OFFLINE-SPRACHEINGABE

Wakeword ohne Internet

Picovoice ist eine Bibliothek, die ohne Cloud-Anbindung intelligente Spracherkennung für verschiedene Betriebssysteme und Plattformen möglich macht.

Der Trend im Bereich von Dienstleistungen der künstlichen Intelligenz geht klar zur Cloud und zur minutenabhängigen Abrechnung. Mit Picovoice existiert eine durchaus leistungsfähige Bibliothek, die losgelöst von der Cloud und damit offline eine „lokale" Spracherkennung für unterschiedliche Betriebssysteme und Plattformen ermöglicht.

Gleich zu Beginn: Ja, es ist möglich und legitim, eine Picovoice-Anwendung komplett ohne Internetverbindung zu nutzen. Andererseits fallen auch dabei Lizenzkosten an – das Produkt ist also keine völlig kostenlose Lösung.

Was Picovoice genau kann und wie man es in der Praxis am besten einsetzt, möchten wir in diesem Artikel auf einer mit Windows 10 laufenden Workstation demonstrieren. Dass das Produkt neben dem hier verwendeten .NET Framework auch .NET Core, Node.js und sogar diverse Mikrocontroller-Systeme unterstützt, sei aus Gründen der Vollständigkeit angemerkt. Nun aber los – die Wakeword-Erkennung wartet auf uns!

Vorbereitungshandlungen

Einst geradezu legendär war Lernout & Hauspies Spracherkennungssoftware Dragon, die den Nutzer – zumindest bis zu einem gewissen Grad – zum Anschließen eines externen Mikrofons in möglichst hoher Qualität nötigte. In Interviews ließen Sprecher des Herstellers immer wieder durchsickern, dass es sich dabei um eine Maßnahme zur „Reduktion der Reklamationen" handelte, ist doch die Qualität der Spracherkennung direkt von der Qualität des Mikrofons abhängig. Der Autor wird in den folgenden Schritten jedenfalls mit einem Auna MIC900 arbeiten; auf Amazon finden sich derartige Exemplare zu einem durchaus guten Preis.

Die zweite Frage ist, wie wir die Kommunikation zwischen der Erkennungs-Engine und diesem Mikrofon abbilden. Die Picovoice-Entwickler empfehlen durch die Bank die Verwendung von OpenAL – unter macOS und Linux würde sich das Paket direkt aus den Paketquellen heraus installieren lassen. Da wir allerdings unter Windows arbeiten werden, ist im ersten Schritt der Besuch der OpenAL-Webseite erforderlich. Im dort herunterladbaren ZIP-Archiv [1] findet sich eine EXE-Datei, die Sie ausführen – damit ist die grundlegende Einrichtung, die wir für Picovoice benötigen, schon abgeschlossen.

Einrichtung der Infrastruktur

Als Erstes wollen wir mit Visual Studio 2019 arbeiten – erstellen Sie eine neue Applikation auf Basis der Vorlage *Konsolen-App (.NET Core)*. Der Name unseres Beispielprogramms soll in den folgenden Schritten *NMGVoiceRecA* lauten.

Nach der Erzeugung des eigentlichen Projektskeletts gilt es der Solution noch einige NuGet-Pakete hinzuzufügen. Erstens benötigen wir die als Porcupine bezeichnete Wakeword-Engine, von der es zum Zeitpunkt der Fertigstellung dieses

Artikels im Repository zwei verschiedene Varianten gab. Entscheiden Sie sich für die aktuellere der beiden, die zu diesem Zeitpunkt die Versionsnummer 1.9.0 aufwies.

Für die Kommunikation mit dem weiter oben installierten OpenAL-Audioframework benötigen wir dann noch OpenTK. Wir entscheiden uns auch hier für die aktuellste Variante – beachten Sie, dass Nicht-Core-Runtimes nur in 3.x-Versionen von OpenTK Unterstützung finden. Diese sind zudem nicht zu hundert Prozent API-kompatibel; die Klasse *ALCaptureDevice* ist in 3.x beispielsweise als *IntPtr* exponiert.

