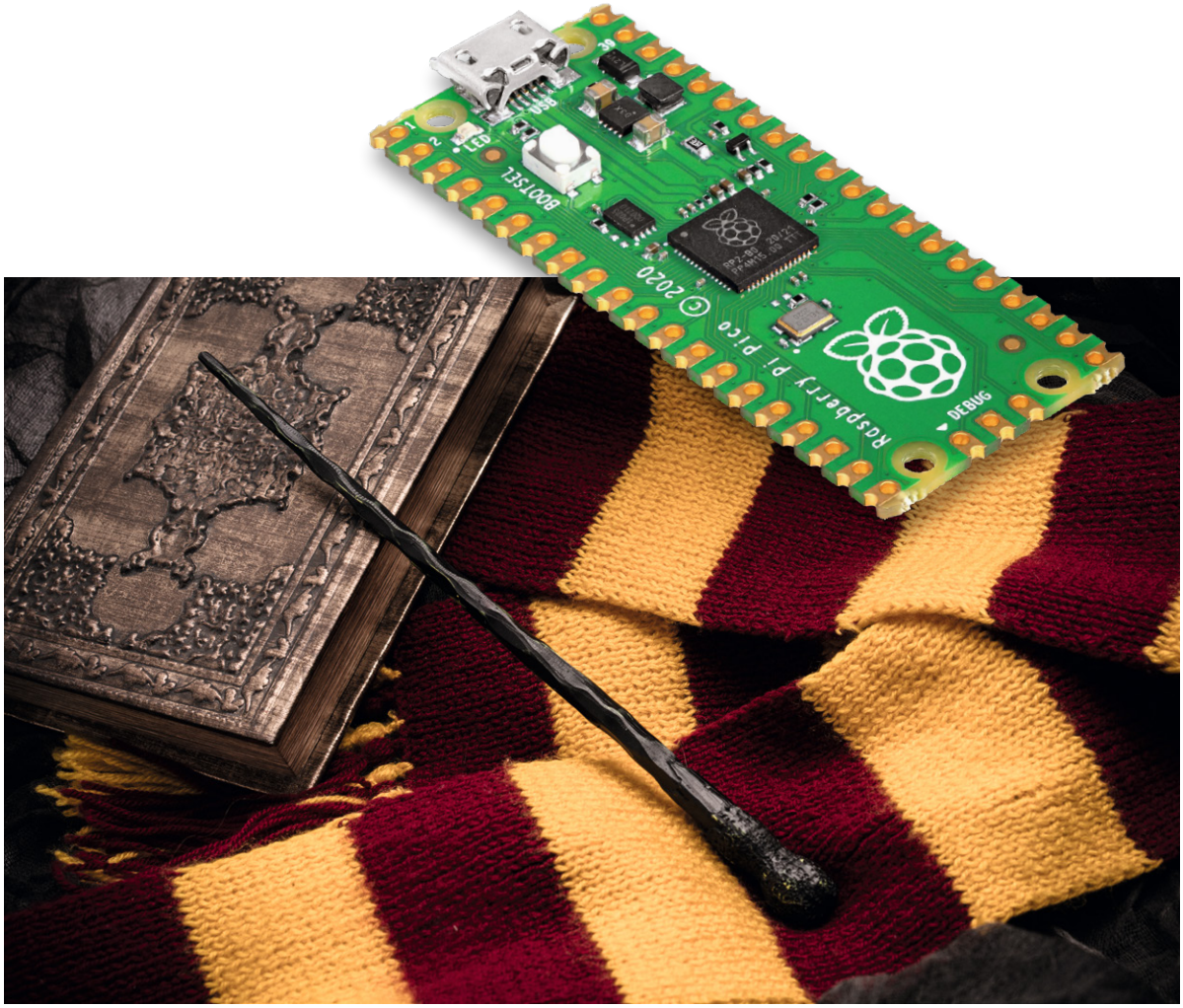
Raspberry Pi Pico

Künstliche Intelligenz und Machine Learning auf dem Raspberry Pi Pico

Teil 4

Oftmals wird angenommen, dass Künstliche Intelligenz, Deep Learning und Maschinelles Lernen (ML) Hochleistungs-CPUs und gigantische Speicherkapazitäten erfordern. Die Implementierung eines ML-Algorithmus in einem Mikrocontroller mit wenigen Kilobyte Speicher und einer Taktrate von einigen Megahertz erscheint da kaum realistisch. Mit einer Light-Version von Tensorflow ist es jedoch möglich, ein neuronales Netzwerkmodell in einen Datensatz umzuwandeln, der lediglich einige Kilobyte Speicherplatz benötigt. Mit diesen Modellen sind dann auch kleinere Controller nicht überfordert.



ML-Modelle auf Mikrocontrollern

Nun stellt sich die Frage, warum man überhaupt daran interessiert sein sollte, ML-Modelle trotz erheblicher Einschränkungen in einem Mikrocontroller zu implementieren. Eine wichtige Anwendung sind hier tragbare und mobile Geräte mit Batteriebetrieb. Da das ML-Modell lokal innerhalb der MCU implementiert wird, ist keine Schnittstelle zu einem Server oder einer Cloud-Anwendung erforderlich. Dadurch wird die Reaktionsgeschwindigkeit verbessert, entsprechend also die sogenannte Latenz reduziert. Zudem wird die Datensicherheit deutlich erhöht.

In diesem Beitrag werden zwei Projekte dazu vorgestellt. Das erste ist vergleichsweise einfach aufgebaut. Allerdings ist es vom historischen Standpunkt aus betrachtet von großer Bedeutung. Die digitale Exklusiv-Oder-Gatterfunktion stürzte nämlich die KI-Forschung in eine tiefe Krise, von der sie sich erst nach vielen Jahren wieder erholte. Das zweite Projekt ist technisch deutlich anspruchsvoller. Hier soll der Pico aus den Daten eines Beschleunigungssensors bestimmte Handgesten erkennen, eine typische Anwendung für mobile Geräte.

Das Perzeptron und der erste „KI-Winter"

Das sogenannte Perzeptron wurde 1943 von McCulloch und Pitts vorgeschlagen. Die Idee beruhte auf der technischen Nachbildung der gerade eben entdeckten Funktion biologischer Neuronen im menschlichen Gehirngewebe. Bereits sehr früh wurde versucht diese Technologie zur Bildklassifizierung einzusetzen. Allerdings waren die Ergebnisse zunächst eher enttäuschend.

Bis heute sind Perzeptrons die einfachsten Bestandteile eines Neuronalen Netzwerks. Ein einzelnes Perzeptron besteht aus einer Verarbeitungseinheit P, zwei Eingängen X1 und X2, einem optionalen Bias-Wert und einem einzelnen Ausgang Y (s. Bild 1). Der Ausgabewert Y wird aus der Summe der gewichteten Eingangsprodukte berechnet. Eine spezielle Aktivierungsfunktion entscheidet, ob das Perzeptron „feuert" oder nicht. Die Aktivierungsfunktion definiert die Übertragungseigenschaften des Perzeptrons. Über den Bias-Wert können die Eingabegrößen bei Bedarf gewichtet werden.

