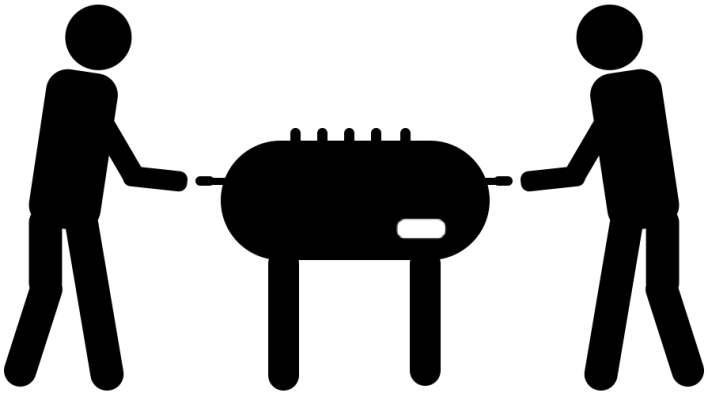
Automatisierung eines Tischkickers



Von [redacted]

Verfasst in Blomberg am HVG

Beim Betreuungslehrer [redacted]

Projektüberblick

Das Folgende Projekt dreht sich um die Automatisierung eines Kickers, um potenzielle Streitfragen, wie wann zählt ein Tor, war das ein Tor, wie viele Punkte gibt ein Tor usw. Aber auch Täuschung und Betrug, durch Manipulation, der analogen Toranzeigen, vorzubeugen. Hierzu habe werden in dem Projekt mehrere Mikrocontroller benutzt, die jeweils andere Aufgaben haben. Eine dieser Aufgaben ist die Torerkennung, eine andere ist Darstellen der Tore auf einer Toranzeige. Beide Teile der Folgenden Projektarbeit funktionieren, wie gewünscht.

Einleitung

Die Idee für das folgende Projekt liegt in dem Gedanken, ein Kickerspiel fairer zu gestalten, indem man die Prozesse der Torerkennung und der Toranzeige automatisiert, da es oftmals zwischen den zwei Spielparteien aufgrund von Toren, deren Zählung und Bepunktung zu Streitigkeiten kommt. Außerdem, da es manchmal vorkommt, dass der Ball vom Tor reflektiert wird. Darüber hinaus kann es ebenfalls durch äußere Beeinflussung dazu kommen, dass die mechanische Toranzeige verrutscht und somit der Punktestand verfälscht wird.

In dem ersten Kapitel des Hauptteils beschäftige ich mich weitergehend mit der Grundidee hinter diesem Projekt. Im nächsten Kapitel beschreibe ich die benutzten Materialien und deren Aufbau. Das dritte Kapitel behandelt dann, wie ich diese eingesetzt habe. Anschließend thematisiere ich die softwaretechnische Lösung für die Torerkennung. Fortlaufend erläutere ich dann die Toranzeige, wie diese hardware- und softwaretechnisch umgesetzt worden ist. Im letzten Kapitel dokumentiere ich dann die praktische Umsetzung des Projekts. Und im Schluss der Projektarbeit erfolgt eine Zusammenfassung des Erreichten und ein Ausblick für die Fortführung des Projekts.

Die benutzten Materialien für das Projekt sind ein Tischkicker, 10 kΩ Fotowiderstände und Widerstände, mehrere Raspberry Pi Pico und zwei Pimoroni Unicorn Packs.

Hauptteil

Kapitel 1: Grundidee zum Projekt

Die Idee für das Projekt liegt, wie schon in der Einleitung erwähnt, in dem Versuch, ein Spiel fairer zu gestalten, durch eine Automatisierung bestimmter Prozesse wie die Erkennung eines Tores, das Zählen der Tore und die unbestechliche Toranzeige, mittels der Technik.
Für die Automatisierung der Torerkennung spricht ferner, dass es manchmal dazu kommt, dass der Ball zwar ins Tor kommt, jedoch zu schnell ist und von der Rückwand des Kickers wieder nach außen, also aus diesem wieder zurück auf das Spielfeld, reflektiert wird.
Hierdurch entstehen meist Missverständnisse bis hin zu Streitigkeiten oder es erfolgt gar der Abbruch des Spieles zwischen den zwei Spielparteien, was zusätzlich das Spielvergnügen beeinträchtigt. Zudem kann es aber auch aufgrund der falschen Bepunktung einzelner Tore zu Missverständnissen kommen, genauso führt auch das Zählen der Tore meistens zu Debatten. Oder auch, wenn jemand (aus Versehen) gegen den Tisch stößt, können die Torpunkte der mechanischen Anzeige verrutschen. Genau so kann man diese aber auch bewusst fälschen, indem man diese zwischendurch nach vorne schiebt oder auch zwei oder drei Punkte weiterschiebt statt einen nach einem Tor.
Um diese und andere Fälle, in denen es wegen eines vermeintlichen oder richtigen Tores zu Streitigkeiten kommt, vorzubeugen, habe ich mir eine Schaltung und ein Programm überlegt, die, mithilfe eines Raspberry Pi Pico W, die Tore erkennt und mit einem Raspberry Pi Pico W und einer LED-Matrix von Pimoroni namens Unicorn Pack [P2023] die Tore anzeigt (Hier zu mehr im Kapitel 3: Softwaretechnische-Lösung).

