

Realidad Virtual

Ingeniería en Mecatrónica

Facultad de Ingeniería - Universidad Nacional de Cuyo

Pizarra Virtual Interactiva

Lucas Trubiano, Renzo Guarise

Julio de 2022, Mendoza, Argentina

Resumen	3
Introducción	4
Desarrollo	5
Tecnologías utilizadas	5
OpenCV	5
Threading	6
Pygame	7
Pybluez	7
Código	7
Funcionalidades	9
Interfaz Gráfica	9
Menú de Dibujo Libre	10
Menú de Preguntas y Respuestas	11
Menú de Juegos o Acciones	12
Hardware y Bluetooth	13
Conclusiones	14
Desarrollos Futuros	14

Resumen

En el presente trabajo se realizó una pizarra virtual interactiva con la que se busca agilizar el aprendizaje de manera virtual. El proyecto consiste de un programa en Python que permite capturar el video de la computadora haciendo uso de la librería “OpenCV” y un dispositivo de hardware “el marcador” realizado en el entorno “Arduino”. Este marcado es capturado por el programa de visión artificial, permitiendo así interactuar con una computadora realizando anotaciones, jugando juegos, dibujar, realizar preguntas y respuesta y demás. Se busca con este proyecto dar una herramienta más para hacer más amena la enseñanza virtual.

Introducción

El proyecto desarrollado consta de una ventana donde hemos montado una pizarra virtual en la cual podremos visualizar una serie de contenidos predefinidos y vamos a poder interactuar con ese contenido para generar una mejor experiencia a la hora de trabajar con ese contenido. Esta pizarra que hemos desarrollado es un prototipo de lo que se podría llegar a hacer para mejorar la experiencia.

Motivaciones

La idea detrás de este proyecto es poder mejorar la experiencia tanto de aprendizaje para los alumnos como la de enseñanza para los profesores. Basándonos en la experiencia que tuvimos de educación a distancia durante la pandemia y pensando en nuevos modelos de educación híbrida. Buscamos nuevas formas de poder interactuar entre profesores, alumnos y el contenido; para así mejorar la capacidad de atención y concentración, la apropiación del conocimiento y el dinamismo de este tipo de actividades de enseñanza como clases, charlas, talleres, etc. Y pensando también en que con las herramientas y tecnologías que tenemos hoy en día las soluciones de educación del futuro no se van a parecer en nada a lo que conocemos hoy en día. Con este proyecto buscamos explorar nuevas formas de educar y que sea un puntapié para empezar a crear nuevos modelos de educación.



Figura 1. Imagen de motivacion.

Desarrollo

Para el desarrollo de este proyecto empezamos con un prototipo de reconocimiento de video a través de la librería “OpenCV” para Python y luego sobre este script se fueron creando distintas funcionalidades en la interfaz gráfica. Esas opciones desarrolladas se dividen en configuración del cursor por un lado (color y trazo). Y las distintas acciones como dibujo libre, preguntas y respuestas y algún juego de interacción. Las mismas se fueron desarrollando de manera modular y están parametrizadas en un json, dando lugar para que el día de mañana sólo con modificar un json sea muy fácil agregar nuevas funcionalidades.

Tecnologías utilizadas

OpenCV

OpenCV es una biblioteca de software libre de visión artificial originalmente desarrollada por Intel y actualmente ofrece una serie de herramientas, bibliotecas, códigos y hardware para visión artificial, detección de movimiento, reconocimiento de objetos, entre muchas otras funcionalidades y soporta modelos de Machine Learning e Inteligencia Artificial.

Con esta biblioteca lo primero que hicimos fue trabajar en el reconocimiento de objetos para poder aislar un objeto en particular según unos parámetros preestablecidos. Esto lo desarrollamos en el script “**detection_parameters.py**” (Figura 2) donde podemos testear los parámetros para filtrar un objeto. Como se ve a continuación podemos aislar objetos según su color, luminosidad y sombras.