Obwohl das Perzeptron zunächst vielversprechend erschien, konnte im Jahr 1969 nachgewiesen werden, dass ein einzelnes Perzeptron eine einfache Exklusiv-Oder (XOR) Funktion

Eingang X1	Eingang X2	Ausgang Y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

nicht „erlernen" konnte. Das Scheitern der „Künstlichen Gehirnzelle" an dieser einfachen Aufgabe führte dazu, dass die Forschungen zum Thema Künstliche Intelligenz in den 70er- und frühen 80er-Jahren nahezu stagnierten. Dieser Zeitraum wurde später als (erster) KI-Winter bezeichnet. Erst 1986 wurde dann ein effizienter Algorithmus entwickelt, der es ermöglichte, zuvor unlösbare Probleme anzugehen. Auch das altbekannte XOR-Gatter konnte damit erzeugt werden. Heutzutage ist der später als „Backpropagation" bekannt gewordene Algorithmus die wichtigste Methode, um neuronale Netze zu trainieren.

Das Perzeptron konnte die XOR-Funktion nicht lernen, da keine einzelne gerade Linie gezogen werden kann, welche das Ausgabemuster eindeutig gruppiert (Bild 2). Das XOR-Ausgabemuster ist also nicht „linear separierbar". Erst mit zwei Linien (Orange in Bild 2) wird die Trennung in einen „0"- und einen „1"- Bereich möglich. Ein einzelnes Perzeptron kann jedoch nur eine einzelne Trennlinie ziehen. Damit wird klar, dass erst ein mehrschichtiges Netz in der Lage ist, die XOR-Funktion korrekt darzustellen.

Das XOR-Gatter als existenzielles Problem

Neben seiner historischen Bedeutung hat das XOR-Beispiel auch den Vorteil, dass es Hardware-technisch sehr einfach aufgebaut werden

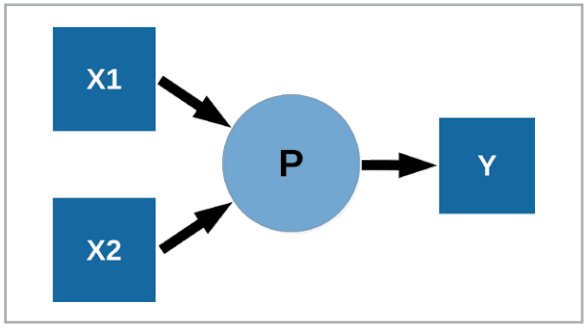


Bild 1: Perzeptron

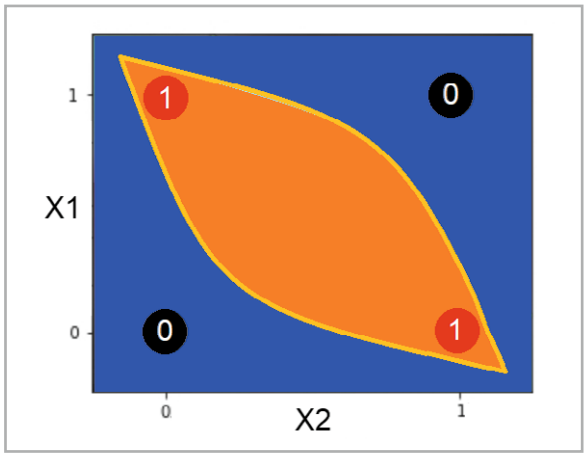


Bild 2: Lösung des XOR-Problems mit zwei Linien

kann. Zwei Taster und eine Leuchtdiode genügen. Die LED wird dabei an Pin 2 des Pico angeschlossen. Die beiden Taster werden über die Pins 21 und 22 abgefragt. Der Aufbau kann also wieder problemlos auf einem EXSB1-Experimentierboard erfolgen. Bild 3 zeigt einen Vorschlag dazu.

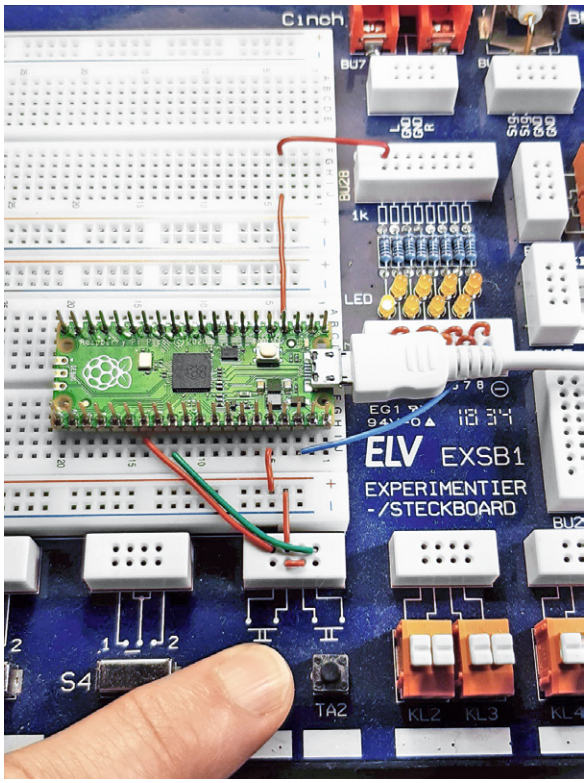


Bild 3: Aufbau auf dem Experimentiersystem EXSB1

Das Auswerten von Taster-Zuständen ist natürlich keine typische Anwendung für Machine Learning- oder KI-Projekte. Dennoch ist das Beispiel nicht nur wegen seiner historischen Bedeutung sehr lehrreich.

Mit einem einfachen Programm kann zunächst die Schaltung getestet werden. Der Code ist bewusst einfach gehalten. Das Programm dient zudem auch als Grundlage für den später vorgestellten TinyML-Code eines trainierten Modells. Als Entwicklungsumgebung kommt die bekannte Arduino IDE zum Einsatz. Obwohl diese seit Kurzem in Version 2.0 verfügbar ist, sollte die ältere Version (1.8.16) verwendet werden, da das neue Hauptrelease bedauernswerterweise noch einige schwerwiegende Fehler bzw. Unzulänglichkeiten enthält. Das Einbinden des Pico in die IDE wurde bereits im ersten Artikel zu dieser Serie ausführlich dargestellt. Bei Bedarf können die Details dazu dort nachgelesen werden.

Der Arduino Sketch für eine klassische XOR-Funktion kann so aussehen (XOR_classic_Arduino.ino):

```
// XOR_classic_Arduino.ino
// Pico @ IDE 1.8.16

#define BUTTON1_PIN 21
#define BUTTON2_PIN 22
#define LED_PIN 2
// #define LED_PIN PICO_DEFAULT_LED_PIN

int button1, button2;

void setup()
{ pinMode(LED_PIN, OUTPUT);
}

void loop()
{ button1 = digitalRead(BUTTON1_PIN);
  button2 = digitalRead(BUTTON2_PIN);
  if (button1 != button2)
    digitalWrite(LED_PIN, HIGH);
  else
    digitalWrite(LED_PIN, LOW);
  sleep_ms(10);
```

Ein kurzer Test bestätigt die Funktion: Wird kein Taster betätigt, leuchtet die LED nicht. Wird nur einer der beiden Taster gedrückt, leuchtet die LED. Beim gleichzeitigen Drücken beider Taster erlischt die LED. Dies ist der wichtige Unterschied zur klassischen und wesentlich einfacheren ODER-Funktion, bei der die LED auch leuchten würde, wenn beide Taster gedrückt sind.

