

Implementatie van een Freeze Detectie Systeem op de ESP32 met Machine Learning: Van Dataset tot Real-Time Voorspellingen

Inhoudsopgave

1. Inleiding en Doelstelling van het Project:	2
Onderzoeksvragen	2
2. Dataset	3
3. Machine learning (ML) trainen	4
3.1 Overzicht van de Machine Learning Modellen in GitLab	4
3.2 Data Preparation	6
3.3 Trainen van de modellen	13
3.4 Model performance	14
4. Overdracht van het Model naar de ESP32	17
4.1 Bestanden en Structuur	17
4.1.1 clf.h (C++ Header File)	17
4.1.2 sketch_mar20a.ino (Arduino Code)	18
4.2 Hoe het Model op de ESP32 kwam	19
4.3 Stappenplan: Het Model Implementeren op de ESP32 met Arduino IDE	20
4.3.1 Benodigdheden (soft- en hardware)	20
4.3.2 Downloaden van de Juiste Software (Windows)	23
4.3.3 Het Implementeren van de Code op de ESP32	33
5. Sensor vastmaken en output aflezen	40
6. 3D Geprinte Behuizing	42
6.1 Modellen	42
6.2 Printen van de Behuizing	43
6.3 Kiezen van het Filament	45

1. Inleiding en Doelstelling van het Project:

Dit project richt zich op de ontwikkeling van een realtime detectiesysteem voor Freezing of Gait (FoG) bij Parkinson-patiënten. FoG is een plotselinge, tijdelijke onmogelijkheid om de voeten vooruit te bewegen, ondanks de intentie om te lopen, en is een van de meest invaliderende symptomen van de ziekte van Parkinson. Dit systeem maakt gebruik van gegevens van een Inertial Measurement Unit (IMU) sensor, die versnellings- en gyroscoopdata verzamelt, gecombineerd met een machine learning-model dat is gedeployed op een ESP32 microcontroller. Het doel is om deze technologie in een draagbare en energiezuinige oplossing te integreren die kan bijdragen aan een tijdige interventie voor patiënten.

Onderzoeksvragen

Het hoofddoel van dit onderzoek is het ontwikkelen van een systeem dat FoG in Parkinson-patiënten kan detecteren in realtime met behulp van een AI-model en IMU-sensorgegevens. Dit doel wordt ondersteund door een aantal subvragen die de technische en wetenschappelijke aspecten van het systeem verder verfijnen:

Hoofdvraag: Hoe kan een AI-model worden geoptimaliseerd en gedeployed op een ESP32-module voor realtime detectie van Freezing of Gait bij Parkinson-patiënten?

Deelvragen:

- Welke gegevens zijn nodig om stappatronen effectief te analyseren?
- Welke open-access datasets zijn geschikt voor deze toepassing?
- Welke AI-modellen zijn het meest geschikt voor de detectie van Freezing of Gait?
- Hoe kan het model geoptimaliseerd worden voor embedded hardware zoals de ESP32?
- Wat zijn de uitdagingen bij het deployen van AI op de ESP32?
- Hoe kan realtime voorspelling worden bereikt op een apparaat met beperkte middelen?

Dit onderzoek beoogt het creëren van een real-time systeem dat nauwkeurige voorspellingen kan doen van FoG-episodes, met als doel om tijdige interventie mogelijk te maken. Het gebruik van het ESP32-platform biedt een goedkope, energiezuinige oplossing voor draagbare toepassingen, wat essentieel is voor het gebruik van het systeem in de dagelijkse zorg van Parkinson-patiënten.

2. Dataset

Het project maakt gebruik van een openbaar dataset van Boari et al. (2021):

- IMU sensordata (versnelling + gyrocoop)
- 35 Parkinson-patiënten die een "draai in plaats"-taak uitvoeren
- Geannoteerde Freezing of Gait (FoG) gebeurtenissen
- Steeksproef frequentie: 128 Hz
- Gebruikte kenmerken: ACC ML, ACC AP, ACC SI, GYR ML, GYR AP, GYR SI

Link naar het artikel van de dataset:

<https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2022.832463/full>

Link naar de dataset:

https://figshare.com/articles/dataset/A_public_dataset_of_video_acceleration_and_angular_velocity_in_individuals_with_Parkinson_s_disease_during_the_turning-in-place_task/14984667/7

- Download IMU.zip.
- De CSV met de gegevens voor staande positie is niet relevant voor de FoG-detectie en is daarom niet gebruikt in ons project.

3. Machine learning (ML) trainen

3.1 Overzicht van de Machine Learning Modellen in GitLab

Ga naar de Gitlab voor het terug vinden van de code:

https://gitlab.fdmci.hva.nl/lampela/fog_project_labspecifiek

Wanneer je op de bovenstaande link klikt, kom je terecht in de omgeving waar alle benodigde bestanden en links naar de dataset beschikbaar zijn (de Gitlab). Hier vind je ook het plan van aanpak voor het project, de presentatie voor de demo, het rapport en de bijlagen. Deze documenten zijn georganiseerd in verschillende mappen:

- Map "Arduino_Model": Bevat het model dat op de ESP is geïnstalleerd.
- Map "FOG_Detectie_MLModellen": Bevat de Python-code voor de vier verschillende machine learning-modellen, inclusief de bijbehorende evaluatiemetrics per model.
- Map "Rapport": Bevat het plan van aanpak, het rapport, de bijlagen en de presentatie.

Deze handleiding richt zich specifiek op de implementatie van het Random Forest-model op de ESP. De bijbehorende code is te vinden in de map:

The screenshot shows a GitLab repository page for 'FoG_Project_LabSpecifiek'. The repository has 24 commits, 1 branch, and 8 MiB of project storage. A table lists files and their details. The 'FoG_Detectie_MLModels' folder is highlighted with a red box. Below the table is a section titled 'Real-Time Freezing of Gait Detection on ESP32 Using AI' with a description of the project.

Name	Last commit	Last update
Arduino_Model	sketch_ino	2 weeks ago
FoG_Detectie_MLModels	Alle modellen + evaluatie	1 week ago
Rapport	presentatie	21 hours ago
README.md	Edit README.md	1 week ago

Real-Time Freezing of Gait Detection on ESP32 Using AI

This repository contains the code, trained model, and deployment instructions for a real-time Freezing of Gait (FoG) detection system. The project combines motion data from an IMU sensor with a machine learning model deployed on an ESP32 WROVER-E (Lolin D32 Pro) to assist in Parkinson's disease monitoring.

Wanneer je de map FOG_Detectie_MLModellen opent (de map die hierboven is omcirkeld), kom je op het volgende scherm terecht:

The screenshot shows a Git commit history for a repository named 'fog_project_labspecific / FoG_Detectie_MLModels'. The commit 'Alle modellen + evaluatie' was authored 1 week ago. A red box highlights the file 'MachineLearningModels_FoG.ipynb' in the list.

Name	Last commit	Last update
..		
MachineLearningModels_FoG.ipynb	Alle modellen + evaluatie	1 week ago

Klik nu op "MachineLearningModels_FoG.ipynb".

Let op: Wanneer een document de extensie ".ipynb" heeft, betekent dit dat het een Python-codebestand is.

Bij het openen van dit bestand zie je de code voor de vier verschillende modellen: Linear SVM, KNN, Decision Tree en Random Forest.

Voor gedetailleerde uitleg over de werking van deze modellen, kun je terecht op de officiële sklearn website: <https://scikit-learn.org/stable/>

The screenshot shows a Jupyter Notebook with two code cells. Cell [1] contains imports for os, random, pandas, numpy, train_test_split, RandomForestClassifier, metrics, export_graphviz, StringIO, pydotplus, Image, and port. Cell [2] contains a function 'load_freeze_data' that lists files in a folder, joins them with a path, reads an Excel file with engine 'xlrd', and initializes X_sequences and y_values lists. Total_rows is calculated as past_rows plus future_rows.

```
In [1]:  
import os  
import random  
import pandas as pd  
import numpy as np  
from sklearn.model_selection import train_test_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn import metrics  
from sklearn.tree import export_graphviz  
from io import StringIO  
import pydotplus  
from PIL import Image  
from micromlgen import port  
  
In [2]:  
def load_freeze_data(folder_path, past_rows=426, future_rows=1):  
    X_sequences, y_values = [], []  
    total_rows = past_rows + future_rows  
  
    for filename in os.listdir(folder_path):  
        file_path = os.path.join(folder_path, filename)  
        df = pd.read_excel(file_path, engine='xlrd')
```

3.2 Data Preparation

In deze preprocessing-pijplijn worden ruwe sensor data uit Excel-bestanden omgezet naar gestructureerde invoer die geschikt is voor het trainen van een machine learning model. De originele data bestaat uit tijdreeksmetingen van zes sensoren: drie versnellingsas (ACC ML, ACC AP, ACC SI) en drie gyroscoopassen (GYR ML, GYR AP, GYR SI).

Voor elke datasequentie voeren we feature engineering uit door deze tijdreeksen samen te vatten in een compacte weergave. In plaats van alle ruwe tijdstappen te gebruiken, berekenen we:

- De totale beweging (som van de waarden)
- De absolute totale beweging (som van de absolute waarden) voor elke van de zes assen.

Dit resulteert in 12 gemaakte features per sequentie (6 sensoren \times 2 samenvattende statistieken), die het algemene bewegingspatroon vastleggen zonder afhankelijk te zijn van de vorm van het ruwe signaal. Dit vermindert de dimensie en helpt het model om beter te generaliseren.

We zorgen ook voor een evenwichtige klasse door een gelijk aantal sequenties met en zonder een freeze-gebeurtenis te selecteren. Ten slotte wordt de data opgesplitst in training-, validatie- en testsets voor een eerlijke model-evaluatie.

Stap 1: inladen van de sensor data uit Excel

Wat: Alle .xls-bestanden in een opgegeven map worden geopend en verwerkt.

Waarom: Elk bestand bevat tijdreeksgegevens van sensoren (versnelling en gyroscoop), samen met hun labels (freeze-gebeurtenis).

Hoe: Dit wordt gedaan met de volgende regel code:

```
def load_freeze_data(folder_path, past_rows=426, future_rows=1):
    X_sequences, y_values = [], []
    total_rows = past_rows + future_rows

    for filename in os.listdir(folder_path):
        file_path = os.path.join(folder_path, filename)
        df = pd.read_excel(file_path, engine='xlrd')
        df = df.sort_values(by='Time [s]', ascending=True)

        X_sequences, y_values = add_file_with_freeze_chunks(X_sequences, y_values, df, total_rows, past_rows)

        print(f"Successfully processed {filename}.")

    return X_sequences, y_values
```

Stap 2: Verwerken van freeze-gebeurtenissen en het maken van feature-sequenties

Wat: De functie `add_file_with_freeze_chunks` verwerkt elke dataframe. Er worden twee soorten samples gecreëerd:

1. True-labels (Freeze-gebeurtenis): Eén sample per freeze-blok.
2. Rest van de labels: Vaste lengte-chunks van de rest van de gegevens.

Waarom: Deze stap bereidt de gegevens voor om een machine learning-model te trainen dat bewegingspatronen herkent die voorafgaan aan een freeze-gebeurtenis. Door slechts één sample per freeze-blok te selecteren, verminderen we redundantie en overfitting. Chunks van de niet-freeze delen zorgen ervoor dat het model een breed scala aan normale en overgangsbewegingen ziet voor robuust leren.

Hoe: Voor elk freeze-blok:

1. Kies één willekeurige rij met voldoende geschiedenis (`past_rows`).
2. De rijen vóór dit punt worden gebruikt om de beweging samen te vatten.
3. Feature engineering wordt toegepast op elke van de 6 sensorassen (3 van de versnellingsmeter en 3 van de gyrocoop). Twee statistieken worden berekend:
 - De totale beweging: som van alle waarden.
 - De absolute totale beweging: som van de absolute waarden.

Dit resulteert in 12 features per sequentie (6 assen \times 2 statistieken). We nemen de absolute waarde omdat, bijvoorbeeld, als een persoon 3 stappen naar rechts zet en daarna 3 stappen naar links, de totale beweging is: $+3 + (-3) = 0$. Dit zou betekenen dat het lijkt alsof de persoon helemaal niet bewoog. Echter, de absolute som zou zijn: $|+3| + |-3| = 3 + 3 = 6$, zodat we nu kunnen zien dat de persoon daadwerkelijk bewoog, alleen in tegenovergestelde richtingen.