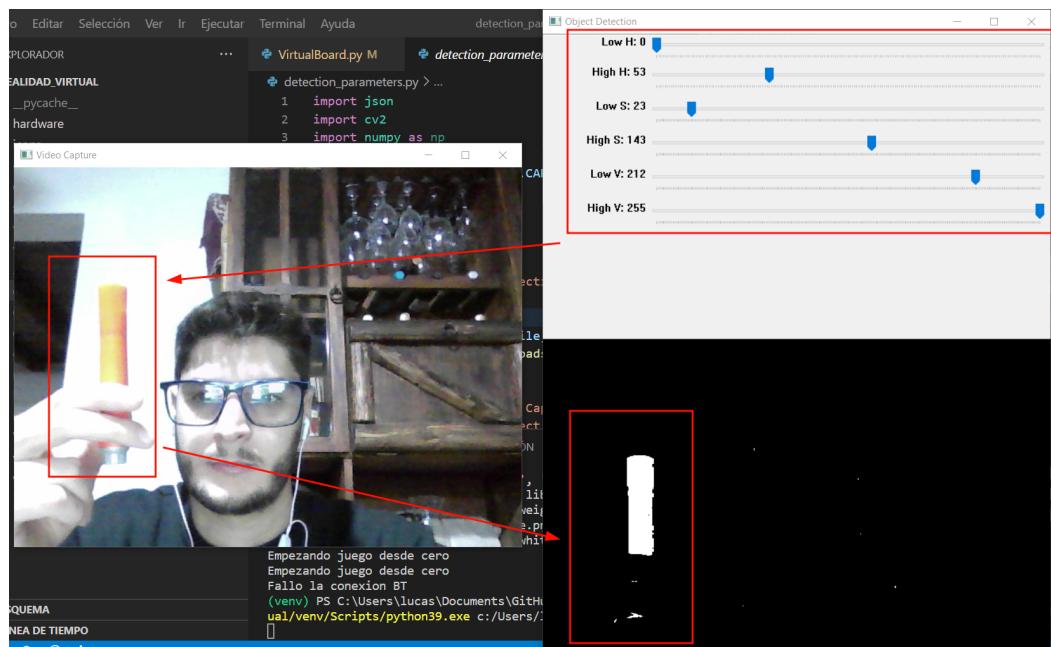


Figura 2. Programa “detection_params.py”.

Una vez seteados los parámetros pasamos al código de “**VirtualBoard.py**” donde se hace todo el procesamiento gráfico.

Lo primero es aplicar los filtros y ver si hay un objeto que podamos trackear. Esto se hace con la función **object_detection** en la cuál aplicamos una máscara a la imagen obtenida y convertida, luego buscamos las áreas o contornos que sean mayores a 800 píxeles cuadrados (para descartar pequeños píxeles o ruidos en la imagen). Una vez encontrados los objetos si pudimos detectar su posición X e Y empezamos a escuchar si el objeto presiona alguna tecla. Y esa información la devolvemos. Es decir, retornamos si hay un objeto detectado, en tal caso su ubicación y la acción que estaba realizando (teclas).

```
def object_detection(self):

    self.mask = cv2.inRange(self.frameHSV, self.object_filter_low, self.object_filter_high)
    # cv2.imshow('rangeColors', self.mask)
    self.mask = cv2.erode(self.mask,None,iterations = 1)
    self.mask = cv2.dilate(self.mask,None,iterations = 2)
    self.mask = cv2.medianBlur(self.mask, 13)
    cnts,_ = cv2.findContours(self.mask, cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:1]

    for c in cnts:
        area = cv2.contourArea(c)
        if area > 800:
            x,y2,w,h = cv2.boundingRect(c)
            x2 = x + w//2

            cv2.circle(self.frame,(x2,y2),self.marker_thickness,self.marker_color,3)
            self.x1 = x2
            self.y1 = y2
        else:
            self.x1, self.y1 = None, None

    # para devolver --> cursor_detected, cursor_position, cursor_key
    if self.x1 == None and self.y1 == None:
        cursor_detected = False
    else:
        cursor_detected = True

    cursor_position = (self.x1,self.y1)

    try:
        if self.BT_Pen.is_connected() and self.BT_Pen.new_data: ...
        else: ...
    except:
        cursor_key = cv2.waitKey(1)

    return cursor_detected, cursor_position, cursor_key
```

Figura 3. Código “object_detection()”.

Luego de detectar el objeto, la otra funcionalidad de la librería OpenCV es dibujar sobre la imagen base (video) para construir la interfaz gráfica.