Pico „erlernt" die XOR-Funktion

Mit „Keras" steht eine Open-Source-Softwarebibliothek zur Verfügung, die eine Python-Schnittstelle zum Erstellen neuronaler Netze bereitstellt. Keras fungiert als High-Level-Schnittstelle für die Tensorflow-Bibliothek, um ein schnelles und einfaches Experimentieren mit neuronalen Netzen zu ermöglichen. In Kombination mit Jupyter-Notebooks kann der Python-Code mit der Keras-Bibliothek in einer einzigen Entwicklungsumgebung interaktiv erstellt, dokumentiert und getestet werden. Jupyter stellt

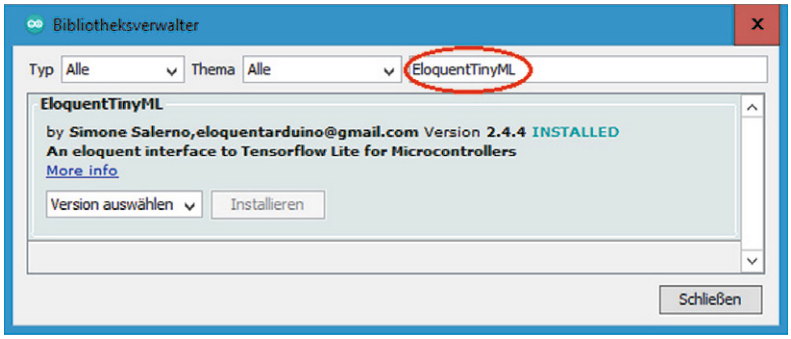


Bild 4: Einbinden der EloquentTinyML Arduino library

standardmäßig alle erforderlichen Bibliotheken zur Verfügung. Installation und Anwendung des Programmpakets wurde in der Artikelserie „Künstliche Intelligenz" ausführlich vorgestellt [1].

Eine detaillierte Darstellung der erforderlichen Arbeitsschritte würde den Rahmen dieses Artikels sprengen, da hier die Anwendung auf dem Pico im Vordergrund steht. Dennoch enthält das Download-Paket die zugehörige Notebook-Datei, mit der ein neuronales Netz zur XOR-Funktion trainiert werden kann. Anwender mit entsprechender Erfahrung können damit ihr eigenes Netz trainieren (XOR_for_Pico.ipynb im Ordner „Jupyter") und testen.

Für die Anwendung auf dem Pico-Board ist im Download-Paket jedoch auch ein fertiges Neuronales Netzwerk hinterlegt, das bereits ein optimal trainiertes Modell enthält (XOR_model.h im Ordner XOR_emu_Arduino). Damit kann die Anwendung auf dem Pico getestet werden, ohne dass das aufwendige Trainingsverfahren im Jupyter-Notebook selbst ausgeführt werden muss. Für die Verarbeitung von neuronalen Netzwerkfunktionen auf dem Pico ist lediglich die EloquentTinyML Arduino library erforderlich. Diese kann wie üblich über die Bibliotheksverwaltung via Library-Manager installiert werden (Bild 4).

Mit EloquentTinyML steht eine universelle Bibliothek zur Verfügung, die es erlaubt, beliebige ML-Modelle auf dem Pico auszuführen. Dazu müssen die Modelle nur aus einem Jupyter-Notebook exportiert werden. Der Umfang der Modelle ist dabei lediglich durch die Speichergröße des Pico bzw. die Arbeitsgeschwindigkeit des Controllers begrenzt.

Nach der Installation kann der Sketch zur Auswertung des Modells (XOR_emu_Adruino.ino) geladen und gestartet werden:

```
// XOR_emu_Adruino.ino
// Pico @ IDE 1.8.16

#include <EloquentTinyML.h>
#include <eloquent_tinyml/tensorflow.h>

#include "XOR_model.h"

#define N_INPUTS 2
#define N_OUTPUTS 1
#define TENSOR_ARENA_SIZE 2*1024    // reserved memory space

Eloquent::TinyML::TensorFlow::TensorFlow
<N_INPUTS, N_OUTPUTS, TENSOR_ARENA_SIZE> tf;

void setup()
{ Serial.begin(115200); delay(1000);
  tf.begin(XOR_model);
  // check if model load correctly
  if (!tf.isOk())
    { Serial.print("ERROR: ");
      Serial.println(tf.getErrorMessage());
      while (true) delay(1000);
    }
}
```

```
void loop()
{ // test for input matrix [(0,0); (0,1); (1,0); (1,1)]
  for (float i=0; i<=1; i++)
  {for (float j=0; j<=1; j++)
    { float input[2] = {i,j};
      float predicted = tf.predict(input);
      Serial.print("in1: "); Serial.print(input[0]);
      Serial.print("\t in2: "); Serial.print(input[1]);
      Serial.print("\t predicted: ");
      Serial.println(predicted);
    }
  }
  Serial.print
ln("_____");
  delay(10000);
}
```

Über die Anweisung

```
#include "XOR_model.h"
```

wird das Modell geladen.

Im Setup kann dieses dann über

```
tf.begin(XOR_model);
```

aktiviert werden.

Die Auswertung des Modells erfolgt danach ganz einfach über die Funktion

```
float predicted = tf.predict(input);
```

Diese wird für alle vier Eingangskombinationen ([0,0]; [0,1]; [1,0]; [1,1]) ausgewertet.

Das Kompilieren des Sketches nimmt eine vergleichsweise lange Zeitspanne (bis zu mehreren Minuten) in Anspruch, da hier alle Funktionen des TinyML-Systems implementiert wurden. Danach zeigt die Ausgabe im seriellen Monitor jedoch, dass das Modell die XOR-Funktion in guter Näherung nachbildet (Bild 5).

Natürlich werden nun nicht mehr die exakten Werte 0 und 1 als Ausgaben geliefert. Vielmehr liefert das Netz die besten Näherungswerte der „neuronalen Aktivität“ (0,04; 0,95; 0,94; 0,07).

Mit dem Sketch (XOR_LED_output_Adruino.ino) wird die Funktion auf die beiden LEDs übertragen. Dazu wurde ein Schwellwert

```
threshold=0.5;
```

verwendet, der darüber entscheidet, ob der Ausgabewert des neuronalen Netzes als 0 oder 1 interpretiert wird. Das Austesten über die Taster bestätigt die korrekte Funktion.

Dies zeigt, dass auch kleine Controller-Boards wie der Pico in der Lage sind, Neuronale Netzwerke auszuwerten. Für das Training dieser Netze ist allerdings ein leistungsfähiges System erforderlich. Auf dem Pico würden die zugehörigen Algorithmen extrem lange Trainingszeiten in Anspruch nehmen. Zudem wäre die Speicherkapazität bei größeren Netzwerkmodellen schnell erschöpft.

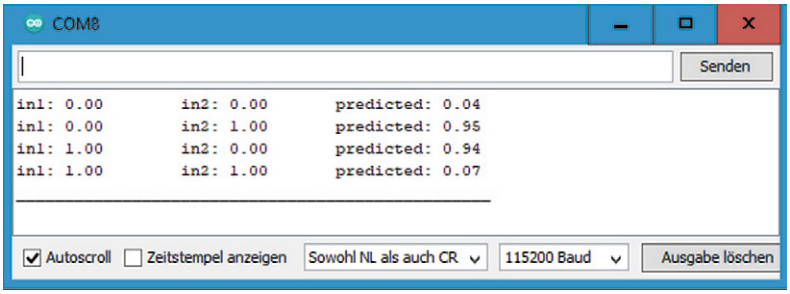


Bild 5: Das neuronale Netz bildet die XOR-Funktion nach.

