DEAL_AI

Brazo mecánico que permite actuar como Dealer o Croupier en el juego del Poker, concretamente la modalidad de Texas Holdem. También ayuda a controlar el estado de la partida.

PROJECT SPRINT #5.
DATE: 25th May 2021

EMIL CARVAJAL 1529821 #1
DENÍS MONTOYA 1527418 #2
MARC GORDÓN 1457460 #3
JOFRE GÓMEZ 1526889 #4

Tabla de contenido

Sprint #5
Date: 25 Mayo 2021

DEAL_AI

Brazo mecánico que permite actuar como Dealer o Croupier en el juego del Poker, concretamente la modalidad de Texas Holdem.
También ayuda a controlar el estado de la partida.

Descripción del Proyecto

DEAL_AI es un robot que ayuda a dirigir el estado de una partida del clásico juego de cartas Poker actuando como Dealer o Croupier.

Básicamente consiste en un brazo mecánico de 4 ejes con la misma estructura de un robot SCARA que tiene acoplada una ventosa para poder distribuir las fichas y las cartas que están en juego. Este robot está diseñado para una mesa semicircular la cual está adaptada a su zona de trabajo. Existe un componente situado al lado de la base el cual le permite girar las cartas simplemente soltándolas por encima del objeto. Para reconocer las fichas y las cartas tiene una cámara colocada en una torre central elevada.

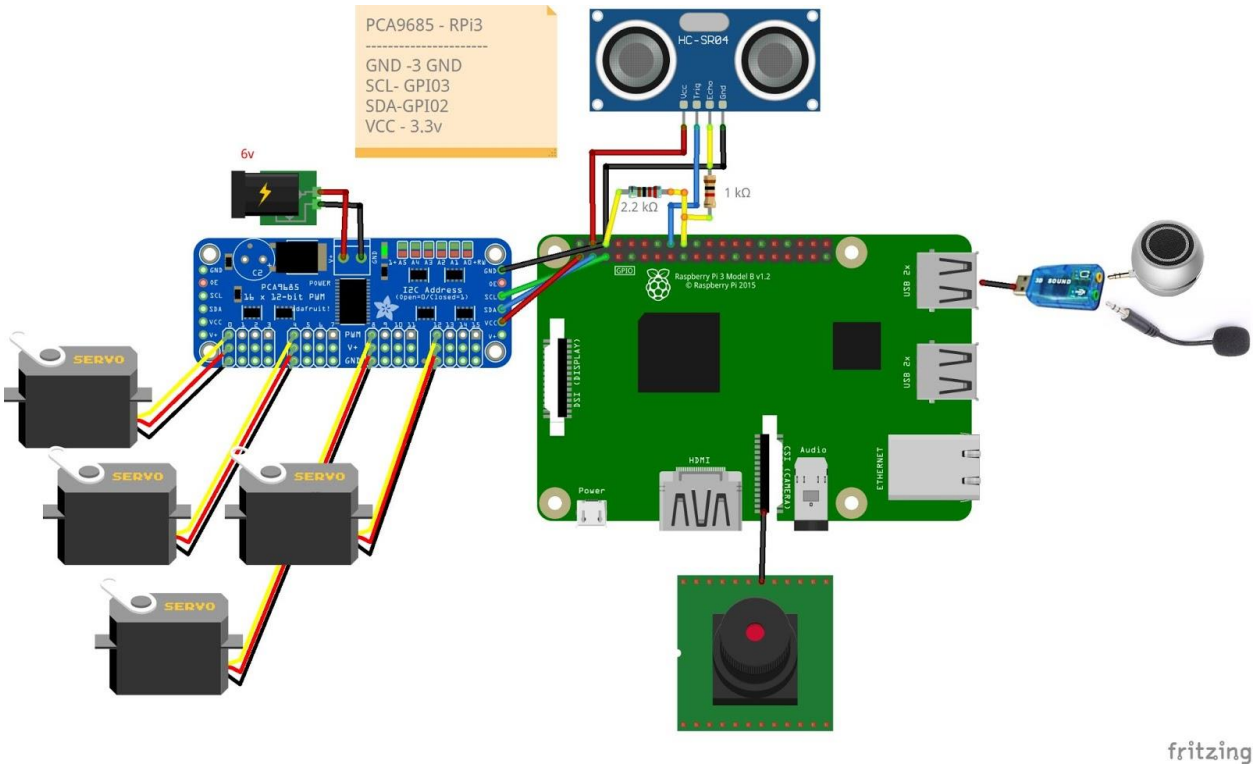
Sus principales funciones son:

- Funciones de Dealer:
  - Flop (mostrar las 3 cartas)
  - Mostar las 5 cartas
  - Quemar cartas
  - Incluye voltear las cartas
  - Repartir Cartas
- Intercambio de fichas mediante IA
- Realizar movimientos y responder preguntas (órdenes captadas por reconocimiento de voz)
- Declarar ganador por medio de visión por computador.