Kapitel 2: Materialien und deren Aufbau

Die benutzen Materialien sind wie schon erwähnt einen Raspberry Pi Pico W, Pimoroni Pico Unicorn Pack, jeweils für die Tor-Seiten eine blaue Leuchtdiode, einen 2,2 kΩ Widerstand und auch einen 10 kΩ Fotowiderstand.
Der Aufbau des Raspberry Pi Pico W ist in dem Datasheet, einmal direkt von Raspberry [Ras2023], aber auch zweitens auf GitHub [GR2023] zu finden. Aber auch ein Pin-Layout von dem Raspberry Pi Pico W findet man bei Heise-Online. Auch ist wichtig zu wissen, dass auf dem Pico W Mirco Python läuft.

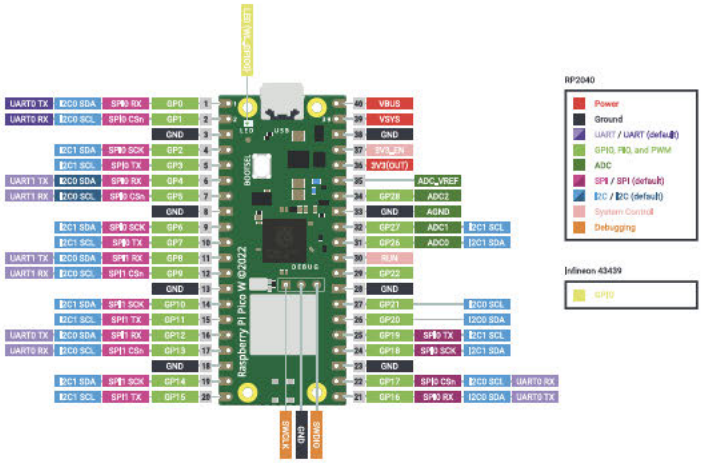


Abbildung 1: Pin-Layout Raspberry Pi Pico W

Genauso wird auch für das Pimoroni Pico Unicorn Pack, direkt vom Hersteller, Pimoroni, ein Datasheet [P2023] bereitgestellt.

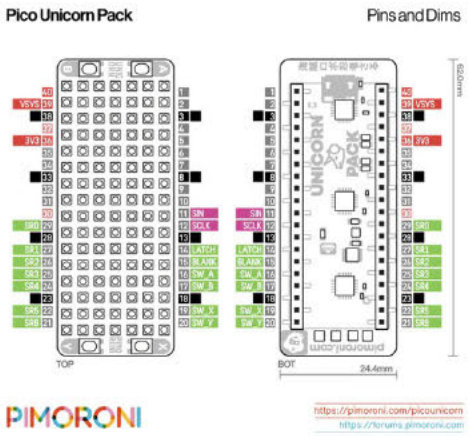


Abbildung 2: Pin-Layout Pimoroni Pico Unicorn Pack

Kapitel 3: Hardwaretechnische Lösung und Aufbau

Bevor man sich die softwaretechnische Lösung des Problems anschauen kann, muss man zuerst eine hardwaretechnische Lösung für das Problem finden. Deshalb schauen wir uns zuerst einmal an, wie sich das Problem bei der Torerkennung lösen lässt und im Folgenden dann, wie ich die Hardware bei der Toranzeige aufgebaut habe.
In meinem Fall musste ich überlegen, mit welcher Methode ich mein Problem lösen wollte. Als Methode zu Erkennung eines Tores wurde sich für eine einfache Lichtschranke entschieden. Hierfür wird eine blaue Leuchtdiode (auch LED genannt) als auch einen 10 kΩ Widerstand und einen 10 kΩ Fotowiderstand benutz. Die Leuchtdiode wurde, direkt gegenüber dem Fotowiderstand montiert.
Der Fotowiderstand und der 10 kΩ Widerstand wurden zu einem Spannungsteiler verbunden. Dieser ist vor allem später für die Software von Vorteil.