4. De gebruikte rijen worden verwijderd om overlap in de training te voorkomen.
5. Voor de rest van de gegevens wordt de dataframe verdeeld in niet-overlappende chunks van `total_rows = past_rows + 1`. Dezelfde 12-feature transformatie wordt toegepast op elke chunk. Het label wordt bepaald op basis van of er een freeze-gebeurtenis plaatsvond in de laatste rij van de chunk.

```
def load_freeze_data(folder_path, past_rows=426, future_rows=1):
    X_sequences, y_values = [], []
    total_rows = past_rows + future_rows

    for filename in os.listdir(folder_path):
        file_path = os.path.join(folder_path, filename)
        df = pd.read_excel(file_path, engine='xlrd')
        df = df.sort_values(by='Time [s]', ascending=True)

        X_sequences, y_values = add_file_with_freeze_chunks(X_sequences, y_values, df, total_rows, past_rows)

        print(f"Successfully processed {filename}.")

    return X_sequences, y_values
```

```
def add_file_with_freeze_chunks(X_sequences, y_values, df, total_rows, past_rows):
    """
    Process a DataFrame by:
    1. Scanning for contiguous chunks of freeze events.
    2. For each chunk, randomly selecting one freeze event (with enough prior rows)
       and extracting the preceding 'past_rows' as a sequence (label=True).
    3. Removing these used rows and splitting the remaining data into sequences
       of length 'total_rows' (using the first 'past_rows' for feature extraction).

    Args:
        X_sequences (list): List to store feature sequences.
        y_values (list): List to store labels.
        df (pd.DataFrame): Input DataFrame.
        total_rows (int): Total number of rows per sequence/chunk.
        past_rows (int): Number of rows to use for feature extraction.

    Returns:
        (list, list): Updated X_sequences and y_values.
    """
    df = df.reset_index(drop=True)
    freeze_indices = df.index[df["Freezing event [flag]"] == 1].tolist()

    freeze_chunks = []
    if freeze_indices:
        current_chunk = [freeze_indices[0]]
        for idx in freeze_indices[1:]:
            if idx == current_chunk[-1] + 1:
                current_chunk.append(idx)
            else:
                freeze_chunks.append(current_chunk)
                current_chunk = [idx]
        freeze_chunks.append(current_chunk)

    used_indices = set()
```

```

for chunk in freeze_chunks:
    valid_indices = [idx for idx in chunk if idx >= past_rows]
    if valid_indices:
        chosen_idx = random.choice(valid_indices)
        seq = df.iloc[chosen_idx - past_rows: chosen_idx]

        data_subset = seq[['ACC ML [g]', 'ACC AP [g]', 'ACC SI [g]',
                           'GYR ML [deg/s]', 'GYR AP [deg/s]', 'GYR SI [deg/s]']]
        abs_sums = data_subset.abs().sum().to_frame().T.rename(
            columns=lambda x: f"{x} abstotalmovement"
        )
        reg_sums = data_subset.sum().to_frame().T.rename(
            columns=lambda x: f"{x} totalmovement"
        )
        x = pd.concat([abs_sums, reg_sums], axis=1)

        X_sequences.append(x)
        y_values.append(True)

        used_indices.update(range(chosen_idx - past_rows, chosen_idx))

remaining_df = df.drop(list(used_indices)).sort_values(by="Time [s]").reset_index(drop=True)

num_sequences = remaining_df.shape[0] // total_rows
for i in range(num_sequences):
    start_idx = i * total_rows
    end_idx = start_idx + total_rows
    chunk = remaining_df.iloc[start_idx:end_idx]

    data_subset = chunk.iloc[:past_rows][['ACC ML [g]', 'ACC AP [g]', 'ACC SI [g]',
                                         'GYR ML [deg/s]', 'GYR AP [deg/s]', 'GYR SI [deg/s]']]
    abs_sums = data_subset.abs().sum().to_frame().T.rename(
        columns=lambda x: f"{x} abstotalmovement"
    )
    reg_sums = data_subset.sum().to_frame().T.rename(
        columns=lambda x: f"{x} totalmovement"
    )
    x = pd.concat([abs_sums, reg_sums], axis=1)
    y = chunk.iloc[total_rows - 1]["Freezing event [flag]"] == 1
    X_sequences.append(x)
    y_values.append(y)

return X_sequences, y_values

```

Stap 3: Het balanceren van de gegevens

Wat: Als er meer False-labels dan True-labels zijn, worden de False-label samples willekeurig geselecteerd om de True-labels in balans te brengen.

Waarom: Dit wordt gedaan om een gebalanceerde dataset te creëren. Ongebalanceerde datasets zorgen ervoor dat modellen een voorkeur ontwikkelen voor het dominante label, wat de prestaties van het model kan beïnvloeden en het vermogen om de minderheidscategorie te voorspellen vermindert.

Hoe:

- De False-labels worden willekeurig geselecteerd om ervoor te zorgen dat het aantal False-labels gelijk is aan het aantal True-labels.
- Dit voorkomt dat het model teveel leert van de dominante klasse (False) en zorgt ervoor dat het model beide klassen even goed leert herkennen.

```
def balance_data(X_sequences, y_values):  
    true_count = y_values.count(True)  
    false_count = y_values.count(False)  
    print(f"Before balancing: True={true_count}, False={false_count}")  
  
    if false_count > true_count:  
        true_indices = [i for i, v in enumerate(y_values) if v]  
        false_indices = [i for i, v in enumerate(y_values) if not v]  
        sampled_false_indices = random.sample(false_indices, true_count)  
        balanced_indices = sorted(true_indices + sampled_false_indices)  
        balanced_X = [X_sequences[i] for i in balanced_indices]  
        balanced_y = [y_values[i] for i in balanced_indices]  
    else:  
        balanced_X, balanced_y = X_sequences, y_values  
  
    print(f"After balancing: True={balanced_y.count(True)}, False={balanced_y.count(False)}")  
    return balanced_X, balanced_y  
  
folder_path = 'C:/Users/lotte/OneDrive/Master AAI/labspecifiek/IMU (1)/IMU'  
X_sequences, y_values = load_freeze_data(folder_path)  
X_balanced, y_balanced = balance_data(X_sequences, y_values)  
  
# Flatten X sequences for model training  
X_flattened = [x.seq.values.flatten() for x_seq in X_balanced]  
X = np.array(X_flattened)  
y = np.array(y_balanced).astype(bool)  
  
# Split data into train (70%), test (15%), validation (15%)  
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)  
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp)  
  
print(f"Data split: Train={len(X_train)}, Test={len(X_test)}, Validation={len(X_val)}")
```

Stap 4: Het platmaken van de feature vectoren

Wat: De data frame-sequenties worden geconverteerd naar platte NumPy-arrays.

Waarom: Machine learning-modellen zoals Random Forest verwachten vervlakte feature vectoren per sample (geen ruwe dataframes).

Hoe: De sequenties in de data frame worden omgezet naar NumPy-arrays door het flattenen van de gegevens. Dit proces zorgt ervoor dat elke sample als een enkele, platte vector wordt gepresenteerd, die de machine learning-modellen kunnen gebruiken voor training. Dit is te zien in de onderstaande code (en afbeelding):

De flattening gebeurt met de volgende regel code:

```
folder_path = 'C:/Users/lotte/OneDrive/Master AAI/labspecifiek/IMU (1)/IMU'
X_sequences, y_values = load_freeze_data(folder_path)
X_balanced, y_balanced = balance_data(X_sequences, y_values)

# Flatten X sequences for model training
X_flattened = [x_seq.values.flatten() for x_seq in X_balanced]
X = np.array(X_flattened)
y = np.array(y_balanced).astype(bool)

# Split data into train (70%), test (15%), validation (15%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp)

print(f"Data split: Train={len(X_train)}, Test={len(X_test)}, Validation={len(X_val)}")
```

Deze stap zorgt ervoor dat de sequenties van sensorgegevens geschikt zijn voor het trainen van het model.

Stap 5: Train/Test/Validatie splitsing

Wat: De gebalanceerde data wordt verdeeld in:

- 70% train data
- 15% test data
- 15% validatie data

Waarom?: Om het model te trainen en te evalueren met ongeziene data. Dit zorgt ervoor dat het model niet overfit op de training data en goed generaliseert naar nieuwe data.

Hoe?: De data wordt gesplitst met behulp van de `train_test_split` functie uit de `sklearn.model_selection` module. De verdeling is als volgt ingesteld:

- 70% van de data wordt gebruikt voor het trainen van het model.
- 15% wordt gebruikt voor het testen van het model.
- 15% wordt gebruikt voor de validatie van het model.

De code hiervoor is als volgt:

```
import os
import random
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.tree import export_graphviz
from io import StringIO
import pydotplus
from PIL import Image
from micromlgen import port

# Split data into train (70%), test (15%), validation (15%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp)

print(f"Data split: Train={len(X_train)}, Test={len(X_test)}, Validation={len(X_val)}")
```

3.3 Trainen van de modellen

In dit hoofdstuk wordt de code kort toegelicht om te verduidelijken hoe de modellen zijn getraind. Nadat de dataset is opgesplitst, worden de modellen getraind. Bij het trainen is het essentieel dat de training plaatsvindt op de trainingsdatasets “X_train” en “y_train”. In de code maken we gebruik van standaard hyperparameters om het model te trainen. Met uitzondering van de modelnaam en de hyperparameters is de code verder identiek. In de onderstaande screenshots wordt geïllustreerd hoe de modellen kunnen worden getraind. Uiteindelijk blijkt dat de Random Forest het beste model is, waardoor het mogelijk zou zijn om uitsluitend dat model te trainen.

Random Forest

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

# Initialize Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, criterion='gini', max_depth=2, random_state=0, bootstrap=True, oob_score=True)

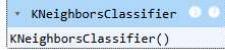
# Train the classifier
model.fit(X_train, y_train)
```



KNN Classifier

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix, roc_curve, auc
import seaborn as sns
import matplotlib.pyplot as plt

# 1. Initialiseer en train het KNN-model
modelknn = KNeighborsClassifier(n_neighbors=5) # Je kunt dit aanpassen (bijv. 3 of 7)
modelknn.fit(X_train, y_train)
```

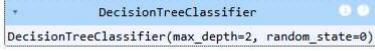


Decision Tree

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier

# 1. Initialiseer en train het Decision Tree-model
modeldt = DecisionTreeClassifier(criterion='gini', max_depth=2, random_state=0)
modeldt.fit(X_train, y_train)
```



Lineair SVC

Lineair SVC

```
from sklearn.svm import LinearSVC  
  
modelsvc = LinearSVC(max_iter=10000, random_state=0)  
modelsvc.fit(X_train, y_train)
```

```
+-----+  
| LinearSVC |  
+-----+  
LinearSVC(max_iter=10000, random_state=0)
```

3.4 Model performance

Nadat het model is getraind moeten de resultaten worden getraind. Hiervoor kijken wij naar de train en test data. Als er wordt geïtereerd op het model is het belangrijk om te itereren op de train en validatie data. De train data zou alleen gezien moeten worden na de laatste iteratie van het model.

```
print("Training Performance:")  
train_pred = model.predict(X_train)  
print(metrics.classification_report(y_train, train_pred, digits=3))  
print(metrics.confusion_matrix(y_train, train_pred))  
  
print("Test Performance:")  
test_pred = model.predict(X_test)  
print(metrics.classification_report(y_test, test_pred, digits=3))  
print(metrics.confusion_matrix(y_test, test_pred))
```

Nadat de performance code is uitgevoerd wordt er een classificatie rapport zichtbaar. In dit rapport kan de precisie, recall en meer waarden worden uitgelezen. De accuracy is hoe vaak het model zegt dat het correct is terwijl de waarde ook echt correct is (true positives) en de recall zegt wanneer het model zegt dat het fout is en dat het echt fout is (true negatives).

```
Training Performance:  
precision    recall    f1-score   support  
  
 False       0.773     0.882     0.824      389  
 True        0.862     0.740     0.797      389  
  
accuracy          0.811      778  
macro avg       0.817     0.811     0.810      778  
weighted avg     0.817     0.811     0.810      778  
  
[[343  46]  
 [101 288]]  
Test Performance:  
precision    recall    f1-score   support  
  
 False       0.835     0.916     0.874      83  
 True        0.908     0.821     0.863      84  
  
accuracy          0.868      167  
macro avg       0.872     0.869     0.868      167  
weighted avg     0.872     0.868     0.868      167  
  
[[ 76   7]  
 [15 69]]
```

Naast het classificatierapport zijn er meer manieren om de performance van de AI modellen te meten. Door de code hieronder kan een ROC curve worden gemaakt.