Componentes electrónicos

- Placa PCA9685
- Raspberry pi 3
- 4 Servomotores
- Cámara
- Adaptador de audio
- Micrófono
- Altavoz
- Ventosa de bomba de aire de vacío
- 5 sensores de ultrasonido

Esquema Hardware



CONEXIONES *(--> Serie // Paralelo)*

SERVO1,2,3,4 (pin,v,gnd) --> PCA9685 ((0,4,8,12) , 5v, GND))

PCA9685 (Vin,GND,SCL,SDA) --> Raspberry pi 3 (5v, GND, GIPO03,GIPO04)

Raspberry pi 3 (CSI camera) --> Camera

Raspberry pi 3 (USB1 port) --> Adaptador de Audio

Adaptador de Audio(in) --> External Microphone

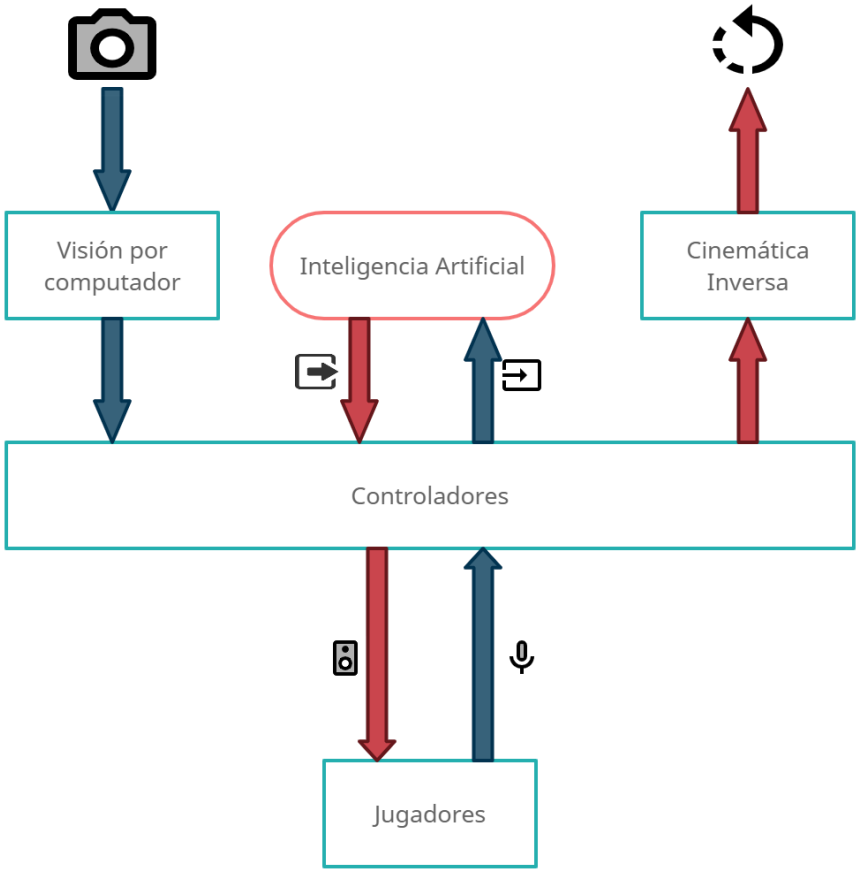
Adaptador de Audio(out) --> External Speaker

HC - SR04 (Vcc,GND,Trig,Echo) --> Raspberry pi 3 (5v, GND, (GND,GIPO23), GIPO24)

- *HC - SR04 (Trig) --> R1(1kΩ) --> Raspberry pi 3(GIPO24)*

  // R2(2.2kΩ) --> Raspberry pi 3(GND)

Arquitectura Software



Visión por computador:

Entendemos este módulo como el que recoge la información de la situación en la zona de juego. Dicha información se utilizará posteriormente para poder establecer las posiciones de los elementos a manipular por el robot (cartas y fichas) y para que pueda determinar la situación en tiempo real de la partida (reconocimiento de las cartas).

Finalmente, para este módulo se ha decidido hacerlo junto con el proyecto final de la asignatura "Visión por Computador". El robot será capaz de detectar tanto cartas como fichas. Además de detectarlos podrá saber sus valores concretos.

Para la **detección de cartas**, se ha limitado una zona para cada jugador conde se encontrarán las cartas y fichas correspondientes de cada jugador.
Se ha implementado una función llamada "**detectarCartes**" que se le pasan por parámetro la región de la imagen del jugador recortada y otra igual pero binarizada (en blanco y negro) y la cantidad de cartas que se esperan que se detecten (2 si son la de los jugadores y 5 si son las de la máquina). Esta función se encarga de tractar las imágenes que recibe para devolver 3 arrays, el 1º contiene una imagen de cada carta, el 2º una imagen del rango / valor de las cartas detectadas (A, 1, 2, 3, …, Q,K) y el 3º contiene una imagen del palo de cada carta detectada (corazón, picas, tréboles, rombos).
También, se ha implementado una función llamada "**detectarValors**" que recibe las imágenes de los valores y tipos obtenidos en la función "**detectarCartes**" y un dataset de valores de cartas y un dataset de valores de palos de Poker. En esta función se usan los dataset que se pasan por parámetro para clasificar los palos y valores de cada una de las imágenes que también recibe y devolver 2 arrays en un formato específico para posteriormente determinar quien es el ganador.
Para poder saber quien es el ganador, se ha implementado una función llamada "**comboJugador**" que mediante los arrays que devuelve "**detectarValors**" le da una puntuación al jugador sobre la mano que tiene y con que combo (pareja, doble pareja, escalera, …) ha obtenido esa puntuación (el jugador que obtenga la mayor puntuación será el ganador).

