa que ejecutamos con un comando más específico que se explicará más detenidamente en el siguiente subapartado**,** que tal y como viene en las especificaciones del fabricante [8], se configura el modo con el que se va mover, en este caso se decide realizarlo por tiempo de trayectoria para tener mayor certeza de en cuanto tiempo tarda en realizar el movimiento.

Velocity-based Profile	Values	Description
Unit	0.229 [rev/min]	Sets velocity of the Profile
Range	0 ~ 32767	'0' represents an infinite velocity

Time-based Profile	Values	Description
Unit	1 [msec]	Sets the time span for the Profile
Range	0 ~ 32737	'0' represents an infinite velocity. Profile Acceleration(108, Acceleration time) will not exceed 50% of Profile Velocity (112, the time span to reach the velocity of the Profile) value.

Figura 5: Diferentes perfiles de velocidad que el fabricante emplea, el que se encuentra en la parte superior corresponde al perfil basado en velocidad y el inferior al que se basado en tiempo. (Fuente: Especificación del fabricante, para el XM430-W350 [8]).

1.3 Formalización previa del programa

Después de realizar la puesta a punto, hay que conocer el funcionamiento de las funciones principales de la librería Dynamixel2Arduino, las cuales se van a explicar a continuación para su entendimiento.

La placa con los motores se comunica a través de un puerto serie que por defecto en los servomotores de la marca trabajan a 57600 baudios. Por tanto, lo primero es configurar para la placa el pin 2 guardado en una variable constante llamada DXL_DIR_PIN. El puerto serie se define en el DXL_SERIAL el cual corresponde al Serial1 (en cada placa se configura de una forma), guardando en esta configuración en la variable dxl:

```
Dynamixel2Arduino dxl(DXL_SERIAL, DXL_DIR_PIN);
```

Como ya se ha comentado anteriormente, queremos que los motores funcionen sincronizados, para ello hay que crear una variable específica:

```
ParamForSyncWriteInst_t sync_write_param;
```

Con esto en el apartado de *setup* de nuestro código (el que sólo se ejecuta una vez) debemos empezar iniciar el monitor serie por el que nos comunicaremos con MATLAB que va a 115200 baudios y el que comunica a los motores, este último se inicializa de la siguiente forma:

```
dxl.begin(57600);
```

El protocolo que emplea los servomotores tiene dos versiones, en nuestro caso para este modelo en concreto es recomendable usar el segundo, por tanto, la función para asignarlo es la que se muestra a continuación:

```
dxl.setPortProtocolVersion(2.0);
```

Para realizar la configuración previa en el *setup* del programa debemos hacer un ping para a los IDs de los motores para corroborar que se encuentran activos y que no hay ningún error de comunicación. Una vez hecho esto, los parámetros que se encuentran en la memoria EEPROM para poder modificarlos, el motor debe estar apagado o lo que es lo mismo con el par desactivado. En consecuencia, ningún tipo de resistencia a ser movido, para ello se desactiva cada uno de los motores para configurar el modo de control (Control de posición basado en corriente) y la corriente máxima (variable *max_current* en el programa) de 2 A, la cual es más que suficiente para la tarea que se realiza.

```
dxl.ping(DXL_ID[i]);

while(!dxl.torqueOff(DXL_ID[i]));

dxl.setOperatingMode(DXL_ID[i],OP_CURRENT_BASED_POSITION);

dxl.setGoalCurrent(DXL_ID[i],max_current,UNIT_MILLI_AMPERE);
```

Por otro lado, se configura el *drive mode* que se explica en el apartado anterior del perfil basado en tiempo con una línea de código aplicando una máscara que vemos a continuación:

```
dxl.writeControlTableItem(DRIVE_MODE,DXL_ID[i],dxl.readControlTableItem(DRIVE_
MODE,DXL_ID[i])|0x04);
```

Y tan solo quedaría establecer los valores del perfil de velocidad y aceleración, estos se almacenan en la RAM, por tanto, pueden ser modificados en cualquier momento, aunque el motor esté encendido y el valor predeterminado asignado es de 5000 ms en el caso de la velocidad y 1500 ms en el de la aceleración.

```
dxl.writeControlTableItem(PROFILE_VELOCITY,DXL_ID[i],speedRobot);


dxl.writeControlTableItem(PROFILE_ACCELERATION,DXL_ID[i],accRobot);
```

Por último, se ha de llevar el robot a la configuración inicial, para ello se establece una posición objetivo a cada motor, en este caso, esto se encuentra definido en un vector llamada POS_INIT, al cual se le pasa a una función creada, posInit, donde se realiza la cinemática directa para realizar la conversión y saber cuál es el punto real. Después se aplica la cinemática inversa con una configuración específica. Por último, se pasan los datos para que puedan ser interpretados por el robot, y con esto, se obtiene qGoal0 con las posiciones de los servos.

```
while(!dxl.torqueOn(DXL_ID[i]));

dxl.setGoalPosition(DXL_ID[i],qGoal0[i]);
```

Por último, se configura la sincronización de los motores y así, más adelante en el bucle principal que todos ellos vayan al mismo tiempo, sin ningún tipo de retraso con las siguientes líneas obtenidas de un ejemplo de la librería y la explicación de un del siguiente recurso multimedia [14].

```
sync_write_param.addr=116;//GOAL_POSITION;

sync_write_param.length=sizeof(int32_t);

sync_write_param.id_count=NUM_SERVOS;
```

for (int i=0;i<NUM_SERVOS;i++)sync_write_param.xel[i].id=DXL_ID[i];

Una vez configurado y sabiendo la posición objetivo hay que llamar a la siguiente función que hará que todos los servos funcionen de forma síncrona:

dxl.syncWrite(sync_write_param);

Con estas funciones específicas se puede mover el robot en una determinada configuración, de forma síncrona para poder realizar nuestra tarea de forma satisfactoria. A continuación, se desarrollarán algunas de las funciones creadas para facilitar el uso del robot.

2. Conexión puerto serie

Es bien sabido que debe haber una comunicación entre el robot, la placa (Arduino UNO) y el ordenador que se esté empleando para poder coordinar todos los movimientos de este. Se realiza mediante un puerto serie, pero además se crea un protocolo de comunicación propio en el que con una trama de datos se obtienen los parámetros necesarios con los que el microcontrolador realiza la función de intérprete para posteriormente ejecutar el movimiento deseado.

En principio la trama debe tener una dirección para referirnos al dispositivo que queremos conectarnos, en este caso, solo tenemos uno, por tanto, no tendrá dirección, dado que ya estaremos conectados al microcontrolador, con el correspondiente comando de MATLAB en que se indica el número de puerto, los baudios y el fin de línea, en nuestro caso es el retorno de carro o CR.

Direccionamiento	**Paquete de datos**	**Checksum**	**CR**
(No especificado)	**(Separados por comas)**		**(\r)**

Podemos observar en la función *lanzamiento.m* del apartado 2.3.1 de los anexos, junto con la función para configurar y conectarse llamada *conexionPuertoserie.m,* que los datos que se envían a través este son las siguientes: posición en coordenadas cartesianas juntos con el ángulo de inclinación, velocidad y aceleración en milisegundos, posición del codo (arriba o abajo), se trata de una variable de tipo booleana al igual que el estado de la pinza. Finalmente se calcula el *checksum*, como su nombre indica es una forma de comprobación mediante suma de los datos, en este caso las variables numéricas, como lo son la posición, velocidad y aceleración. El Arduino UNO, leerá los datos y calculará también el *checksum* para comprobar que los datos son correctos, si este cálculo diese un número diferente al recibido por la trama se enviaría al robot a la posición inicial, a la espera de recibir un nuevo dato, tal y como se ha mencionado en la función desarrollada llamada *leerPuertoSerie*.