```

from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Stap 1: Voorspel waarschijnlijkheden voor de positieve klasse (FoG = True)
y_prob = model.predict_proba(X_test)[:, 1]

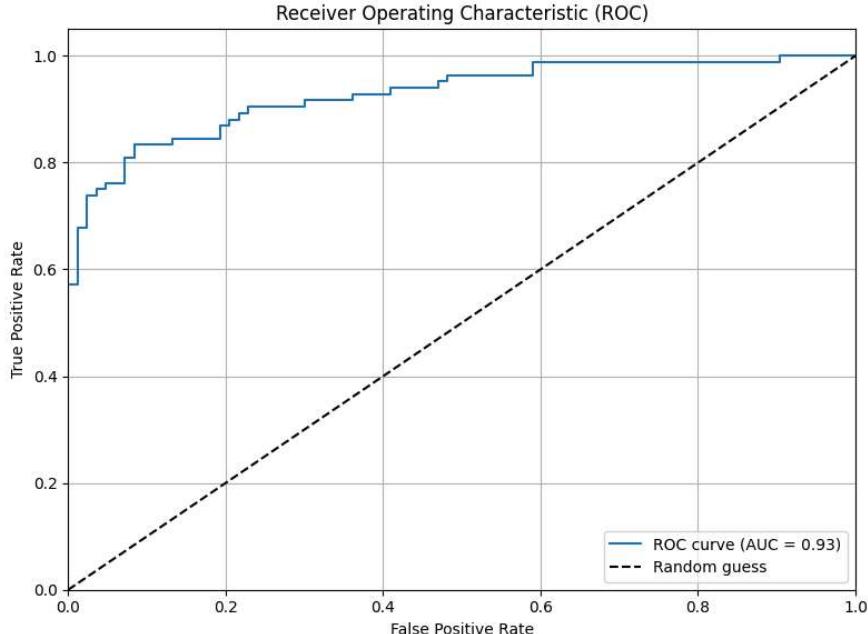
# Stap 2: Bereken ROC-waarden
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

# Stap 3: Plot de ROC-curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"ROC curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], 'k--', label='Random guess')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic (ROC)")
plt.legend(loc="lower right")
plt.grid(True)
plt.tight_layout()
plt.show()

```

In de ROC curve kan worden gezien op de X axis van de tabel hoeveel true positives er zijn en hoeveel true negatives er zijn op de Y axis. Als de lijn in de buurt van de linkerbovenhoek komt is de performance van het model hoog. Als de lijn symmetrisch is, dan is het model even goed in alle klassen classificeren.

1]:



De laatste metric waar we gebruik van maken is de confusion matrix. Hierin kan gedetailleerd worden gezien hoeveel fouten waar zijn gemaakt in het model. De code om de confusion matrix te maken staat hieronder.

```

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

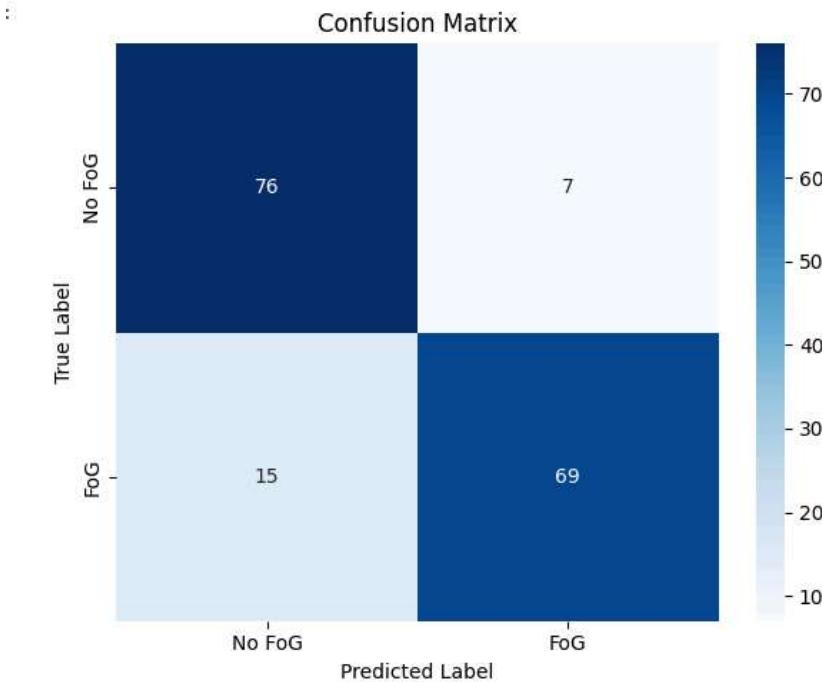
# Stap 1: Maak voorspellingen
y_pred = model.predict(X_test)

# Stap 2: Genereer confusie matrix
cm = confusion_matrix(y_test, y_pred)
labels = ['No FoG', 'FoG'] # Pas aan als je klassen anders noemt