Para la **detección** de las **fichas** no se utilizan las mismas zonas que para las cartas porques estas no son usadas ya que no contienen el area donde el jugador puede dejar sus fichas para que el robot las detecte, sume su valor y pueda recogerlas y llevarlas a la banca de fichas para su intercambio. Por lo que lo primero ha sido determinar esas áreas de cambio

valorFichas(img_Robot): Función que se encarga de determinar el valor de una ficha. Esta recibe una imagen de la área determinada por el jugador que pide cambio de fichas. La imagen contiene varias fichas, pero al estar apiladas solo ve la que está más por encima.

Los pasos que emplea son:

- Eliminación del ruido
- Encuentra el área del fondo
- Encuentra el área segura que pertenece a la ficha
- Encuentra la región desconocida (bordes)
- Etiquetado de las manchas que detecta
- Adiciona 1 a todas las etiquetas para asegurra que el fondo sea 1 en lugar de cero
- Se marca la región desconocida con ceros
- Utiliza técnica "*watershed*" para dividir las manchas
- Se binariza la imagen
- Buscamos los contornos las manchas restantes con la función cv2.findContours()
- Recortamos la mancha
- Le añadimos brillo mediante la función **increase_brightness()**
- Realizamos una media de pixeles y vemos que color es más cercano para asociarle

Cinemática inversa:

En este módulo se aplicará a los servo motores la cinemática inversa calculada para permitir al robot alcanzar el objeto o la posición deseada. Básicamente tras haber establecido la posición (módulo IA) el controlador ordenará a los componentes hardware la cinemática a realizar.

Lo principal para poder empezar a desplazar nuestro robot hacia las posiciones deseadas, es calcular las ecuaciones de movimiento del robot mediante una tabla de Denavit Hartenberg (tabla de D-H).

Para ello, hacemos los correspondientes cálculos en función de la diferencia entre las distancias y orientaciones de un servo-motor y otro. Al aplicar los cálculos correctos, obtenemos la siguiente matriz resultante para la cinemática inversa de un robot SCARA de 4 ejes:

$$\begin{bmatrix} 1.0\cos(\theta_1+\theta_2-\theta_4) & -1.0\sin(\theta_1+\theta_2-\theta_4) & 0 & la\cos(\theta_1)+lb\cos(\theta_1+\theta_2) \\ 1.0\sin(\theta_1+\theta_2-\theta_4) & 1.0\cos(\theta_1+\theta_2-\theta_4) & 0 & la\sin(\theta_1)+lb\sin(\theta_1+\theta_2) \\ 0 & 0 & 1.0 & -1.0d_3-1.0l_4+1.0lc \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

En esta ecuación conocemos los valores de *la, lb, lc* y *l4* que equivalen a la distancia de cada brazo o *link*. Los datos que desconocemos son ϑ1, ϑ2, ϑ4 y d3, que equivaldrían a los ángulos de rotación de los servo-motores 1, 2 y 4 (los cuales son rotacionales) y la altura a la que está situada el manipulador (controlado por el servo-motor 3 que es de tipo prismático). Así pues, programamos un módulo básico para el cálculo de estas coordenadas:

```
def colocar(manchas, z):
    retCode = sim.simxSetJointTargetPosition(clientID, joint3, 0, sim.simx_opmode_oneshot)

    time.sleep(2)

    eq1 = 0.2 * cos(theta1) + 0.2 * cos(theta1 + theta2) - manchas[0][0]
    eq2 = 0.2 * sin(theta1) + 0.2 * sin(theta1 + theta2) - manchas[0][1]
    eq3 = 0.105 - d3 - z
    try:
        qFinal=nsolve((eq1,eq2,eq3),(theta1,theta2,d3),(1,1,1),prec=5)
    except:
        print('No se encuentra solución')
        qFinal=[0,0,0]

    retCode = sim.simxSetJointTargetPosition(clientID, joint1, qFinal[0], sim.simx_opmode_oneshot)
    retCode = sim.simxSetJointTargetPosition(clientID, joint2, qFinal[1], sim.simx_opmode_oneshot)
    retCode = sim.simxSetJointTargetPosition(clientID, joint3, -qFinal[2], sim.simx_opmode_oneshot)

    time.sleep(2)
    setEffector(0)
    time.sleep(1)
    retCode = sim.simxSetJointTargetPosition(clientID, joint3, 0, sim.simx_opmode_oneshot)
    time.sleep(1)
```

Se utiliza el módulo base de la función "colocar" para ejercer toda la cinemática del robot. El núcleo de esta es la función "nsolve" que encontrar las incógnitas de un sistema de ecuaciones. Tanto las incógnitas como las ecuaciones se pasan como parámetros. En nuestro caso, se pasa como ecuaciones los resultados de aplicar la tabla D-H en un robot de 4 ejes SCARA y, para el movimiento, se encuentran los ángulos de los ejes rotacionales de los 2 primeros servo-motores. La altura no obstante será determinada mediante un sensor de ultrasonido, capaz de determinar la distancia a la que se encuentra la válvula de vacío (manipulador del robot) del objeto en cuestión (cartas o fichas).

Problema: La orientación del robot no es la predefinida, el robot está orientado en el eje de ordenadas en vez del de abscisas.

