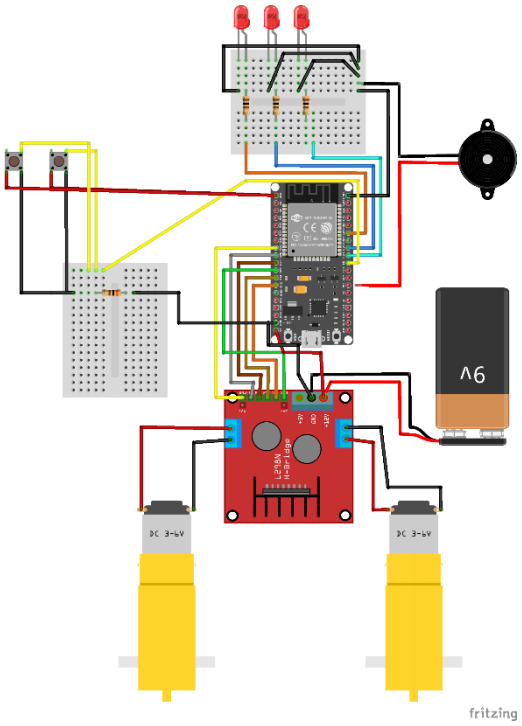
Bluetooth-gesteuerter Mini-AutoScooter mit Kollisionssensorik



Materialien:

- ❑ ESP32-WROOM NodeMCU
- ❑ H-Brücke(L298N): Dual H-Bridge Motor Driver
- ❑ Motor: 2-mal DC-Motor (3-6V)
- ❑ Widerstände: 3-mal 100 Ohm, 1-mal 300 Ohm
- ❑ LED: 3-mal Rot
- ❑ Pieper: Piezo Buzzer
- ❑ Strom: 9V Batterie und Anschluss
- ❑ Berührungserkennung: 2-mal Blech/Alufolie, 2-mal Gummiband, 2-mal schmale Metallröhre (ca. 2cm Durchmesser)
- ❑ MorphBot3D: 3-mal Plastik-Blöcke
- ❑ MorphBot3D: 1-mal H-Brücke-Kreuz
- ❑ MorphBot3D: 2-mal normale Kreuze
- ❑ MorphBot3D: 2-mal bearbeitete Kreuze (T-Form)
- ❑ MorphBot3D: 2-mal Fläche (Klebefläche für das Breadboard)
- ❑ 1-mal Breadboard

In diesem Projekt wird die Idee eines ferngesteuerten Auto-Scooters realisiert. Hierbei geht es um eine Art Spielkonzept, in welchem verschiedene Modelle eines Auto-Scooters nach unserer Bauanleitung gegeneinander antreten können.

Abschnitt 1: Funktionalität einer LED und der Komponenten des Roboters

Eine Leuchtdiode LED ist ein elektronisches Bauteil, das Strom in Licht umwandelt. Im Gegensatz zu einer normalen Glühlampe benötigt eine LED nur sehr wenig Energie und ist daher besonders effizient. LEDs haben eine Anode (+) und eine Kathode (-) – das bedeutet, dass sie nur in eine Richtung Strom leiten.

Damit eine LED nicht beschädigt wird, benötigt sie einen Vorwiderstand, der den Strom begrenzt. Ohne diesen Widerstand würde zu viel Strom fließen, wodurch die LED zerstört werden könnte. Der Widerstandswert wird so gewählt, dass die LED den richtigen Strom erhält, üblicherweise etwa 10-20 mA.

Der Roboter nutzt zwei DC-Motoren, die die Hinterräder antreiben. Diese Motoren benötigen eine H-Brücke, um ihre Drehrichtung zu ändern und ihre Geschwindigkeit zu steuern. Die H-Brücke funktioniert wie eine elektrische Weiche und ermöglicht es, die Motoren vorwärts oder rückwärts laufen zu lassen. Die Geschwindigkeitsregelung erfolgt über Pulsweitenmodulation (PWM), wobei die Motoren je nach Signal langsamer oder schneller drehen.