Threading

En programación, la técnica que permite que una aplicación ejecute simultáneamente varias operaciones en el mismo espacio de proceso se llama Threading. A cada flujo de ejecución que se origina durante el procesamiento se le denomina hilo o

subproceso, pudiendo realizar o no una misma tarea. En Python, el módulo threading hace posible la programación con hilos.

En este proyecto se utilizan hilos para dividir el flujo de ejecución, teniendo así tres hilos: uno para la ejecución del programa principal, otro para la ejecución del módulo bluetooth y otro para el juego simon.

Pygame

Pygame es un conjunto multiplataforma de módulos Python diseñados para escribir videojuegos. Incluye gráficos por computadora y bibliotecas de sonido diseñadas para usarse con el lenguaje de programación Python.

En el programa se utiliza el módulo mixer para la reproducción de sonidos del juego simon.

Pybluez

PyBluez es un módulo de extensión de Python escrito en C que brinda acceso a los recursos de Bluetooth del sistema de manera modular y orientada a objetos. Está escrito para Windows XP (pila Bluetooth de Microsoft) y GNU/Linux (pila BlueZ).

Este módulo es utilizado para realizar la conexión bluetooth entre la computadora y el marcador

Código

El programa se encuentra compuesto por tres clases: MyPlayer, InteractivePen y VirtualBoard. A continuación se detalla su función y sus métodos principales.

- **Class MyPlayer(threading.Thread):** Clase para la reproducción de sonidos. Permite la reproducción de sonidos cuando un marcador se posiciona sobre una de las teclas del juego "Simon", si hay un sonido reproduciendo evita su reproducción. Esta clase hereda de la clase Thread para la creación de hilos y agrega la clase mixer para la reproducción de sonidos.
 - **def __init__(self):** Método para inicialización de la clase. Inicializa el hilo y el objeto mixer para la reproducción de sonidos.
 - **def play(self,sound_name: str=None):** Método principal el cual es llamado por la clase VirtualBoard para la reproducción de sonidos del juego Simon.
 - **def run(self):** Método que es corrido por el hilo.
 - **def is_playing(self):** Método para verificar si se está reproduciendo sonido.
- **Class InteractivePen(threading.Thread):** Clase para realizar la conexión vía bluetooth del marcador. La clase hereda de la clase Thread para realizar el proceso de búsqueda y conexión en un hilo, de esta manera la interacción

bluetooth se realiza de manera paralela al proceso permitiendo que la aplicación corra de manera fluida. Se agrega la clase bluetooth para la conexión.

- **def __init__(self):** Método para inicialización de la clase.
- **def initialize(self):** Busca el dispositivo y realiza la conexión requerida vía bluetooth.
- **def run(self):** Método que es corrido por el hilo.
- **def is_connected(self):**
- **def disconnect(self):**

- **Class VirtualBoard():** Clase principal del programa. Agrega la clase cv2, clase de visión artificial que permite el reconocimiento del marcador.
 - **def __init__(self):**
 - **def cursor_simon(self,cursor):** Método para realizar la interacción en el juego "Simon". Detecta el marcador y resalta los colores.
 - **def validate_simon(self,respuesta,consigna):** Verifica la correcta selección de secuencia del juego Simon.
 - **def draw_simon(self,fill="",reprod=True):** Esta función dibuja en la pantalla el juego del simon y también puede pintar algún cuadrante.
 - **def draw_gui(self):** Dibuja la interfaz gráfica. Posiciona los botones en la pantalla.
 - **def object_detection(self):** Realiza la detección del marcador lo que permite la interacción con la pizarra.
 - **def menu_over(self,cursor_position):** Recibe la posición del marcador y la tecla y si está dentro de un menú ejecuta las acciones correspondientes. Dibuja el menú según corresponda, marcar si estoy encima de algo
 - **def menu_action(self,menu, id_menu, cursor_position, cursor_key):** Despliega todo los menús y los posiciona en la pizarra.
 - **def menu_creation(self):** Crea el menú.
 - **def end_board(self):** Termina el loop.
 - **def start_board(self):** Despliega un loop el cual se encarga de darle la interactividad a la pizarra.