Pico-KI

Das XOR-Beispiel zeigt, dass Deep Learning und Künstliche Intelligenz auch auf kleinen Systemen möglich ist. Die Auswertung eines trainierten Netzwerks ist also auch für Mikrocontroller bzw. eingebettete, autonome Systeme durchführbar. Da das ML-Modell in diesen Fällen lokal innerhalb der MCU implementiert ist, ist weder eine Cloud-Schnittstelle noch ein Zugriff auf externe Server-Farmen o. Ä. erforderlich. Die Verwendung von ML-Algorithmen auf eingebetteten Geräten bietet damit einige wichtige Vorteile:

- **Keine hohen Bandbreitenerfordernisse:**
  ML-Algorithmen auf kleinen Controllern können aussagekräftige Informationen aus Daten extrahieren, die ansonsten aufgrund von Bandbreitenbeschränkungen nicht zugänglich wären.
- **Geringe Latenz:**
  Geräteinterne ML-Modelle können in Echtzeit auf Eingaben reagieren und so Anwendungen ermöglichen, die nicht realisierbar wären, wenn sie von der Netzwerklatenz abhängig wären.
- **Hervorragende Wirtschaftlichkeit:**
  Durch die Verarbeitung von Daten auf dem Gerät selbst vermeiden eingebettete ML-Systeme hohe Kosten für die Übertragung von Daten über ein Netzwerk und deren Verarbeitung in einer Cloud.
- **Hohe Zuverlässigkeit:**
  Systeme, die von internen Modellen gesteuert werden, sind von Natur aus zuverlässiger als solche, die auf eine Verbindung zur Cloud angewiesen sind.
- **Guter Datenschutz:**
  Wenn Daten auf einem eingebetteten System verarbeitet und nicht in Cloud-Systeme übertragen werden, ist die Privatsphäre der Benutzer gut geschützt und es besteht nur ein geringeres Missbrauchsrisiko.

All diese Vorteile führen zu dem großen Interesse am Einsatz von KI-System auf autonomen Systemen. Die Entwicklung von TinyML und anderen vergleichbaren Systemen zeigt, wie wichtig diese Variante der KI geworden ist.

Das nächste Anwendungsbeispiel demonstriert zudem, dass ML-Technologie auf Mikrocontrollern nicht nur auf die einfache Auswertung von Taster-Zuständen beschränkt ist. Vielmehr können auch komplexe Sensordaten erfasst, vorverarbeitet und schließlich ausgewertet werden. Da die meisten Controller bereits über integrierte Analogeingänge verfügen, können vielfältige Daten ohne zusätzlichen Hardwareaufwand gesammelt und verarbeitet werden.

Ein elektronischer „Zauberstab“

Wer hätte nicht gerne einen Zauberstab, der es gestattet, mit einer Bewegung des Handgelenks die erstaunlichsten Wunder zu vollbringen. Spätestens seit J. K. Rowling mit „Harry Potter“ einen unglaublichen Hype ausgelöst hat, träumen Kinder und Jugendliche davon, einen Zauberstab von Mr. Ollivander zu besitzen.

Mit elektronischen Mitteln kann man diesen Wunsch zumindest in bescheidenen Ausmaßen näherkommen. Mithilfe des Raspberry Pi Pico, einem analogen Beschleunigungsmesser und etwas Künstlicher Intelligenz kann man ein System aufbauen, das in der Lage ist, Handgesten zu erkennen.

Für erste Versuche sollen drei verschiedene geometrische Formen verwendet werden:

- Schlange: S-förmige Bewegung
- Zauberkreis: kreisförmige Handbewegung
- Block: Lineare Vor- und Zurückbewegung der Hand

Zusätzlich wird noch der „Ruhezustand" erfasst. Ansonsten würde das System immer versuchen eine der drei Formen zu erkennen, auch wenn der Sensor bzw. „Zauberstab" überhaupt nicht bewegt wird.

Zur Erkennung der Bewegung wird ein kostengünstiger Beschleunigungssensor eingesetzt. In der einfachsten Variante genügt bereits ein eindimensionaler Sensor wie der ADXL(1)50. Inzwischen sind aber 3-D-Sensoren weit verbreitet, sodass diese vorzugsweise eingesetzt werden. Auch hier würde das Auswerten eines einzelnen Kanals in vielen Fällen bereits genügen. Bei Verwendung aller drei Raumrichtungen sind die Ergebnisse allerdings zuverlässiger.

Da der Pico über drei separate Analogeingänge verfügt, kann ein Sensor mit analogen Ausgangssignalen verwendet werden. Eine mögliche Variante für 3-D-Sensoren ist damit der ADXL330 oder der ADXL335. Beide Sensoren sind auf Breakout-Boards fertig montiert erhältlich. Falls andere Sensoren eingesetzt werden sollen, kann das nachfolgend vorgestellte Programm aber auch leicht an diese angepasst werden.

Das Erfassen von Trainingsdaten und das Trainieren eines TinyML-Modells erfolgt mithilfe von „Edge Impulse Studio". Dieses Tool ist kostenlos als Online-Version verfügbar [2]. Es basiert auf Google Tensorflow und erlaubt die Erstellung von Machine-Learning-Modellen sowie deren Portierung auf kleine Controller wie den Raspberry Pi Pico.

Mit dieser abgespeckten Version von Tensorflow (bzw. Tensorflow lite) ist es möglich, auch größere neuronale Netzwerkmodelle in Minimal-Versionen umzuwandeln, die nur wenige Kilobyte Speicherplatz benötigen. Die Rechenleistung des Pico ist zudem vollkommen ausreichend, um diese Modelle in Echtzeit auszuführen. Natürlich kann Tensorflow lite keine hochkomplexen Modelle konvertieren. So wird etwa die Arbeit mit optischen Mustererkennungsmodellen schnell unpraktikabel. Wenn man allerdings die gegebenen Einschränkungen beachtet, können durchaus einige interessante Projekte realisiert werden.

Umsetzung mit Edge Impulse

Edge Impulse erlaubt es, ein TinyML-Modell von Grund auf neu zu entwickeln. Von der Datenerfassung bis zur Modellbereitstellung stehen dort alle erforderlichen Werkzeuge zur Verfügung. Zunächst muss hierfür ein entsprechendes Nutzerkonto angelegt werden.

Der nächste Schritt ist dann die Datenerfassung mit dem ausgewählten Sensor, im vorliegenden Fall also einem Beschleunigungsmesser. Daten sind in verschiedenen Formen, wie etwa im CSV- oder JSON-Dateiformat, importierbar.

Die Edge Impulse Command-Line-Funktion (CLI) erlaubt es, Daten direkt von der Hardware (Pico mit angeschlossenem Sensor) zu erfassen. Prinzipiell sind auch andere MCUs wie Arduino oder der ESP32 verwendbar, um Messwerte über die Serielle Schnittstelle aufzunehmen. Allerdings ist es vorteilhaft, dieselbe MCU und denselben Sensor zur Erfassung von Trainingsdaten und für die Modellvalidierung zu verwenden. Dadurch wird das Training des Netzwerkes präziser und Fehlanpassungen können vermieden werden.

Das Edge Impulse CLI kann problemlos auf einem PC installiert werden. Dabei ist zu beachten, dass das Interface NodeJS und Python erfordert. Falls noch nicht vorhanden, sind diese Programm also vorher zu installieren. Nach erfolgreichem Download können Daten mit einem „edge-impulse-data-forwarder"-Interface erfasst werden.