# Stap 3: Plot de confusie matrix
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.tight_layout()
plt.show()

```

Op de confusie matrix staan op de Y axis de waarden zoals ze in de dataset zijn geklassificeerd en op de X axis als zoalde waarden door het AI model zijn geklassificeerd. Zo kan je zien in welke classificaties het model goed en slecht is.

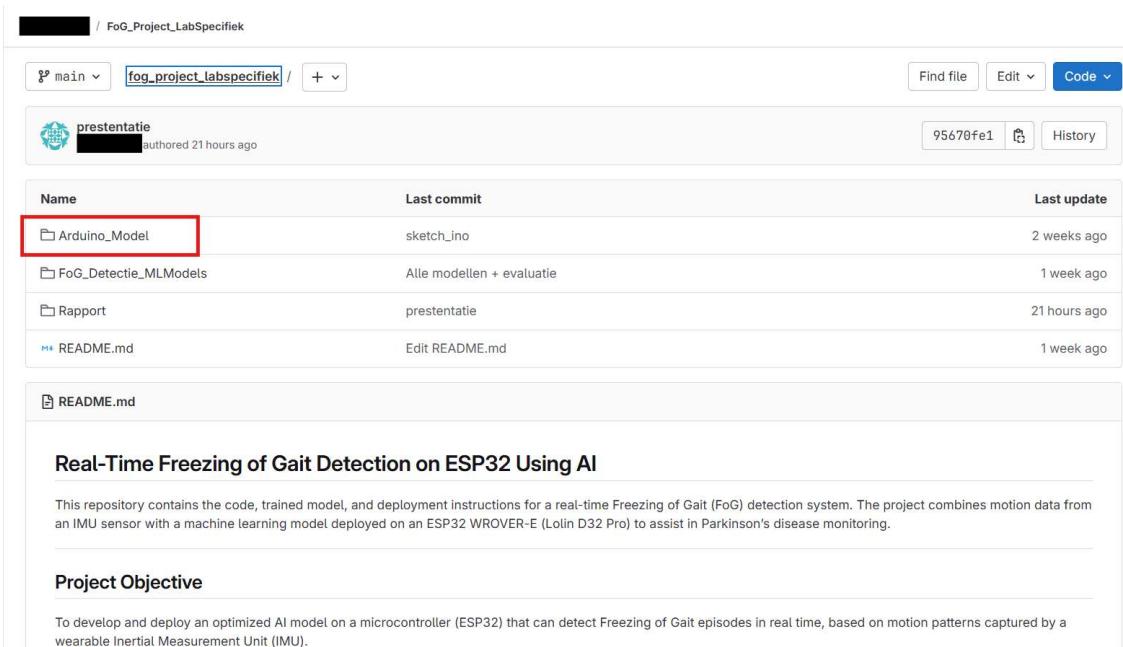


4. Overdracht van het Model naar de ESP32

In dit hoofdstuk leggen wordt uitgelegd hoe het originele Python-model voor het detecteren van freezes (zoals geanalyseerd in de MachineLearningModels_FoG.ipynb) is omgezet naar een werkende implementatie op de ESP32 microcontroller. Deze transformatie maakt gebruik van C++-code en specifieke bibliotheken om het model te draaien zonder de krachtige rekenkracht van een computer.

4.1 Bestanden en Structuur

Als je terugkomt op het beginscherm van de Gitlab omgeving, zie je weer het overzicht van alle mapjes. Klik hier op het mapje "Arduino_Model". Hier is de code terug te vinden die geïmplementeerd is op de ESP.



The screenshot shows a Gitlab repository interface. At the top, there's a navigation bar with 'main' (dropdown), 'fog_project_labspecific' (selected tab), and '+'. On the right are 'Find file', 'Edit' (dropdown), and 'Code' (dropdown). Below the navigation is a commit card for 'prestentatie' (authored 21 hours ago, commit 95670fe1) with 'History' and a copy icon. The main area lists files and folders:

Name	Last commit	Last update
Arduino_Model	sketch_ino	2 weeks ago
FoG_Detectie_MLModels	Alle modellen + evaluatie	1 week ago
Rapport	prestentatie	21 hours ago
README.md	Edit README.md	1 week ago

Below the table is a section titled 'Real-Time Freezing of Gait Detection on ESP32 Using AI' with a description: 'This repository contains the code, trained model, and deployment instructions for a real-time Freezing of Gait (FoG) detection system. The project combines motion data from an IMU sensor with a machine learning model deployed on an ESP32 WROVER-E (Lolin D32 Pro) to assist in Parkinson's disease monitoring.' A 'Project Objective' section follows, stating: 'To develop and deploy an optimized AI model on a microcontroller (ESP32) that can detect Freezing of Gait episodes in real time, based on motion patterns captured by a wearable Inertial Measurement Unit (IMU).'

De code bestaat uit verschillende belangrijke onderdelen die samenwerken om het model op de ESP32 te laten draaien:

4.1.1 clf.h (C++ Header File)

Dit bestand bevat de implementatie van het Random Forest-model, dat de kern van de voorspelling vormt. Het model is omgezet van een Python RandomForestClassifier naar een C++ versie die geschikt is voor gebruik op de ESP32. Het belangrijkste onderdeel is de RandomForest class, die een predict functie bevat. Deze functie ontvangt een array van 12 waarden (features) en doorloopt verschillende beslisregels gebaseerd op de getrainde bomen in het Random Forest. Voor elke boom in het bos wordt een beslissing gemaakt of de input tot een freeze (True) of geen freeze (False) leidt. Het model telt de stemmen van alle bomen en kiest de meerderheid als de uiteindelijke voorspelling.

Belang van dit bestand:

- Het clf.h bestand zorgt voor de implementatie van het machine learning-model in een formaat dat compatibel is met de ESP32. Het bevat de logica voor de voorspelling door het Random Forest.
- Het maakt het mogelijk om de getrainde modellen (oorspronkelijk in Python) te exporteren en toe te passen in een embedded omgeving zoals de ESP32, die beperkt is in geheugen en rekenkracht.

The screenshot shows a GitHub repository interface for a project named 'Fog_Project_LabSpecific'. The repository path is 'fog_project_labspecific / Arduino_Model'. The main tab is selected, showing a list of files. One file, 'clf.h', is highlighted with a red box. The table below lists the files with their names, last commit details, and last update times.

Name	Last commit	Last update
..		
.gitkeep	adruino_Model dir	2 weeks ago
clf.h	rf.h file for arduino	2 weeks ago
sketch_mar20a.ino	sketch_ino	2 weeks ago

4.1.2 sketch_mar20a.ino (Arduino Code)

In dit bestand vinden we de hoofdlus (loop) van de code die draait op de ESP32. De belangrijkste taak van deze code is om gegevens van de MPU6050 sensor (versnellingsmeter en gyroscoop) in te lezen, te verwerken, en vervolgens een voorspelling te maken met het getrainde Random Forest-model. De sensor wordt via I2C verbonden en verzamelt continu versnellings- en gyroscoopdata. Deze gegevens worden vervolgens omgezet in de 12 inputkenmerken die het model vereist.

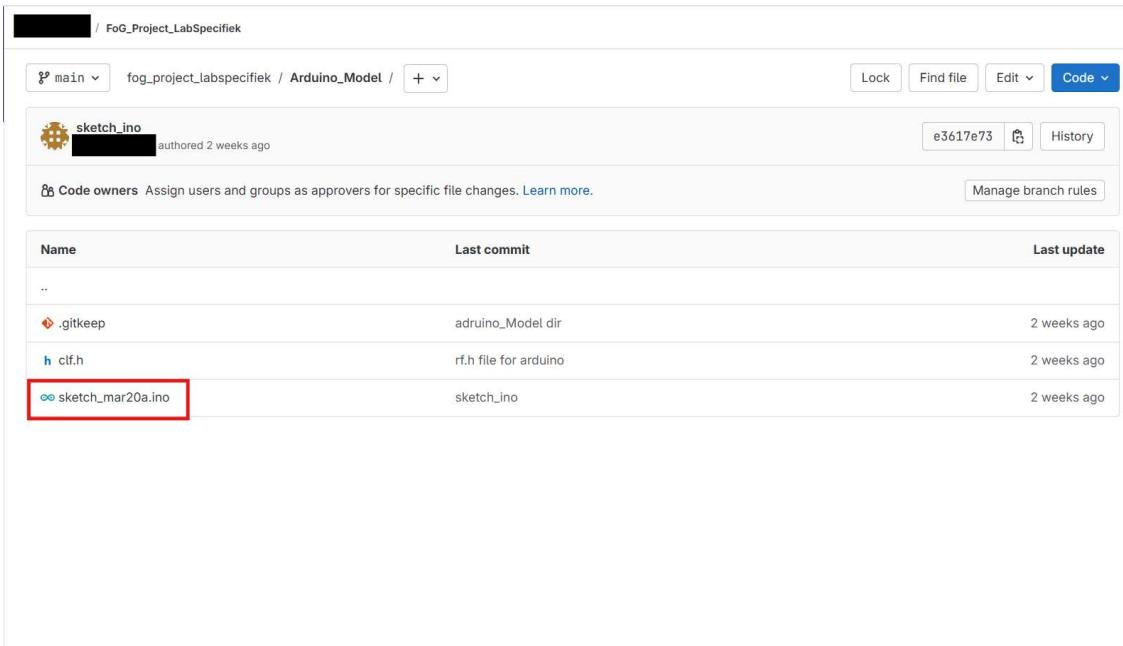
Functie van de sketch_mar20a.ino:

- MPU6050 Sensor: De code leest versnellings- en gyroscoopdata van de sensor in, zet deze om naar g-waarden voor versnelling en graden per seconde voor rotatie (de eenheden die tijdens de training zijn gebruikt).
- Feature Extractie: De sensor verzamelt de data in een buffer. Zodra de buffer vol is, worden de absolute waarden en de som van de versnellings- en gyroscoopmetingen over de verzamelde data berekend. Deze worden als features gebruikt voor de voorspelling.
- Modelvoorspelling: Na de feature-extractie wordt het Random Forest-model gebruikt om te voorspellen of een freeze heeft plaatsgevonden (output is 1 voor freeze, 0 voor geen freeze).
- Bluetooth Communicatie: De voorspellingen worden via Bluetooth verzonden naar een gekoppeld apparaat, zodat de gebruiker op de hoogte wordt gesteld van een

gedetecteerde freeze.

Het belang van dit bestand:

- Verwerken van sensordata: Dit bestand zorgt voor het verzamelen en verwerken van de gegevens van de sensor, en maakt het mogelijk om deze data te gebruiken in de machine learning-predictie.
- Draadloze communicatie: Het maakt Bluetooth-communicatie mogelijk, zodat het systeem in real-time freeze-detecties kan doorgeven aan een gekoppeld apparaat, bijvoorbeeld een smartphone of computer.
- Runnen van machine learning op embedded systeem: Het maakt het mogelijk om een getraind model op een ESP32 te draaien en voorspellingen te doen met behulp van de beperkte hardware van de microcontroller.



The screenshot shows a GitHub repository interface. At the top, there's a navigation bar with 'main' selected, followed by 'fog_project_labspecific / Arduino_Model'. On the right side of the top bar are buttons for 'Lock', 'Find file', 'Edit', and 'Code'. Below the top bar, there's a commit card for 'sketch_arduino.ino' authored 2 weeks ago. To the right of the commit card are buttons for 'e3617e73', 'History', and 'Manage branch rules'. Below the commit card, there's a note about 'Code owners' and a link to 'Learn more'. The main content area is a table showing file details:

Name	Last commit	Last update
..		
.gitkeep	adruino_Model dir	2 weeks ago
clf.h	rf.h file for arduino	2 weeks ago
sketch_arduino.ino	sketch_arduino.ino	2 weeks ago

The last row, 'sketch_arduino.ino', has a red box drawn around it.

4.2 Hoe het Model op de ESP32 kwam

Om het getrainde machine learning-model, dat oorspronkelijk in Python ontwikkeld is, naar de ESP32 over te zetten, hebben we gebruikgemaakt van de **micromlgen-bibliotheek** in Python. Deze bibliotheek converteert een getraind model naar C++-code, die rechtstreeks op embedded systemen zoals de ESP32 kan worden gebruikt. De Arduino IDE 2.3.6 werd hierbij ingezet om de code te schrijven, te compileren en naar de ESP32 te uploaden.

1. Modeltraining in Python (hoofdstuk 3)

In de originele Python-code werd het Random Forest-model getraind met behulp van de scikit-learn-bibliotheek. Het model werd getraind op gegevens van de MPU6050-sensor uit de gebruikte dataset (hoofdstuk 2), die versnellings- en gyroscoopdata verzamelt. Het doel van het model was om te voorspellen of een "freeze" (bevriezing van de beweging) was opgetreden op basis van deze sensorinformatie. De training vond plaats op een dataset die bestond uit bewegingsdata, waarbij de freezing events werden gemarkerd.

2. Model Exporteren

Na het trainen van het model in Python, werd het getransformeerd naar C++ door gebruik te maken van de micromlgen-bibliotheek. Deze bibliotheek genereert C++-code die de functionaliteit van het getrainde model reproduceert, zodat het rechtstreeks op een embedded systeem kan draaien. Het resultaat van deze stap was de **clf.h** header file die de Random Forest-logica bevatten.

3. Inbedden in de ESP32