Solución: Se invierten los ejes en el cálculo de la posición, así pues, nuestras coordenadas realmente son (y, x).

Para facilitar la faena de los programadores, se crean módulos estándar que serán utilizados en las funciones principales, así pues, para repartir cartas a un jugador, en vez de reescribir el código de la cinemática inversa, se utiliza un módulo de movimiento estándar que es llamado por la función de mayor nivel.

Solución: Al utilizar un sistema de jerarquía de funciones, podemos volver a trabajar con el sistema de coordenadas (x,y) en general, hemos realizado el cambio a (y,x) en la función de menor nivel pero nos permite trabajar de manera normal y cómoda de esta forma.

Con este sistema establecido, generamos las siguientes funciones de alto nivel:

- **PosicionJugador(nJugadores)**: Se pasa como parámetro 'nJugadores' que corresponde al número de participantes que jugarán a póker. La función calcula y devuelve las posiciones geométricas de cada jugador y el ángulo correspondiente (entre 90 y -90 grados porque es una mesa semicircular y el robot está orientado en el ángulo 0 grados).

- **DarCartaJugador(x, y, angulo):** Se pasa como parámetro la posición **central** de un jugador. En esta función se le suma y resta a esta posición un coeficiente pequeño que determina la separación entre una carta y otra, pues en esta modalidad de póker ser reparten dos cartas a cada jugador. Una vez aplicado este cálculo, se determina de la posición central un total de 2 posiciones, que equivalen a las posiciones de cada carta. Puesto que se colocan las cartas de los jugadores en el perímetro del área, al colocar el eje rotacional en el ángulo 0 grados, las cartas quedan orientadas apuntando hacia el centro de la mesa.
- **RevelarCartasCentrales():** En esta función se revelan las cartas correspondientes cuando da inicio una nueva ronda. En esta modalidad de póker, se revelan un total de 5 cartas: 3 cartas en la primera ronda, 1 carta en la segunda y 1 carta en la tercera.
- **Ir_a_flipeador():** Tras analizar y probar varios modelos de piezas 3D que permitan voltear una carta automáticamente, hemos encontrado la pieza necesaria para lograrlo. Tras situar la pieza en un lugar específico de la zona de juego, el brazo robótico únicamente debe colocar encima de la pieza la carta que desea voltear, rotando dicha carta para que quepa en la entrada y se voltee automáticamente, deslizándose hasta una pequeña zona donde se recogerá la carta.

  Problema: El brazo no es capaz de elevarse lo suficientemente alto como para colocar la carta en la parte superior de la pieza 3D.

  Solución: Se han modificado las distintas dimensiones del robot, en concreto se ha incrementado la altura de la base del robot y se ha elevado el eje prismático, permitiéndonos depositar la carta en la parte superior del volteador.

  Problema derivado: En algunos casos, para repartir las cartas a algunos jugadores, el brazo colisiona con el volteador de cartas, bloqueando el único medio de acceso hacia la posición del jugador.

  Solución: Se ha modificado la función de nivel bajo **CogerCarta()** la cual, al recoger la carta de la baraja, la eleva lo máximo posible.

De este modo, cada vez que se quiere acceder a una posición problemática, el brazo robótico se desplazará por encima de la pieza 3D, evitando colisiones innecesarias y permitiendo alcanzar cualquier posición sin impedimentos .

- **QuemarCarta()**: En muchos juegos de cartas de apuestas antes de repartir cartas o de revelar cartas se realiza una acción denominada como "quemar carta" que simplemente consta en coger la primera carta del mazo y colocarla en una pila de descartes.

Problema 1: La función 'nsolve' muestra problemas para encontrar el camino óptimo que ha de recorrer el robot y, en ciertas ocasiones para llegar a la coordenada de destino realiza más vueltas de las necesarias (gira más de 360 grados).

Solución 1: En el módulo del cálculo de la cinemática inversa se aplica el módulo a la solución para así obtener un resultado entre 0 y 360 grados:

$$angulo = angulo \,\%\, 2\pi$$

En este caso, '*angulo*' es el parámetro que nos devuelve la función 'nsolve' que equivaldría a la rotación que debería ejercer el servo-motor para alcanzar la coordenada en cuestión. 2π equivale a 360 grados en radianes.

Problema derivado 1.1: La operación módulo (%) no tiene en cuenta el signo del ángulo, es decir, si la variable 'angulo' es positiva, el resultado será positivo, pero en caso de ser un valor negativo, el resultado que devuelve es positivo cosa que no nos interesa, pues hay una gran diferencia entre girar 45 y -45 grados.

Solución 1.1: Puesto que no hay ninguna función que permita realizar la operación de módulo devolviendo el signo correcto, se detecta el signo antes de aplicar la operación y, en caso de ser negativo, se calcula el módulo con el valor positivo y luego se negativiza el resultado.

Problema derivado 1.2: La variable '*angulo*' ahora está comprendida entre 0 y 360 grados, pero en ciertas ocasiones el brazo gira casi 360 grados para acceder

a una posición cercana si se desplazara en dirección contraria. Por ejemplo, si el robot quiere situarse en el centro del 4to cuadrante (cuadrante (x, -y) entre los ángulos 270 y 360) podría acceder de manera rápida si rotara -45 grados, pero la función nos devuelve un valor de 315, por tanto, el brazo rota un total de 315 grados lo cual no es óptimo.