3. Simulación de la optimización en CoppeliaSim

La parte de la simulación ha sido realizada por Hannes Deruytter [25], un alumno de intercambio que ha realizado su tesina de máster en la UPV. Él se encargó de diseñar

la plataforma de espuma y realizar la optimización Bayesiana en MATLAB combinado al software de simulación CoppeliaSim.

El diseño de la superficie/plataforma fue sometido a diferentes pruebas, como se puede apreciar, se pretendía que tuviera una forma bastante irregular y el agujero se encontrara al final de ésta, tal y como se muestra en la siguiente figura donde se aprecia la superficie junto al OpenMANIPULATOR-X importado. este último. desde un archivo STL:

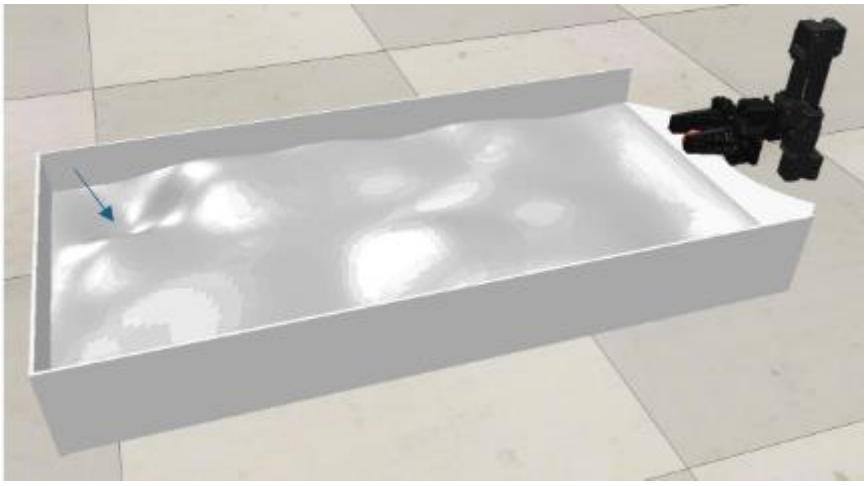


Figura 6: Entorno de prueba de la simulación. (Fuente: Master thesis by Hannes Deruytter [25]).

Los resultados no fueron los esperados, por eso se cambió el diseño a uno más sencillo, tras diferentes ensayos, se llegó a la conclusión que se podría realizar los experimentos con la herramienta de la optimización bayesiana. En conclusión, el entorno final desarrollado en su simulación se muestra a continuación:

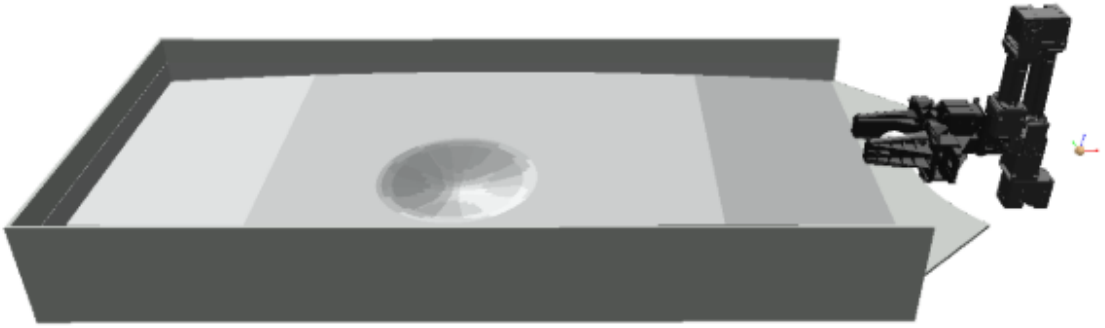


Figura 7: Entorno de simulación final. (Fuente: Master thesis by Hannes Deruytter [25]).

Se inicia el análisis establecido la función objetivo que se quiere optimizar:

$$J = \alpha \cdot {dist_{fin}}^2 + \beta \cdot \sum_{i=1}^{n} {dist_{tray}}_i^2$$

Donde α es la variable que asigna un peso al valor de $dist_{fin}$ la cual es donde la pelota se ha parado, mientras que β asigna el peso al sumatorio de la distancia de la trayectoria $dist_{tray}$. Esta función refleja es cómo de lejos se ha quedado del objetivo (el hoyo). Cuanto menor sea el valor obtenido significará que los hiperparámetros que se han ejecutado en la predicción cada vez son más precisos. Los datos que se obtienen de la trayectoria es un vector de cincuenta puntos, aplicado también al experimento realizado en la realidad.

Algunos de los experimentos realizados, empleando la función de adquisición de *expected improvement* con diferentes ratios de exploración. Se trata de un parámetro que cuanto mayor es el valor, el robot realizará una mayor exploración, es decir. intentará buscar por otras zonas en las que probablemente no se encuentre el mínimo de nuestra función objetivo. En la simulación se prueba desde 0.1 hasta 1.5 mediante diferentes intervalos, realizando 150 ensayos en cada uno de ellos.

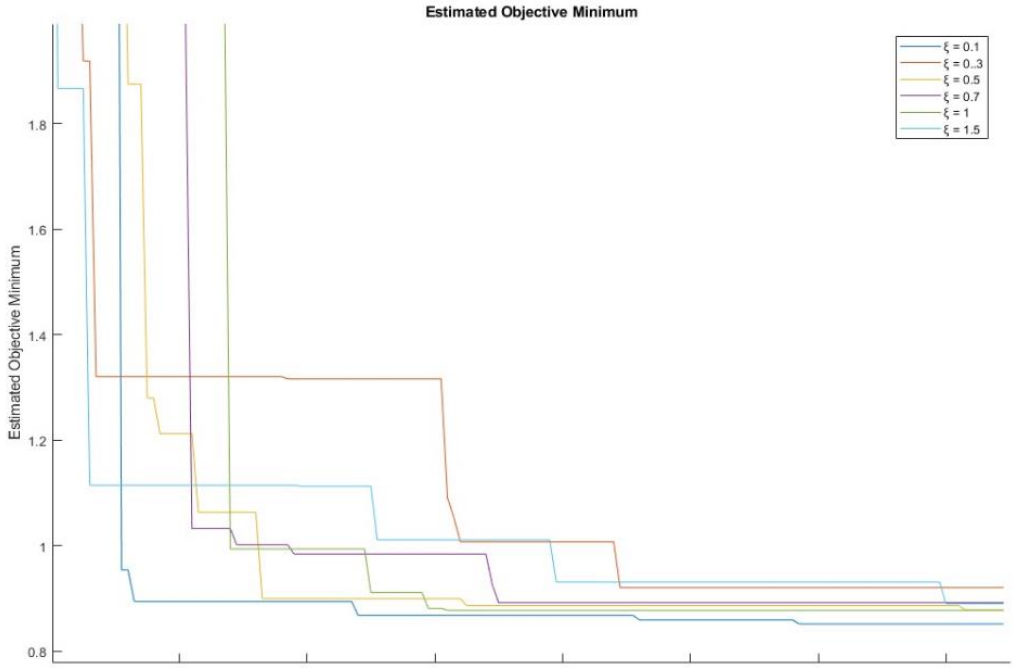


Figura 8: Simulación de la optimización con EI mediante diferentes ratios de exploración. (Fuente: Master thesis by Hannes Deruytter [25]).