Die Cross-Plattform-Tauglichkeit von MSIL macht uns an dieser Stelle einen Strich durch die Rechnung. Nach dem Bereitstellen der NuGet-Pakete konfrontiert uns Visual Studio während der Kompilation mit der folgenden Fehlermeldung: *Konflikt zwischen der Prozessorarchitektur des Projekts „MSIL", das erstellt wird, und der Prozessorarchitektur des Verweises, "C:\Users\tamha\.nuget\packages\porcupine\.9.0\lib\netstandard2.0\Porcupine.dll", "AMD64".*

So verlockend es auf den ersten Blick sein mag, diesen Fehler zu ignorieren, so unzulässig ist es – die Picovoice-Bibliothek enthält nativen Code, was zur Laufzeit beim Laden der Bibliothek zu einem Programmabsturz führt.

Erfreulicherweise ist es nicht kompliziert, das Problem zu beheben: Klicken Sie auf *Erstellen* | *Konfigurations-Manager*, um das Konfigurations-Verwaltungswerkzeug zu laden. Dann klicken Sie auf die Combobox unter *Aktive Projektmappenplattform* und entscheiden sich dort für die Option *Neu*.

Visual Studio blendet daraufhin den in Bild 1 gezeigten Dialog ein, in dem Sie sich in der Rubrik *Neue Plattform eingeben oder auswählen* für die Option *x64* entscheiden. Nach dem Bestätigen durch einen Klick auf *OK* wählen Sie *x64* als Zielplattform aus – Bild 2 zeigt die korrekte Konfiguration.

Nach dem Schließen des Fensters können wir noch eine Neu-Kompilation befehligen; der Konflikt ist fortan verschwunden.

Nicht alles ist ein Indianer!

Das Picovoice-SDK steht – zumindest laut den Hinweisen in der NuGet-Konsole und Co. – theoretisch unter der Apache-Lizenz. Warum dies dennoch zu einer kommerziellen Nutzung nicht ausreicht, wollen wir uns anhand eines kleinen Beispiels ansehen. Hierzu öffnen wir die Hauptdatei unserer Konsolenanwendung und schreiben ihr im ersten Schritt insgesamt drei Members ein:

```
class Program
{
  static short[] myFramebuffer;
  static Porcupine myPorcupine;
  static ALCaptureDevice myDevice;
```

Während *myPorcupine* einen Verweis auf die Wakeword-Engine darstellt, ist das derzeit leer initialisierte Array ein Zwischenspeicher für die später von OpenAL angelieferten Audio-Samples. *myDevice* dient für das Audio-API dann zur Verwaltung eines Verweises auf das vom Betriebssystem zurückgegebene Audiogerät.

Im nächsten Schritt können wir auch schon damit beginnen, Initialisierungshandlungen vorzunehmen:

```
static void Main(string[] args) {
  myPorcupine = Porcupine.Create(keywords: new
    List<String> { "red", "green", "blue" });
  myFramebuffer = new short[myPorcupine.FrameLength];
  myDevice = OpenTK.Audio.OpenAL.ALC.CaptureOpenDevice(
    null, myPorcupine.SampleRate, ALFormat.Mono16,
    myPorcupine.FrameLength * 2);
}
```

Am wichtigsten ist die Erzeugung der Porcupine-Instanz, die in ihrem Konstruktor eine Liste zu aktivierender Wakewords erwartet. Nach dem erfolgreichen Durchlaufen von *Create* verwenden wir die in der Instanz angelegten Parameter, um damit den Framebuffer-Zwischenspeicher und die *ALCaptureDevice*-Geräteinstanz zu parametrieren. *myPorcupine.FrameLength * 2* ist dabei übrigens kein Tippfehler – die in *FrameLength* enthaltenen Informationen sind in Shorts, während die OpenAL-Engine die Puffergröße zwangsweise in Bytes erwartet.

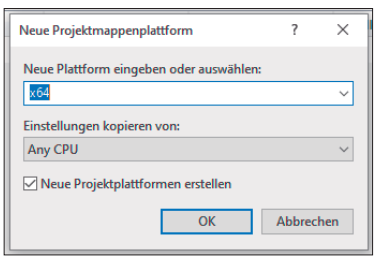
Die Ausführung des Programms führt dann zu dem in Bild 3 gezeigten Fehler. Die Ursache dafür liegt darin begründet, dass das Picovoice-SDK in Porcupine von Haus aus nur ein gutes Dutzend vorgegebener Wakewords unterstützt – möchte man selbst definierte Wakewords verwenden, so ist die Konsole im Backend erforderlich, die nur gegen Zahlung von Lizenzgebühren nutzbar ist.