Solución 1.2: Los brazos robóticos con capacidad de rotación de 360 grados pueden dividirse en dos zonas de movimiento óptimo: de 0 a 180 grados y de 0 a -180 grados (que equivaldría a la zona comprendida entre 180 y 360 grados). Si el robot quiere rotar 181 grados, es más eficaz que rote -179 grados, pues llegará a la misma posición rotando una cantidad más pequeña de grados. Por tanto, se determina un condicional que si detecta que la variable '*angulo*' supera los 180 grados (que en radianes equivale a π), entonces se le resta 360 grados a '*angulo*' (equivalente a 2π), así pues, con el mismo caso de antes, si tenemos 181 grados y le restamos 360, obtenemos -179 que sería nuestro recorrido óptimo.

Solución final: Juntando todas estas soluciones, obtenemos un módulo capaz de corregir cualquier ángulo y transformarlo en uno óptimo (si es que no lo era antes):

```
qFinal=nsolve((eq1,eq2,eq3),(theta1,theta2,d3),(1,1,1),prec=5)
for i in range(2):
    if abs(qFinal[i]) > np.pi:
        if qFinal[i] < 0:
            qFinal[i] = qFinal[i] * (-1)
            qFinal[i] = qFinal[i] % (2*np.pi)
            if qFinal[i] > np.pi:
                qFinal[i] = qFinal[i] - (2*np.pi)
            qFinal[i] = qFinal[i] * (-1)
        else:
            qFinal[i] = qFinal[i] % (2*np.pi)
            if qFinal[i] > np.pi:
                qFinal[i] = qFinal[i] - (2*np.pi)
```

Ejemplos de módulos de bajo nivel sería CogerItem() que activa la válvula de vacío, CogerCarta() que automatiza la acción de coger una carta de la baraja (utilizando la función CogerItem()), DejarItem() que desactiva la válvula de

vacío… En general son funciones sencillas que son llamadas varias veces por los módulos de alto nivel.

Inteligencia Artificial:

Main:

Este es el núcleo del robot, donde todos los inputs son transmitidos a este módulo el cual se encargará de dar una serie de órdenes al robot. Estas órdenes pasarán por el controlador en concreto y establecerá una acción: o bien utilizará el altavoz para dar un mensaje a los jugadores o bien realizará un movimiento para alcanzar una posición calculada previamente.

Este main representa el sistema del juego Poker en la modalidad Holdem Texas pero con ciertos cambios que se adaptan a una partida cotidiana entre amigos.
Por eso, cada jugador empieza con una cantidad de fichas determinadas. El sistema esta organizado por "Manos" y "Rondas". Denominamos **Manos** a 4 Rondas completas donde en la última se decide quien es el jugador que ha ganado la Mano y se lleva todo el dinero invertido. Una **Ronda** es terminada cuando todos los jugadores activos de la Mano han apostado la misma cantidad de fichas. En la Ronda 0 se reparten las cartas, en la Ronda 1 se colocan 3 cartas centrales en la mesa y en la Ronda 2 i 3 se "quema" una carta y se coloca otra junto a las centrales. "Quemar" una carta no es más que coger una carta y descartar-la colocándola en una posición de la mesa.

El sistema controla la secuencia de juego y mantiene un estado de los jugadores que no se han retirado para pasar a la siguiente ronda.
Controla en todo momento los jugadores que están activos y los que han abandonado la mesa (no se contemplan para las siguientes Manos y no se les reparte cartas).

Cuando es el turno de un jugador se le pregunta, por comandos de voz, què acción desea realizar. Solo se ejecutan las acciones que ordena si se encuentran en el sistema. Estas **acciones** son típicas del juego como:

- "Paso": el jugador no desea subir la apuesta.
- "Igualo, Voy": el jugador sigue la apuesta máxima y coloca sus fichas en la mesa.
- "Me retiro": Se retira de la Mano y no se le preguntará nada hasta la próxima Mano.

- "Abandono": Abandona la mesa y ya no se le tendrá más cuenta en el juego.
- "Cambio a la alta/baja": El jugador coloca sus fichas en su área de cambio según si desea cambio a valores altos o bajos. Implementado por el **Modulo de Cambio.**
- Preguntas de estado: Se le puede preguntar al sistema información sobre el estado del juego:

  Ej: "Dealer actual?", "Ronda actual?, "Quién es la ciega grande?", "Apuesta máxima?"…

Cuando se ha llegado a la última Ronda de una Mano el sistema dice a los jugadores que han permanecido al final de la Mano que muestren sus cartas.
El sistema reconoce todas las cartas que hay en la mesa gracias al módulo de visión por computador y determina el jugador que ha ganado y la combinación de cartas con lo que lo ha hecho.

Para las siguientes Manos el sistema de juego es el mismo. Se termina el juego cuando solo queda un jugador en la mesa.

Módulo de cambio:

Módulo que controla el sistema de cambio de fichas. Se implementa en 3 etapas: reconocimiento de las fichas que se desean cambiar y llevarlas a la banca, tratamiento del valor total de las fichas para establecer cuantas monedas y de qué tipo ha de devolver y por último la secuencia de movimientos para colocar las fichas de cambio.

Funciones:

- **recogerFichasMonton(area):** Por medio de un área establecida controla el módulo de visión por computador de las fichas para poder reconocerlas y luego a través de las funciones de cinemática inversa coloca esas fichas en su determinada posición de la banca.

