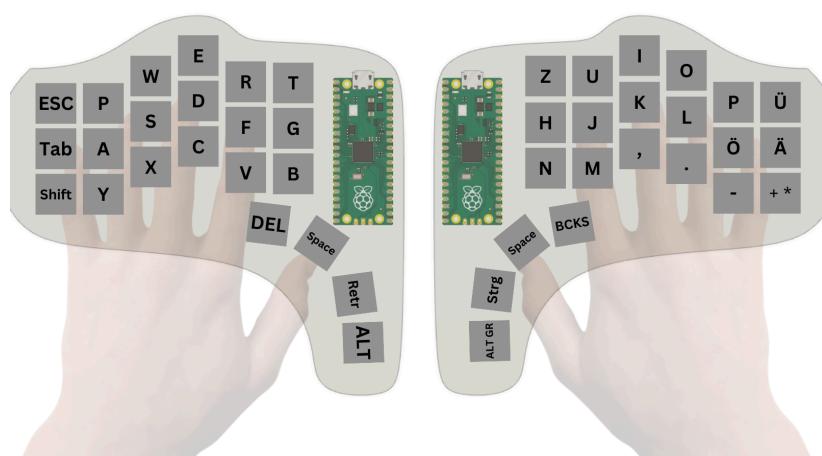


## Entwicklerdokumentation

# Ergonomische Tastatur

**SoSe 2024, Smart Technology**  
*Lukas Krämer*



# Einführung

Die Ursprünge unserer heutigen Computertastaturen liegen in der zweiten Hälfte des 19. Jahrhunderts, als in Amerika die moderne Schreibmaschine patentiert wurde. Nach anfänglichen Experimenten, was die Anordnung der Tasten anging, setzte sich (abgesehen von regionalen Abweichungen, wie zum Beispiel die deutschen Umlaute) im Wesentlichen eine Art von Tastaturen durch: Das sogenannte Staggered Keyboard mit dem QWERTY Layout, bei dem die drei horizontalen Buchstabenreihen in ihrer vertikalen Position zueinander verschoben sind:

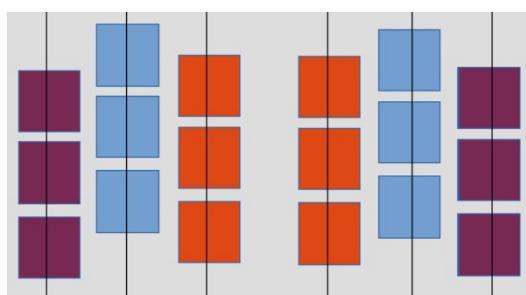


*Bild 1: "Staggered Keyboard" mit QWERTY Layout*

Der Hauptgrund hierfür war mechanisch bedingt, da diese Anordnung die Wahrscheinlichkeit für ein Verklemmen der Schreibmaschinenhebel bei zu schnellem Tippen unwahrscheinlicher machte. Im Folgenden ist bei Bezug auf das Tastaturlayout primär die mechanische Tastenanordnung gemeint und nicht die darauf gedruckten Buchstaben, wenngleich es auch hierfür viele verschiedene Anordnungsmöglichkeiten gibt, die zum Teil erhebliche Ergonomie bzw. Geschwindigkeitsvorteile aufweisen.

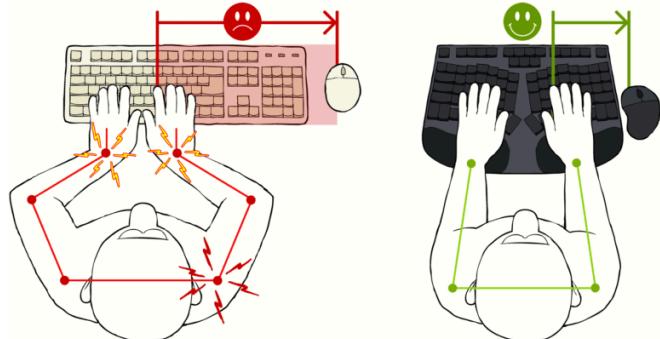
Offensichtlicherweise sind unsere heutigen digitalen Computertastaturen nicht mehr auf diese Weise eingeschränkt, allerdings setzte sich das nunmehr 150 Jahre alte Layout bis heute durch.

Dadurch entstand die Motivation für dieses Uniprojekt, eben diesen Status Quo kritisch zu hinterfragen und darauf basierend eine eigene Tastatur zu entwickeln. Nach Abwägung der Vor- und Nachteile von verschiedenen Tastaturtypen, fiel die Wahl schlussendlich auf eine sogenannte "Columnar staggered" Tastatur, bei der die senkrechten Tastenreihen waagerecht zentriert aber vertikal zueinander verschoben sind, um die unterschiedliche Länge der menschlichen Finger zu kompensieren.