Funcionalidades

Interfaz Gráfica

Otro aspecto de la aplicación es la interfaz gráfica de usuario que consta de una serie de menús que están parametrizados dentro de un archivo “**gui.json**” aquí adentro están configurados los menús, su ubicación, logo y nombre. Según el nombre luego se ejecutan distintas acciones dentro de una función llamada **menu_action**. La interfaz se describe en la siguiente imagen.



Figura 4. Interfaz gráfica.

A la **izquierda** se observan **grosor** y **color**, para configurar el cursor y cómo se ve la interfaz.

En la parte **superior-izquierda** encontramos un botón **back**, que es para salir del menú en el cual estemos.

Luego en la parte **superior** tenemos las distintas acciones:

- **Preguntas-Respuestas, Juegos y Dibujo Libre.** Estas son algunas de las varias que se podrían crear.
- Siguiendo con la parte **superior-derecha** encontraremos que podemos **borrar** el lienzo de dibujo o cualquier cosa que hayamos dibujado sobre la imagen.

Finalmente en la parte **inferior-derecha** encontramos el botón de **salir**, para cerrar la aplicación.

Menú de Dibujo Libre

Esta opción permite utilizar la pantalla como un pizarrón pudiendo escribir y realizar dibujos de cualquier tipo. Además permite seleccionar el trazo y color del marcador a utilizar. En la figura 5 y 6 se puede apreciar las opciones de trazo y color respectivamente

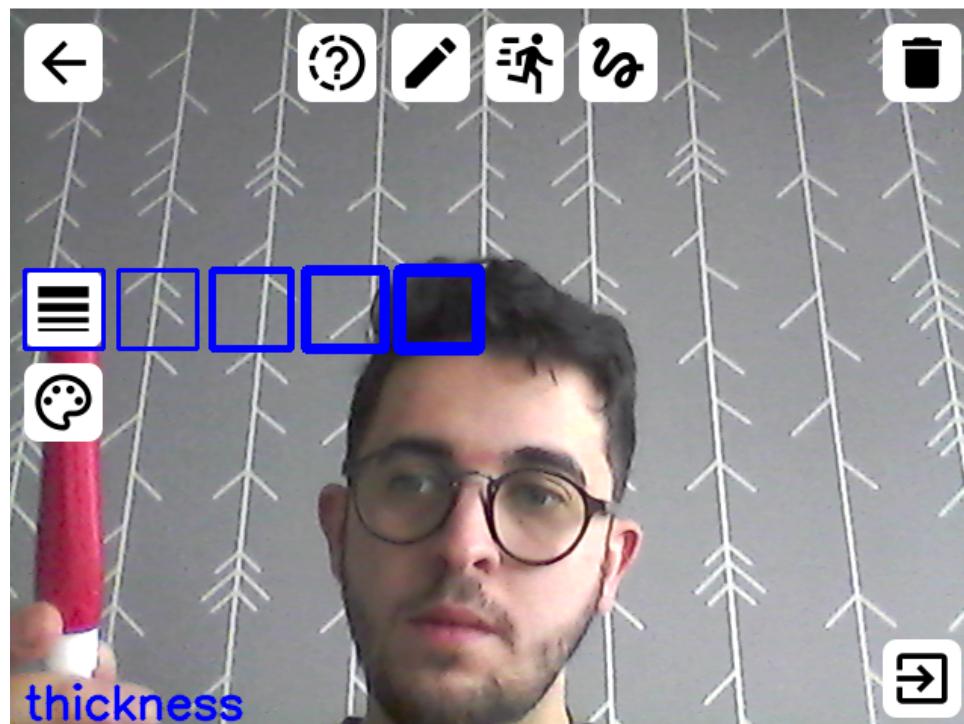


Figura 5. Menú de trazo.