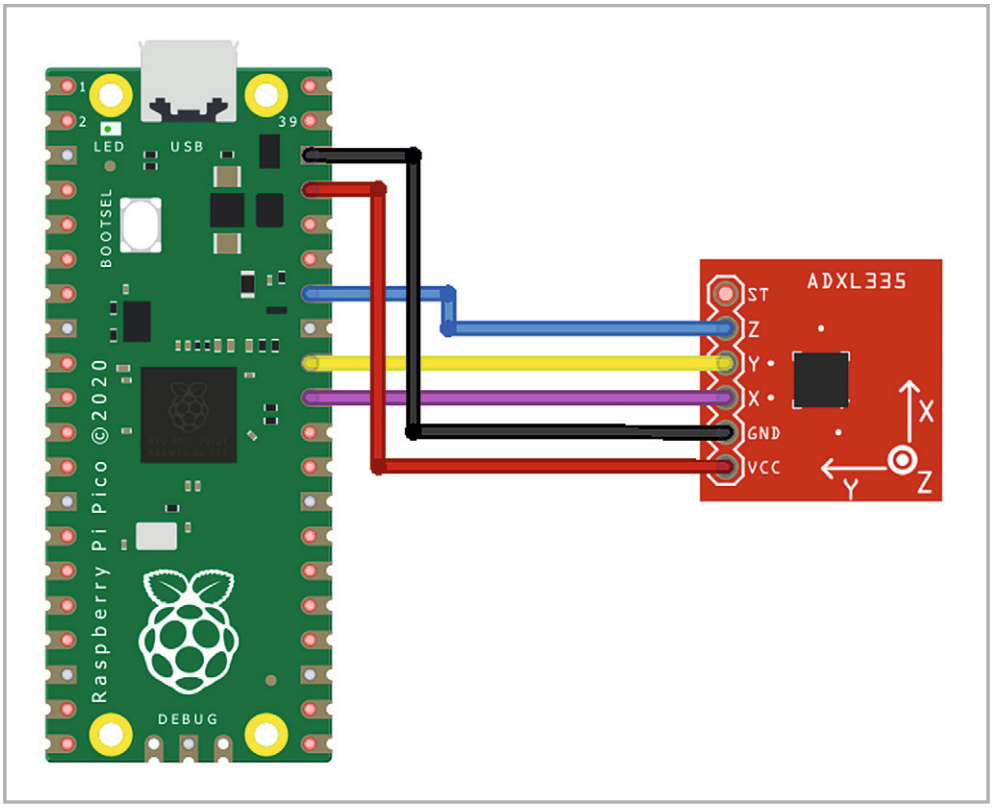


Bild 6: Anschluss des Beschleunigungssensors an den Pico

Die Verkabelung zwischen der Pico-Platine und dem Beschleunigungsmesser ist in Bild 6 dargestellt. GPIO-26, 27 und 28 entsprechen den ADC-Kanälen 0, 1 und 2. Diese werden verwendet, um die analogen Ausgänge x, y und z des Beschleunigungsmessers auszulesen.

Das Format zum Senden der Daten über eine Serielle Schnittstelle ist sehr einfach. Die x-, y- und z-Werte des Sensors werden direkt seriell, d. h. lediglich durch ein Komma getrennt, übertragen. Man kann entweder MicroPython oder die Arduino-IDE verwenden, um die Sensordaten an den Edge Impulse Data Forwarder zu senden. Im folgenden Programm soll wieder die Arduino IDE verwendet werden, um die Daten im 10-ms-Takt zu übertragen (s. MagicWand_Arduino.ino):

```
// MagicWand_Arduino.ino
// Pico @ IDE 1.8.16

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

void loop()
{ Serial.print(analogRead(A0)); Serial.print(",");
  Serial.print(analogRead(A1)); Serial.print(",");
  Serial.println(analogRead(A2));
  delay(10);
}
```

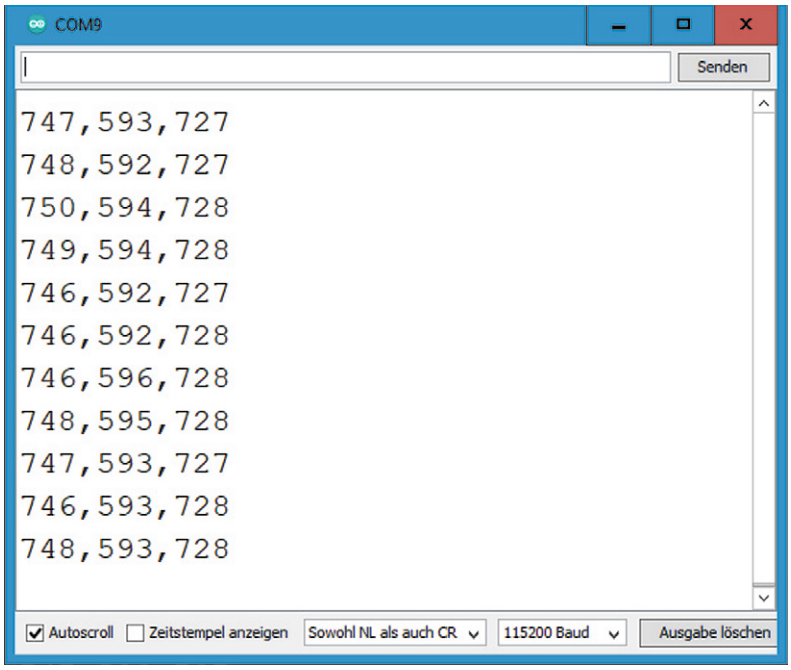


Bild 7: Datenausgabe auf der Konsole der Arduino IDE

Theoretisch liegt die Übertragungsfrequenz der Daten damit bei 100 Hz. Nach dem Ausführen des Codes werden die Daten auf der Konsole ausgegeben. Das Programm liefert beispielsweise die Datensequenz im seriellen Monitor wie in Bild 7.

Nun muss in der Eingabeaufforderung von Windows der Befehl

edge-impulse-data-forwarder

ausgeführt werden.

Beim ersten Mal wird nach Anmeldeinformationen, Board-Namen (optional), Benennung der Daten (z. B. Sensor-x, Sensor-y, Sensor-z) gefragt. Die Datentransferrate wird automatisch erkannt. Bild 8 zeigt, dass eine Rate von 98 Hz detektiert wurde. Der theoretische Wert von 100 Hz wird aufgrund von zusätzlichen Verzögerungen im Programmablauf erwartungsgemäß nicht exakt erreicht, da die MCU auch Zeit benötigt, um die Print-Anweisungen auszuführen. Als Ergebnis ist die praktische Abtastfrequenz daher etwas kleiner als 100 Hz.

Dann kann die in der letzten Zeile ausgegebene Web-Adresse, also

https://studio.edgeimpulse.com/studio/➲
xxxxxx/acquisition/training

in einem beliebigen Browser aufgerufen werden.

Hier wird im Auswahlmenü zunächst „Devices“ aktiviert. Dort erscheint der „MagicWand“ als aktiver Eintrag.

Im Menü „Date Acquisition“ können nun Trainingsdaten gesammelt werden. Dazu sind die passenden Bezeichnungen (Label) wie „Schlange“, „Zauberkreis“ oder „Block“ einzutragen.

Dann wird mit dem Sensor die zugehörige Bewegung ausgeführt. Für die Schlange eine S-fömige Bewegung, dann der Kreis und schließlich eine schnelle Vor- und Zurückbewegung für den „Block“.

Diese Prozedur wird nun für alle Muster mehrmals durchgeführt.

Ein Datensatz mit ca. 12 bis 30 Proben reicht für erste Tests vollkommen aus.