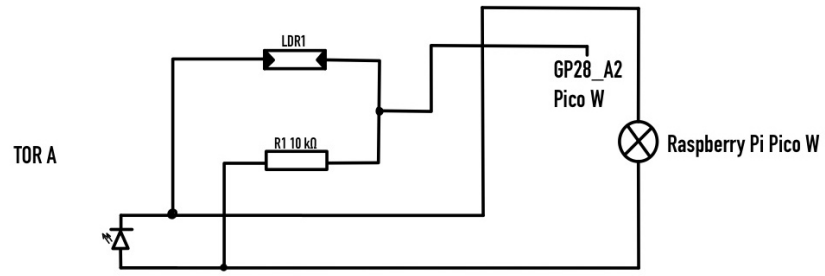


Abbildung 3: Schaltplan Tor A / Torsensor A

Aus dem Schaltplan kann man nun die Schaltung für die eine Torseite herauslesen. Wie man sieht, wird die LED direkt von der Spannungsquelle, hier der Raspberry Pi Pico W, versorgt. Aber auch der Fotowiderstand und der Widerstand R1 sind jeweils mit einer Leitung der Stromquelle angeschlossen. Jedoch gehen hier die freien Enden zusammen zu einer Leitung, die dann wiederum als Eingangsquelle auf den Pin 28_A2 gelegt wird. Diese ermöglicht die Messung. Des Weiteren kann man im gesamten Schaltplan sehen, dass ein Kabel zu einem weiteren Raspberry Pi Pico W und gleichzeitig auch zwei Kabel zu einem Raspberry Pi 4b führen. Diese Kabel dienen der Kommunikation unter den einzelnen Mikrocontrollern, damit diese, nachdem mein Teil des Projekts ein Tor erkannt hat und jeweils über die Kabel ein Signal gesendet hat, ihre Aufgabe im Projekt ausführen können.

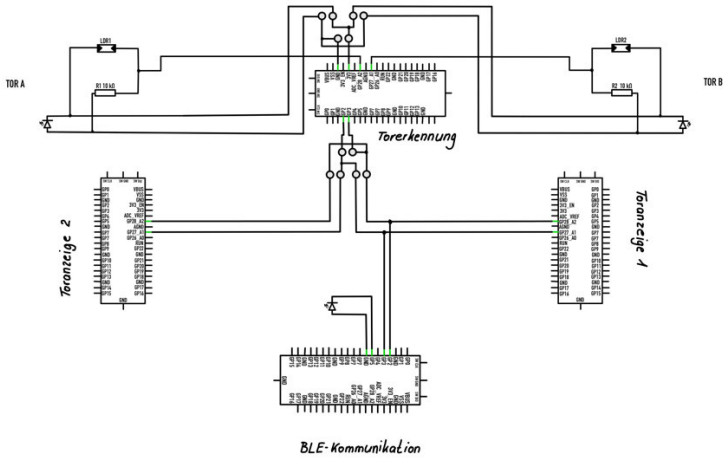


Abbildung 4: Kompletter Schaltplan

Auch kann man erkennen, dass die Torsensoren über einen analogen Pin, Pin-GP28_A2 und Pin-GP27_A1, angeschlossen sind. Die Torsensoren sind die Schaltungen, die jeweils auf den Torseiten zu einer Schaltung zusammengefasst (s. Abbildung 3) sind.

Kapitel 4: Softwaretechnische Lösung

Da man über die Hardware-Lösung informiert ist, kann man sich jetzt der Software-Lösung des Problems widmen. Im ersten Teil dieses Kapitels wird das Programm zur Torerkennung thematisiert und im zweiten Teil dieses Kapitels geht es dann um das Programm der Toranzeige.
Um das Programm der Torerkennung im weiteren Verlauf besser zu verstehen, wird anhand des folgenden PAP (Programm-Ablauf-Plan) die Vorstellung erklärt, sowie was und wie das Programm ausführen soll.
Die zwei zusätzlich benutzen Bibliotheken sind machine [MP2023A] und utime [MP2023B].

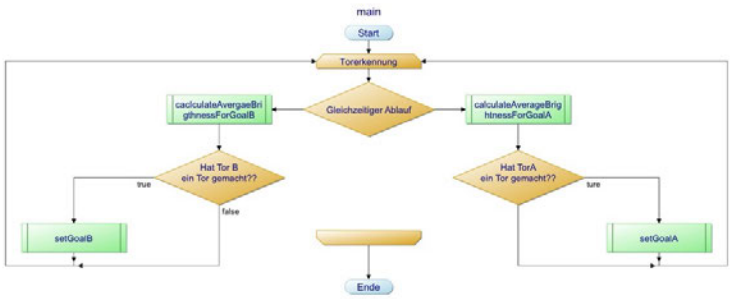


Abbildung 5: PAP-Torerkennung

Der Ablauf des Programmes ist, dass es in einer sich immer wiederholenden Schleife zeitgleich überprüft, ob bei Tor A oder Tor B ein Tor erzielt werde. Dafür berechnet das Programm immer die durchschnittliche Helligkeit. Falls nun bei Tor A oder Tor B ein Tor erzielt, worden sein sollte, führt das Programm den jeweils dafür vorgesehenen Teil des Programms aus.
Da man jetzt im Groben den Ablauf des Programmes verstanden hat, kann man sich das Programm inzwischen an dem Programmcode bzw. Quellcode schrittweise anschauen.

Kapitel 4.1: Kommunikation über GPIO-Pins

Damit die Picos miteinander kommunizieren können, sind die Ausgänge des Picos, auf dem das Torerkennungsprogramm läuft, und die Eingänge der Picos, die die Anzeigen steuern, jeweils als GPIO-Pin [EK2023A] initialisiert worden. Der Pico, auf dem die Torerkennung läuft, sendet, sobald dieser ein Tor erkannt hat, in der Funktion *Ausgabe Tor A oder B (setGoalA/setGoalB)* ein Signal über die Ausgangspins an die Picos der Toranzeigen. In dem der Ausgangspin, welche auf TorA und TorB aufgeteilt sind, zuerst auf null gesetzt wird, zu dem wartet das Programm 500ms [MP2024B]. So dass sicher gegangen werden kann, dass der Pin [MP2024C] Aus ist und kein Signal darüber gesendet wird. Nachdem das Programm 500ms gewartet hat, wird der jeweilige Pin auf High bzw. An gesetzt, wie auch gerade wartet das Programm 500ms, damit der andere Pico genug Zeit hat, diese Veränderung zu erkennen. Zuletzt wird der Pin wieder in den Ausgangszustand, auf null bzw. Aus gesetzt.