Zusätzlich besitzt der Roboter einen Piezo-Speaker, der einfache Töne erzeugen kann. Der ESP32 steuert sowohl die Motoren als auch den Lautsprecher und dient als zentrale Recheneinheit des Roboters.

Aufgabe 1

1	**Aufgabe Wissensfragen**
1a	Erkläre mit eigenen Worten, welche Bauteile in dem Roboter verwendet werden. Beschreibe ihre Funktion im Projekt und gib ein weiteres mögliches Einsatzgebiet (falls du eines kennst) für jedes Bauteil an.
1b	Warum benötigen die Leuchten in unserem Roboter einen Widerstand? Warum haben wir uns für einen 100-Ohm-Widerstand entschieden? Welche Farbcodierung hat dieser Widerstand?

Abschnitt 2: Bluetooth Verbindung mit dem PS4-Controller

Bluetooth ist eine Technologie für die drahtlose Kommunikation zwischen Geräten über kurze Distanzen. Sie ermöglicht den Datenaustausch zwischen verschiedenen elektronischen Geräten ohne physische Verbindung. In unserem Fall nutzen wir Bluetooth, um eine Verbindung zwischen einem ESP32-Mikrocontroller und einem PS4-Controller herzustellen.

Der ESP32 ist ein leistungsfähiger Mikrocontroller mit integriertem Bluetooth, der sich hervorragend für unser Projekt eignet. Durch die Verbindung mit einem PS4-Controller können wir dessen Eingabemöglichkeiten wie Buttons, Joysticks und Bewegungssensoren für unsere eigenen Projekte nutzen.

In dem beigefügten Code-Beispiel ist eine mögliche Grundlage gegeben, wie sich im Allgemeinen ein Projekt mithilfe einer Bluetooth-Verbindung zu einem PS4-Controller umsetzen lässt (Code-Beispiel-1). Hierbei sind bereits verschiedene Methoden enthalten:

- `onConnectedController`
- `onDisconnectedController`
- `processControllers`

Diese sind ausschließlich für die Verbindung mit dem Controller zuständig und können für dieses Projekt unverändert bleiben.

Entscheidend für unseren Kontext ist vor allem die Methode `processGamepad`() und `loop`().

Hinweis:

- Vor dem Laden des Code-Beispiels müssen alle Schritte der Installationsanleitung erfüllt sein. Außerdem befindet sich eine genauere Erklärung zu dem Codebeispiel in den Kommentaren im Code.
- Um den Controller mit dem ESP32 verbinden zu können, muss dieser im Pairing Modus sein. Dieser wird mit den Tasten „PS + Share“ gestartet. Im Anschluss blinkt die LED schnell doppelt. Bei weiteren Problemen mit der Verbindung ESP32 und Controller neustarten.
- Der PS4-Controller muss geladen sein.

Aufgabe 2

2	**Herstellen einer Bluetooth Verbindung mit einem PS4-Controller**
2a	Ladet das Beispiel zur Herstellung einer Bluetooth Verbindung zum PS4-Controller auf euren ESP32. Prüft, ob bei dem Drücken der Taste X auch eine entsprechende Ausgabe im Seriellen-Monitor erscheint.
2b	Modifiziert das Programm so, dass eine Ausgabe kommt, sobald die Tasten L2 *(throttle)* und R2 *(brake)* gedrückt wurden

Abschnitt 3: Steuerung des Fahrzeugs

Um die Steuerung des Autoscooters umzusetzen, müssen die Motoren zunächst an die H-Brücke angeschlossen werden. Anschließend wird die H-Brücke mit dem ESP32-Mikrocontroller verbunden. Wie die Bauteile korrekt verbunden werden, kann dem Schaltbild auf Seite 1 entnommen werden.