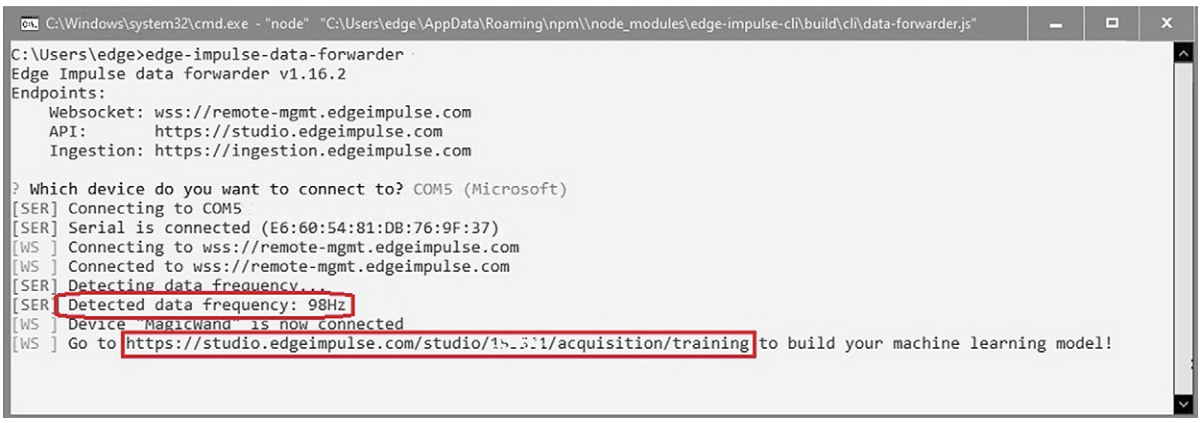


Bild 8: Starten des Edge-Datenerfassungsmoduls

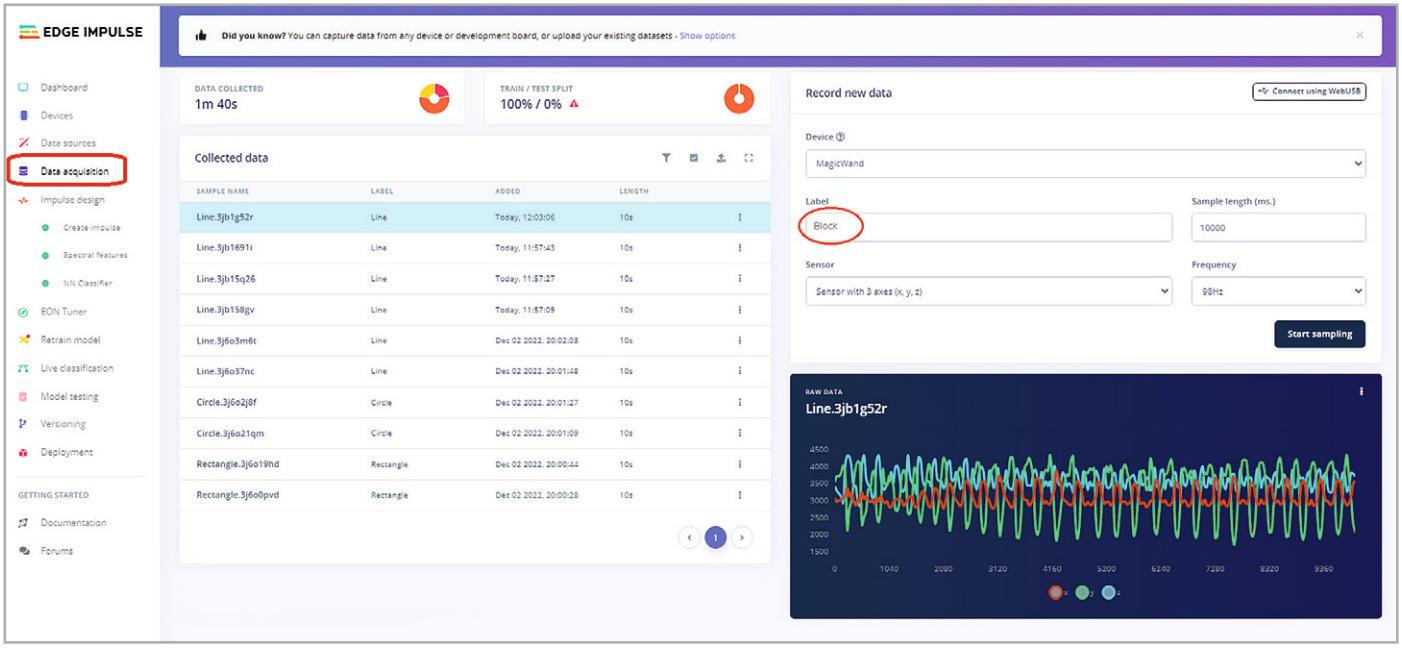


Bild 9: Datenerfassung in Edge

Wenn ein kompletter Datensatz erfasst wurde, werden die Messwerte rechts unten in einem grafischen Fenster dargestellt (Bild 9). Hier kann man die Aufzeichnungen kontrollieren. Fehlmessungen, z. B. aufgrund von mangelhaften Kontakten, falschen Kanalzuordnungen etc., können dann sofort entfernt bzw. korrigiert werden.

Bild 10 zeigt typische Ergebnisse für die drei Bewegungsarten. Man erkennt, dass die einzelnen Bewegungen zwar komplexe, aber deutlich unterscheidbare Datenmuster erzeugen.

Training und Test

Im nächsten Schritt werden die Daten so konditioniert, dass sie für das Training eines neuronalen Netzes verwendbar sind. Unter

Impulse Design -> Create Impulse

kann ein Zeitfenster für die Datensegmentierung festgelegt werden. Da die Bewegungen kontinuierlich sind, müssen Fenster definiert werden, welche die Daten in fest definierte Segmente unterteilen. Hier können zunächst die voreingestellten Werte übernommen werden. Bevor die Rohdaten in ein neuronales Netz eingespeist werden, müssen sie vorverarbeitet werden. Dieses Vorgehen hilft dabei, die wesentlichen Merkmale zu separieren. Da es sich bei den Sensordaten um Zeitreihen handelt, ist eine Darstellung im Frequenzbereich nach Durchführung einer FFT (Fast Fourier Transformation), d. h. einer Spektralanalyse, eine gute Wahl. Edge stellt automatisch geeignete Parameter für die Datensätze zur Verfügung. Bei Bedarf können diese dem aktuellen Problem angepasst werden.

Nach dem Abspeichern der Parameter können die Ergebnisse in der Registerkarte „Spectral Features" überprüft werden. Auch hier können die voreingestellten Parameter übernommen werden.