Los resultados efectuados muestran que el mejor resultado obtenido (el más bajo) es el de ratio de exploración de 0.1, con una solución de J=0.8519. Mientras que experimentos como el de ratio 0.3 ni tan si quiera llega al valor mínimo bajar de un cierto valor propuesto, en este caso 0.9. En la siguiente tabla se muestran los datos de entrada con la solución y en que iteración realiza el experimento con menor puntuación.

ξ	Iteraciones hasta J menor de 0.9	Mejor ángulo (º)	Mejor velocidad (m/s)	Mejor solución J
0.1	14	-0.010	0.958	0.851
0.3	+150	-0.009	0.799	0.921
0.5	32	-0.008	0.908	0.878
0.7	71	0.137	1.094	0.892
1	60	0.117	1.041	0.878
1.5	141	0.014	1.063	0.890

Tabla 1: Resultados de los mejores experimentos para cada uno de los diferes ratios de exploración. (Fuente: Fuente: Master thesis by Hannes Deruytter [25]).

A continuación, se expone el modelo obtenido tras realizar la simulación con EI y ratio de exploración de 0.1, donde claramente se puede apreciar que para las velocidades y ángulos cercanos al mejor consiguen se obtienen valores muy similares de la misma magnitud, haciendo ver que claramente ahí se encuentra la solución, es decir, acertar la pelota en el hoyo objetivo.

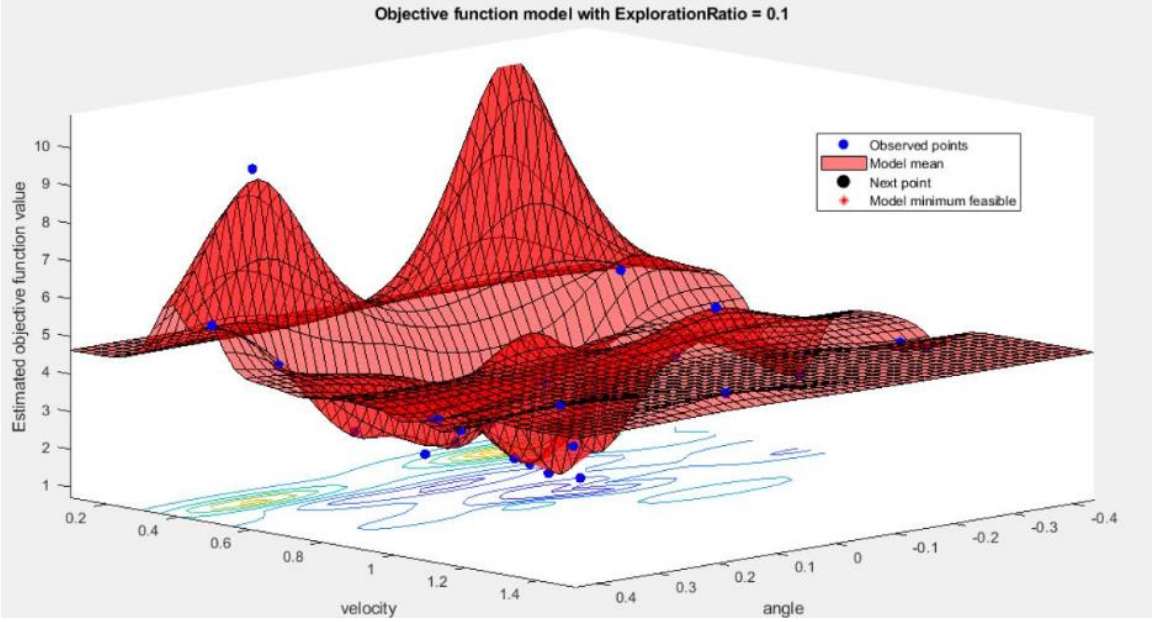


Figura 9: Modelo obtenido tras la simulación de la optimización con EI ratio de exploración. (Fuente: Master thesis by Hannes Deruytter [25]).

Realizó más experimentos y comparaciones, con diferentes funciones de adquisición, kernels y demás, no obstante, para este proyecto valoramos la idea general de la simulación realizada que sirve de base para comparar medianamente y saber el punto de inicio en el que se desarrolla el proyecto [25].