Bevor die Steuerung im Code implementiert wird, sollte das Fahrzeug so weit zusammengebaut sein, dass es fahrbereit ist. Dafür müssen der ESP32, die H-Brücke und die Motoren an den Klötzen befestigt und die vordere Rolle angebracht werden. Auch dazu hilft das Bild des Fahrzeugs als Referenz.

Für die Steuerung im Code wird der bestehende Code, der die Verbindung zwischen dem ESP32 und dem PS4-Controller herstellt, erweitert. Dies umfasst die Ergänzung einer Hilfsmethode, die die Motoren aktiviert, sowie die Anpassung der processGamepads()-Methode durch zusätzliche if-Statements. Diese prüfen, ob bestimmte Tasten gedrückt wurden, und führen entsprechend den passenden Code aus. Dazu wird ein weiteres Codebeispiel zur Orientierung bereitgestellt.

Die Bluepad32-Bibliothek bietet zahlreiche Funktionen für die Verbindung und Nutzung eines PS4-Controllers. Im folgenden Abschnitt sind vor allem die Funktionen zum Auslesen der Tasten relevant.

Grundprinzip der Steuerung:

- Überprüfen, welche Tasten gedrückt sind.
- Je nach Eingabe werden die Motoren entsprechend gesteuert.
- Die Prüfung der Tasten erfolgt in der loop()-Funktion, um den aktuellen Status kontinuierlich abzufragen.

Aufgabe 3

3	**Implementieren der Steuerung**
Aufgabe 3a	Fügt das Codebeispiel in euren bestehenden Code ein und überprüft, ob das Fahrzeug tatsächlich vorwärts fährt. (Beachtet dabei, dass die processGamepad-Methode nicht komplett ersetzt wird. Stattdessen soll die bereits vorhandene Methode erweitert werden.) **Hinweis:** Falls beim Testen des Codebeispiels die R2-Taste (throttle()) nicht wie geplant das Fahrzeug vorwärtsfahren lässt, sondern rückwärtsfahren bewirkt, müssen die Motorkabel (in1, in2) umgesteckt werden.
Aufgabe 3b	Verwendet das Codebeispiel, um die processGamepad()-Methode zu erweitern und die Funktionalitäten für Vorwärts- und Rückwärtsfahren zu implementieren. **Hinweis:** Um die Erweiterung umzusetzen, ergänzt die moveMotors(bool vorwaerts, bool rueckwaerts)-Methode so, dass sie Rückwärtsfahren ermöglicht. Passt außerdem die processGamepad()-Methode an, sodass nicht nur geprüft wird, ob "R2 gedrückt" ist, sondern auch, ob "L2 gedrückt" ist.
Aufgabe 3c	Nun, da der Autoscooter vorwärts und rückwärts fahren kann, soll als nächstes die Lenkung implementiert werden. Für die Lenkung spielt das Auslesen der Funktion ctl->AxisX() eine zentrale Rolle. **Tipp:** Prüft zunächst, ob die Eingabe für Vorwärts- oder Rückwärtsfahren erfolgt. Anschließend kontrolliert ihr, ob der Joystick nach links, rechts oder in der Neutralstellung (nicht bewegt) ist. Ruft basierend auf diesen Eingaben die moveMotors-Methode mit den entsprechenden Parametern auf.

Programmierbefehle für Vorwärts und Rückwärts Fahren:

Befehl	`ctl->throttle()`
Parameter	Keine Parameter
Rückgabewert	Die throttle-Funktion liefert den Wert der R2-Taste zurück. Der Rückgabewert liegt im Bereich von 0 (nicht gedrückt) bis 520 (vollständig gedrückt).

Beispiel	`1 if (ctl->throttle() > 10) {`
	`2  // Aktion ausführen`
	`3 }`

Befehl	`ctl->brake()`
Parameter	Keine Parameter
Rückgabewert	Die brake-Funktion liefert den Wert der L2-Taste zurück. Der Rückgabewert liegt zwischen 0 (nicht gedrückt) und 1020 (vollständig gedrückt).