- **cambiar_fichas_greedy(suma_fichas,mode):** Algoritmo *greedy* (sacado de la asignatura de la mención de Computación "Analisis y disseny d'algortimes") implementado para saber qué fichas ha de devolver el robot. Incorpora 2 modos para devolver fichas con valores bajos o máximos.

- **dar_Cambio(cambio, q):** Recibe qué fichas se han de devolver y la matriz de la posición donde el jugador dejó sus fichas de cambio.

Interacción Robot-Jugador

El sistema es capaz de escuchar (mediante **speech-recognition**) y contestar (mediante gTTS y playsound). En el apartado de "Jugadores" se darán más detalles sobre este módulo.

Controladores: Esta estructura es la que nos hace de puente entre todos los módulos existentes y nos permite la transmisión de entradas (inputs) y salidas (outputs) a sus respectivos módulos.

Todo y que los controladores en el diseño van como un módulo a parte, se ha decidido finalmente unificarlos junto a los principales ficheros como el main, speakAndListen etc…

Jugadores:

Representa el colectivo de seres humanos que están realizando una partida con el Deal AI. Estos son los que generarán los inputs principales, los cuales determinarán al robot qué algoritmo utilizar y con qué módulos operar.

En cuanto a este módulo, se han implementado unas funciones de interacción robot-humano. Se han usado los módulos/librerías **speech_recognition** (para el reconocimiento de voz, así dando la oportunidad de controlar las acciones del robot), gTTS (para poder leer con voz sintética (text-to-speech) las respuestas dadas por el robot) y playsound (para poder reproducir sonidos y poder leer los resultados obtenidos por el gTTS).

Amazing contributions

Deal AI cuenta con las siguientes funciones:

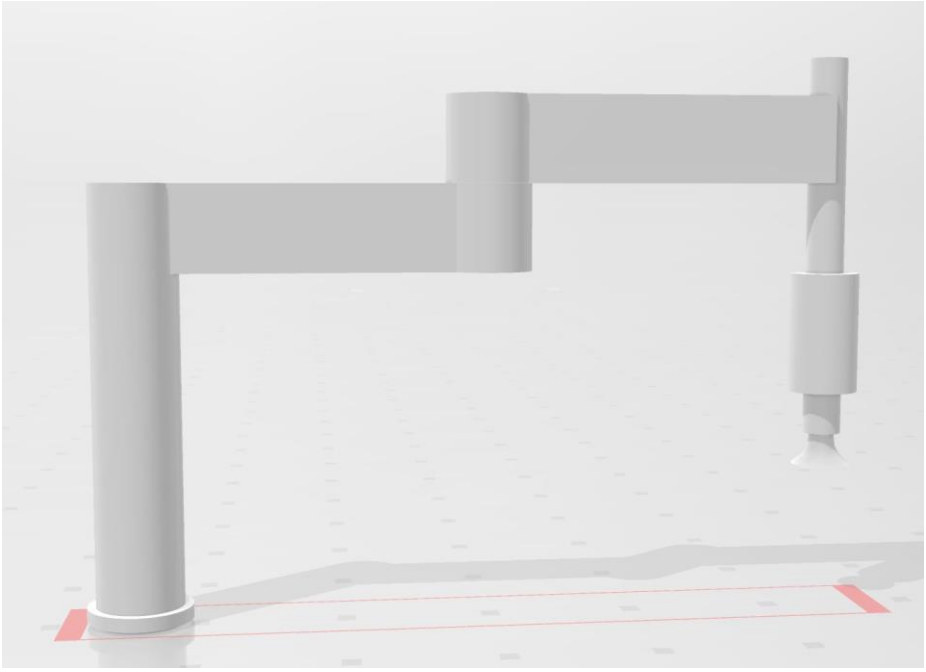
- Seguimiento automático de una partida en vivo de póker texas holdem.
- Repartición de cartas a los jugadores.
- Volteo de cartas automático.
- Intercambio de fichas mediante algoritmos greedy.
- Colocación de cartas compartidas mediante visión por computador.
- Explicación de la situación actual mediante voz (text to speech).
- Avance de la partida mediante comandos de voz.
- Resolución de preguntas básicas mediante el módulo de sonido.
- Reacción ante importunos.

En caso de realizar todos y cada uno de los objetivos creemos que la nota debería equivaler a un excelente. La máxima nota podría ser obtenida dependiendo de la eficiencia del robot ante las situaciones mencionadas anteriormente.