De gegenereerde clf.h werd vervolgens toegevoegd aan de Arduino-sketch (bestandsnaam: sketch_mar20a.ino). Samen met de nodige bibliotheken voor de sensor (Adafruit_MPU6050) en voor draadloze communicatie via Bluetooth (BluetoothSerial), werd de code klaargemaakt om op de ESP32 te draaien. De Arduino IDE 2.3.6 werd gebruikt om de code te schrijven, te testen en uiteindelijk te uploaden naar de ESP32-microcontroller. Dit zorgde ervoor dat de ESP32 de sensor kon uitlezen en de voorspellingen in real-time kon uitvoeren.

4. Realtime Voorspellingen

Zodra de ESP32 draait, leest hij de gegevens van de MPU6050-sensor in real-time uit. Deze gegevens worden gebruikt om een voorspelling te maken met het getrainde Random Forest-model, dat in C++ op de ESP32 draait. Het resultaat van de voorspelling wordt via Bluetooth naar een gekoppeld apparaat gestuurd. Dit maakt het mogelijk om een freeze in real-time te detecteren en onmiddellijk een waarschuwing te sturen naar een ander apparaat, zoals een smartphone of computer.

In het volgende hoofdstuk (4.3) wordt een gedetailleerd stappenplan/tutorial gegeven over hoe je deze software op een ESP32 installeert, zodat je zelf een real-time freeze-detectie systeem kunt maken met behulp van de Arduino IDE en ESP32.

4.3 Stappenplan: Het Model Implementeren op de ESP32 met Arduino IDE

4.3.1 Benodigdheden (soft- en hardware)

1. Benodigde Software

In hoofdstuk 4.3.2 wordt de installatie en configuratie van de benodigde software stap voor stap uitgelegd. Dit zorgt ervoor dat je de juiste programma's hebt om het systeem te programmeren en te testen.

2. De Hardware

- LOLIN D32 PRO

Dit is een ontwikkelbord met een ESP32 microcontroller. De ESP32 is het hart van het systeem en wordt gebruikt voor het uitvoeren van het machine learning-model en het verwerken van de sensorgegevens. Het bord heeft verschillende I/O-pinnen voor het aansluiten van sensoren en andere randapparatuur.



- IMU 6050: voor het meten van de gyro en accelero data

De IMU (Inertial Measurement Unit) 6050 is een sensormodule die zowel een accelerometer als een gyroscoop bevat. Deze sensoren worden gebruikt om de beweging van het lichaam (specifiek de stappen en de houding) te meten, wat essentieel is voor de Freezing of Gait (FoG) detectie. De IMU 6050 levert data die wordt gebruikt om te bepalen of een Parkinson-patiënt een freeze van het gangpad ervaart.



- Jumper wires

Dit zijn flexibele draadjes die worden gebruikt om verbindingen te maken tussen het ontwikkelbord (LOLIN D32 PRO) en de sensoren (zoals de IMU 6050). Ze zijn essentieel voor het correct aansluiten van de hardwarecomponenten.

Koppelen:

In de tabel hieronder staat hoe je aan de hand van de jumping wires de LOLIN D32 Pro koppelt aan de IMU 6050.

LOLIN D32 PRO	IMU 6050
3V	VCC
GND	GND
DAC2	SCL
DAC1	SDA



- Lipo Accu

De Lipo-batterij (Lithium Polymer) biedt de benodigde stroomvoorziening voor de ESP32 en andere aangesloten hardware. Het is belangrijk om een geschikte batterij te kiezen die voldoende capaciteit heeft om het systeem langdurig van stroom te voorzien.



- Micro USB kabel: voor het koppelen van de LOLIN32 (board waar de ESP op ligt)
Deze kabel wordt gebruikt voor het verbinden van het LOLIN32-ontwikkelbord met de computer. Via deze verbinding kun je de code uploaden naar de ESP32 en de voeding leveren aan het bord. De Micro USB-kabel wordt ook gebruikt om de ESP32 via een computer op te laden.



4.3.2 Downloaden van de Juiste Software (Windows)

Voordat je kunt beginnen met het implementeren van het model op de ESP32, moet je de juiste software en tools op je Windows-pc installeren. Volg de onderstaande stappen om alles in te stellen:

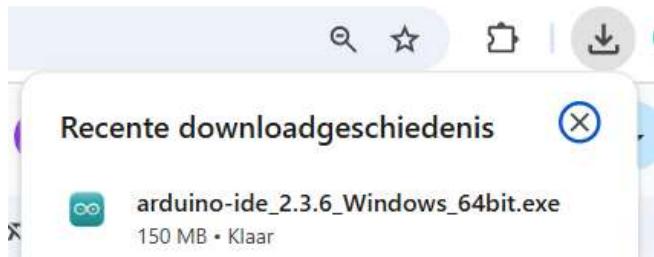
1. Downloaden en Installeren van de Arduino IDE

De eerste stap is het downloaden van de Arduino IDE, de software waarmee we de code schrijven en uploaden naar de ESP32. Volg de onderstaande stappen:

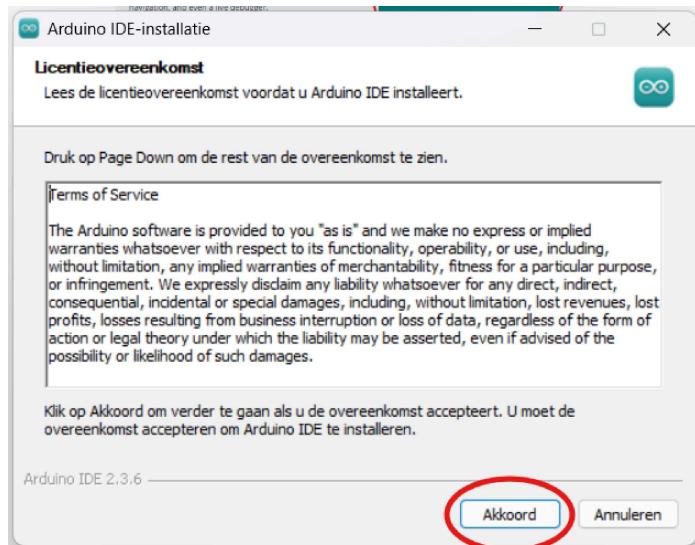
1. Ga naar de officiële Arduino website: <https://www.arduino.cc/en/software/#ide>
2. Onder het gedeelte voor Windows, klik je op Windows Win 10 and newer om het installatiebestand te downloaden. (Klik telkens op “just download”).

The screenshot shows the Arduino website's download page. At the top, there are links for Professionals, Education, and Makers. Below that is a navigation bar with Products, Community, Documentation, a SHOP button, a search icon, and a Cloud icon. The main content area features a section for the Arduino Cloud Editor, followed by a large image of a hand interacting with a computer screen displaying the Arduino IDE interface. Below this is a 'Downloads' section. On the left, there's a thumbnail for the Arduino IDE 2.3.6 release, which includes a brief description and a link to the documentation. On the right, under 'DOWNLOAD OPTIONS', there are links for Windows (Win 10 and newer, 64 bits; MSI installer; ZIP file), Linux (AppImage 64 bits (X86-64); ZIP file), and macOS (Intel, 10.15: "Catalina" or newer, 64 bits; Apple Silicon, 11: "Big Sur" or newer, 64 bits). A red circle highlights the 'DOWNLOAD OPTIONS' section.

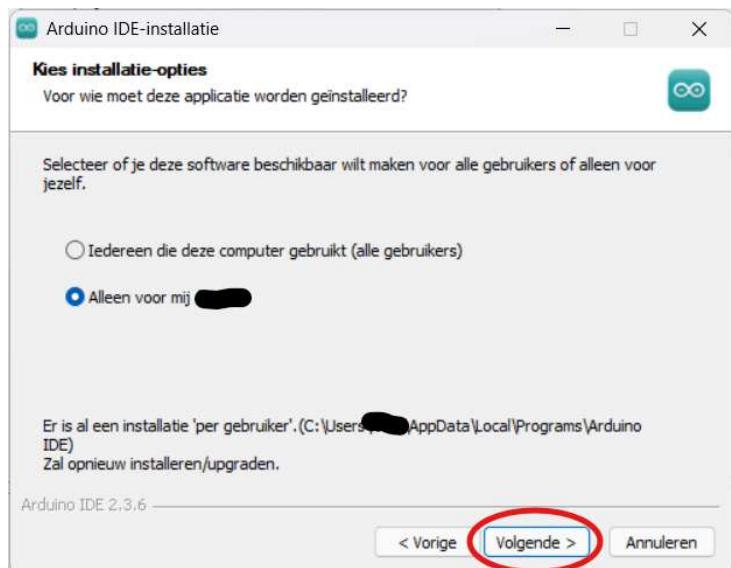
3. Nadat het bestand is gedownload, open je het installatiebestand en volg je de installatie-instructies om de Arduino IDE op je computer te installeren.



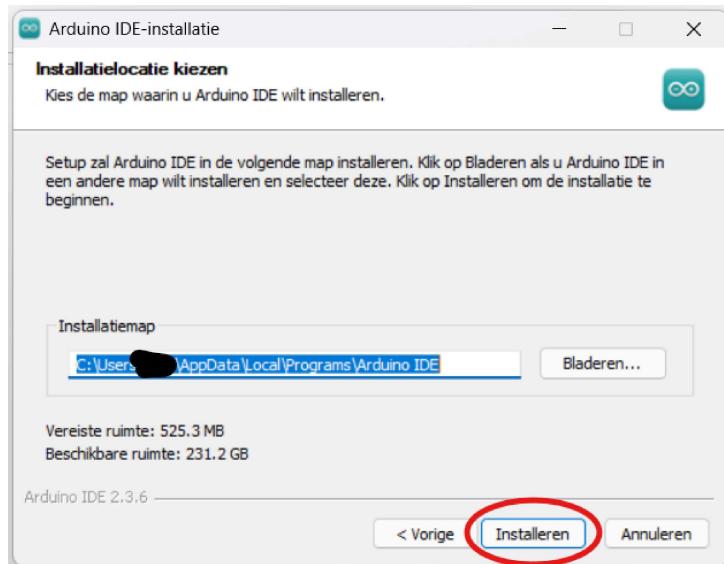
Open de arduino-ide_2.3.6_Windows_64.exe



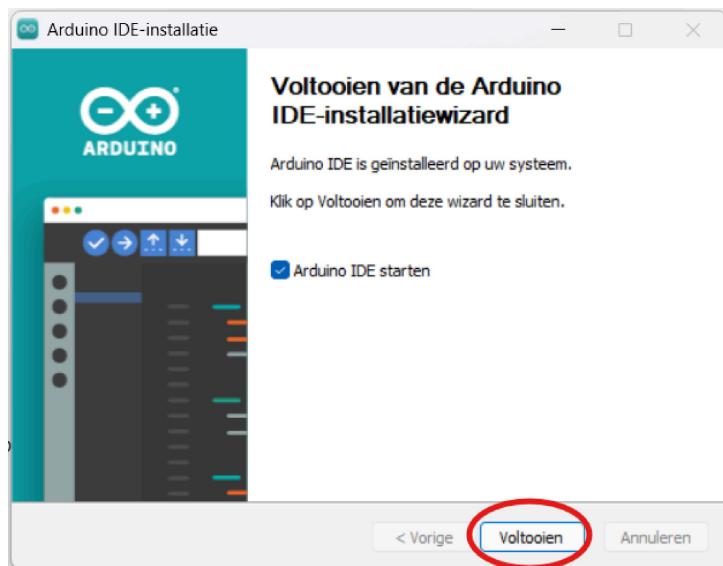
Klik op "Akkoord"



Klik op "Volgende >"



Klik op "Installeren"



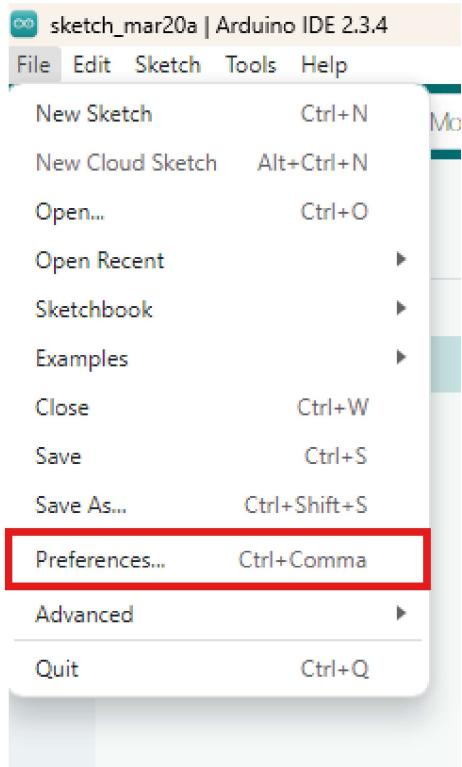
Klik op "Voltooien"

4. Zodra de installatie is voltooid, open je de Arduino IDE

2. Installeren van de ESP32 Board Support in de Arduino IDE

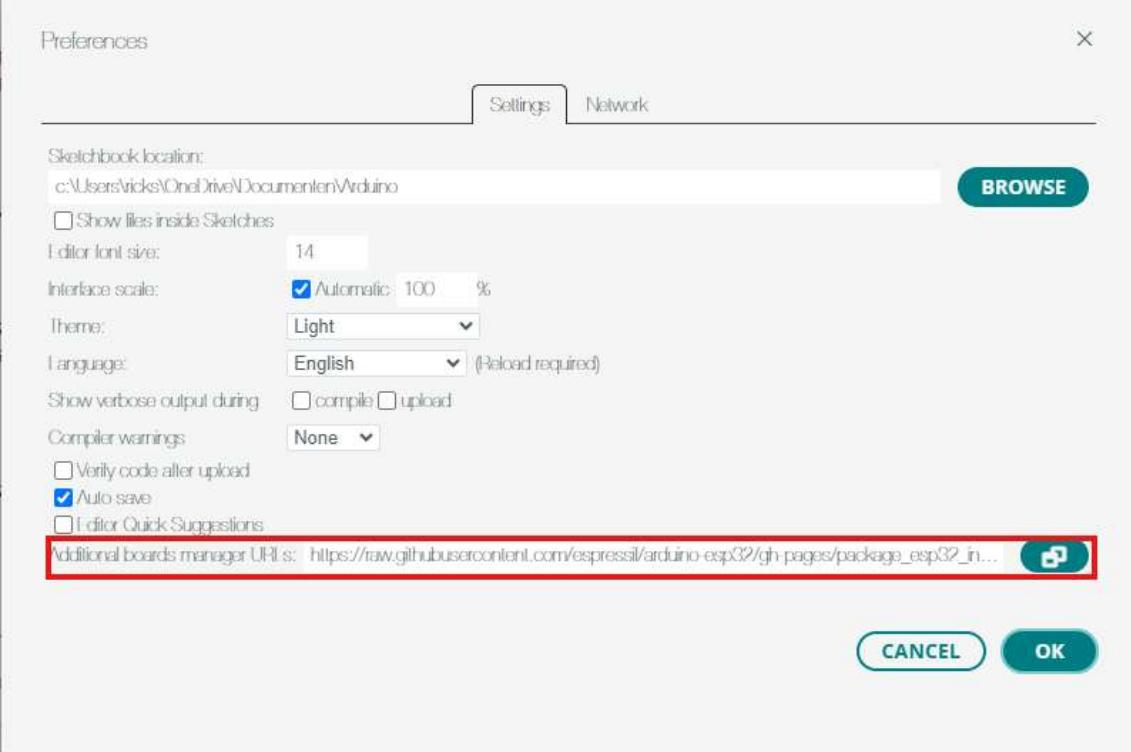
De Arduino IDE ondersteunt standaard geen ESP32-borden, dus je moet eerst de ESP32 boardmanager toevoegen. Volg deze stappen:

1. Open de Arduino IDE.
2. Ga naar File > Preferences in de bovenste menubalk.

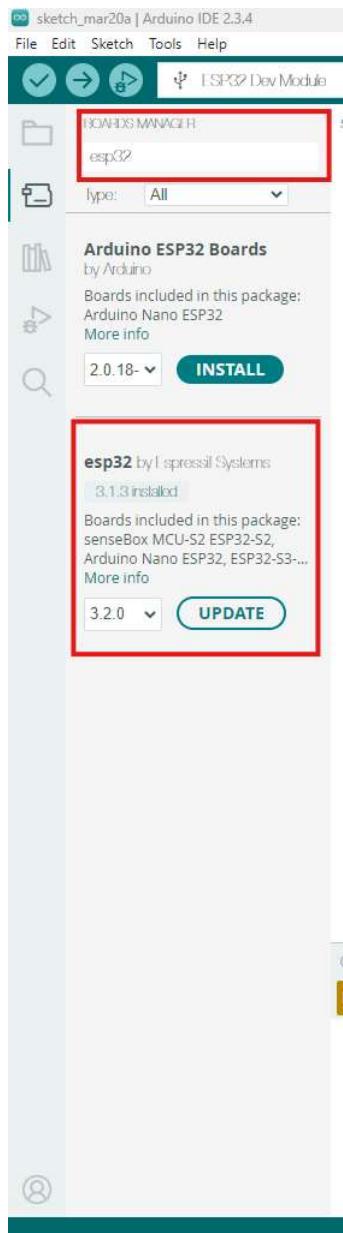


In het venster dat opent, zoek je naar het veld Extra Board Manager URL's. Voeg de volgende URL toe:

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json



3. Klik op OK om de wijzigingen op te slaan.
4. Ga naar Tools > Board > Boards manager
5. Zoek naar ESP32 in de zoekbalk.
6. Klik op Installeren naast het ESP32 by Espressif Systems en wacht totdat de installatie is voltooid.



Na deze stappen kun je het ESP32-bord selecteren in de IDE.

3. Installeren van de Adafruit MPU6050 Bibliotheek

Om gegevens van de MPU6050 sensor te kunnen lezen, gebruiken we de Adafruit MPU6050 bibliotheek. Volg deze stappen om deze bibliotheek te installeren:

1. Open de Arduino IDE en ga naar Sketch > Libraries → Manage libraries
2. Zoek naar Adafruit MPU6050 in de zoekbalk.
3. Klik op Installeren naast de juiste bibliotheek.

LIBRARY MANAGER

MPU6050

Type: All

Topic: All

MPU6050 by Electronic
Cats

1.4.3 installed

MPU6050 Arduino Library.
MPU-6050 6-axis
accelerometer/gyroscope...
[More info](#)

1.4.3 **REMOVE**

Adafruit MPU6050 by
Adafruit

2.2.6 installed

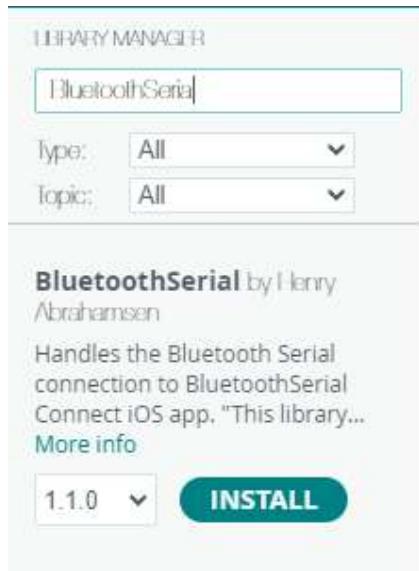
Arduino library for the
MPU6050 sensors in the
Adafruit shop Arduino library...
[More info](#)

2.2.6 **REMOVE**

4. Installeren van de BluetoothSerial Bibliotheek

Voor draadloze communicatie gebruiken we de BluetoothSerial bibliotheek, die standaard beschikbaar is voor de ESP32 in de Arduino IDE. Volg de onderstaande stappen om deze bibliotheek te installeren:

1. Ga naar Sketch > Libraries → Manage libraries in de Arduino IDE.
2. Zoek naar BluetoothSerial in de zoekbalk.
3. Controleer of de bibliotheek al geïnstalleerd is. Zo niet, installeer de bibliotheek.



5. Installeren van de micromlgen Bibliotheek in Python (Samenvatting)

Voor het omzetten van het getrainde Python-model naar C++ gebruiken we de micromlgen bibliotheek. Dit proces zorgt ervoor dat het model kan worden geëxporteerd naar een C++-bestand, geschikt voor de ESP32. Deze stappen zijn al voor je uitgevoerd, en je kunt het benodigde C++-bestand direct downloaden uit de GitLab-repository.

Wat we hebben gedaan:

- Modeltraining in Python: Het model is getraind met behulp van scikit-learn in Python, op basis van data van de MPU6050-sensor. Het model maakt gebruik van versnellings- en gyrocoopdata om te voorspellen of er een freeze is opgetreden.
- Exporteren naar C++: Na het trainen van het model in Python, hebben we het model omgezet naar C++ met behulp van de micromlgen bibliotheek. Dit resulteerde in het C++-bestand clf.h, dat de functionaliteit van het getrainde model bevat.