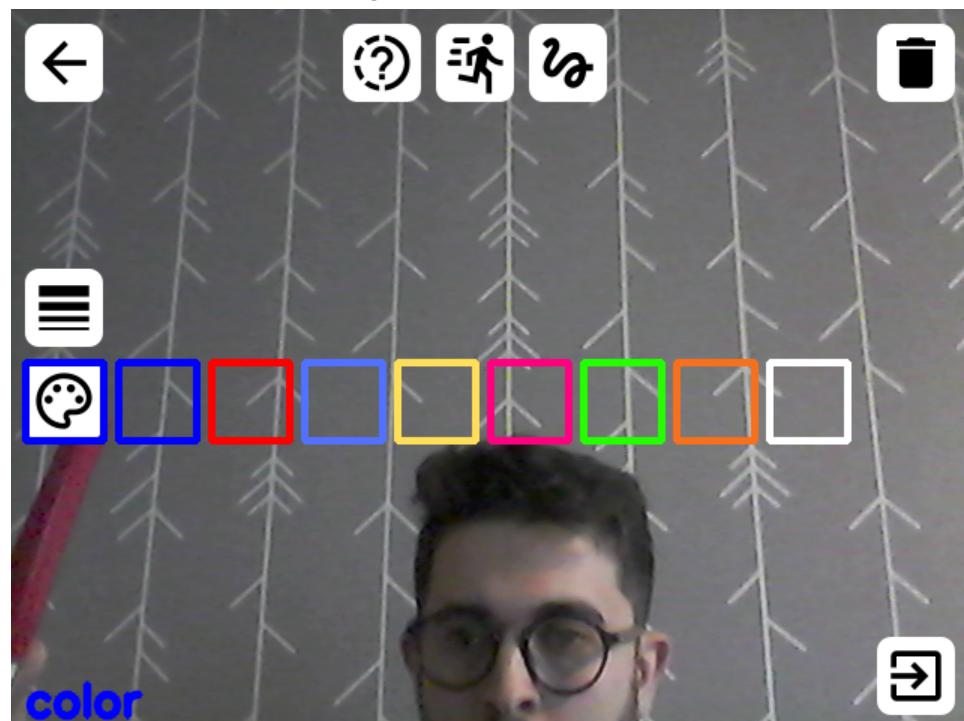


Figura 6. Menú de color.

Menú de Preguntas y Respuestas

La aplicación posee la posibilidad de desplegar un juego de preguntas de respuesta, en este demo el juego consiste en preguntas sobre la traducción al inglés de animales como se ve en la figura 7. El juego es completamente personalizable mediante el archivo “**questions.json**” (en la figura 8 se puede apreciar el contenido del mismo) en el mismo se pueden setear las preguntas y respuesta deseadas.



Figura 7. Juego preguntas y respuestas.

```
{
  "correct_logo": "questions/answers/check-decagram.png",
  "incorrect_logo": "questions/answers/close-circle.png",
  "animales": {
    "question": "Como se dice @animal en Ingles?",
    "images": "questions/animals",
    "max_options": 3,
    "languages": [
      "ingles", "español"
    ],
    "lista": [
      ["bat", "murcielago"],
      ["bird", "pajaro"],
      ["cat", "gato"],
      ["dog", "perro"],
      ["duck", "pato"],
      ["fish", "pez"],
      ["pig", "cerdo"],
      ["rabbit", "conejo"],
      ["snake", "serpiente"]
    ]
  }
}
```

Figura 8. Contenido archivo questions.json

Menú de Juegos o Acciones

En este menú se buscaba desarrollar alguna funcionalidad que ofreciera desafíos cognitivos para el usuario de la aplicación, y así poner a prueba las capacidades de la persona y entrenar alguna función particular del cerebro. Por eso fue que luego de algunas consultas nos propusimos a desarrollar virtualmente el juego del Simon.



Figura 9. Imagen ilustrativa del juego.

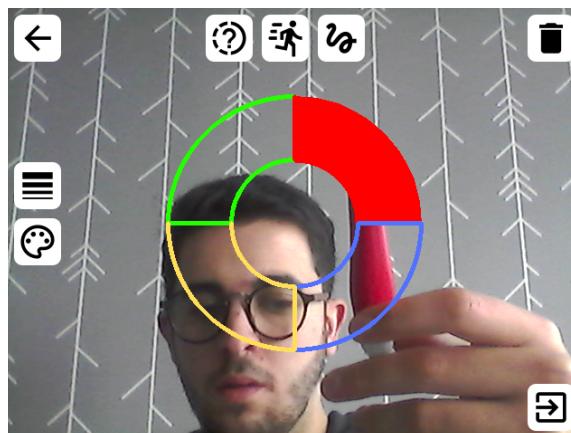


Figura 10. Imagen del juego desarrollado.