Vorerst werden wir deshalb mit den bereits enthaltenen Keywords arbeiten. Dazu passen wir den Konstruktor im ersten Schritt nach folgendem Schema an, um nur noch in der SDK-Distribution enthaltene Wakewords zu verwenden:

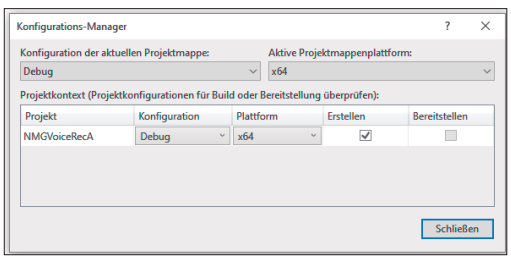
```
static void Main(string[] args) {
  myPorcupine = Porcupine.Create(keywords: new
  List<String> { "blueberry", "bumblebee", "computer",
  "grapefruit" });
```

Einrichtung der Audio-Pipeline

Unsere nächste Aufgabe besteht darin, das ALCaptureDevice zum Zurückliefern von Audio-Samples zu animieren. Dabei handelt es sich im Prinzip um eine Abfolge von Zah- ▶



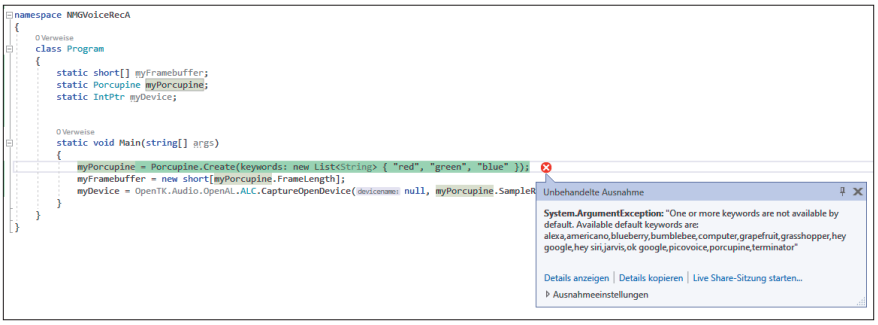
Dieses Fenster erlaubt die Einschränkung der Zielplattformen (Bild 1)



Sobald unser Projekt als Plattform „x64" aufweist, sind wir startklar (Bild 2)

len, welche jeweils die aktuell am AD-Wandler der Soundkarte anliegenden Spannungswerte repräsentieren. Nicht wissenschaftlich beschrieben handelt es sich damit um eine Art „Datenquelle“, die permanent – genauer gesagt von der Samplerate gesteuert – Daten ausspeit.

Unsere nächste Aufgabe ist deshalb, diese Informationen einzusammeln und danach auszugeben. Da das OpenAL-API nicht wirklich auf Listenern basiert, wollen wir an dieser Stelle – ganz wie in der Mikrocontroller-Programmierung – auf eine Endlosschleife setzen.



Die Initialisierung von Porcupine schlägt mit einem Verweis auf eine ungültige Stichwort-Liste fehl (Bild 3)

Ihre erste Aktion besteht darin, das Gerät zum Anliefern von Informationen zu animieren. Danach überprüfen wir permanent, ob im Sample-Puffer des Audiogeräts schon genug Informationen verfügbar sind, um der Porcupine-Engine einen vollständigen Daten-Frame offerieren zu können:

```
Task.Factory.StartNew(() => {
  ALC.CaptureStart(myDevice);
  while (true) {
    int samplesAvailable = ALC.GetAvailableSamples(
       myDevice);
    if (samplesAvailable > myPorcupine.FrameLength) {
      ALC.CaptureSamples(myDevice, ref myFramebuffer[0],
      myPorcupine.FrameLength);
```

Modelle der künstlichen Intelligenz können mit den übergebenen menschenlesbaren Strings im Allgemeinen nichts anfangen. Die meisten Modell-Generatoren verwerfen diese Informationen deshalb vergleichsweise früh und arbeiten nur mit numerischen IDs zur Beschreibung der einzelnen Klassen, siehe dazu auch den Kasten Was ist eine Klasse?