```
from micromlgen import port

# Exporteer het model naar een C-headerbestand
c_code = port(model)
with open("clf.h", "w") as f:
    f.write(c_code)

print("Model exported to C header file for ESP32!")

Model exported to C header file for ESP32!
```

Mocht je dit zelf willen toepassen, hoeft de code hierboven alleen toegevoegd te worden aan het einde van je model. In dit geval heette het random forest model ‘model’ en zetten we het over naar ‘clf.h’.

1. Klaar voor gebruik: Het gegenereerde clf.h bestand is al beschikbaar in de GitLab-repository, zodat je het eenvoudig kunt downloaden en gebruiken in de ESP32-code.

6. Downloaden van het bestand

Je kunt het benodigde clf.h bestand downloaden vanuit de GitLab-repository, waar het al voor je klaarstaat. Dit bestand bevat het getrainde model en kan direct worden geïmplementeerd in de Arduino-code op je ESP32.

Door deze stappen zijn de benodigde voorbereidingen al getroffen, en hoeft je je geen zorgen te maken over het exporteren of omzetten van het model naar C++, het bestand is al beschikbaar voor gebruik.

The screenshot shows a GitLab repository interface. At the top, there's a navigation bar with 'main' and 'fog_project_labspecifiek / Arduino_Model'. On the right, there are buttons for 'Lock', 'Find file', 'Edit', and 'Code'. Below the navigation, there's a commit history for the 'sketch_ino' branch, authored 2 weeks ago by a user with a blacked-out profile picture. The commit message is 'Code owners Assign users and groups as approvers for specific file changes. Learn more.' There's a link to 'Manage branch rules'. The main part of the screen shows a table of files in the 'Arduino_Model' directory:

Name	Last commit	Last update
..		
↳ .gitkeep	adruino_Model dir	2 weeks ago
h clf.h	rf.h file for arduino	2 weeks ago
sketch_mar20a.ino	sketch_ino	2 weeks ago

A red box highlights the 'clf.h' file in the list. At the bottom right of the interface, there's a small 'Office' button.

In de volgende stappen leggen we uit hoe je de code op de ESP32 kunt zetten en in real-time voorspellingen kunt maken.

4.3.3 Het Implementeren van de Code op de ESP32

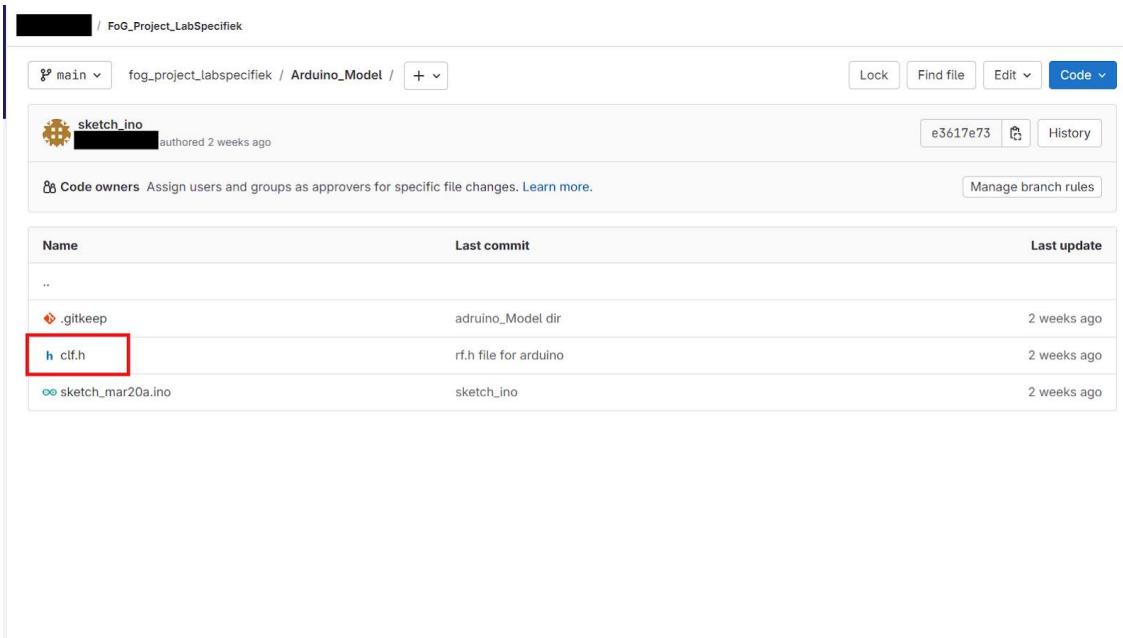
In dit hoofdstuk leer je hoe je de benodigde bestanden kunt downloaden van GitLab en de code naar de ESP32 kunt uploaden. Dit omvat de volgende stappen:

1. Downloaden van de Code van GitLab
2. Openen van de Code in de Arduino IDE
3. Instellen van de ESP32 in de Arduino IDE
4. Uploaden van de Code naar de ESP32

1. Downloaden van de Code van GitLab

De code die nodig is voor de ESP32 staat in een GitLab-repository. Volg de onderstaande stappen om de code naar je computer te downloaden:

1. Open een browser en ga naar de GitLab-repository waar de code is opgeslagen.



The screenshot shows a GitLab repository interface. At the top, there's a navigation bar with 'main' dropdown, 'fog_project_labspecifiek' project name, 'Arduino_Model' branch, and a '+' button. To the right are 'Lock', 'Find file', 'Edit', and 'Code' buttons. Below the navigation is a file list with a single item: 'sketch_ino'. On the right side, there are buttons for 'e3617e73', 'History', and 'Manage branch rules'. Below the file list, a commit history table is shown:

Name	Last commit	Last update
..		
.gitkeep	adruino_Model dir	2 weeks ago
clf.h (highlighted with a red box)	rf.h file for arduino	2 weeks ago
sketch_mar20a.ino	sketch_ino	2 weeks ago

Klik op het 'clf.h' bestand