Beispiel	`1 if (ctl->brake() > 10) {`
	`2  // Aktion ausführen`
	`3 }`

Programmierbefehle für Lenken:

Befehl	`ctl->AxisX()`
Parameter	Keine Parameter
Rückgabewert	Die AxisX-Funktion gibt die Auslenkung des rechten Joysticks zurück. Die Werte reichen von -520 (Joystick vollständig nach links gedrückt) bis 520 (Joystick vollständig nach rechts gedrückt), wobei 0 den Ruhelagewert darstellt.

Beispiel	`1 if (ctl->AxisX() > 10) {`
	`2  // Aktion ausführen`
	`3 }`

Hinweis:

Beim PS4-Controller kann es vorkommen, dass der Ruhelagewert von ctl->throttle(), ctl->brake() und ctl->AxisX() nicht bei 0 liegt, sondern beispielsweise bei 5.

Um zu verhindern, dass das Fahrzeug fälschlicherweise konstant geradeaus fährt, sollte ein Schwellenwert festgelegt werden, ab dem eine Taste als tatsächlich gedrückt gilt.

In unseren Code-Beispielen wird ein Wert von „10“ verwendet, ab dem eine Taste als gedrückt angesehen wird.

Codebeispiele

Beispiel Code für Vorwärts Fahren

```
// Hilfsmethode moveMotors:
// Beispiele:
// - moveMotors(true, true)  => geradeausfahren
// - moveMotors(false, false) => rückwärtsfahren
void moveMotors(bool linksVorwaerts, bool
rechtsVorwaerts) {
    if (linksVorwaerts && rechtsVorwaerts) {
        digitalWrite(in1, HIGH);
        digitalWrite(in2, LOW);
        digitalWrite(in3, HIGH);
        digitalWrite(in4, LOW);
    }
}

// processGamepad prüft Controllereingaben.
// Wird in loop() kontinuierlich für jeden
Controller aufgerufen.
void processGamepad(ControllerPtr ctl) {
    // R2-Taste auslesen
    if (ctl->throttle() >= 10) {
        moveMotors(true, true);
    }
}
```

Codebeispiele

Beispiel Code für Motoren anschalten	

```
1  // Linker Motor: Vorwärts
2  digitalWrite(in1, HIGH);
3  digitalWrite(in2, LOW);
4
5  // Rechter Motor: Vorwärts
6  digitalWrite(in3, HIGH);
7  digitalWrite(in4, LOW);
8
9  // Linker Motor: Rückwärts
10 digitalWrite(in1, LOW);
11 digitalWrite(in2, HIGH);
```

Abschnitt 4: Treffererkennung am Fahrzeug

Der Mikrocontroller kann an seinen digitalen Pins nicht nur Spannungen ausgeben, sondern auch einlesen. Dies funktioniert mit der Funktion digitalRead(pin) und liest den Wert an dem im Parameter angegebenen Pin, entweder HIGH (1) oder LOW (0). Um den Pin als Eingang festzulegen, verwendet man die Funktion pinMode(pin, mode) im setup, wobei man den mode auf INPUT setzt.

Um dies beispielsweise mit einem Taster zu realisieren, schließt man den Taster an der einen Seite mit einer Spannungsquelle an und verbindet den Taster dann auf der anderen Seite mit einem Widerstand, der zu GND führt und mit einem digitalen Pin. Der Widerstand wird auch als „PULLDOWN-" Widerstand bezeichnet, da er die Spannung am Eingangspin immer auf 0V „herunterzieht". Dadurch liest der digitale Pin den Wert LOW, solange der Taster nicht gedrückt ist. Wird der Taster nun gedrückt, schließt sich der Stromkreis und der Pin liest ein HIGH.