Im Fall der hier genutzten Porcupine-Instanz werden die für die einzelnen Klassen verwendeten IDs einfach aus der Reihenfolge abgeleitet, über die der Entwickler die Strings beim Konstruktor anliefert. Die erste Aufgabe unseres Programms besteht deshalb darin, die Funktion *myPorcupine.Process* aufzurufen und ihren Rückgabewert auszuwerten:

```
int keywordIndex = myPorcupine.Process(myFramebuffer);
if (keywordIndex >= 0) {
  switch (keywordIndex) {
    case 0:
      Console.WriteLine("Blueberry");
      break;
```

Features	Personal	Enterprise
Commercial Use		✓
Train Rhino Speech-to-Intent Models	✓	✓
Pre-trained Porcupine Wake Word Models	✓	✓
Train Porcupine Wake Word Models		✓
Microcontroller Access		✓
Community Support	✓	✓
Email Support		✓
Engineering Services		✓
Custom Platform Support		✓
# Devices	1	Custom
# Voice Interactions (per month)	1000	Custom
All models generated with Picovoice Console expire after 30 days. Users can retrain models before expiry to keep them functional.		

Bild: Picovoice.ai [2]

Die Erzeugung eigener Wakewords ist nur der kommerziellen Konsole vorbehalten (Bild 4)

```
    case 1:
      Console.WriteLine("Bumblebee");
      break;
    case 2:
      Console.WriteLine("Computer");
      break;
    case 3:
      Console.WriteLine("Grapefruit");
      break;
  }
}
```

Sofern Porcupine aus dem eingegebenen Daten-Frame kein Wakeword ableiten konnte, liefert die Engine einen negativen Zahlenwert zurück. Werte größer oder gleich null informieren stattdessen über die erfolgreiche Erkennung – wir nutzen in diesem Fall die Methode *Console.WriteLine*, um Statements in Richtung der Konsole auszugeben.

Da ein per *Task.Factory.StartNew* losgelassener Thread die Programmausführung nicht garantiert, müssen wir in der Methode *Main* zu guter Letzt noch eine weitere Endlosschleife platzieren – alternativ dazu bieten sich auch die diversen *Thread.Sleep*-Methoden an:

```
while (1 == 1) ;
```

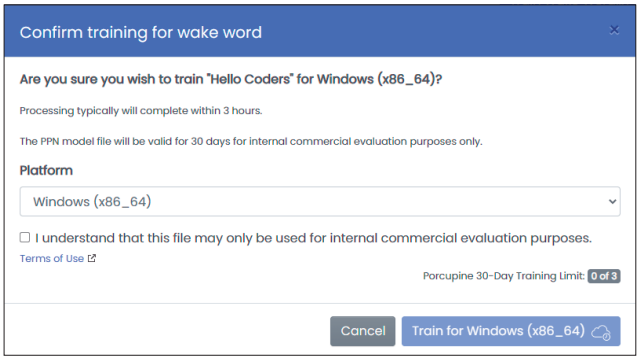
Was ist eine Klasse?

In der Welt der künstlichen Intelligenz bezeichnet man die Sortier-Kategorien, in die ein Modell die eingehenden Informationen unterteilt, als Klassen.

An dieser Stelle ist das Programm zur Ausführung bereit. Starten Sie die Applikation und sprechen Sie eines der vier Wakewords in das Mikrofon, um sich an der Ausgabe des jeweiligen Wortes in der Konsole zu erfreuen. In Tests des Autors funktionierte dies sogar mit seinem relativ akzentbehafteten „Austrian English".

Arbeit mit der Konsole

Nachdem wir uns in den bisherigen Abschnitten von der grundlegenden Funktionalität des Picovoice-SDK überzeugt haben, wollen wir nun dem Werkzeug ein eigenes Wakeword einschreiben. Hierzu benötigen wir die weiter oben erwähnte Konsole.



Der Trainingsprozess ist durchaus zeitintensiv (Bild 5)

Picovoice bietet zwei Versionen seines Produkts an [2], die sich – siehe Bild 4 – im Funktionsumfang unterscheiden.

Unsere nächste Aktion besteht darin, einen Picovoice-Account anzulegen. Dazu öffnen Sie die Picovoice-Konsole [3] in einem Browser Ihrer Wahl, klicken auf die Variante *Enterprise Console* und entscheiden sich im daraufhin erscheinenden Login-Fenster für die Option *Create a New Account*.

Der Account-Generierungsprozess selbst läuft komplett ohne menschliche Interaktion und ist binnen weniger Sekunden erledigt. Wichtig ist dabei nur das Übergeben einer gültigen E-Mail-Adresse, weil Picovoice einen zur Aktivierung des Accounts notwendigen Code an diese Adresse sendet.