```
1 #pragma once
2 #include <csstdarg>
3 namespace Eloquent {
4     namespace ML {
5         namespace Port {
6             class RandomForest {
7                 public:
8                     /**
9                      * Predict class for features vector
10                     */
11                 int predict(float *x) {
12                     uint8_t votes[2] = { 0 };
13                     // tree #1
14                     if (x[11] <= -11028.732421875) {
15                         if (x[2] <= 484.77857971191406) {
16                             votes[0] += 1;
17                         }
18                         else {
19                             votes[1] += 1;
20                         }
21                     }
22                 }
23             };
24         };
25     };
26 }
```

Klik op de download knop rechts boven.

Doe hetzelfde voor het bestandje "sketch_mar20a.ino" in de "Arduino_model" map.

De bestanden die je nodig hebt zullen onder andere het volgende bevatten:

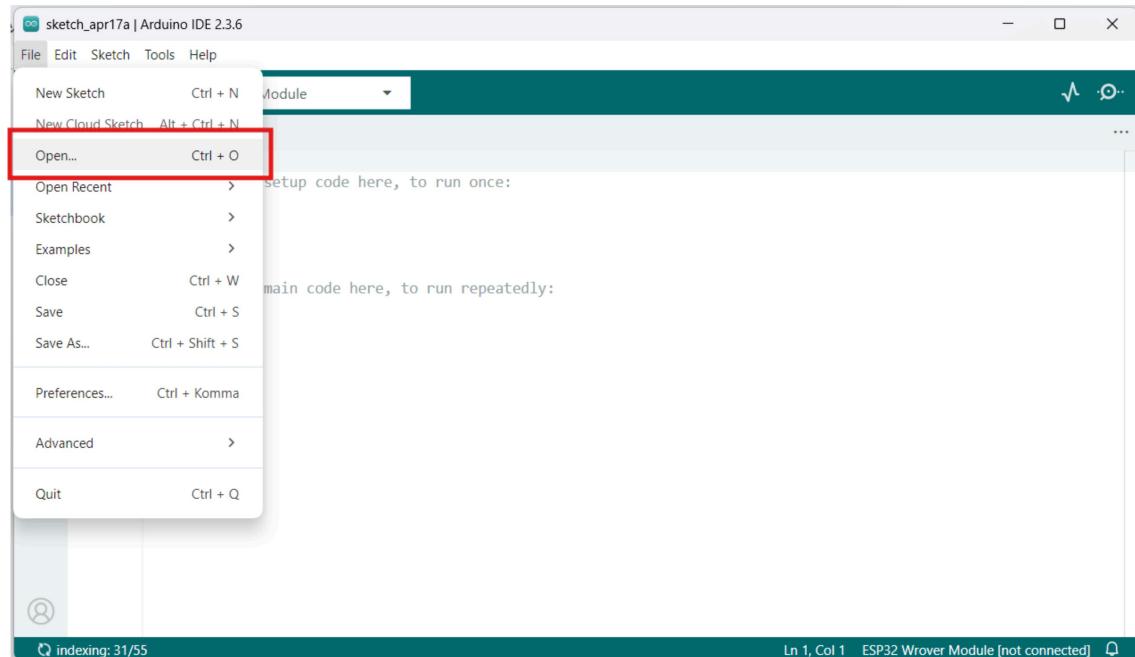
- **clf.h:** De header-bestand met het getrainde Random Forest-model.
- **sketch_mar20a.ino:** De Arduino-code voor de ESP32, inclusief het ophalen van sensorgoedjes en het uitvoeren van voorspellingen.

2. Openen van de Code in de Arduino IDE

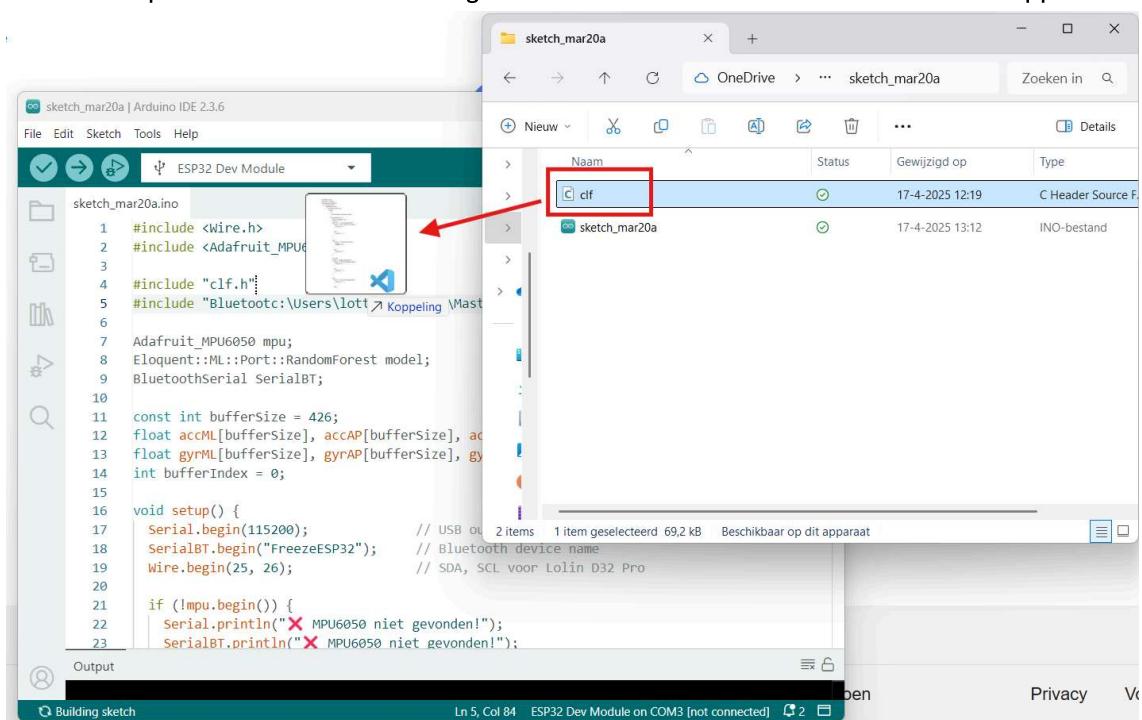
Nu je de code hebt gedownload, is de volgende stap om de bestanden te openen in de Arduino IDE.

1. Open de Arduino IDE.

2. Ga naar File > Open... in de bovenste menubalk.



3. Navigeer naar de map waar je de gedownloade bestanden hebt uitgepakt en selecteer het bestand **sketch_mar20a.ino**.
4. Klik op Openen om het bestand in de Arduino IDE te openen.
5. Sleep het **clf.h** bestand vervolgens vanuit de verkenner ook in de arduino applicatie



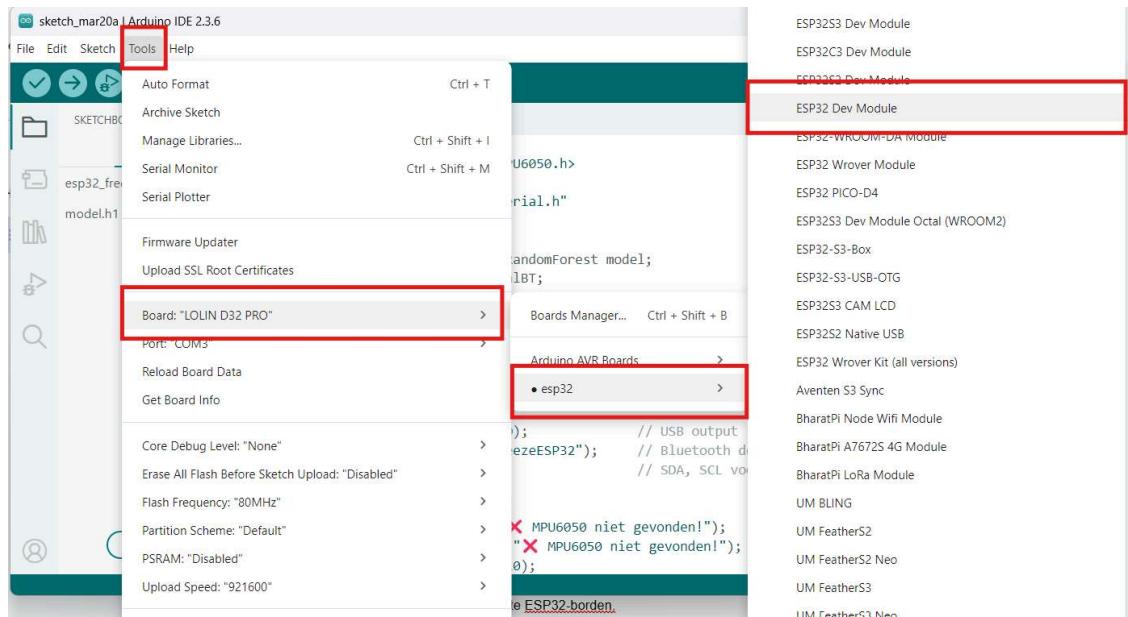
De Arduino IDE zou nu de volledige code voor de ESP32 moeten laden, inclusief de benodigde bibliotheken en de code die je eerder hebt gedownload.

3. Instellen van de ESP32 in de Arduino IDE

Voordat je de code kunt uploaden naar de ESP32, moet je ervoor zorgen dat de juiste instellingen in de Arduino IDE zijn geconfigureerd. Verbind de ESP32 via een micro-USB-kabel met je computer.

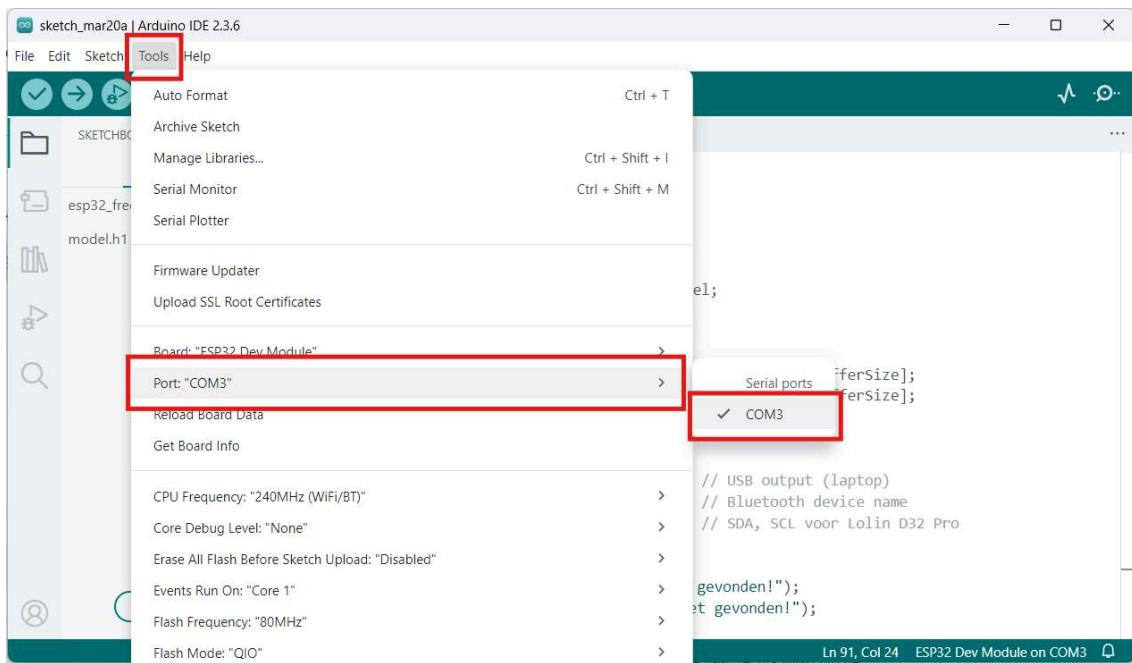
1. Selecteer het juiste board:

- Ga naar Tools > Board > ESP32 Dev Module.
- Als je het ESP32-bord niet kunt vinden, controleer dan of je de boardmanager voor ESP32 hebt geïnstalleerd, zoals beschreven in hoofdstuk 4.3.1.



2. Selecteer de juiste poort:

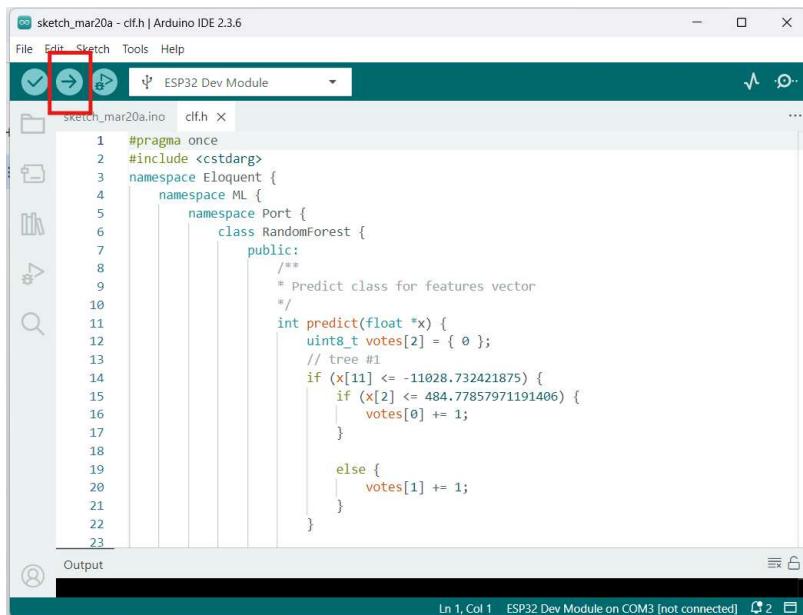
- Ga naar Tools > Port en selecteer de juiste poort waaraan de ESP32 is verbonden (bijvoorbeeld COM3 voor Windows).



4. Uploaden van de Code naar de ESP32

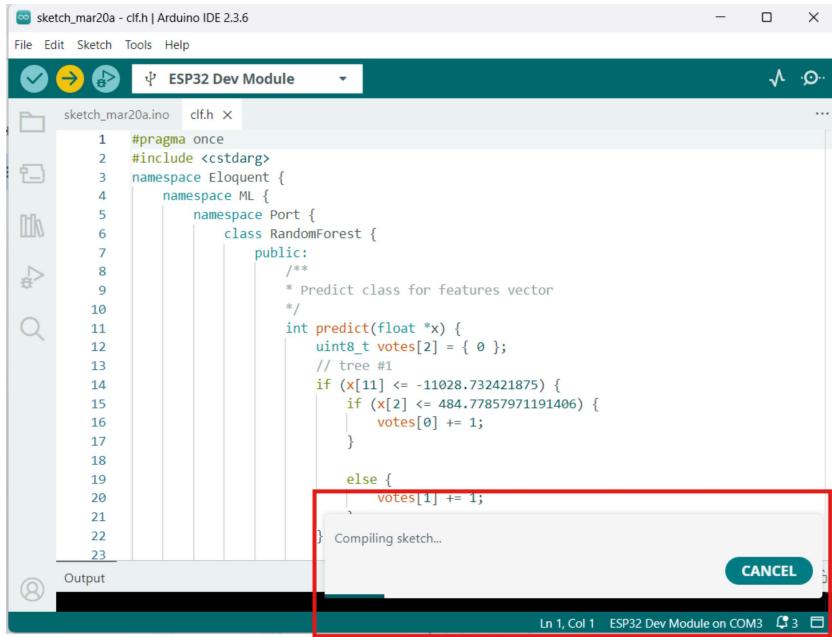
Nu alles is ingesteld, ben je klaar om de code naar de ESP32 te uploaden.

1. In de Arduino IDE, klik op de knop Uploaden (de pijl die naar rechts wijst) om de code naar de ESP32 te sturen.



2. De IDE zal de code compileren en vervolgens uploaden naar de ESP32. Dit kan eveneens duren.

- Zodra de upload is voltooid, zou de ESP32 automatisch moeten beginnen met het uitvoeren van de code.



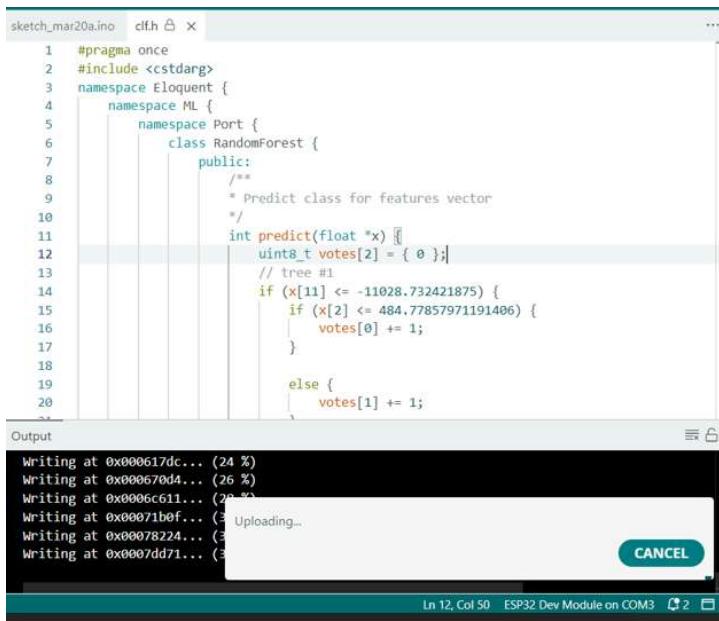
```
sketch_mar20a - clf.h | Arduino IDE 2.3.6
File Edit Sketch Tools Help
ESP32 Dev Module
sketch_mar20a.ino clf.h X
1 #pragma once
2 #include <cstdint>
3 namespace Eloquent {
4     namespace ML {
5         namespace Port {
6             class RandomForest {
7                 public:
8                     /**
9                      * Predict class for features vector
10                     */
11                     int predict(float *x) {
12                         uint8_t votes[2] = { 0 };
13                         // tree #1
14                         if (x[1] <= -11028.732421875) {
15                             if (x[2] <= 484.77857971191406) {
16                                 votes[0] += 1;
17                             }
18                         else {
19                             votes[1] += 1;
20                         }
21                     }
22                 } Compiling sketch...
23             }
24         }
25     }
26 }
```

Output

Ln 1, Col 1 ESP32 Dev Module on COM3 C 3

CANCEL

> Eerst verschijnt er Compiling sketch...



```
sketch_mar20a.ino clf.h X
1 #pragma once
2 #include <cstdint>
3 namespace Eloquent {
4     namespace ML {
5         namespace Port {
6             class RandomForest {
7                 public:
8                     /**
9                      * Predict class for features vector
10                     */
11                     int predict(float *x) {
12                         uint8_t votes[2] = { 0 };
13                         // tree #1
14                         if (x[1] <= -11028.732421875) {
15                             if (x[2] <= 484.77857971191406) {
16                                 votes[0] += 1;
17                             }
18                         else {
19                             votes[1] += 1;
20                         }
21                     }
22                 }
23             }
24         }
25     }
26 }
```

Output

Writing at 0x000617dc... (24 %)
Writing at 0x000670d4... (26 %)
Writing at 0x0006c611... (28 %)
Writing at 0x00071bf0... (30 %) Uploading...
Writing at 0x00078224... (32 %)
Writing at 0x0007dd71... (34 %)

CANCEL

Ln 12, Col 50 ESP32 Dev Module on COM3 C 2

> Erna zal het model uploaden

The screenshot shows the Arduino IDE interface. The code editor window displays the file 'sketch_mar20a.ino' with several lines of C++ code related to sensor initialization and connection. Below the code editor is the 'Output' window, which shows the progress of the upload process. The output text reads:

```
Writing at 0x0001005d... (51 %)
Writing at 0x0001c3d1... (93 %)
Writing at 0x000121b6b... (95 %)
Writing at 0x00012763b... (97 %)
Writing at 0x00012d629... (100 %)
Wrote 1173472 bytes (723664 compressed) at 0x00010000 in 10.4 seconds (effective 906.8 kbit/s)
Hash of data verified.
```

At the bottom of the interface, a status bar indicates 'Ln 4, Col 14 ESP32 Dev Module on COM3'.

Het model is nu succesvol geüpload op de ESP.

5. Serial Monitor

Na het uploaden kun je de Seriële Monitor gebruiken om te controleren of de ESP32 correct werkt.

1. Klik op Hulpmiddelen > Serial Monitor in de Arduino IDE.
2. Stel de baudrate in op 115200 om de berichten te lezen die door de ESP32 worden verzonden.
3. Je zou nu de uitvoer moeten zien van de ESP32, inclusief berichten zoals:
 - ✓ MPU6050 verbonden. Bluetooth gestart.
 - ● Feature vector:" gevuld door de sensorwaarden.
 - ⚠ Freeze gedetecteerd!" of ✓ Geen freeze gedetecteerd.

Als alles correct is ingesteld, zou je nu de real-time voorspellingen van de ESP32 moeten kunnen zien. Deze voorspellingen worden via Bluetooth naar een gekoppeld apparaat gestuurd, zoals een smartphone of een computer.

5. Sensor vastmaken en output aflezen

Bij de sensor en behuizing zit een batterij. Deze sluiten we aan op de sensor, wat ervoor zorgt dat de sensor begint te werken. We plaatsen de sensor om het been dicht bij de enkel, dit doen we met een band die we om de sensor heen hebben geknoopt, zie afbeelding.



Vervolgens maken we contact met de sensor door middel van een bluetooth verbinding. Dit doen we als volgt:

1. Download de Bluetooth Serial app op je telefoon:
https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal&pli=1 (let op deze is alleen beschikbaar voor Android, je moet dus een Android telefoon hebben)
2. Zorg dat je Bluetooth aanstaat, open de app en zoek bij devices naar FreezeESP32 Maak hier verbinding mee.
3. Ga naar de terminal in de app, je zou moeten zien dat de ESP aan het verbinden is of al verbonden is.
4. Wanneer de terminal bevestigt dat de verbinding succesvol is, ben je klaar om de sensor te gebruiken. Loop een stuk en imiteer een freeze met de sensor aan je been (dicht bij de enkel) en zie de output op je telefoon in de terminal (zie afbeelding)