Ein Treffer soll zählen, sobald die elastisch befestigte Metallröhre so weit eingedrückt wird, sodass sie die Blechwand des Fahrzeugs berührt. Ähnlich wie

bei einem Taster kann man nun das Blech mit der Spannungsquelle 3,3V verbinden und die Metallröhre mit einem digitalen Pin, sowie über einen Widerstand mit GND. Analog zum gedrückten Taster ist der Stromkreis nun geschlossen, sobald die Röhre das Metall berührt, wodurch der Pin ein HIGH bzw. 1 liest. Somit kann man mit digitalRead(beruhrungskabel) prüfen, ob ein Treffer stattgefunden hat.

Aufgabe 4

4	**Berührungserkennung**
4a	Ist das Fahrzeug noch am Leben und wird getroffen, sollen die Leben um 1 verringert werden. Damit man nach einem Treffer für 3 Sekunden nicht noch einmal getroffen werden kann, muss der Zeitpunkt der letzten Berührung gespeichert werden. Implementiert die dafür nötige if-Abfrage in loop().
4b	Zusatzaufgabe: Konsolenausgabe der Leben Ergänzt den Code, sodass nach einem Treffer die Anzahl der Leben in der Konsole ausgegeben werden, zum Beispiel: Leben: 2 Leben: 1

Um den Zeitpunkt der Berührung in **long** letzteberuhrung zu speichern, verwendet man millis().

Programmierbefehl zum Auslesen der Zeit:

millis();	
Beschreibung	Gibt die Anzahl von Millisekunden zurück, seit das Arduino-Board das aktuelle Programm gestartet hat.
Parameter	keine
Rückgabewert	Anzahl der Millisekunden seit dem Programmstart. Datentyp: unsigned long

Ob das Fahrzeug mindestens 3 Sekunden seit dem letzten Treffer getroffen wurde, wird in dem boolean istfahrzeuggetroffen geprüft:

```
bool istfahrzeuggetroffen = digitalRead(beruhrungskabel) == 1 && millis() - letzteberuhrung > 3000;
```

Für die Konsolenausgabe verwendet man Serial.printf();

Abschnitt 5: Lebensanzeige

Aufgabe 5

Hierzu muss zunächst die bisherige Schaltung um die LEDs und den Pieper erweitert werden (siehe Schaltplan)

5a	**Lebensanzeige und Trefferwarnung durch Speaker** Implementiere eine Funktion, die den aktuellen "Lebenszustand" des Fahrzeugs anzeigt: - Je nach verbleibenden Leben sollen 3, 2 oder 1 LED leuchten. - Bei jeder Berührung soll ein Leben abgezogen werden, und ebenfalls ein kurzer Piepton ertönen. - Wenn keine Leben mehr übrig sind, dürfen keine LEDs mehr leuchten.
5b	**Cooldown, Lebensverwaltung und Animation** Implementiere eine "Cooldown"-Funktion: - Nach einer Kollision soll das Fahrzeug für 3 Sekunden unverwundbar sein (keine weiteren Leben können in dieser Zeit verloren gehen). Programmiere eine Funktion, die das Fahrzeug bei 0 Leben deaktiviert: - Das Fahrzeug darf sich nicht mehr bewegen, wenn alle Leben aufgebraucht sind. Füge eine Reset-Funktion hinzu: - Wenn die "X"-Taste des PS4-Controllers gedrückt wird, sollen die Leben des Fahrzeugs zurückgesetzt werden (auf 3). - Während des Resets sollen die LEDs nacheinander aufleuchten. **Bonus**: Entwickle eine visuelle Animation für das "Nachladen" der Leben (z.B. LEDs blinken in einer bestimmten Reihenfolge).