4. Código Arduino

```
#include <Dynamixel2Arduino.h>
#include <SoftwareSerial.h>//Se incluyen las librerías
específicas
SoftwareSerial soft_serial(7, 8); // DYNAMIXELShield
UART RX/TX
#define DXL_SERIAL Serial//Definiciones
#define DEBUG_SERIAL soft_serial
#define PINZA_ABIERTA 1150 //1081 abierto total
#define PINZA_CERRADA 2625
#define CODO_ARRIBA true
#define CODO_ABAJO false
#define POS_INIT {0.0 * PI / 180.0, 90.0 * PI / 180.0,
0.0 * PI / 180.0, 0.0 *
PI / 180.0}
const int DXL_DIR_PIN = 2; // DYNAMIXEL Shield DIR PIN
Dynamixel2Arduino dxl(DXL_SERIAL, DXL_DIR_PIN);
typedef struct //Declaraciones de estructuras
{
double x;
double y;
double z;
double qt;
```

```
} robotPosition;
typedef struct
{
double l1;
double l2;
double l21;
double l22;
double l3;
double l4;
} robotParams; // Definimos la estructura de los
parametros
typedef struct
{
double q1;
double q2;
double q3;
double q4;
} robotJoints;
const int NUM_SERVOS = 5;//Definición de constantes y
variables
const uint8_t DXL_ID[NUM_SERVOS] = {11, 12, 13, 14, 15};
const robotParams parameters = {0.077, 0.130, 0.128,
0.024, 0.124, 0.126};
unsigned long _currentTime;
unsigned long _lastUpdate;
robotJoints joints = {0.0 , 0.0 , 0.0 , 0.0 };
robotJoints q1 = {0.0 , 0.0 , 0.0 , 0.0 };
int32_tq Goal0[NUM_SERVOS]={joints.q1*4095/(2*PI),(-
joints.
q2+(3*PI/2))*4095/(2*PI),(-joints.q3+PI)*4095/(2*PI),(-
joints.q4+PI)*4095/(2*PI),
1081};
unsigned int speedRobot = 5000;//en ms el tiempo que
tarda en llegar
unsigned int accRobot = 1500;//en ms el tiempo que tarda
en acelerar
bool estadoCodo = true;
bool estadoPinza = true;
double max_current = 2000; //Corriente máxima en mA
ParamForSyncWriteInst_t sync_write_param;
using namespace ControlTableItem;
robotPosition directKinematics(const robotJoints
&joints, const robotParams
&params);//Definición de las funciones
robotJoints inverseKinematics(const robotPosition
&target, const robotParams
&params,const bool estadoCodo);
void calcula_qGoal(const robotJoints &joints,int32_t
qGoal[NUM_SERVOS],int
estado_pinza);
```

```
robotPosition leerPuertoSerie(unsigned int
&speedRobot,unsigned int &accRobot,
bool &estadoCodo, bool &estadoPinza);
void setup() {
DEBUG_SERIAL.begin(115200);
while(!DEBUG_SERIAL);
// Baudios para comunicarse con los motores de Dynamixel
dxl.begin(57600);
delay(100); // Tiempo de espera para que pueda arrancar
// Potocolo que emplea "2.0"
dxl.setPortProtocolVersion(2.0);
robotPosition target = posInit();//Establece los
posición del robot en la
inicial
joints = inverseKinematics(target,
parameters,CODO_ABAJO);//Clacula la
cinematica inversa para la posición inicial
calcula_qGoal(joints,qGoal0,PINZA_ABIERTA);//Pasa de
radianes a parámetros
que van de 0 a 4095
delay(200);
for (int i=0;i<NUM_SERVOS;i++)//Bucle for donde se
inicializa la
configuración de cada uno de los servos
{
while(!dxl.torqueOff(DXL_ID[i]));
dxl.setOperatingMode(DXL_ID[i],OP_CURRENT_BASED_POSITION
);
dxl.writeControlTableItem(DRIVE_MODE,DXL_ID[i],dxl.
readControlTableItem(DRIVE_MODE,DXL_ID[i])|0x04);
dxl.writeControlTableItem(PROFILE_VELOCITY,DXL_ID[i],spe
edRobot);
dxl.writeControlTableItem(PROFILE_ACCELERATION,DXL_ID[i]
,accRobot);
dxl.setGoalCurrent(DXL_ID[i],max_current,UNIT_MILLI_AMPE
RE);
while(!dxl.torqueOn(DXL_ID[i]));
dxl.setGoalPosition(DXL_ID[i],qGoal0[i]);
}
delay(5000);
sync_write_param.addr=116;//Estas líneas configuran para
el funcionamiento de
forma síncrona
sync_write_param.length=sizeof(int32_t);
sync_write_param.id_count=NUM_SERVOS;
for (int
i=0;i<NUM_SERVOS;i++)sync_write_param.xel[i].id=DXL_ID[i
];
_lastUpdate=millis();
delay(5000);
}
```

```
void loop() {
if(DEBUG_SERIAL.available() > 0){//Detecta si hay algo
en el puerto serie
robotPosition target1 =
leerPuertoSerie(speedRobot,accRobot,estadoCodo,
estadoPinza);//Lee el puerto serie
q1 = inverseKinematics(target1, parameters,
estadoCodo);//Calcula la cinematica
inversa
calcula_qGoal(q1,qGoal0,PINZA_ABIERTA);//Pasa de
radianes a parámetros que van
de 0 a 4095
for (int i=0;i<NUM_SERVOS;i++){
dxl.writeControlTableItem(PROFILE_VELOCITY,DXL_ID[i],spe
edRobot); //Actualiza
el valor del tiempo trayectoria de la velocidad
dxl.writeControlTableItem(PROFILE_ACCELERATION,DXL_ID[i]
,accRobot);
//Actualiza el valor del tiempo trayectoria de la
aceleración
sync_write_param.xel[i].id=DXL_ID[i];
memcpy(sync_write_param.xel[i].data,&qGoal0[i],sizeof(in
t32_t));
}
dxl.syncWrite(sync_write_param);//Mueve los motores de
forma síncrona
delay(200);
}
}
//Funciones principales
robotPosition directKinematics(const robotJoints
&joints, const robotParams
&params)//Cálculo de la cinemática directa
{
robotPosition target;
double r;
double q22;
double alpha3;
alpha3 = atan2(params.l22, params.l21);
q22 = (PI / 2) - joints.q2;
r = (params.l21 * sin(q22) + params.l22 * cos(q22)) +
params.l3 * cos(-q22 +
joints.q3) + params.l4 * cos(-q22 + joints.q3 +
joints.q4);
target.x = r * cos(joints.q1);
target.y = r * sin(joints.q1);
target.z = params.l1 + params.l21 * sin(joints.q2) +
params.l22*sin(-q22)+
params.l3 * sin(joints.q3-q22) + params.l4 *
sin(joints.q4 + joints.q3-q22);
target.qt = joints.q3 + joints.q4-q22;
```

```
return target;
}
robotJoints inverseKinematics(const robotPosition
&target, const robotParams
&params,const bool estadoCodo)//Cálculo de la cinemática
indirecta
{
robotJoints joints;
double Pmx,Pmy,Pmz;
double r,h,s;
double alpha1,alpha2,alpha3;
double beta1,beta2;
//Calculo punto de muñeca
joints.q1=atan2(target.y,target.x);
Pmx = target.x-params.l4*cos(joints.q1)*cos(target.qt);
Pmy = target.y-params.l4*sin(joints.q1)*cos(target.qt);
Pmz = target.z-params.l4*sin(target.qt);
//Calculo de distancias
h = Pmz - params.l1;
r=sqrt(pow(Pmx,2)+pow(Pmy,2));
s=sqrt(pow(r,2)+pow(h,2));
//Angulos auxiliares
alpha1=atan2(h,r);
alpha2=acos((-
pow(params.l3,2)+pow(s,2)+pow(params.l2,2))/(2*s*params.
l2));
alpha3=atan2(params.l22,params.l21);
beta1=atan2(params.l21,params.l22);
beta2=acos((-
pow(s,2)+pow(params.l3,2)+pow(params.l2,2))/(2*params.l3
*params.
l2));
//Calculo q
if( estadoCodo == true ){//Condición codo
joints.q2=alpha1-alpha2+alpha3;
joints.q3=((2*PI-beta2)+beta1)-PI;
}
else{
joints.q2=alpha1+alpha2+alpha3;
joints.q3=(beta1+beta2-PI);
}
joints.q4=target.qt-joints.q3+(PI/2-joints.q2);
return joints;
}
void calcula_qGoal(const robotJoints &joints,int32_t
qGoal[NUM_SERVOS],int
estado_pinza)//Pasa los radianes a valores entre 0 y
4095, lo que interpreta el
el servomotor
{
qGoal[0]=round(joints.q1*4095/(2*PI));
```

```
qGoal[1]=round((-joints.q2+(3*PI/2))*4095/(2*PI));
qGoal[2]=round((-joints.q3+PI)*4095/(2*PI));
qGoal[3]=round((-joints.q4+PI)*4095/(2*PI));
qGoal[4]=estado_pinza;
}
robotPosition leerPuertoSerie(unsigned int
&speedRobot,unsigned int &accRobot,
bool &estadoCodo, bool &estadoPinza){//Se encaraga de
leer la trama y separar los
datos
robotPosition target;
int posComa;
String datos[8] =
{"0","0","0","0","0","0","0","0"},trama = "";
int posPuntoyComa;
double checksumCalculado,checksumRecibido;
double tolerancia = 0.1;
trama = DEBUG_SERIAL.readStringUntil('\r');//Lee hasta
llegar el retono de
carro
if(trama != "" )//Condición de trama que no este vacía
{
for (int i = 0; i < 7; i++) {//Lectura de los datos
seprados mediante ","
posComa = trama.indexOf(',');
if (posComa != -1) {
datos[i] = trama.substring(0, posComa );
trama.remove(0, posComa + 1);
}
}
posPuntoyComa = trama.indexOf(';');
datos[7]=trama.substring(0, posPuntoyComa );//Lectura
último dato
trama.remove(0, posPuntoyComa + 1);
checksumRecibido=trama.toDouble();//Lectura del checksum
recibido
for (int i = 0; i < 6; i++) {
checksumCalculado += datos[i].toDouble();//Cálculo del
checksum, a partir
de los datos
}
if(abs(checksumRecibido) - abs(checksumCalculado) <
tolerancia)//Comprobación
de que el checksum calculado y recibido son iguales con
una cierta tolerancia de
cálculo debido a los decimales
{
target.x = datos[0].toDouble();//Guarada los datos
recibidos en su lugar
correspondiente
target.y = datos[1].toDouble();
```

```
target.z = datos[2].toDouble();
target.qt = datos[3].toDouble();
speedRobot = datos[4].toInt();
accRobot = datos[5].toInt();
if(datos[6] == "true") estadoCodo = CODO_ARRIBA;
else estadoCodo = CODO_ABAJO;
if(datos[7] == "true") estadoPinza = PINZA_ABIERTA;
else estadoPinza = PINZA_CERRADA;
return target;//Devuelve la posición que recibida
}
else{
return posInit();//Si el checksum no es coincidente, se
vuelve a la posición
inicial
}
}
}
robotPosition posInit(void)//Función para la posición
inicial, calcula su
cinemática directa
{
robotJoints q = POS_INIT;
robotPosition target = directKinematics(q, parameters);
return target;
}
```