Die Darstellung im Feature Explorer zeigt, wie gut die drei Bewegungsgeometrien separiert werden können. Eine klare Trennung führt zu einer hohen

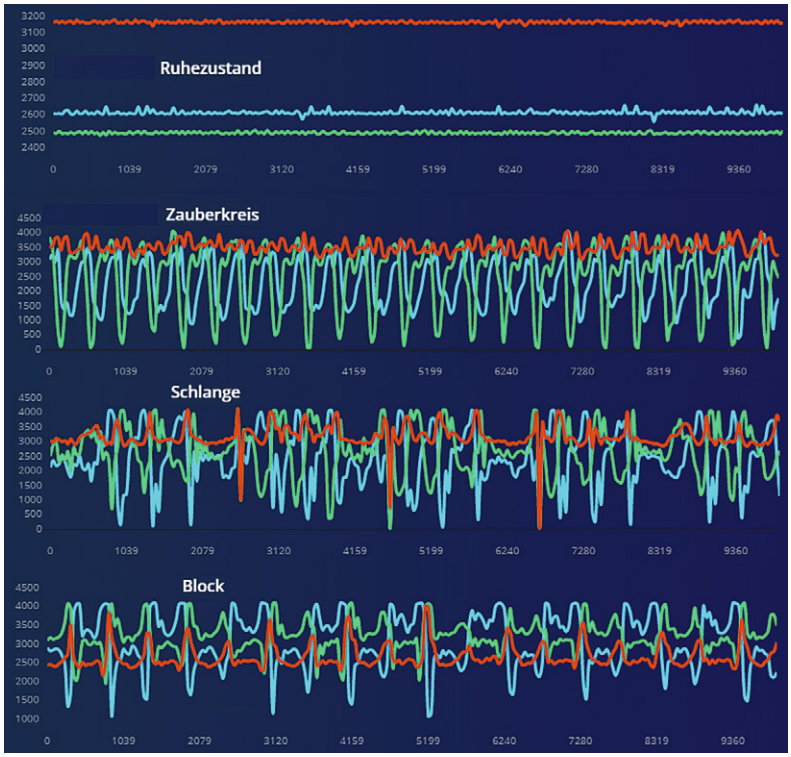


Bild 10: Die drei Bewegungsmuster und der Ruhezustand

Erfolgsrate beim Trainieren des neuronalen Netzes. Wenn sich Datenbereiche stark überlappen, können sie möglicherweise nicht gut erkannt werden. In diesem Fall kann es angebracht sein, den Datensatz durch das Aufnehmen zusätzlicher Bewegungsmuster zu erweitern.

Damit sind die Rohdaten für die Einspeisung in ein Neuronales Netzwerk bereit. Im „Classifier" wird die Architektur des Neuronalen Netzwerks definiert. Die erste (drei Eingänge) und die letzte Ebene (ein Ausgang) werden automatisch zugewiesen. Für den Anfang kann eine Schicht mit 20 Neuronen gefolgt von einer weiteren Schicht mit 10 Neuronen hinzugefügt werden. Danach wird ein erster Trainingslauf gestartet. Das Trainieren nimmt einige Zeit in Anspruch. Schließlich wird das Ergebnis in Form einer Matrix dargestellt (Bild 11).

Bild 11: Trainingsergebnis

Zusätzlich wird die Verteilung der Datenpunkte grafisch ausgegeben. Hier wird deutlich, dass der Zauberstab die Bewegungen bereits recht gut erkennt. Lediglich die „Block"-Bewegung wird in einigen Fällen (8,3%) mit der „Schlange" verwechselt. In Anbetracht des geringen Umfangs des Datensatzes ist das ein sehr respektables Ergebnis. Die grafische Auswertung zeigt zudem, dass die einzelnen Bewegungen gut getrennt werden konnten. Durch das Optimieren der Parameter oder die Aufnahme weiterer Datensätze könnte das Resultat jedoch auch noch weiter verbessert werden.

Im sogenannten Feature Explorer können die Daten auch in einer dreidimensionalen Darstellung angezeigt werden. Neben den Rohdaten können hier auch Werte wie die Mittelungen oder Standardabweichungen als Achsen ausgewählt werden. Bei komplexen Datensätzen ergeben sich hier vielfältige Möglichkeiten, um die Trennung der Daten zu optimieren. Bild 12 zeigt ein Beispiel für eine spezielle Auswahl.

Insbesondere wenn die direkte Auswertung der Rohdaten keine gute Trennung der Ausgabewerte erlaubt, können zusätzliche Größen wie die Standardabweichung einer Messreihe zu deutlichen Verbesserungen führen. Im hier betrachteten Beispiel des Zauberstabs lassen die gewählten Bewegungsmuster allerdings bereist recht gute Zuordnungen zu.

Bild 12: 3-D-Darstellung im sogenannten „Feature Explorer"

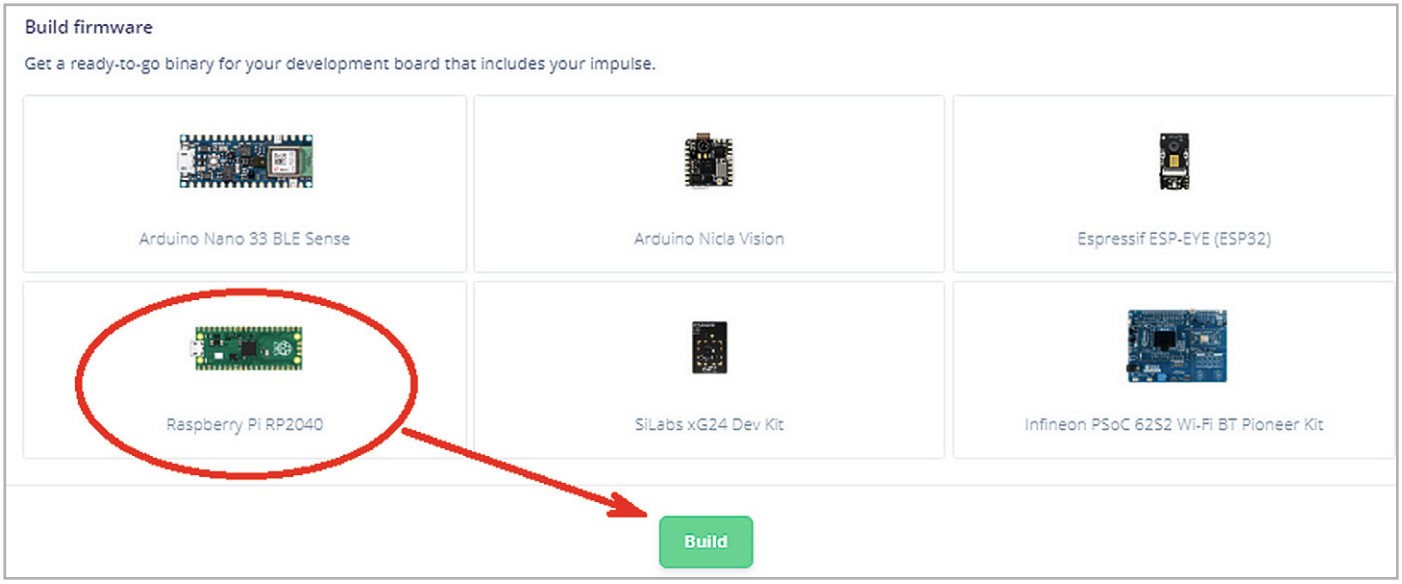


Bild 13: Erstellen der uf2-Datei für den Pico

Stand-alone-Anwendungen

Im letzten Schritt muss das Modell nun noch auf den Pico übertragen werden. Hierfür stehen drei Möglichkeiten zur Verfügung:

- Erstellung einer C++-Bibliothek
- Erstellung einer Arduino-Bibliothek
- Erstellung einer Binärdatei für die direkte Übertragung zum Controller

Alle Varianten haben ihre spezifischen Vor- und Nachteile. Die Übertragung der C++-Bibliothek auf den Pico erfordert den Einsatz von Visual Studio und ist mit erheblichem Aufwand verbunden.

Die Erstellung einer Arduino-Bibliothek und deren Einsatz in der zugehörigen IDE ist zwar deutlich einfacher, jedoch zeigt sich schnell, dass dieser Weg nicht sehr zuverlässig arbeitet. Hier treten immer wieder Fehler bei der Übernahme des Netzwerkmodells auf.