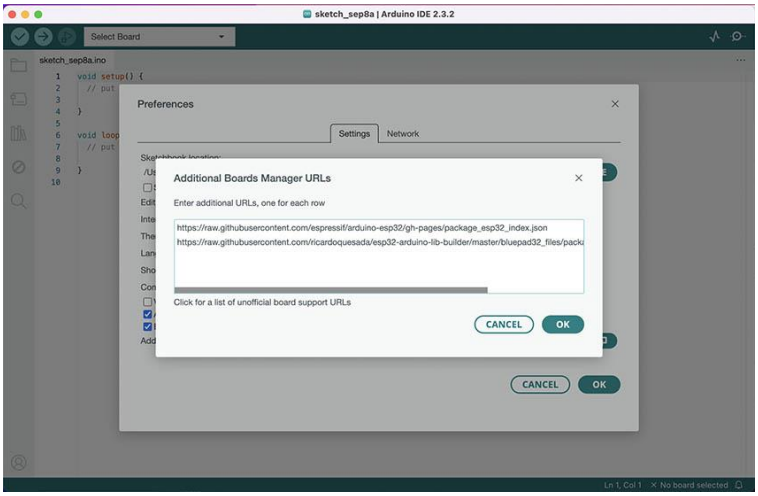
Code-Beispiel-1 (Aufgabe 2)

```
#include <Bluepad32.h>

//  Zur Verwaltung aller verbundenen Contoller
ControllerPtr myControllers[BP32_MAX_GAMEPADS];

//  Diese Methode wird immer aufgerufen, wenn ein neuer Controller angeschlossen wird.
//  Hierbei wird ein Controller in Form eines ControllerPtr-Objektes als Parameter übergeben, welcher im Anschluss der
Liste an
//  verbundenen Controllern angehängt wird.
//
//  Es können bis zu 4 Controller gleichzeitig angeschlossen werden.
void onConnectedController(ControllerPtr ctl) {
    bool foundEmptySlot = false;
    for (int i = 0; i < BP32_MAX_GAMEPADS; i++) {
        if (myControllers[i] == nullptr) {
            Serial.printf("CALLBACK: Controller is connected, index=%d\n", i);
            ControllerProperties properties = ctl->getProperties();
            Serial.printf("Controller model: %s, VID=0x%04x, PID=0x%04x\n", ctl- >getModelName().c_str(),
properties.vendor_id, properties.product_id);
            myControllers[i] = ctl;
            foundEmptySlot = true;
            break;
        }
    }
    if (!foundEmptySlot) {
        Serial.println("CALLBACK: Controller connected, but could not found empty slot");
    }
}

//  Dient dem Trennen der Verbindung mit einem konkreten Controller
void onDisconnectedController(ControllerPtr ctl) {
    bool foundController = false;
    for (int i = 0; i < BP32_MAX_GAMEPADS; i++) {
        if (myControllers[i] == ctl) {
            Serial.printf("CALLBACK: Controller disconnected from index=%d\n", i);
            myControllers[i] = nullptr;
            foundController = true;
            break;
        }
    }
    if (!foundController) {
        Serial.println("CALLBACK: Controller disconnected, but not found in myControllers");
    }
}

//  In diese Methode findet die tatsächliche Verabeitung der aktuellen Daten des Controllers statt.
//  Das übergebene "ControllerPtr"-Objekt beinhaltet alle verfügbaren Informationen über den Controller.
//
//  In unserem Fall können wir hier auf sich ändernde Daten, wie das Drücken der Taste "X", oder das bewegen eines
Joysticks reagieren.
void processGamepad(ControllerPtr ctl) {
//== PS4 X Taste = 0x0001 ==//
  if (ctl->buttons() == 0x0001) {
      // Code, wenn X Taste gedrückt wird.
      Serial.println("X funktioniert!");
```