5. Código MATLAB

5.1 Vision

5.1.1 visionCapture.m

```
function [X,Y] = visionCapture

tvec=[396.5078;-193.6990;160.1134];%Vectroeres de traslación y rotación
(extrínsecos)
rvec=[-0.0202;3.1399;-0.0023];
camIntrinsicsK =
[113.537587310038,0,308.357053124234;0,113.583810409203,246.707770402621;0,0,
1];%Matriz intrínseca

% Conectar a una cámara
camera = cv.VideoCapture(0); % Cambia el número según la cámara que estés
utilizando % Cambia el número según la cámara que estés utilizando
camera.set('FrameWidth', 640);%Cambio de resolución de la cámara
camera.set('FrameHeight', 480);
camera.set('FPS', 30);%FPS con los que captura

XYZ=[0;0;0];%Inicialización de variables
iteracion=1;
X=zeros(50, 1);
Y=zeros(50, 1);
u=0;
```

```
v=0;
area=0;


% Definir rango de color amarillo
yellowBajo = [20, 100, 100];
yellowAlto = [30, 255, 255];


while camera.isOpened()%Bucle enecendiendo la cámara

    frame = camera.read;%Captura de la imágen y guardado en un variable
llamada frame
    frameHSV = cv.cvtColor(frame, 'RGB2HSV');%Pasa el Frame a HSV

    maskYellow = cv.inRange(frameHSV, yellowBajo, yellowAlto);%Creacion de
mácasra con el frame y el rango del color amarillo
    maskYellowvis = cv.bitwise_and(frame, frame, 'Mask',
maskYellow);%Operando AND, aplicando la máscara
    maskYellowvis_gray = cv.cvtColor(maskYellowvis, 'RGB2GRAY');%Se pasa a
escala de grises
    contornosR = cv.findContours(maskYellowvis_gray, 'Mode', 'External',
'Method', 'Simple');%Se busca el numero de contornos
    %bucle para el amarillo
    for index = 1:numel(contornosR)

        contour = contornosR{index};
        area = cv.contourArea(contour);%Calculo del area del contorno
        if area > 10

        M = cv.moments(contour); %Calculo del centroide
        if M.m00 == 0 % Condicion de si es cero ponerlo a 1 para que no de
error
           M.m00= 1;
        end
        u = int32(M.m10/M.m00);% Posición del centroide en píxeles
        v = int32(M.m01/M.m00);



 [XYZ] = calculaCoordenadasMreal(u,v,rvec,tvec,camIntrinsicsK); %Conversión
de píxeles a coordenadas reales


        end
    end

    if  area > 10
    X(iteracion,1) = XYZ(1)/1000;%Lo pasamos a metros y guaradamos en un
vector
    Y(iteracion,1) = XYZ(2)/1000;
    iteracion=iteracion+1;
    pause(0.05);
    end
    if iteracion > 50%Cuando se capturen 50 coordenadas se para la función
        break;
    end

end
```

```
end
```

5.1.2 camera_detectaAmarillo.m

```
clear all;
close all;

tvec=[396.5078
 -193.6990
  160.1134];%Datos extrínsecos
rvec=[  -0.0202
    3.1399
   -0.0023];
camIntrinsicsK =   [113.537587310038   0     308.357053124234
0     113.583810409203    246.707770402621
0     0     1]; %Datos intrínsecos

XYZ=[0;0;0];
% Conectar a una cámara
camera = cv.VideoCapture(0); % Cambia el número según la cámara que estés
utilizando % Cambia el número según la cámara que estés utilizando
camera.set('FrameWidth', 640);
camera.set('FrameHeight', 480);
camera.set('FPS', 60);
iteracion=1;
u=0;
v=0;
pause(2);

% Definir rango de color amarillo
yellowBajo = [20, 100, 100];
yellowAlto = [30, 255, 255];



while camera.isOpened()
    frame = camera.read;

    frameHSV = cv.cvtColor(frame, 'RGB2HSV');
    maskYellow = cv.inRange(frameHSV, yellowBajo, yellowAlto);
    maskYellowvis = cv.bitwise_and(frame, frame, 'Mask', maskYellow);
    maskYellowvis_gray = cv.cvtColor(maskYellowvis, 'RGB2GRAY');%Comprobars
si es HSV2GRAY
    contornosR = cv.findContours(maskYellowvis_gray, 'Mode', 'External',
'Method', 'Simple');
    %bucle para el amarillo
    for index = 1:numel(contornosR)
        contour = contornosR{index};
        area = cv.contourArea(contour);
        if area > 10

        M = cv.moments(contour);
        if M.m00 == 0
           M.m00= 1;
        end
        u = int32(M.m10/M.m00);
        v = int32(M.m01/M.m00);
```

```
        frame = cv.circle(frame,[u,v],7,'Color',[0,255,0],'Thickness',-1);

        [XYZ] = calculaCoordenadasMreal(u,v,rvec,tvec,camIntrinsicsK);
         x_str = num2str(XYZ(1));
         y_str = num2str(XYZ(2));

        % Concatenar las cadenas de caracteres
        infoString = ['{' x_str '}{' y_str '}'];
        nuevoContorno = cv.convexHull(contour);
        frame =
cv.putText(frame,infoString,[u+10,v],'FontFace','HersheySimplex','FontScale',
0.75, 'Color', [0,255,0],'LineType','AA');
        frame = cv.drawContours(frame, {nuevoContorno}, 'Color', [255, 0, 0],
'Thickness', 3);

        end
    end
    imshow(frame);
    f = gcf; % Obtener la figura actual
    % Obtener los valores deseados
    X(iteracion) = XYZ(1);
    Y(iteracion) = XYZ(2);
    iteracion=iteracion+1;

    if f.CurrentCharacter > 0 %Salir se presiona cualquier tecla
          break;
    end

end
```