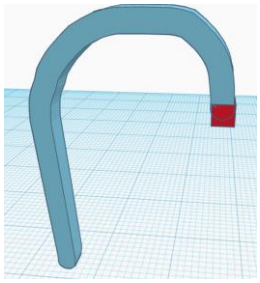
Componentes y piezas 3D extras

- Brazo
- Soporte cámara
- Pieza “flip” de cartas
- Pieza contenedora de fichas
- Soporte cartas

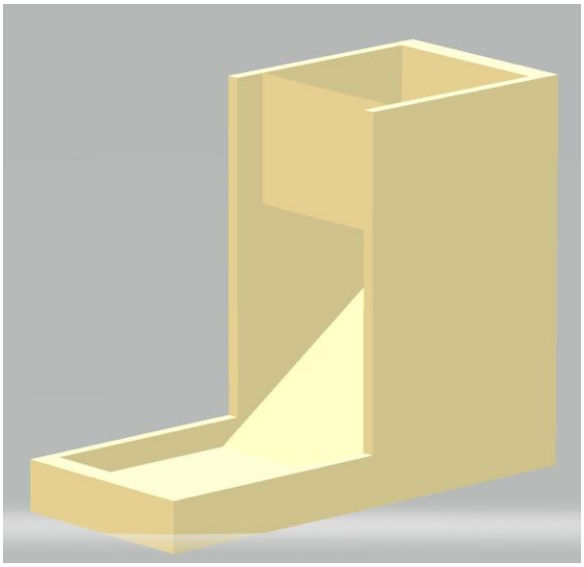
1) **<u>Brazo:</u>** Componentes unidos del brazo robótico (total de 4 piezas por separado).



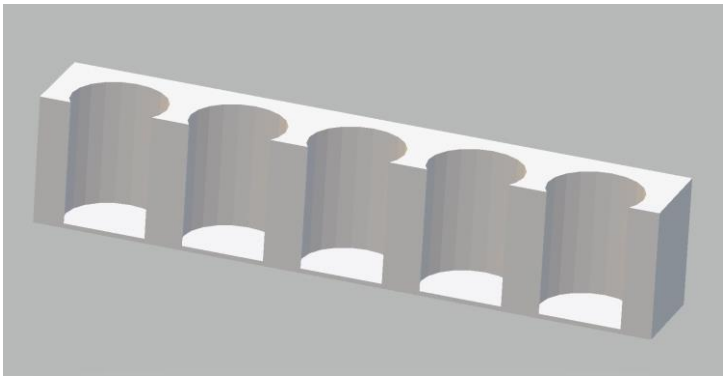
2) **<u>Soporte de cámara:</u>** Pieza que sostiene la cámara de tal modo que nos permite obtener una vista cenital de la mesa así evitando el ser tapada por el propio brazo y consiguiendo un sistema de cálculos más simple.



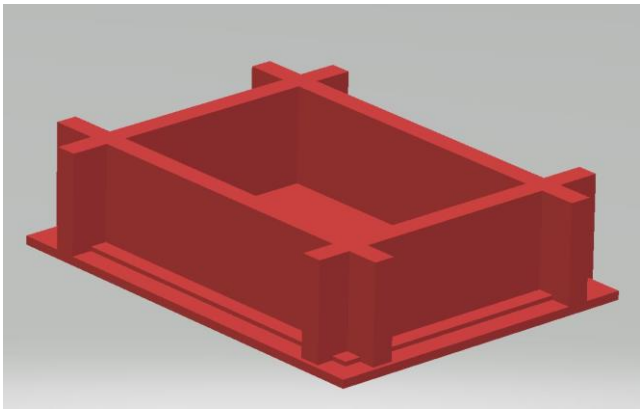
3) **<u>Pieza “flip” de cartas:</u>** Pieza con dos conductos que, al introducir una carta por ésta, le da la vuelta y la expulsa por el otro conducto. De este modo conseguimos un “flip” automático de las cartas haciendo el robot independiente para esta tarea. La carta termina en una badeja y De este modo podemos tener en todo momento controlada la carta.



4) **<u>Contenedor de fichas:</u>** Pieza estática que nos facilitará el proceso de clasificación de fichas y además, servirá de contenedor.



5) **<u>Soporte cartas:</u>** Pieza sencilla que nos servirá como soporte para la baraja. De este modo aseguramos que la baraja se mantiene estática en un lugar y no se desmonta al coger una carta.



Se ha decidido hacer el modelo simple de los propuestos en el primer sprint, siguiendo los siguientes razonamientos: Teniendo en cuenta que la dimensión de una carta es de 64mm x 88mm, el hueco interno del soporte de las cartas se ha ampliado en 1'5mm para cada uno de los 4 lados, estableciendo pues un hueco interior de 67mm x 91mm, así pues, hemos decidido que el ancho de cada pared del receptáculo sea de 4.5mm, el resultado final es este, una pequeña caja sin tapa de dimensiones 76mm x 100mm x 25mm (suponiendo que una baraja de cartas tenga un tamaño máximo de 20mm de altura, así pues dejamos 5mm como margen).

Estrategia de simulación

La estrategia de simulación que seguiremos será la de ofrecer una simulación de una partida real y en vivo, mostrando en ésta todas las funcionalidades descritas en los apartados anteriores. De este modo podremos demostrar la funcionalidad real de nuestro robot además de mostrar que en un caso práctico real da la talla, es decir, cumple todas las funcionalidades presentadas.

En cuanto al sistema o simuladores que usaremos, este va a ser CoppeliaSim ya que lo estamos aprendiendo a usar en las clases de problemas, nos sirve perfectamente para nuestro robot y además, se puede enlazar con Python que será el lenguaje que usaremos para programar los módulos de detección de sonido, visión por computador, la propia simulación de la partida etc…

Los módulos simulados dentro de la partida serán las principales funcionalidades mencionadas en los apartados anteriores.

De las primeras simulaciones hemos obtenido los siguientes resultados y conclusiones:

Las simulaciones sobre el módulo de voz han sido satisfactorias. Se han detectado correctamente las “key words” en varias pruebas que hemos hecho con distintos sujetos.