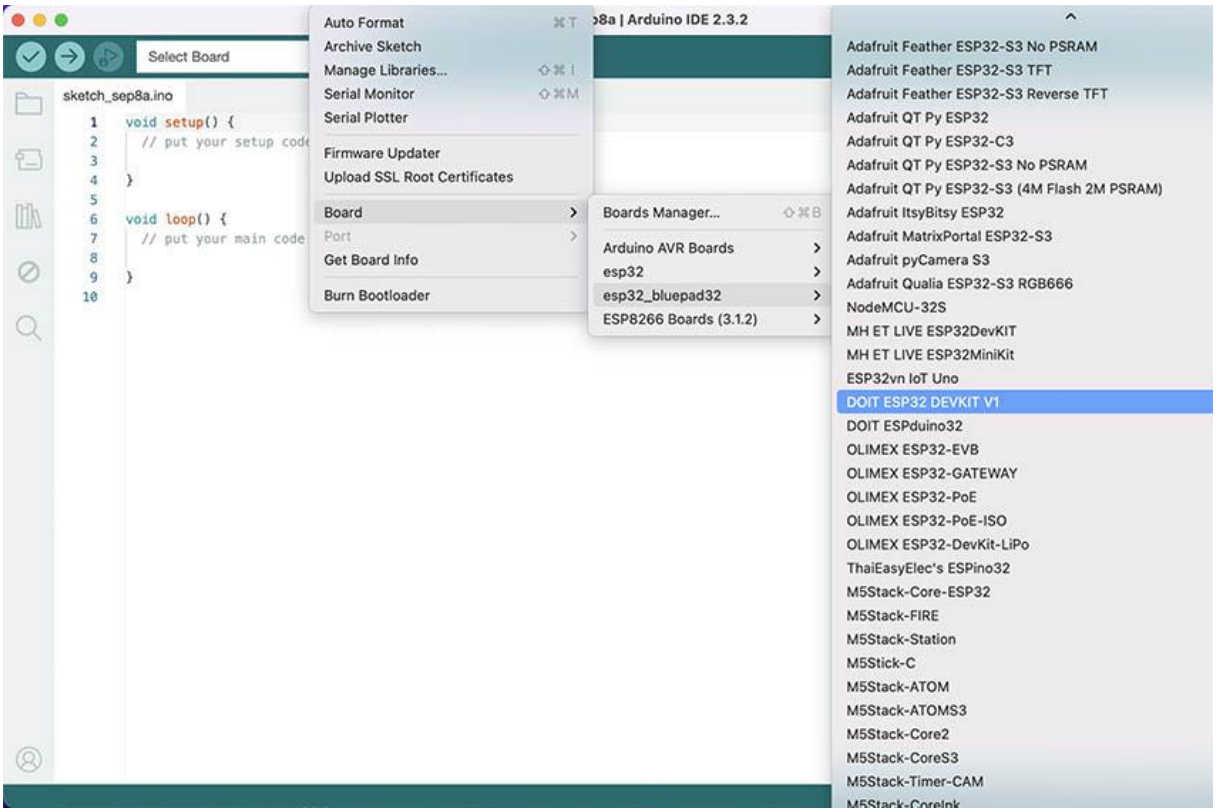
```
  }

  //== LINKER JOYSTICK - LINKS ==//
  if (ctl->axisX() <= -25) {
    // Code, wenn der linke Joystick nach links gedrückt wird.
    // HINWEIS: Ein perfekter Controller sollte, sobald der Joystick nicht mehr gedrückt ist,
    //          Den Wert 0 aufweisen.
    //          Da die meisten Controller aber eine leichte Abweichung von diesem Wert haben,
    //          wird hier erst bei einem Wert ab 25 reagiert.
    //          Ist der Controller Maximal nach Links geneigt, wird der Wert -508 erreicht.
  }

  //== LINKS JOYSTICK - RECHTS ==//
  if (ctl->axisX() >= 25) {
    // Code, wenn der linke Joystick nach rechts gedrückt wird
  }

}

//  Hier werden alle Controller duchgegangen, und für jeden die "processGamepad(ControllerPtr ctl)" Methode aufgerufen.
void processControllers() {
    for (auto myController : myControllers) {
        if (myController && myController->isConnected() && myController->hasData()) {
            if (myController->isGamepad()) {
                processGamepad(myController);
            } else {
                Serial.println("Unsupported controller");
            }
        }
    }
}

void setup() {
    Serial.begin(115200);

    //  Hier wird die Bluetooth-Verbindung gestartet, und die Bluetooth-Addresse des ESPs dargestellt.
    //  Im Weiteren Verlauf des Programms findet die eigentliche Verbindung mit einem PS4-Controller im Hintergrund
statt.
    const uint8_t* addr = BP32.localBdAddress();
    Serial.printf("BD Addr: %2X:%2X:%2X:%2X:%2X:%2X\n", addr[0], addr[1], addr[2], addr[3], addr[4], addr[5]);
    BP32.setup(&onConnectedController, &onDisconnectedController);
    BP32.enableVirtualDevice(false);
}

void loop() {
    //  Hier wird überprüft, ob neue Daten vom Controller empfangen wurden. Wenn dem so ist wird die Verarbeitung
gestartet ("processControllers()").
    //  Sobalt eine Verbindung besteht werden permanent neue Daten gesendet, auch wenn sich nichts am Contoller verändert
hat.
    bool dataUpdated = BP32.update();
    if (dataUpdated) {
        processControllers();
    }
    delay(150);
}
```