5.1.3 calibrarReferencia.m

```
clear all;
close all;
camera = cv.VideoCapture(0); % Cambia el número según la cámara que estés
utilizando % Cambia el número según la cámara que estés utilizando
    camera.set('FrameWidth', 640);
    camera.set('FrameHeight', 480);

    camIntrinsicsK =   [113.537587310038     0      308.357053124234
0      113.583810409203    246.707770402621
0      0      1];
    coefDistorsion=[0.0104;-0.0010;0.0002;-0.0012];
uc=0;
vc=0;
uv=0;
vv=0;
um=0;
vm=0;
urs=0;
vrs=0;
pause(2);
% Definir coordenadas de los circulos de referencia en el mundo real
centrosMreal={[785,16,0],[17,379,0],[780,385,0],[17,16,0]};
% Definir rango de color Cyan
cyanBajo = [85, 50, 50];
```

```
cyanAlto = [120, 255, 255];

% Definir rango de color Verde
verdeBajo = [40, 50, 50];
verdeAlto = [85, 255, 255];
% Definir rango de color Rosa
rosaBajo = [150, 50, 50];
rosaAlto = [170, 255, 255];
% Definir rango de color Naranja
marronBajo = [7, 150, 150];
marronAlto = [30, 255, 255];

while camera.isOpened()
    frame = camera.read;
    frameHSV = cv.cvtColor(frame, 'RGB2HSV');
     %mascara y contorno Cyan
    maskCyan = cv.inRange(frameHSV, cyanBajo, cyanAlto);
    maskCyanvis = cv.bitwise_and(frame, frame, 'Mask', maskCyan);
    maskCyanvis_gray = cv.cvtColor(maskCyanvis, 'RGB2GRAY');
    contornosCyan = cv.findContours(maskCyanvis_gray, 'Mode', 'External',
'Method', 'Simple');
     %mascara y contorno Rosa
    maskRosa = cv.inRange(frameHSV, rosaBajo, rosaAlto);
    maskRosavis = cv.bitwise_and(frame, frame, 'Mask', maskRosa);
    maskRosavis_gray = cv.cvtColor(maskRosavis, 'RGB2GRAY');
    contornosRosa = cv.findContours(maskRosavis_gray, 'Mode', 'External',
'Method', 'Simple');
    %mascara y contorno Verde
    maskVerde = cv.inRange(frameHSV, verdeBajo, verdeAlto);
    maskVerdevis = cv.bitwise_and(frame, frame, 'Mask', maskVerde);
    maskVerdevis_gray = cv.cvtColor(maskVerdevis, 'RGB2GRAY');
    contornosVerde = cv.findContours(maskVerdevis_gray, 'Mode', 'External',
'Method', 'Simple');
    %mascara y contorno Naranja
    maskMarron = cv.inRange(frameHSV, marronBajo, marronAlto);
    maskMarronvis = cv.bitwise_and(frame, frame, 'Mask', maskMarron);
    maskMarronvis_gray = cv.cvtColor(maskMarronvis, 'RGB2GRAY');
    contornosMarron = cv.findContours(maskMarronvis_gray, 'Mode', 'External',
'Method', 'Simple');
        %bucle para el Cyan
     for index = 1:numel(contornosCyan)
        contourC = contornosCyan{index};
        areaC = cv.contourArea(contourC);
        if areaC > 10

        Mc = cv.moments(contourC);
        if Mc.m00 == 0
           Mc.m00= 1;
        end
        uc = int32(Mc.m10/Mc.m00);
        vc = int32(Mc.m01/Mc.m00);

        frame = cv.circle(frame,[uc,vc],7,'Color',[0,255,0],'Thickness',-1);

        nuevoContornoC = cv.convexHull(contourC);

        frame = cv.drawContours(frame, {nuevoContornoC}, 'Color', [0, 0,
255], 'Thickness', 3);
```

```
        end
    end

    %bucle para el Rosa
    for index = 1:numel(contornosRosa)
        contourRs = contornosRosa{index};
        areaRs = cv.contourArea(contourRs);
        if areaRs > 4

        Mrs = cv.moments(contourRs);
        if Mrs.m00 == 0
            Mrs.m00= 1;
        end
        urs = int32(Mrs.m10/Mrs.m00);
        vrs = int32(Mrs.m01/Mrs.m00);

        nuevoContornoRs = cv.convexHull(contourRs);
        % frame =
cv.putText(frame,infoString,[uc+10,vc],'FontFace','HersheySimplex','FontScale
', 0.75, 'Color', [0,255,0],'LineType','AA');
        frame = cv.drawContours(frame, {nuevoContornoRs}, 'Color', [128, 0,
128], 'Thickness', 3);

        end
    end
    %bucle para el Verde
    for index = 1:numel(contornosVerde)
        contourV = contornosVerde{index};
        areaV = cv.contourArea(contourV);
        if areaV > 1

        Mv = cv.moments(contourV);
        if Mv.m00 == 0
            Mv.m00= 1;
        end
        uv = int32(Mv.m10/Mv.m00);
        vv = int32(Mv.m01/Mv.m00);

        frame = cv.circle(frame,[uv,vv],7,'Color',[0,255,0],'Thickness',-1);
        nuevoContornoV = cv.convexHull(contourV);
        frame = cv.drawContours(frame, {nuevoContornoV}, 'Color', [0, 255,
0], 'Thickness', 3);

        end
    end
     %bucle para el Naranja
    for index = 1:numel(contornosMarron)
        contourM = contornosMarron{index};
        areaM = cv.contourArea(contourM);
        if areaM > 1

        Mm = cv.moments(contourM);
        if Mm.m00 == 0
            Mm.m00= 1;
        end
        um = int32(Mm.m10/Mm.m00);
        vm = int32(Mm.m01/Mm.m00);
        frame = cv.circle(frame,[um,vm],7,'Color',[0,255,0],'Thickness',-1);
```

```
        nuevoContornoM = cv.convexHull(contourM);

        frame = cv.drawContours(frame, {nuevoContornoM}, 'Color', [255, 165,
0], 'Thickness', 3);

        end
     end
     %Resultados
     centrosPvirtual = {[uc,vc],[uv,vv],[um,vm],[urs,vrs]}
     [rvec, tvec, success] = cv.solvePnP(centrosMreal,centrosPvirtual,
camIntrinsicsK,'DistCoeffs',coefDistorsion);

    imshow(frame);
    f = gcf; % Obtener la figura actual


    if f.CurrentCharacter > 0
          break;
    end

end
```

5.1.4 calculocoordenadasMreal.m

```
function [XYZ] = calculaCoordenadasMreal(u,v,rvec,tvec,camIntrinsicsK)

uv = [u;v;1];%Se ponen los datos en una matriz
R = cv.Rodrigues(rvec);%Calculo de la matriz de rotación
Vtri = inv(R)*(inv(camIntrinsicsK)*double(uv));%
Vbi = inv(R)*tvec;
s = 0 + Vbi(3) / Vtri(3);%Factor de escala
XYZ = inv(R)*( inv(camIntrinsicsK)*double(uv)*s-tvec);%Calculo coordenadas
reales

end
```