Kapitel 4.2: generelle Schleife (while True)

Die while-True-Schleife ist das Herzstück des Programms, da dieses ohne die Schleife nur einmalig laufen würde. In der while-True-Schleife werden jeweils die einzelnen Programmteile des Programms ausführen, darunter die Funktionen zur Berechnung der durchschnittlichen Helligkeit der Torsensoren (beidseitig) als auch die Abfrage, ob diese dann ein Tor erzielt haben.

Kapitel 4.3: Funktion *Berechne durchschnittliche Helligkeit für Tor A und B (caluclateAverageBrightnessForGoal…)*

In diesen Funktionen werden jeweils die durchschnittliche Helligkeit für die jeweiligen Torsensoren A und B gleichzeitig ablaufend in zwei verschieden Funktionen berechnet.

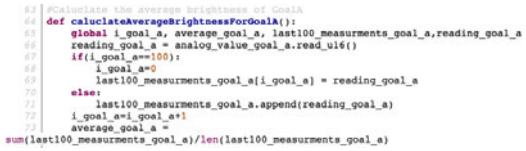


Abbildung 6: Bsp.: Funktion *Berechne durchschnittliche Helligkeit*

Zuerst, in Linie 64, werden die folgenden Variablen als globale Variablen definiert und erstellt, i_goal_a, avergae_goal_a, last100_measurments_goal_a und reading_goal_a. Darauffolgend wird beschrieben, dass die Variable reading_goal_a dem Analogwert des Torsensors B zugewiesen werden. Daraufhin startet eine Abfrage, ob i_goal_a den Wert 100 hat, denn dann soll es den Wert von i_goal_a wieder auf null setzen. Dieser Wert wird dann in das Array last100_measurments_goal_a in die eckigen Klammern eingesetzt (Linie 69), welches dann wiederrum der Variable reading_goal_a zugewiesen wird. Wenn die Abfrage nicht ergibt, dass die Variable i_goal_a gleich 100 ist, wird erneut ausgesagt, dass last100_measurments_goal_a abhängig von reading_goal_a sind (Linie 71). Danach wird, außerhalb der Abfrage, i_goal_a den Wert 1 (Linie 72) und berechne dann den Durchschnittswert für das Tor A, average_goal_a (Linie 73).
Für die Tor-Seite B werden zeitgleich die gerade beschriebenen Schritte durchgeführt.
Fortgehend setzte ich wieder alle Variablen, die ich zum Berechnen der durchschnittlichen Helligkeit brauche, wieder auf null.

Kapitel 4.4: Funktion *Prüfe, ob A/B ein Tor gemacht hat* (checkIfAHasScored…Goal)

In diesen Funktionen wird jeweils gleichzeitig, eine Abfrage gestartet, ob Tor A oder B ein Tor erzielt hat (wie man in dem folgenden Bild sehen kann).

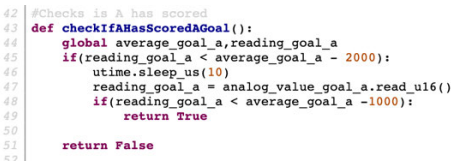


Abbildung 7: Bsp.: Funktion zur *Abfrage, ob bei Tor A ein Tor erzielt wurde*.

Auch hier gebe ich wieder globale Variablen an, und zwar average_goal_a und reading_goal_a (Linie 44). Direkt in der nächsten Zeile starte ich dann die Abfrage, ob der average_goal_a minus 2000 kleiner ist als der ausgegebene Wert von reading_goal_a. Wenn dies zutrifft, warte ich zehn Mikrosekunden (Linie 46), direkt danach weise ich der Variable reading_goal_a den Analogwert des Torsensor A zu (Linie 47). Fortgehend wird eine neue Abfrage, ob der Analogwert des Torsensors kleiner ist als die durchschnittliche Helligkeit minus 1000. Sobald diese Abfrage richtig ist, gebe ich aus, dass die Abfrage richtig ist und dass das Programm weiter in der Systematik vorgehen soll. Ist der Wert der ersten Abfrage (Linie 44) nicht erreicht, sage ich aus, dass diese nicht richtig ist und er das ganze wiederholen soll.
Zeitgleich, wie man auch in der Grundfunktion while True (Kapitel 4.2) sieht, läuft die gleiche Funktion für die Torseite B ab und fragt dort genau das Gleiche ab.