De las simulaciones generales no han habido resultados satisfactorios debido a ciertos factores: primero, el modelo que tenemos de carta hace movimientos raros que no entendemos (sale volando sin sentido, sin ser tocada, lo hace de forma automática) y debemos preguntar al tutor. El otro problema encontrado es que el brazo hace movimientos no coherentes cuando le enviamos a un punto, cosa que nos ha hecho llegar a la conclusión que cierta parte del código de cinemática inversa está erróneo.

Simulaciones para módulo de Visión por computador:

Inicialmente, se hicieron pruebas con imágenes con 1 o mas cartas sin ningún tipo de inclinación y con resoluciones bajas (512x256), esto causaba varios problemas:

- La cámara está bastante elevada y cada carta ocupa unos pocos píxeles, lo que hacía imposible reconocer tanto los valores como el tipo de la carta.
- Al no tener inclinación las cartas, hacía que los casos probados no fuesen realistas ya que un jugador no porqué dejar la carta encima de la mesa totalmente recta.
- Algunos parámetros dependían de la resolución, lo que si en un futuro se quiere cambiar solo la cámara resultaría un problema. Esto también tenía repercusión en algunas partes del código donde en futuras simulaciones con cartas con inclinación hacía que saltasen errores.

En posteriores simulaciones con cartas con inclinación, donde se aumentó la resolución de la cámara (1024x512) y se cambiaron parámetros para que escalasen con la resolución. Finalmente obtuvimos los resultados que esperábamos, detectaba los valores y tipos de cartas con un porcentaje de acierto muy alto y decía quién era el ganador sin ningún tipo de problema.

Simulaciones cinemática inversa:

En cuanto al sistema de repartir y manipular cartas, nuestro grupo se encuentra con un problema relacionado con las físicas del simulador que no ha sido capaz de solventar. Se ha buscado una solución para casi todos los apartados en los que ocasionaba error, pero en el módulo de dar la vuelta a las cartas no se ha podido encontrar una manera real de hacerlo, por tanto, no hemos tenido éxito simulándolo.

No obstante, se ha llevado a cabo la simulación y comprobación de los módulos básicos de manipulación de cartas y estos han sido los resultados:

- Gestión correcta de una partida de entre 2 y 5 jugadores.
- Es capaz de revelar las 5 cartas correctamente e incluso puede revelar hasta un total de 7 cartas sin obstruir la zona de juego (aplicable para otras modalidades de póker).
- Buena compenetración entre los distintos módulos de gestión de cartas.
- Cálculo computacional de las ecuaciones rápido.

Sin embargo, un problema que hemos encontrado es la diferencia de tiempos entre la ejecución de código y el simulador. Al ejecutar, concretamente, una función de espera (en nuestro caso, de la librería time, time.sleep(segundos)) empieza a contar los segundos de espera desde que es ejecutada y utiliza como referencia el tiempo real. Esto, al intentarlo sincronizar con la ejecución, nos es un problema debido que la velocidad de ejecución del simulador (sus frames por segundo), depende de la capacidad del ordenador, por tanto, si no se dispone de un ordenador con buenos recursos, los tiempos no irán sincronizados, pues si un movimiento del robot tarda 1 segundo para ir de un punto a otro en 60fps, tardará el doble si va a 30fps y, como el tiempo de espera depende del tiempo del ordenador y no el de la ejecución, significa que se han de colocar tiempos mayores y, por consecuencia, esperas más elevadas en ordenadores que tienen peores recursos. En nuestro caso nuestra media de fps rondaba los 13.

Posibles riesgos y plan de contingencia

Risk #	**Description**	**Probability** (High/Medium/Low)	**Impact** (High/Medium/Low**)**	**Contingency plan**
1	Que la superficie de la ficha no sea completamente plana y la válvula de vacío no coja correctamente la ficha	Low	High	1- Utilizar fichas personalizadas 2- Válvula con más potencia
2	Válvula sin potencia para coger fichas (al levantarlas caen por su propio peso)	Low	High	Cambiar válvula
3	Giros bruscos hacen caer las fichas y / o cartas	Medium	Medium	1- Movimientos más lentos 2- Reducir el movimiento al llegar al punto de destino
4	Cartas se doblan debido a que las personas tienen contacto con ellas y hace que la válvula no las coja correctamente	Medium	Medium	Utilizar válvula con más potencia
5	Relacionado con el punto 4: las cartas se doblan y como consecuencia el aparato de darle la vuelta a las cartas deja de funcionar	Low	High	Cambiar diseño aparato que le da la vuelta a las cartas
6	Confundir ficha con algún decorado de la mesa	Low	Medium	Poner un manto para aumentar contraste
7	Que al dejar las fichas en la caja, reboten y se salgan como consecuència	Low	Low	Ajustar distancia con sensores

Referencias

Este Proyecto ha estado inspirado por los siguientes proyectos de internet:

#1 https://github.com/OriolMoreno/C.A.R.L.E.S

#2 https://www.banggood.com/DIY-4DOF-Pump-RC-Robot-Arm-Educational-Kit-With-Metal-Digital-Servo-For-Arduino-p-1424994.html?utm_campaign=7858775_1424994&utm_content=1087&p=CS12047858752016125&cur_warehouse=CN