Este juego consiste en 4 colores (distribuidos en forma de anillos divididos en 4 cuadrantes) y donde van apareciendo secuencias de sonidos y colores que debemos repetir. A medida que el juego avanza la secuencia es cada vez más larga y hay que tratar de retener lo más posible la misma.

Es un juego relativamente simple ya que consta de 2 etapas: Mostrar la secuencia y luego escuchar la secuencia que ingresa el usuario. Y si ambas coinciden se va agregando un paso más a esta secuencia. Además al ir mostrando la secuencia vemos una serie de combinaciones entre colores y sonidos.

Este juego no tiene fin sino que la idea es ver hasta dónde puede llegar el usuario sin equivocarse y logrando retener en orden correcto la secuencia.

Hardware y Bluetooth

Respecto al apartado de hardware de este proyecto, hemos analizado qué objetos podrían ayudarnos a interactuar con esta pantalla. En ese análisis vimos que lo más viable y cómodo era tener un pequeño marcador, el cuál fuera detectable por la cámara y que a su vez nos permitiera ejecutar acciones en la pantalla. Por lo cual decidimos incorporar un módulo de bluetooth al marcador para así poder disparar eventos como: apretar botones para realizar, activar o desactivar distintas acciones en la pantalla.

Así nuestro dispositivo consta de un pequeño microcontrolador que se encarga de leer eventos que provienen de presionar botones. Luego ese evento es transmitido por bluetooth a la computadora y adicionalmente este objeto debe tener un color particular y área que sea fácilmente detectable por la cámara. Adjuntamos un diagrama de cómo se comunican el hardware con el software, en una forma esquemática.

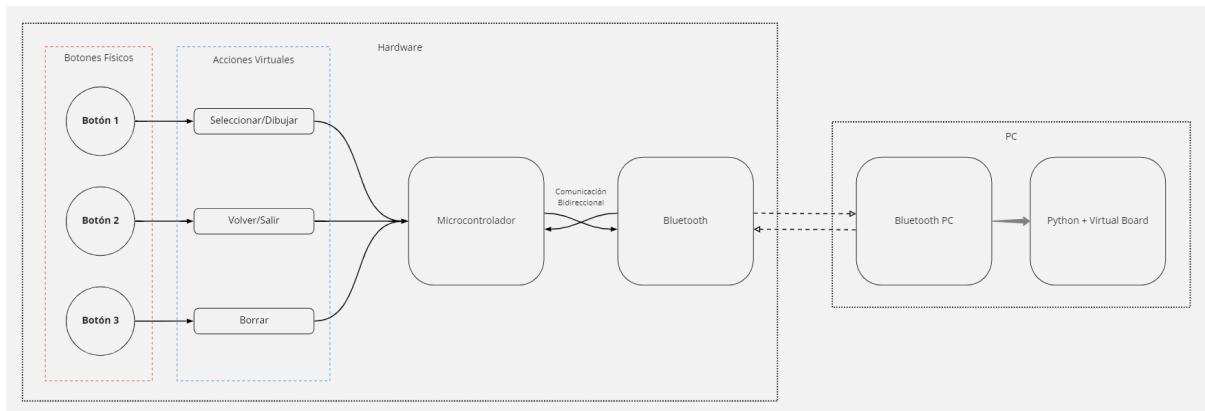


Figura 11. Esquema funcional del hardware

En cuanto al software utilizado en el microcontrolador es el siguiente y consta de una etapa de configuración de pines y puestos; y otra con la lógica funcional del dispositivo.

```
// Pines de comunicación BT
const int rxpin = 2;
const int txpin = 3;

// Variable para almacenar comunicaciones
char k = 'A';

// Creamos bluetooth
SoftwareSerial bluetooth(rxpin, txpin);

// Pines de los pulsadores
const int button1pin = 8;
const int button2pin = 9;
const int button3pin = 10;

void setup()
{
    // Configuramos pines de entrada
    pinMode(button1pin, INPUT_PULLUP);
    pinMode(button2pin, INPUT_PULLUP);
    pinMode(button3pin, INPUT_PULLUP);

    // Puerto serial listo!!
    Serial.begin(9600);
    Serial.println("Serial ready");

    // Puerto bluetooth listo!!
    bluetooth.begin(9600);
    bluetooth.println("Bluetooth ready");
    Serial.println("Bluetooth ready");
}

void loop()
{
    // Transmisión de datos bidireccional
    if (bluetooth.available()){
        Serial.write(bluetooth.read());
    }
    if (Serial.available()){
        bluetooth.write(Serial.read());
    }

    // Leer pulsadores y enviar
    int button1state = digitalRead(button1pin);
    if (button1state == LOW) {
        Serial.println("1");
        bluetooth.println("1");
    }

    int button2state = digitalRead(button2pin);
    if (button2state == LOW) {
        Serial.println("2");
        bluetooth.println("2");
    }

    int button3state = digitalRead(button3pin);
    if (button3state == LOW) {
        Serial.println("3");
        bluetooth.println("3");
    }

    // Esperar unos milisegundos para no sobrecargar
    delay(10);
}
```