Kapitel 4.5: Funktionen *Ausgabe Tor A und Tor B (setGoalA/B)*

Hier werden die Ausgangs-Pins, Pin 2: GoalA (Pin zur Toranzeige „TorA“), Pin 3: GoalB (Pin zur Toranzeige „TorB“) und Pin 4: Bluetooth-Modul zuerst auf HIGH gesetzt, daraufhin wird dann 500 Millisekunden. Bevor dann die Ausgangs-Pins einmal ausgeschalte bzw. sie auf LOW gesetzt werden und dann wieder 500 Millisekunden wartet und sie wieder auf HIGH gesetzt bzw. eingeschaltet werden.

Kapitel 5: Toranzeige

Kapitel 5.1: Grundidee und hardwaretechnische Lösung

Ferner soll die Toranzeige automatisiert werden, denn, wie schon in der Einleitung erwähnt, kann es dazu kommen, dass eine der Spielparteien seine mechanische Anzeige bewusst einen zu viel verschiebt. Des Weiteren wird durch die Automatisierung die Streitfrage, wie viel ein Tor zählt, oder ob es sich um ein Tor handelt, oder wie viele Tore bisher schon erzielt wurden, verhindert.
Bei der LED-Matrix handelt es sich um, da in der Einleitung erwähnte Pimoroni Pico Unicorn Pack, was als Aufsatz auf den Pico gesteckt wird, womit der Pico die Rechnen

Leistung für die Anzeige, als auch das Torzählen bereitstellt. Hierzu werden die zwei ausgehenden Kabel von dem Torerkennungs-Pico an die Pins 4 und 6 des Pico für die Toranzeige Angeschlossen (s.h. Abbildung 4: Kompletter Schaltplan).
Diesen Aufbau gibt es auf beiden Seiten.

Kapitel 5.2: Softwaretechnische Lösung

Um das Programm nun einfacher zu verstehen, folgt nun der Programm-Ablauf-Plan, kurz PAP. In dem das Programm als, eine Art Diagramm dargestellt wird.

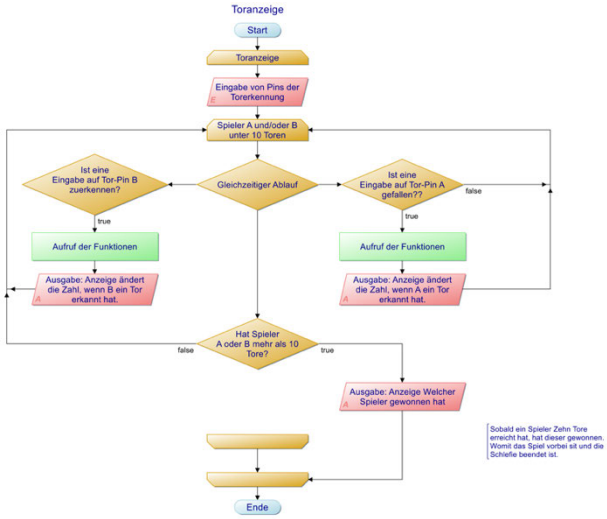


Abbildung 8: PAP-Toranzeige

In dem PAP, lässt sich erkennen, dass es immer eine Abfrage gibt, die überprüft, ob ein Tor gefallen ist. Diese Abfrage ist die Basis des Programmes. Des Weiteren ist erkennbar, dass, sobald, die sich immer wiederholende Abfrage ein Tor erkennt, das jeweilige Tor auf der Anzeige um einen Punkt nach oben gelegt wird. Ferner lässt sich auch registrieren, dass sobald ein Spieler mehr als zehn Tore erzielt haben sollte, das Programm beendet wird und angezeigt wird, welcher Spieler gewonnen hat.

Kapitel 5.2.1: Kommunikation über GPIO-Pins

Wie schon bei der Torerkennung erwähnt, laufen das Programm der Torerkennung und das Programm der Anzeigen auf mehreren verschiedenen Mikrokontrollern (Picos). Um nun kommunizieren zu können, brauchen die Picos eine hardwaretechnische-Schnittstelle, die in das Programm softwaretechnisch Implementiert ist, dazu werden die GPIO-Pins [MP2024C] der jeweiligen Mikrokontrollern genutzt. In dem Programm der Toranzeigen sind die zwei Eingangspins als PULL-DOWN [MP2024D], damit diese erst

reagieren, wenn sie ein Signal von dem andern Pico erhalten [EK2023B]. Dies soll die Fehleranfälligkeit der Toranzeigen minieren, da sie sonst ein Signal erkennen würde, welches nicht vorhanden ist. Dazu läuft eine permanente Abfrage ab, ob sich der Zustand von dem Eingangs-Pin verändert hat. Sobald diese erkennt, dass sich eine Veränderung ergeben hat, wird der Rest des folgenden Programmes ausgeführt.