5.2 Mallado

5.2.1 dibujarPlots_v2.m

```
load ('griddatafindef2.mat');%Carga los datos del mapeado
%script para dibujar y analizar los datos del mapeado
goal=[X_final Y_final];
tolerance=0.05;


data.points=all_inputs;
data.trajectories=cell(size(all_inputs,1),1);
data.succeed=[];
data.failed=[];
data.Jsucceed=[];
data.Jfailed=[];
close all;
```

```
for i=1:size(all_inputs,1)
    data.trajectories{i}=[Resultados.X{i} Resultados.Y{i}];

    if (isGoalReached(data.trajectories{i},goal,tolerance)==1)
        data.succeed=[data.succeed;i all_inputs(i,:)];
    else
        data.failed=[data.failed;i all_inputs(i,:)];
    end
end

figure;
plot(data.succeed(:,2),data.succeed(:,3),'bx');
hold on;
plot(data.failed(:,2),data.failed(:,3),'ro');
hold off;
title('Mapeado 20x20 aciertos(x)/fallos(o)');
xlabel('Angulo_q1(°)');
ylabel('Tiempo del movimiento (ms)');

figure;
hold on;
lg=cell(length(data.succeed),1);
for i=1:length(data.succeed)
    idx=data.succeed(i,1);
    traj=data.trajectories{idx};
    plot(traj(:,1),traj(:,2));
    x=[traj(1,1)-0.2 traj(1,1)];
    y=[traj(1,2)+0.2 traj(1,2)];
    lg{i}=sprintf('(%.1d,%.1d)',data.points(idx,1),data.points(idx,2));

%a=annotation('textarrow',x,y,'String',sprintf('(%.1d,%.1d)',data.points(idx,
1),data.points(idx,2)));
end
hold off;
xlim([0 0.8]);
ylim([0 0.4]);
title('Trayectorias cumplidas del objetivo en el mapeado');
xlabel('Eje X(m)');
ylabel('Eje Y(m)');
%axis equal;

legend(lg);


Jfinal=zeros(400,1);
minmax_tray = minmaxDatos(data.trajectories);
for i=1:1:400
datosLanzamiento=data.trajectories{i};
Jfinal(i,1)=calculaNuevaJ(datosLanzamiento,goal,minmax_tray);
%Jfinal(i,1)=calculaJ(datosLanzamiento(:,1),datosLanzamiento(:,2));
end
figure;

L=reshape(Jfinal(:),size(Atest));
surf(Atest,Vtest,L);
hold on;
```

```
scatter3(data.succeed(:,2),data.succeed(:,3),Jfinal(data.succeed(:,1)),20,'g'
,'filled',...
        'MarkerEdgeColor','k');
%plot3(data.succeed(:,2),data.succeed(:,3),Jfinal(data.succeed(:,1)),'ro',Lin
eWidth=3);
title('Mapeado 20x20');
xlabel('Angulo q1(°)');
ylabel('T.movimiento (ms)');
zlabel('J (función de coste)')
shading interp;view([15 50]);
hold off;
floor(Jfinal(1,1))
for i=1:size(all_inputs,1)


    if ( Jfinal(i,1)<=12  )
        data.Jsucceed=[data.Jsucceed;i all_inputs(i,:) ];

    else
        data.Jfailed=[data.Jfailed;i all_inputs(i,:)];
    end
end

figure;
plot(data.Jsucceed(:,2),data.Jsucceed(:,3),'c^');
hold on;
plot(data.succeed(:,2),data.succeed(:,3),'bx');
plot(data.Jfailed(:,2),data.Jfailed(:,3),'ro');
hold off;
title('Mapeado 20x20 aciertos(x)/cernaco a acierto(Δ)/fallos(o)');
xlabel('Angulo_q1(°)');
ylabel('Tiempo del movimiento (ms)');

figure;
hold on;


for i=1:length(data.Jsucceed)
    if(Jfinal(data.Jsucceed(1,1),1)>6)
    idx=data.Jsucceed(i,1);
    traj=data.trajectories{idx};
    plot(traj(:,1),traj(:,2));
    x=[traj(1,1)-0.2 traj(1,1)];
    y=[traj(1,2)+0.2 traj(1,2)];

    end
end
hold off;
xlim([0 0.8]);
ylim([0 0.4]);
title('Trayectorias cercanas al objetivo en el mapeado');
xlabel('Eje X(m)');
ylabel('Eje Y(m)');
```

5.2.2 isGoalReached.m

```
function [res]=isGoalReached(trajectory,goal,tolerance)
```

```
N=floor(0.05*size(trajectory,1));%Comprueba con una tolerancia si se ha
acertado cogiendo los útlimos 5 puntos de cada disparo
sum=0;
for i=1:N
    sum=sum+(norm(trajectory(end-i+1,:)-goal)<tolerance);
end
res=sum>0;
end
```

5.2.3 minmaxDatos.m

```
function res = minmaxDatos(data)
%Calcula la longitud de la trayectoria para cada punto del mapeado y
%devuelve su valor normalizado junto a su índice
long_trayectorias_total=zeros(length(data),1);
for i=1:1:length(data)
long_trayectorias_total(i,1)=sum(sqrt(sum((diff(data{i}).^2),2)));
end

[minimo_long_trayectorias,min_index]=min(long_trayectorias_total);
[maximo_long_trayectorias,max_index]=max(long_trayectorias_total);
res=[min_index minimo_long_trayectorias; max_index maximo_long_trayectorias];
end
```

5.3 Experimentos

5.3.1 lanzamiento.m

```
function [Xdata,Ydata] = lanzamiento(angulo_q1, velocidad)
%% Datos y conexión

acc = "500"; % Tiempo de aceleración en ms
disp(angulo_q1);% Se imprime por el terminal el valor de ángulo
disp(velocidad);% Se imprime por el terminal el valor de tiempo del perfil de
velocidad
estadoPinza="true";%Pinza cerrada
velocidad_posDisparoInicial=3000;% Tiempo en ms del perfil de velocidad para
el posicinamiento
OpenRB150=conexionPuertoserie(15,115200,"CR");%Conexión puerto serie
pause(4);
%% Llamada a bucles


lanzamimientoCorrecto = 0;%Inicialización a cero del condicionante de
lanzamiento

while(lanzamimientoCorrecto == 0)
    permisoLanzamiento = 0;%Inicialización a cero del condicionante de
permiso del lanzamiento
    while permisoLanzamiento == 0
        estadoCodo="false";%Posicion codo
        [Px,Py,Pz,qt] = cinematica_directa_funcion(angulo_q1,110,-
80,60);%Configuración de posicionamiento
        Checksum=string(double(Px) + double(Py) +
double(Pz)+double(qt)+double(velocidad_posDisparoInicial)+double(acc));%Calcu
lo de checksum
```

```
msg=Px+","+Py+","+Pz+","+qt+","+velocidad_posDisparoInicial+","+acc+","+estad
oCodo+","+estadoPinza+";"+Checksum;%Mensaje en un string
        writeline(OpenRB150,msg);%Envio de mensaje por el puerto serie
        pause(3);%Pausa de posocionamiento
        fprintf(' [1] Lanzar experimento\n');%Elección entre lanzar o volver
a posicionar
        fprintf(' [0] Volver a posicionar\n');
        permisoLanzamiento = input('Elección: ');
    end

    estadoCodo="true";
    [Px,Py,Pz,qt] = cinematica_directa_funcion(angulo_q1,18,85,-5);
    Checksum=string(double(Px) + double(Py) +
double(Pz)+double(qt)+double(velocidad)+double(acc));

msg=Px+","+Py+","+Pz+","+qt+","+velocidad+","+acc+","+estadoCodo+","+estadoPi
nza+";"+Checksum;
    writeline(OpenRB150,msg);
    [Xdata,Ydata] = visionCapture;%Llamada a la función que captura las
coordenadas
    pause(velocidad/1000);
    fprintf(' [1] Lanzamiento Correcto\n');%Comprobación del correcto
lanzamiento
    fprintf(' [0] Error al lanzar\n');
    lanzamimientoCorrecto = input('Elección: ');
 end



end
```