Fig 12. Código BT en hardware de Arduino.

En cuanto al código del lado de Python que recibe estos eventos, el código es el siguiente para la parte de inicialización, es decir, el bloque de código que busca y elige los dispositivos de confianza.

```

with bluetooth.BluetoothSocket() as self.sock: #bluetooth.RFCOMM

def initialize(self):
    # Iniciamos un nuevo thread
    threading.Thread.__init__(self)

    self.status = True

    print("Buscando dispositivos BT...")
    self.nearby_devices = bluetooth.discover_devices()
    nombres = list(map(bluetooth.lookup_name, self.nearby_devices))

    if InteractivePen.auto_recognize_device in nombres:
        print("Encontramos el dispositivo: ",InteractivePen.auto_recognize_device)
        selection = nombres.index(InteractivePen.auto_recognize_device)
        # ya tengo mi dispositivo
        self.bd_addr = self.nearby_devices[selection]

    else:
        print("Dispositivos encontrados:")
        for i, device in enumerate(nombres):
            print(i+1,":",device)

        selection = int(input(" > ")) - 1
        print("Dispositivo seleccionado:", bluetooth.lookup_name(self.nearby_devices[selection]))
        self.bd_addr = self.nearby_devices[selection]

    self.port = 1

    self.start()

    try:
        print("Conectando a:", self.bd_addr)
        self.sock.connect((self.bd_addr, self.port))
        self.connected = True
        print("Dispositivo BT conectado")
    except:
        self.status = False
        self.connected = False
        print("Fallo la conexion BT")

    try:
        while self.status:
            data = self.sock.recv(1024)
            if not data:
                break
            # print("Recibido: ", data)
            data = data.decode("utf-8")
            if data.find("1") >= 0:
                self.new_data = True
                self.data = "b" # mayor prioridad es simular "b"
            elif data.find("2") >= 0:
                self.new_data = True
                self.data = "d" # segundo es "d"
            elif data.find("3") >= 0:
                self.new_data = True
                self.data = "c" # tercero es "c"
            else:
                self.new_data = False
                self.data = ""
            print("BT:",self.data)
    
```

Fig 13. (a) Inicialización de la conexión BT de lado de la PC. (b) Bucle en tiempo real.

Una vez que está configurada la conexión Fig. 13 (a) (todavía no se ha iniciado la misma) hay que proceder a ejecutar un procesamiento en paralelo que esté en tiempo real atendiendo los eventos que vienen del BT. Y el código para hacer esto es el que se muestra en la Fig. 13 (b).

Conclusiones

En el desarrollo del presente proyecto se logró realizar satisfactoriamente la aplicación propuesta obteniendo un proyecto que busca mejorar el aprendizaje de los alumnos de una forma lúdica y didáctica pero también no acontando únicamente su alcance a este si no que se buscó un diseño y desarrollo de software intentando dejar la puerta a posibles nuevos usos y mejoras.

Teniendo en cuenta lo mencionado anteriormente y luego de numerosos testeos hemos encontrado el producto final satisfactorio y acorde a nuestros objetivos.

Desarrollos Futuros

Como desarrollos futuros se propone la extensión del apartado de preguntas y respuestas, el agregado de juegos como dónde está la bolita o juegos de memoria (Figura 11). También se buscará en un futuro la implementación en algún dispositivo más inmersivo como lentes de realidad virtual.



Figura 11. Juego de memoria y donde está la bolita.

Recursos

- <https://github.com/renzo-guarise/RepositorioRV>
- [Proyecto Final RV - Google Drive](#)
- https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html