Kapitel 5.2.2: generelle Schleife (while True)

Wie auch schon bei der Torerkennung kommt eine while-True-Schleife zum Einsatz, da diese wieder der Name schon suggeriert, dass die Schleife eine Endlosschleife ist. Ferner ist das wichtig, da sonst das Programm nur einmal durchlaufen würde und wieder von vorne gestartet werden müsste. In der Schleife wird die Anzeige der Zahlen gestartet, auch wird eine Abfrage gestartet, ob der Durchschnittswert der Toreingangspins sich verändert hat. Wenn das der Fall ist, wird dann die Toranzeige für das jeweilige Tor um einen Punkt ergänzt. Ferner findet simultan eine Abfrage nach dem Punktestand der Spieler statt, denn sobald einer der Spieler zehn Tore erreicht hat, hat dieser das Spiel gewonnen.

Kapitel 5.2.3: Funktion *Zeige Punktestand an* (displayScore*)*

Diese Funktion beschreibt, wie das Display die Anzeige Hand haben soll und ist auch die Basis Funktion für das ganze Programm. Ferner führt sie die einzelnen (noch Folgenden) Funktionen, zu einer Funktion zusammen. Auch beinhaltet sie eine Abfrage, ob Tor A oder Tor B ein Tor erzielt haben.

```
159 #List of the function that combines the functions /displaySinglChar , /drawReactangle
and /displayDots into one function. To simplify the use
160 def displayScore(structPlayerA, structPlayerB):
161     clearDisplay()
162     displaySingleNumber(structPlayerA,0)
163     displayDots(playerA, 1)
164     char_width = len(characterGoals[structPlayerA.goal_count][0])
165     offset_player_b = char_width+3
166     displaySingleNumber(structPlayerB,offset_player_b)
167     if(playerA.has_scored):
168         flashReactangle(playerA.flashInformation,playerA.colourInformation)
169         playerA.has_scored = False
170     elif (playerB.has_scored):
171         flashReactangle(playerB.flashInformation,playerB.colourInformation)
172         playerB.has_scored = False
```

Abbildung 9: Bsp.: Funktion *Zeige Punktestand an*

In dem Bild lässt sich erkennen, dass direkt eine Funktion, und zwar die Funktion clearDisplay, aufgerufen wird. Darauffolgend wird eine weiter Funktion, displaySingleNumber, aufgerufen, dabei wird diese aber mit der Zusatzinformation und die Klasse, structPlayerA, aufgerufen. Diese Information bzw. Klasse gibt dem Programm die nötigen Informationen, die für die Anzeige der Punkt von SpielerA

notwendig sind. Weitergehend wird die Breite der Ziffern gleich, der Liste der Ziffern, definiert als character, mit dem Zusatz diese vorerst nur für SpielerA anzuzeigen, gesetzt. Als nächstes wird definiert, wo der aktuelle Punkte Stand von SpielerB stehen sollen. Fortlaufend werden dann, auch für SpielerB, die Punkte, durch die Funktion, displaySingleNumber, angezeigt. Ebenso wird die Funktion mit Zusatzinformation bzw. die Klasse structPlayerB, ergänzt. Nun wird noch eine Abfrage gestartet, ob SpielerA oder SpielerB ein Tor erzielt haben. Sollte einer der Beiden Spieler, ein Tor erzielt haben wird die Funktion flashRectangle mit dem jeweiligen, dazu gehörigen player.flashInformation Funktion und der player.colourInformation aufgerufen. Die Funktion player.colourInformation, sagt dem Pico in welcher Farbe das Rechteck blinken soll, da SpielerA und SpielerB zwei verschiedene Farben haben. Und des Weiteren wird, wenn einer der Spieler ein Tor erzielt hat, der Punktestand auf der Anzeige geupdatet und um einen Punkt erhöht. Danach wird, damit nicht dauerhaft ein Spieler vermeintlich ein Tor schießt, die jeweilige Funktion wieder auf Flasch gesetzt, bis ein neues Tor erkannt wurde.

Kapitel 5.2.4: Funktion *Setzte bzw. Resettet Display zurück* (cleardisplay*)*

Diese Funktion setzt die Anzeige der Toranzeige, auf allen Pixeln der Matrix auf schwarz und resettet somit die Toranzeige.

Kapitel 5.2.4: Funktion *Zeige einzelne Ziffer* (displaySingleNumber*)*

Diese Funktion beschreibt den Prozess der Darstellung der einzelnen Ziffern.
Hierfür wird in einer For-Schleife, für Zeilen im Bereich der Displayhöhe-2, die Breite der Ziffern gleich, der Liste der Ziffern, definiert als „character“, gesetzt. Daraufhin wird in einer zweiten For-Schleife, für Spalten im Bereich der Ziffernbreite, die Koordinaten x, mit Spalte + 2 + die Variable offset_second_number, und y, mit Zeilen + 1, definiert. Fortgehend wir auch hier eine Abfrage gestartet, die Abfragt, ob einer der Spieler ein Tor erlangt hat, sobald dieses dem Wert Richtig entspricht, werden die jeweiligen Pixel der Matrix überschrieben. Dabei kommt auch die Klasse structPlayer, mit der Farbergänzung, vor.