*Bild 2: "Columnar staggered" Tastatur*

Zusätzlich sollte die Tastatur aus zwei Hälften bestehen, um ein unergonomisches Abknicken der Handgelenke zu vermeiden und an die Schulterbreite angepasst werden zu können (Erwachsene können so beispielsweise die beiden Hälften weiter auseinander ziehen als wenn ein Kind die Tastatur benutzt).



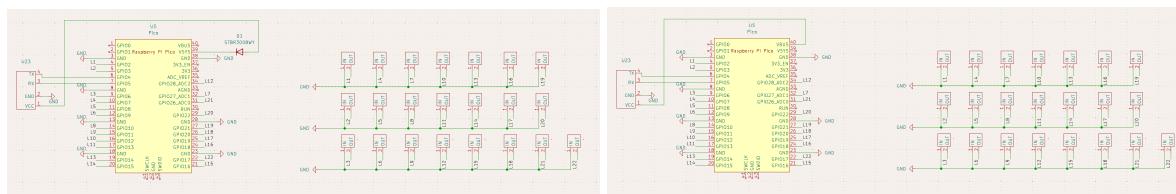
*Bild 3: Vergleich zwischen geradem und eingeknickten Handgelenken*

Zu guter Letzt sollten für das Projekt keine Membrantasten, wie in vielen Tastaturen, verwendet werden, sondern stattdessen echte mechanische Schalter, die bei der Benutzung haptisches Feedback geben und deutlich langlebiger sind.

## Hardware

Neben den Schaltern und den beiden Raspberry Pi Picos, die später die Tasten auslesen und Tastendrücke an den PC schicken müssen, sind die beiden selbst entworfenen Platinen, für die rechte und linke Hälfte, wesentliche Bestandteile dieses Projekts.

Für das Erstellen der Leiterplatten wurden dafür zunächst die Schaltpläne in KiCad erstellt. Glücklicherweise besitzt der Pi Pico genug GPIO Pins, um alle Tasten einzeln anschließen zu können, was den Hardwareaufwand verglichen mit anderen Lösungen erheblich reduziert hat. Außerdem wurden im Projekt die eingebauten Pull-Up Widerstand genutzt, sodass auch diese nicht separat auf der Platine platziert werden mussten.



*Bild 4: Schaltplan für die linke und rechte Hälfte*

Um das Layout neben der Skizze auf dem Deckblatt auch in der Praxis verifizieren zu können, wurde ein kurzer Test in Fusion 360 designet und anschließend mit dem 3D Drucker produziert.



Bild 5: Probedruck des Layouts

Der erste Eindruck war zufriedenstellend, sodass nun die tatsächlichen Platinen entworfen und gefertigt werden konnten.



Bild 6: fertiger Leiterplattenentwurf in KiCad 8.0

Nach der Bestückung und ersten Power-On Tests konnte es also mit der Software weitergehen.



Bild 7: fertig bestückte Tastatur

## Software

Die beiden Raspberry Pi Picos wurden vollständig in CircuitPython programmiert, da es dort eine leicht nutzbare Bibliothek für die Emulation von Tastaturen gibt. Geplant war zunächst, dass die beiden Hälften untereinander kommunizieren und nur die rechte Seite mit dem PC verbunden wird. Aufgrund von Hardwareproblemen wurde dieser Plan allerdings kurzfristig verworfen, sodass nun beide Hälften per USB Kabel mit dem Computer verbunden werden.

Als Folge dessen ist der Code jedoch erheblich vereinfacht worden, außerdem ist er für die beiden Seiten nun abgesehen von den emulierten Tasten nahezu identisch, weshalb hier nur die Grundarchitektur näher erläutert wird.

Zum Programmstart werden zunächst die benötigten Bibliotheken eingebunden, nennenswert ist hiervon allerdings nur “adafruit\_hid”, welche die Tastaturemulation übernimmt.

Anschließend folgt ein sogenanntes Dictionary, in dem man “Schlüssel-Wert-Paare” miteinander verknüpfen kann. In diesem Fall wird hier sowohl der entsprechende Pin mit der zu emulierenden Taste abgespeichert als auch ein Boolean-Wert (entspricht: Taste gedrückt) und eine Zahl (entspricht Zeitstempel, wann Taste zuletzt gedrückt wurde). Falls gewünscht, kann hierdurch auch das Layout der Tastatur maßgeblich verändert und beliebig angepasst werden.