Nach der erfolgreichen Anmeldung bietet uns die Picovoice-Konsole sowohl die Erzeugung von Modellen für die Porcupine-Wakeword-Engine als auch für ihre Kollegin Rhino an – Rhino kümmert sich dabei um eine als „Speech to Intent" bezeichnete Operation, die wir uns in einem Folgeartikel näher ansehen wollen.

Klicken Sie deshalb auf den blauen Button *Porcupine*, um sich daraufhin in der Wakeword-Liste der Konsole wiederzufinden. Interessant ist hier, dass das Eingabefenster über eine Combobox das Festlegen „beliebiger" Sprachen erlaubt – zum Zeitpunkt der Fertigstellung dieses Artikels war hier allerdings nur die Option für Englisch aktiviert.

Für einen ersten Test wollen wir eine ganze Phrase eingeben, weshalb wir im diesbezüglichen Eingabefeld den Text *Hello Coders* platzieren. Klicken Sie danach auf den hellblauen Knopf *Train Wakeword*. Porcupine blendet daraufhin den in Bild 5 gezeigten Dialog ein, in dem wir uns für die Option *Windows-Plattform* entscheiden. Beachten Sie, dass der Trainingsprozess – je nach Auslastung der Server – bis zu drei Stunden Zeit in Anspruch nimmt und dass Sie mit dem kostenlosen Evaluationskonto zudem nur drei derartige Trainingsvorgänge lostreten dürfen.

Nach dem erfolgreichen Trainingsprozess sendet das Backend Ihnen eine E-Mail, die Sie zum Wiedereinstieg ins Backend auffordert – der Download-Knopf liefert nach der Zustimmung zu einer weiteren rechtlichen Meldung eine Datei nach dem Schema *hello_coders_windows_23.2.2021_v1.9.0.zip*. Berücksichtigen Sie dabei, dass die in dieser Datei enthaltenen Modelldaten nur 30 Tage lang funktionieren – wichtig ist vor allem die PPN-Datei, die Sie in einem für Sie bequem zugänglichen Ort im Dateisystem platzieren. Der Autor wird in den folgenden Schritten mit *C:\picovoice\hello_coders_windows_2021-02-23-utc_v1_9_0.ppn* arbeiten.

Jede PPN-Datei ist dabei nur für ein Schlüsselwort verantwortlich, weshalb die angepasste Version des Konstruktors bei unserem einwertigen Modell folgendermaßen aussieht:

```
static void Main(string[] args) {
  myPorcupine = Porcupine.Create(keywordPaths: new
    List<string> { "C:\\picovoice\\hello_coders_
    windows_2021-02-23-utc_v1_9_0.ppn" });
  myFramebuffer = new short[myPorcupine.FrameLength];
```

Wichtig ist hier vor allem die Verwendung der Eigenschaft *keywordPaths*, über die der Parser PPN-Dateien entgegennimmt. Zur Laufzeit erweist sich das System übrigens als extrem streng – ist es auf „Hello Coders" trainiert, so wird „Hello Coder" reproduzierbar abgelehnt.

Fazit

Wakeword-Erkennung ermöglicht das schnelle Entgegennehmen einzelner Kommandos – dank Picovoice auch ohne Cloud-Verbindung. Wer statt Porcupine Rhino verwendet, kann im Rahmen der Wakeword-Sequenzen Parameter anliefern: eine Aufgabe, die wir uns in einem Folgeartikel [4] in dieser dotnetpro-Ausgabe ansehen wollen. ■

[1] OpenAL 1.1 Windows Installer, https://openal.org/downloads/oalinst.zip
[2] Picovoice-Versionen, https://picovoice.ai/docs/quick-start/console-signup
[3] Picovoice-Konsole, https://console.picovoice.ai
[4] Tam Hanna, Formelle Spracherkennung, dotnetpro 8/2021, Seite 12 ff., www.dotnetpro.de/A2108Picovoice2

Tam Hanna
entwickelt Programme für verschiedene Plattformen, betreibt Online-Newsdienste zum Thema und steht für Fragen, Trainings und Vorträge gern zur Verfügung. Sie erreichen ihn unter der E-Mail-Adresse **tamhan@tamoggemon.com**.

dnpCode A2108Picovoice