Kapitel 5.2.6: Klasse *Spieler-Informationen* (structPlayer*)*

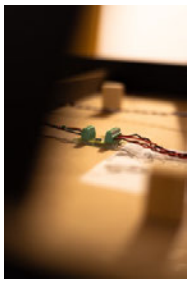
Diese Klasse dient der Vereinfachung, um auf die wichtigsten Informationen, die bisher alle in einzelnen Klassen und Funktionen standen, für die Spieler zu speichern. Dies wird durch die Schaffung der Funktion _init_(self) übernommen, _init_(self) hat hier den Sinn, dass wir die Folgenden Attribute besser nutzen können und diese, bis auf die Farben, immer gleich sind, sowohl bei SpielerA als auch bei SpielerB.
Als erstes wird die colourInformation mit der Klasse structColourInformation, in der die Grundlegenden Farbinformationen für das Projekt gespeichert sind, gleichsetzt. Drauffolgend setzten wir auch flashInformation gleich der Klasse structFlashInformation, die wichtigsten Informationen wie Blinkzeit (flash_time_in_ms) und Blinkanzahl (flash_count) enthält, setzt. Im Weiteren wird dieses gerade erstellte Attribut um die Blinkzeit (flash_time_in_ms) in Millisekunden, um den Wert 250ms, ergänzt. Des Weiteren setzten wir die Blinkanzahl (flash_count) auf vier. Diese Beiden veränderung des Attributs bezwecken jetzt, dass das Rechteck 4-mal in 250ms aufblinkt. Jedoch wird dieser Wert nachdem ausführen, durch die Klasse structFlashInformation, wieder zum Ursprung überschrieben. Auch wird in dieser Klasse der Punktestand (goal_count) definiert auf null.

Kapitel 6: Praktische Umsetzung

Kapitel 6.1: Anfang und theoretische Arbeit

Zu Anfang des Projektes wurde ein einfaches 3D-Modell des Kickers erstellt, um sich dieses bessere Visualisieren zu können, wenn dieser nicht direkt neben einem steht. Außerdem wurde die Torerkennung erst auf einem Steckbrett bzw. einem Bradboard, da man hier schneller Komponenten austauschen kann und es nicht zu Wackelkontakten, auf Grund von Kaltlötstellen oder nicht richtig stecken Kabeln kommen kann.

Kapitel 6.2: praktische Umsetzung/Anbau an den Kicker



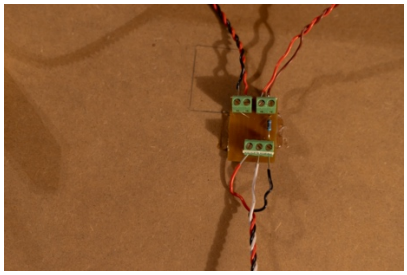
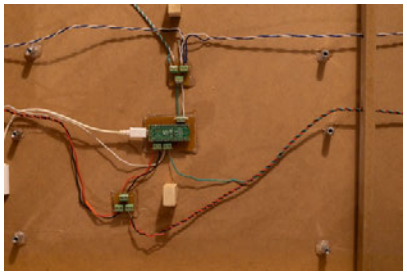


Abbildung 10: Kabelführung & Leiterplatine Torseiten (von der Seite und Oben)

Wie man auf den Bildern erkennen kann, wurde für die Verteilung von Strom, den Signal zum Pico und einem Spannungsteiler mit einem 2,2 kΩ Widerstand und einem 10 kΩ Fotowiderstand, eine Leiterplatine genutzt. Auf dieser sind Klemmen montiert, diese kommen zum Einsatz, für eine einfacherer Fehlerlösung, als auch Variable Spannungsteiler einbauen zu können und falls es zu einem Defekt in einem Modul oder Kabel kommt um dieses schneller und kosten günstiger auszuwechseln.



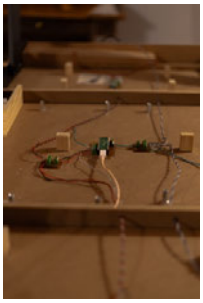


Abbildung 11: Kabelführung & Leiterplatine Pico Torerkennung (von oben und der Seite)

Auf diesen Bildern lassen sich mehrere Module erkennen. Zentral in der Mitte des linken Bildes, ist der Raspberry Pico mit dem Torerkennungsprogramm, wie man gut erkennt bekommen die Kabel der Torseiten wieder zum Pico zurück. Dieser ist, um diesen bei einem Defekt einfacher austauschen zu können auf einer Leiterplatine montiert. Auch bei dieser Leiterplatine sind wieder Klemmen an den Rändern zu erkennen. Diese sind in diesem Fall dazu da, um die Kabel nicht am Pico festlöten zu müssen.
Am Unteren Rand des linken Bildes, ist eine weitere Leiterplatine montiert. Diese dient als Stromverteiler, so dass nicht mehrere Kabel an den Pico gelötet werden müssen und wie bei den anderen Modulen, soll auch dieses wieder zum einfacheren Austausch von defekten dienen. Des Weiteren lässt sich am oberen Bildrand auch ein Verteilungsmodul erkennen, dieses soll die Ausgegebenen Daten, die für die Toranzeige sind, auf die zwei Toranzeigen Verteilen. Zuletzt noch zwei Bilder, auf den der gesamte Kicker dargestellt ist.