Nachdem alle Pins als Eingang mit Pull-Up Widerstand definiert wurden, prüft der Pico nun kontinuierlich, ob eine bestimmte Taste gedrückt wurde und diese Taste zuvor noch nicht gedrückt war (repräsentiert durch den besagten Boolean-Wert im Dictionary). Falls dies der Fall ist, wird die Taste gedrückt und der Zeitstempel im Dictionary aktualisiert, also auf die aktuelle Systemzeit gesetzt.

Falls ein zuvor gedrückter Schalter losgelassen wird, sendet der Pico den entsprechenden Befehl ebenfalls an den Computer.

Damit ist die Grundfunktionalität der Tastatur vollständig, selbst Shortcuts wie “Strg + C” sind möglich. Wer allerdings eine normale Tastatur gewöhnt ist, wird feststellen, dass dennoch eine Funktion fehlt:

Hält man beispielsweise die Leertaste für längere Zeit gedrückt, sollten auch mehrere Leertasten erscheinen. Dies lässt sich durch den gespeicherten Zeitstempel ebenfalls leicht implementieren, indem man prüft, ob eine Taste für mehr als 50ms nicht losgelassen wurde, also der Zeitstempel älter als 50ms ist. In diesem Fall sollte der entsprechende Befehl nämlich erneut an den Computer gesendet werden.

## Verbesserungspotential

Neben der Tatsache, dass die Leiterbahnen auf der rechten Platine im Gegensatz zur linken verhältnismäßig chaotisch verlegt sind, ließe sich das Projekt durch Nutzung von direkter Kommunikation zwischen den beiden Hälften erheblich verbessern, da nun nicht mehr zwei USB-Kabel benötigt werden würden. Außerdem

wüssten die Hälften jeweils voneinander, welche Tasten auf der anderen gedrückt sind, was neue Möglichkeiten für eigene Shortcuts bieten würde. Diese sind zwar kein Bestandteil von gewöhnlichen Tastaturen, aber theoretisch ließe sich beispielsweise programmieren, dass beim Herunterdrücken von Alt und J eine Pfeiltaste emuliert wird (inspiriert von der Navigation im Texteditor VI). Aktuell würde dies jedoch nur dazu führen, dass der PC den Shortcut Alt+J empfängt. Auch ein Gehäuse oder zumindest kleine Standfüße wären wünschenswert, damit sich die Tastatur auf glatten Oberflächen nicht so leicht verschieben lässt.

## Wartung

In der Theorie sollte die Tastatur robust sein, da sie keine aktiv bewegten Teile wie Motoren oder Servos enthält. Nichtsdestotrotz kann mit der Zeit natürlich einer der Schalter verschleißen und damit nicht mehr ordnungsgemäß funktionieren. In diesem Fall kann er abgelötet und gegen einen neuen ersetzt werden.

Auch einer der Mikrocontroller könnte beschädigt werden (zum Beispiel durch ein ausgelaufenes Getränk...), wobei sich diese sogar ohne Nutzung eines Lötkolbens tauschen lassen.

## Weiterentwicklung

Die nächsten sinnvollen Schritte, um das Projekt noch weiter zu verbessern, sind in meinen Augen die Wiederherstellung von direkter Kommunikation zwischen den Mikrocontrollern und das Fertigen eines passenden Gehäuses.

Zusätzlich lässt sich die Tastatur natürlich beliebig erweitern, beispielsweise durch einen Lautstärkeregler oder eine Beleuchtung.

Entsprechende Weiterentwicklungen können gerne als Pull-Request im folgenden GitHub Repository hinzugefügt werden:

<https://github.com/kraemerlukas314/PinkyPi/>

## Einrichten der Entwicklungsumgebung

Für die Entwicklung wurde hauptsächlich der Editor Thonny genutzt. Bevor ein neuer Pi Pico jedoch programmiert werden kann, muss zunächst die CircuitPython Firmware aufgespielt werden. Dazu sind die folgenden Schritte zu befolgen:

1. Pi Pico mit dem PC verbinden, während die Taste "Bootsel" gedrückt wird
2. Thonny starten
3. "Werkzeuge -> Optionen -> Interpreter" öffnen
4. CircuitPython (Generic) auswählen
5. Auf "CircuitPython installieren oder aktualisieren (U2F)" klicken

Der Code wurde auf Version 9.0.5 entwickelt und getestet.

Abschließend sollte der “lib” Ordner aus diesem [GitHub repo](#) auf den Speicher des Picos kopiert werden.

## Teileliste

- [2x Raspberry Pi Pico W](#)
- [Gateron Tasten](#)
- [Tastenkappen](#)
- 2x Micro USB Kabel
- 10x M3 Schrauben, 8mm lang
- 10x M3 Muttern

## Werkzeug

- Lötkolben
- Lötzinn
- M3 Innensechskant
- Zange