Installationsanleitung für notwendige Bibliotheken

- Installiert zunächst die neuste Version der Arduino IDE (hier Version 2.3.2)
- Unter „File→Preferences“ findet ihr das Feld „Additional boards manager URLs“.
  Fügt hier, in seperaten Zeilen, die Folgenden URLs ein, und klicke „OK“:
  a. `https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json`
  b. `https://raw.githubusercontent.com/ricardoquesada/esp32-arduino-lib-builder/master/bluepad32_files/package_esp32_bluepad32_index.json`



- Unter “Tools→Board→Board-Manager“ installiere folgende Bibliotheken in der jeweils neusten Version. Warte nun, bis die Installation vollendet ist:
  a. *esp32 by Espressif Systems*
  b. *esp32_bluepad32 by Ricardo Quesada*
- Wähle nun unter „Tools→Board→esp32_bluepad32“ das „*DOIT ESP32 DEVKIT V1*“ aus
- Gehe nun auf „Tools→Port“.
  Wenn der ESP32 angeschlossen ist, müsste dieser nun als ein Port (bspw. COM6) aufgeführt werden


Select Board
sketch_sep8a.ino
1 void setup() {
2 // put your setup code
3
4 }
5
6 void loop() {
7 // put your main code
8
9 }
10
Auto Format ⌘T
Archive Sketch
Manage Libraries... ⇧⌘I
Serial Monitor ⇧⌘M
Serial Plotter
Firmware Updater
Upload SSL Root Certificates
Board
Port
Get Board Info
Burn Bootloader
Boards Manager... ⇧⌘B
Arduino AVR Boards
esp32
esp32_bluepad32
ESP8266 Boards (3.1.2)
Adafruit Feather ESP32-S3 No PSRAM
Adafruit Feather ESP32-S3 TFT
Adafruit Feather ESP32-S3 Reverse TFT
Adafruit QT Py ESP32
Adafruit QT Py ESP32-C3
Adafruit QT Py ESP32-S3 No PSRAM
Adafruit QT Py ESP32-S3 (4M Flash 2M PSRAM)
Adafruit ItsyBitsy ESP32
Adafruit MatrixPortal ESP32-S3
Adafruit pyCamera S3
Adafruit Qualia ESP32-S3 RGB666
NodeMCU-32S
MH ET LIVE ESP32DevKIT
MH ET LIVE ESP32MiniKit
ESP32vn IoT Uno
DOIT ESP32 DEVKIT V1
DOIT ESPduino32
OLIMEX ESP32-EVB
OLIMEX ESP32-GATEWAY
OLIMEX ESP32-PoE
OLIMEX ESP32-PoE-ISO
OLIMEX ESP32-DevKit-LiPo
ThaiEasyElec's ESPino32
M5Stack-Core-ESP32
M5Stack-FIRE
M5Stack-Station
M5Stick-C
M5Stack-ATOM
M5Stack-ATOMS3
M5Stack-Core2
M5Stack-CoreS3
M5Stack-Timer-CAM