Abbildung 12: Gesamter Kicker (zu Hause [Privat])

Abbildung 13: Gesamter Kicker (JuFo)

Schluss

Zusammenfassung und Ausblick

Zusammenfassend lässt sich sagen, dass ich mit meinem Projekt, die am Anfang beschriebene Problematiken behoben habe. Indem ich diese sowohl hardwaretechnisch als auch softwaretechnisch verbessert habe.

Zurzeit befindet sich eine Lösung in Arbeit, um die Toranzeige noch über Bluetooth mit einer Handy-App darzustellen bzw. verknüpfen. In dieser App wird es möglich sein verschiedene Spielmodi auszuwählen, zum Beispiel: Normal (bis Zehn-Tore), Zeitmodus, ein Führungsmodus (bei diesem Modus muss einer der Spieler, wie beim Tischtennis mit zwei Punkten führen und mit diesen Vorsprung Zehn Tore erreichen) oder gar einen Unendlichen Modus. Die erwähnte steht schon im Grundgerüst, ist aber noch nicht einsatzfähig und benötigt noch zeit zum Bearbeiten, weshalb sie nur im Schluss dieser Ausarbeitung erwähnt wird. Im Anhang sind App-Design, wie eine Kurze App-vorschau enthalten.

Weiterführend wäre es möglich noch die Balleingabe zu automatisieren, sodass auch hier ein Streitfaktor, das anderehens des Balles verhindert werden kann. Dies könnte man lösen, indem man eine Art Drehplatte baut. In bzw. auf der die Bälle gelagert werden, bevor sie auf das Spielfeld geschoben werden. Hierzu müsste man die Drehplatt mithilfe eines weiteren Mikrokontroller und einem Motor ausstatten, um diese zu automatisieren.

Literaturverzeichnis

[EK2023A] ELEKTRONIK KOMPENDIUM: „Raspberry Pi Pico: GPIO-Belegung“ in elektronik-kompendium.de, Stand 10.01.2024, letzter Zugriff 20:20 10.01.2024, URL: https://www.elektronik-kompendium.de/sites/raspberry-pi/2611051.htm

[EK2023B] ELEKTRONIK KOMPENDIUM: „Raspberry Pi Pico: GPIO-Eingang beschalten“ in elektronik-kompendium.de, Stand 10.01.2024, letzter Zugriff 18:00 12.03.2024, URL: https://www.elektronik-kompendium.de/sites/raspberry-pi/2611131.htm

[GR2023] GITHUB RASPBERRY: „Raspberry Pi Documentation - Raspberry Pi Pico and Pico W.pdf“ in GitHub.com, Stand 13.5.2023, letzter Zugriff 17:17 02.10.2023, URL: https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html

[GP2023] GITHUB PIMORONI: „pimoroni-pico“ in Github.com, Stand 2023, letzter Zugriff 17:21 02.10.2023, URL: https://github.com/pimoroni/pimoroni-pico/releases/tag/v1.20.6

[MP2024A] MICROPYTHON: „machine — functions related to the hardware“ in docs.micropython.org, Stand 2023, letzter Zugriff 17:16 02.10.2023, URL: https://docs.micropython.org/en/latest/library/machine.html

[MP2024B] MICROPYTHON: „utime – time related functions“ in docs.micropython.org, Stand 2024, letzter Zugriff 17:15 12.03.2024, URL: https://docs.micropython.org/en/v1.15/library/utime.html

[MP2024C] MICROPYTHON: „GPIO Pins“ in docs.micropython.org, Stand 2024, letzter Zugriff 17:30 12.03.2024, URL: https://docs.micropython.org/en/latest/esp8266/tutorial/pins.html

[MP2024D] MICROPYTHON: „class Pin – control I/O pins“ in docs.micropython.org, Stand 2024, letzter Zugriff 17:35 12.03.2024, URL: https://docs.micropython.org/en/latest/library/machine.Pin.html

[P2023a] PIMORONI: „PIM546-Pico-Unicorn-Pack“ in okdo.com, Stand 2023, letzter Zugriff 20:37 30.9.2023, URL: https://www.okdo.com/wp-content/uploads/2021/06/PIM546-Pico-Unicorn-Pack.pdf

[Ras2023] RASPBERRY PI: „datasheets.raspberrypi.com_picow_pico-w-datasheet.pdf“ in Raspberrypi.com, Stand 2023, letzter Zugriff 17:18 02.10.2023, URL: https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf

Abbildungsverzeichnis

Unterstützungsleistungen

[redacted], Ausbildungsleiter, KEB Automation KG, hat mir bei der Beschaffung der folgenden Materialien geholfen, Beschaffung und Bereitstellung von zwei Raspberry Pi Picos W, von einer LED-MATRIX (Pimoroni UnicornPack) und Herstellung (Fräsen) von selbst designten Leiterplatinen explizit für das Projekt.

Anhänge:

Auf den folgenden Seiten sind die wichtigsten Anhänge, des Projekts zu finden.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die im Literaturverzeichnis angegebenen Hilfsmittel verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

_____	_____	_____
Ort	Datum	Unterschrift