5.3.2 lanzamiento2.m

```
function [J] = lanzamiento2(angulo_q1, velocidad)
%% Datos

acc = "500"; % en ms
disp(angulo_q1);
disp(velocidad);
estadoPinza="true";
velocidad_posDisparoInicial=3000;
OpenRB150=conexionPuertoserie(15,115200,"CR");
goal=[0.3680,0.15];


minmax_tray= [329.0000    0.0783;236.0000    1.3311];
X=[];
Y=[];
pause(4);
%% Llamada a bucles while
lanzamimientoCorrecto=0;
 while(lanzamimientoCorrecto==0)
    permisoLanzamiento=0;
    while permisoLanzamiento==0
        estadoCodo="false";
        [Px,Py,Pz,qt] = cinematica_directa_funcion(angulo_q1,110,-80,60);
```

```
        Checksum=string(double(Px) + double(Py) +
double(Pz)+double(qt)+double(velocidad_posDisparoInicial)+double(acc));

msg=Px+","+Py+","+Pz+","+qt+","+velocidad_posDisparoInicial+","+acc+","+estad
oCodo+","+estadoPinza+";"+Checksum;
        writeline(OpenRB150,msg);
        pause(3);
        fprintf(' [1] Lanzar experimento\n');
        fprintf(' [0] Volver a posicionar\n');
        permisoLanzamiento = input('Elección: ');
    end

    estadoCodo="true";
    [Px,Py,Pz,qt] = cinematica_directa_funcion(angulo_q1,18,85,-5);
    Checksum=string(double(Px) + double(Py) +
double(Pz)+double(qt)+double(velocidad)+double(acc));

msg=Px+","+Py+","+Pz+","+qt+","+velocidad+","+acc+","+estadoCodo+","+estadoPi
nza+";"+Checksum;
    writeline(OpenRB150,msg);
    [Xdata,Ydata] = visionCapture;%LLamada a la función que detecta la pelota
y devuelve las coordenadas

    fprintf(' [1] Lanzamiento Correcto\n');
    fprintf(' [0] Error al lanzar\n');
    lanzamimientoCorrecto = input('Elección: ');

    J=calculaNuevaJ([Xdata Ydata],goal,minmax_tray);% A diferencia de
lanzamiento.m en este script se calcula la J

  end


end
```

5.3.3 conexionPuertoserie.m

```
function [arduino] = conexionPuertoserie(numPuerto,baudios,finalTrama)
% Función de configuración del puerto serie
numPuertoCompleto = "COM"+string(numPuerto)
arduino = serialport(numPuertoCompleto,baudios);%Se asignan los baudios a los
que va a trabajar, los mismos que en el Arduino
configureTerminator(arduino,finalTrama);%Al final de la trama añade el
terminador por ejemplo el retorno de carro

end
```

5.3.4 cinemática_directa_funcion.m

```
function [Px,Py,Pz,qt] = cinematica_directa_funcion(q1,q2,q3,q4)
%% Datos del robot
l1=0.077;
l2=0.130;
l21=0.128;
l22=0.024;
l3=0.124;
l4=0.126;

%% Datos de entrada
```

```
q1=q1*(pi/180);%Se pasa de grados a radianes con esta operación los datos de
entrada
q2=q2*(pi/180);
q22=(pi/2)-q2;
q3=q3*(pi/180);
q4=q4*(pi/180);
qt=q3+q4-q22;
alpha3=atan2(l22,l21);
%% Calculos
r=(l21*sin(q22)+l22*cos(q22))+l3*cos((-q22+q3))+l4*cos((-
q22+q3+q4));%Distancia de la proyeccion del robot con el suelo
Px=r*cos(q1);
Py=r*sin(q1);
Pz=l1+l21*sin(q2)+l22*sin(-q22)+l3*sin(q3-q22)+l4*sin(q4+q3-q22);
end
```

5.3.5 RobotyVision.m

```
%Script para el comprobar el funcionamiento del robot y la vison artificual
%al mismo tiempo

for cont=1:1:10 %10 iteraciones
[Xdata,Ydata] = lanzamiento(14.9, 730)% Envio de datos de lanzamiento

 Resultados2.X{cont}=Xdata;%Guardar los datos recibidos
 Resultados2.Y{cont}=Ydata;

end

goal=[0.368 0.150];%Objetivo en metros
minmax_tray= [329.0000    0.0783;236.0000    1.3311%Datos de longitud de la
trayectoria normalizada con sus índices

for i=1:1:length(Resultados2.X(:))%Calculo de la función objetivo
Jnueva(i,1)=calculaNuevaJ([Resultados2.X{i},Resultados2.Y{i}],goal,minmax_tra
y);
end

figure;%Representación de las trayectorias
plot(Resultados2.X{1},Resultados2.Y{1})
xlim([0 0.8])
ylim([0 0.4])
hold on
for i=2:1:length(Resultados2.X(:))
plot(Resultados2.X{i},Resultados2.Y{i})
end
hold off;
```

5.3.6 calculaNuevaJ.m

```
function [J] = calculaNuevaJ(datosLanzamiento,goal,minmax_tray)
%Calculo de la ffunción objetivo
N=floor(0.1*size(datosLanzamiento,1));
lastData=datosLanzamiento(end-N+1:end,:);%ltimos cinco puntos

media=mean(lastData);%Media de los últimos cinco puntos
```

```
dist_media=norm(media-goal);%Distancia de la media de los últimos veinte
puntos (VALOR FINAL)

minimo_long_trayectorias=minmax_tray(1,2);
maximo_long_trayectorias=minmax_tray(2,2);
long_trayectoria=sum(sqrt(sum((diff(datosLanzamiento).^2),2)));
long_trayectoria_norm=(long_trayectoria-
minimo_long_trayectorias)/(maximo_long_trayectorias-
minimo_long_trayectorias);% Calculco de la lingitude de la trayectoria
normalizada

distancia_recorrida=zeros(1,50);
for i=1:1:size(datosLanzamiento,1)
distancia_recorrida(1,i)=(norm(datosLanzamiento(i,:)-goal));%Distancia
recorrida para cada punto
end
[minimaDistanciaTrayectoria] = min(distancia_recorrida);%Distancia más
cercana al objetivo
alpha=15;%Valores de importancia para cada término
beta=50;
omega=5;
J=alpha*dist_media+beta*minimaDistanciaTrayectoria+omega*long_trayectoria_nor
m;%Función objetivo
end
```

5.3.7 bayesoptfunction.m

```
clear all;
close all;
vars = optimizableVariable('angulo_q1', [-15, 15]); % Por ejemplo, si el
rango de x es [-10, 10]
vars2 = optimizableVariable('velocidad', [650, 750]); % Por ejemplo, si el
rango de x es [-10, 10]

funcion = @(x) lanzamiento2(x.angulo_q1,x.velocidad);%Declaración de la
función

results = bayesopt(funcion, [vars vars2],'IsObjectiveDeterministic',false,
'AcquisitionFunctionName','probability-of-
improvement','MaxObjectiveEvaluations',15,ExplorationRatio=0.1);

plot(results,'all');%Dibuja todas las figuras de los resultados
```

6. Tabla resumen ODS

Se añade una tabla específica de los ODS que están relacionados con el presente proyecto, tal y como se especifica en el boletín oficial de UPV número 118/2022.

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No procede
ODS 1. Fin de la pobreza.			X	
ODS 2. Hambre cero.				X
ODS 3. Salud y bienestar.	X			
ODS 4. Educación de calidad.				X
ODS 5. Igualdad de género.				X
ODS 6. Agua limpia y saneamiento.			X	
ODS 7. Energía asequible y no contaminante.	X			
ODS 8. Trabajo decente y crecimiento económico.		X		
ODS 9. Industria, innovación e infraestructuras.	X			
ODS 10. Reducción de las desigualdades**.**				X
ODS 11. Ciudades y comunidades sostenibles**.**		X		
ODS 12. Producción y consumo responsables.		X		
ODS 13. Acción por el clima.		X		
ODS 14. Vida submarina.				X
ODS 15. Vida de ecosistemas terrestres.				X
ODS 16. Paz, justicia e instituciones sólidas.				X
ODS 17. Alianzas para lograr objetivos.				X

Tabla 2: Resumen del grado de importancia de las ODS. (Fuente: Elaboración propia).