Auch das direkte Erstellen einer Binärdatei läuft nicht immer völlig fehlerfrei. Allerdings ist diese Methode vergleichsweise einfach. Man muss hierzu nur das Icon RP2040 auswählen und dann den Compiler-Prozess über „Built“ starten (Bild 13).

Anschließend wird eine Firmware-Datei im Binärformat für den Pico erzeugt. Diese kann nun wie jede andere Firmware auf den Pico übertragen werden. Dazu wird die USB-Verbindung zunächst getrennt. Beim Wiederherstellen muss der BOOTSEL-Button auf dem Pico gedrückt werden, bis der Pico als Laufwerk im Windows-Explorer erscheint. Dann kann die neue Datei auf den Pico kopiert werden. In einem cmd-Fenster von Windows wird nun die Anweisung

„edge-impulse-run-impulse“

eingegeben. Daraufhin meldet sich der Pico mit den Ausgabewerten der neuen Firmware (Bild 14).

Nun kann man die drei „Zauberfiguren“ mit dem Beschleunigungssensor in der Hand ausführen (im Beispiel nach einem kurzen Ruhezustand die Schlange, dann der Block und schließlich der Kreis). Man erkennt, dass das System die drei gewählten Figuren gut unterscheiden kann. Zudem wird auch der

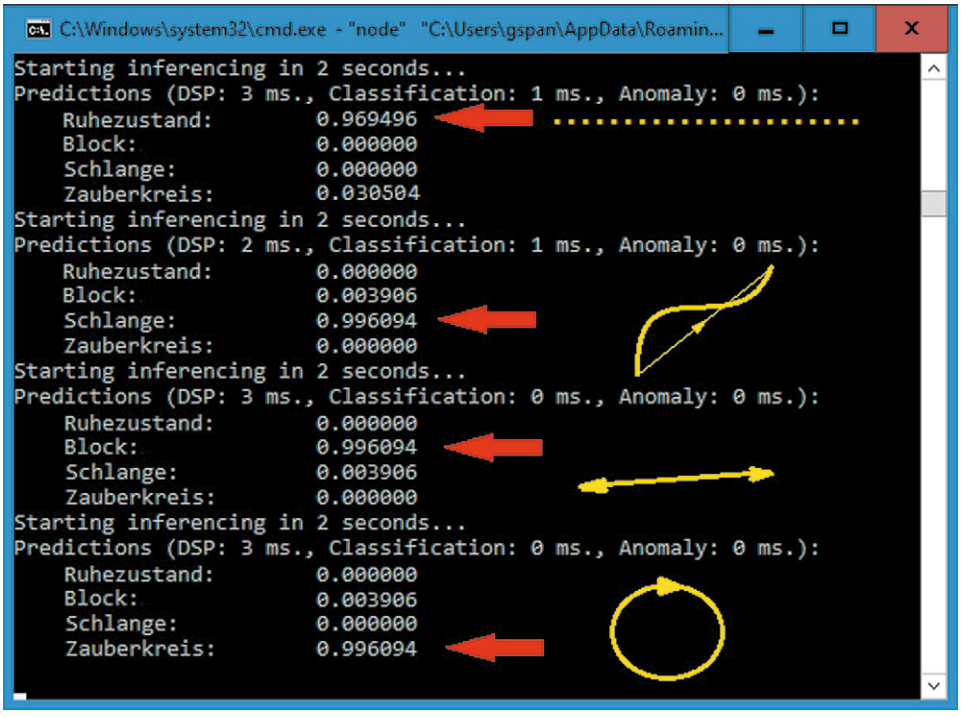


Bild 14: Ausgabewerte der „Zauberstab“-Firmware

Ruhezustand als solcher erkannt. Natürlich hängt die Qualität der Zuordnung davon ab, wie gut die Bewegungen mit den im Training verwendeten Mustern übereinstimmen.

Falls die Ergebnisse beim ersten Probelauf noch nicht zufriedenstellend sind, kann man die folgenden Optimierungen vornehmen:

1. Aufzeichnung zusätzlicher Trainingsdaten: Umfangreichere Trainingsdaten führen praktisch immer zu einer Verbesserung des Modells.
2. Modifizierung der Netzwerkstruktur: Mit mehreren Schichten und zusätzlichen Neuronen lässt sich die Genauigkeit des Netzwerks deutlich steigern.
3. Optimierung der Datenverarbeitungsparameter: Hier wurden zunächst nur die voreingestellten Werte übernommen. Durch Anpassung der Parameter sind zweifellos weitere Verbesserungen möglich.

Das Beispiel zeigt, dass auch mit dem Pico bereits komplexere Sensordaten über ML-Algorithmen ausgewertet werden können. Mithilfe der integrierten Analog-digital-Wandler können die Messwerte mit hoher Qualität direkt über den Controller erfasst werden. Nach einem weitgehend automatisierten Trainingsverfahren kann dann ein Neuronales Netzwerk erzeugt werden, das eine Zuordnung, d. h. eine Klassifizierung der Daten, ermöglicht. So werden aus den Rohdaten, wie etwa den hier verwendeten Beschleunigungswerten, bestimmte Muster extrahiert.

Material	Artikel-Nr.
Raspberry Pi Pico	251905
Experimentierboard EXSB1	153753
Beschleunigungsmesser mit Breakout-Board, z. B. ADXL330, ADXL335 oder ADXL(1)50	

Natürlich könnte man die erkannten Muster (Block, Kreis etc.) nun auch nutzen, um bestimmte Aktionen zu steuern. So ließe sich der Block etwa dazu verwenden, ein Fernsehgerät auszuschalten. Der Kreis könnte den aktuellen Programmkanal wechseln und die Schlange die Lautstärke regeln. Aber auch ein Roboterfahrzeug könnte so auf „magische Weise" gesteuert werden – der Fantasie sind hier keine Grenzen gesetzt ...

Ausblick

Mit diesem Artikel findet die vierteilige Serie zum Raspberry Pi Pico ihr Ende. Die Arbeit mit Edge zeigt, dass auch kleine Controller wie der Pico in der Lage sind, trainierte Modelle effizient zu nutzen. Allerdings wird deutlich, dass man es hier mit einer neuen und noch keineswegs voll ausgereiften Technologie zu tun hat. Insbesondere der Export der Modelle zum Controller arbeitet noch nicht sehr zuverlässig und man kann nur hoffen, dass in naher Zukunft verbesserte Tools verfügbar sein werden.

Dennoch erlaubt der Pico sehr preisgünstige KI-Anwendungen, und das Interesse an entsprechenden Applikationen wächst mit rasantem Tempo. Zudem ist der Pico durchaus auch in der Lage, komplexere ML-Algorithmen zu verarbeiten. So ist es möglich, beispielsweise über eine ArduCam auch visuelle Anwendungen umzusetzen. Allerdings sind hier dann bereits Lösungen mit einem klassischen Raspberry mit PiCam schon effizienter und auch kostengünstiger.

Trotzdem bleibt die Anwendung des bislang kleinsten und preisgünstigsten Mitglieds der Raspberry-Pi-Familie auch auf dem Gebiet der Künstlichen Intelligenz und des Machine Learnings ein hochinteressantes und extrem spannendes Entwicklungsfeld. ELV

i Weitere Infos

[1] ELV Journal 4/2021: Neuronale Netze - Aufbau und Training: Artikel-Nr. 252174
[2] Edge Impulse: https://www.edgeimpulse.com/

Download-Paket: Artikel-Nr. 253377

Alle Links finden Sie auch online unter: de.elv.com/elvjournal-links