```
13:18:02.451 Connecting to VELORETTI ...
13:18:03.476 Connection failed: gatt status 133
13:18:18.761 Connecting to c01s2434965724347437 ...
13:18:19.114 Connection failed: gatt status 133
13:18:28.969 Connecting to FreezeESP32 ...
13:18:29.637 Connected
13:18:38.432 Connection lost
13:18:49.777 Connecting to FreezeESP32 ...
13:18:56.200 Connection failed: read failed, socket might closed or timeout, read ret: -1
13:19:13.268 Connecting to FreezeESP32 ...
13:19:14.151 Connected
13:19:17.313 ✓ Geen freeze gedetecteerd.
13:19:22.620 ✓ Geen freeze gedetecteerd.
13:19:27.323 ✓ Geen freeze gedetecteerd.
13:19:31.740 ✓ Geen freeze gedetecteerd.
13:19:35.855 ✓ Geen freeze gedetecteerd.
13:19:40.180 ✓ Geen freeze gedetecteerd.
13:19:44.407 ✓ Geen freeze gedetecteerd.
13:19:49.300 ✓ Geen freeze gedetecteerd.
13:19:53.094 ✓ Geen freeze gedetecteerd.
13:19:57.954 ✓ Geen freeze gedetecteerd.
13:20:01.941 ✓ Geen freeze gedetecteerd.
13:20:07.072 ⚠ Freeze gedetecteerd!
13:20:10.996 ✓ Geen freeze gedetecteerd.
13:20:15.699 ✓ Geen freeze gedetecteerd.
13:20:20.214 ⚠ Freeze gedetecteerd!
13:20:25.302 ⚠ Freeze gedetecteerd!
13:20:30.197 ⚠ Freeze gedetecteerd!
13:20:38.973 Connection lost
13:26:19.708 Connecting to FreezeESP32 ...
13:26:27.329 Connected
13:27:18.650 Connection lost
13:28:26.070 Connecting to FreezeESP32 ...
13:28:32.502 Connection failed: read failed, socket might cl
```

M1 M2 M3 M4 M5 M6 M7 > ||| O <

6. 3D Geprinte Behuizing

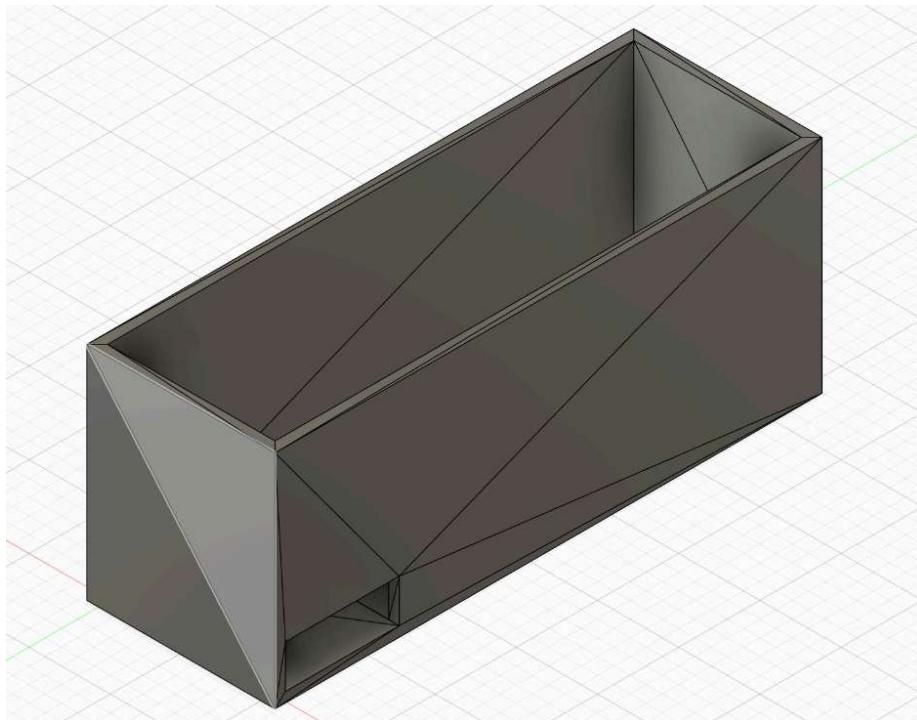
Benodigdheden

- Fusion360
- PrusaSlicer
- PETG-filament

6.1 Modellen

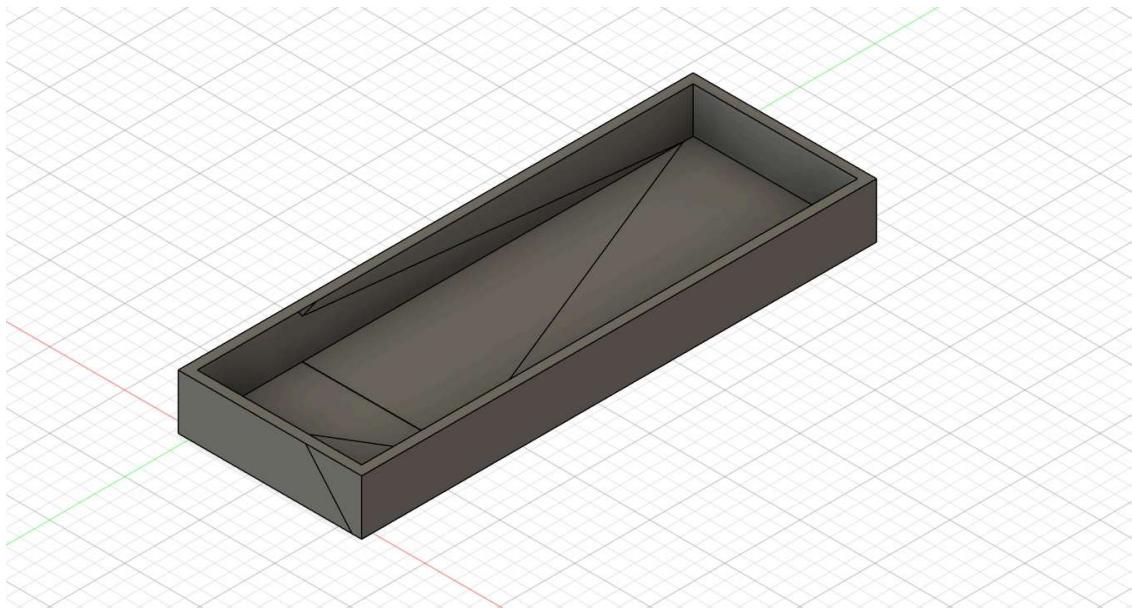
De afmetingen van de behuizing zijn afhankelijk van de componenten, die licht kunnen variëren afhankelijk van de configuratie. Voordat je de Fusion360-software opent, moet je eerst de afmetingen van de ESP32 en de IMU 6050-sensor (accelerometer en gyroscoop) meten.

Daarnaast moet de locatie en de grootte van de SSD worden gemeten, evenals de aansluiting voor de accu. Dit kan per apparaat iets verschillen. Zorg ervoor dat er bovenin wat extra ruimte wordt gelaten om te voorkomen dat de kabels onterecht gebogen worden. Het uiteindelijke 3D-model van de behuizing moet er ongeveer als volgt uitzien:



Om de componenten veilig binnen de behuizing te plaatsen, is het ook noodzakelijk om een bovenkant (dak) voor de behuizing te ontwerpen. De eenvoudigste manier om dit te doen is door de wanden van de behuizing uit te rekken naar de hoogte van de componenten en vervolgens de openingen aan de zijkanten op te vullen. Het bovenste deksel is echter niet permanent bevestigd, zodat het indien nodig kan worden verwijderd voor toegang tot de resetknop van de Lolin32, wat essentieel is tijdens de test- en herprogrammeerfasen. Het bovenste deksel kan naar wens worden geplaatst of verwijderd.

Het 3D-model van de behuizing, inclusief het bovenste deksel, zou er als volgt uit moeten zien:



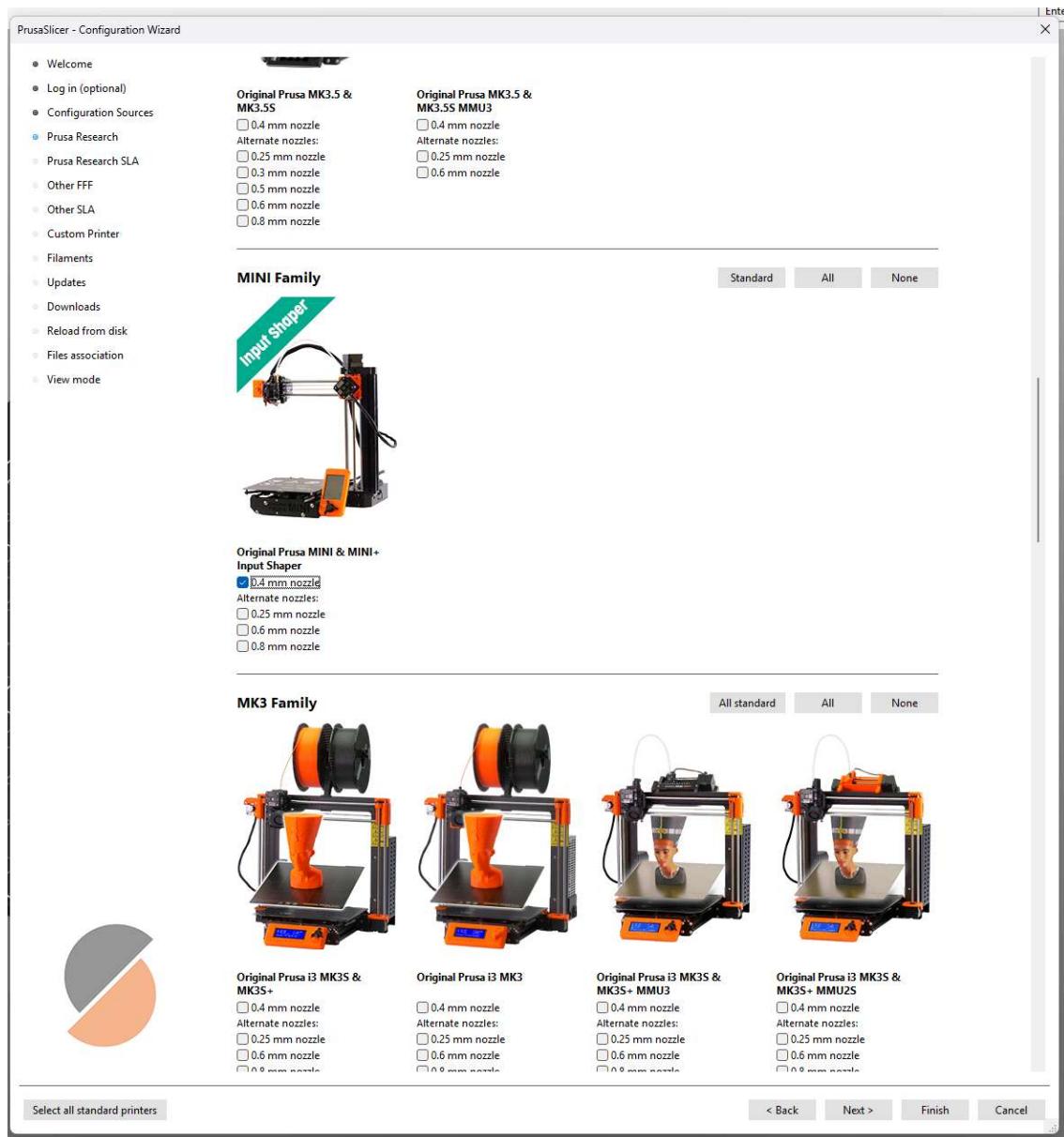
Deze 3D modellen moeten vervolgens worden geëxporteerd als .fbx bestand.

Exporteren van het 3D-model

Nadat het ontwerp is voltooid, moet het model worden geëxporteerd als een .fbx-bestand vanuit Fusion360.

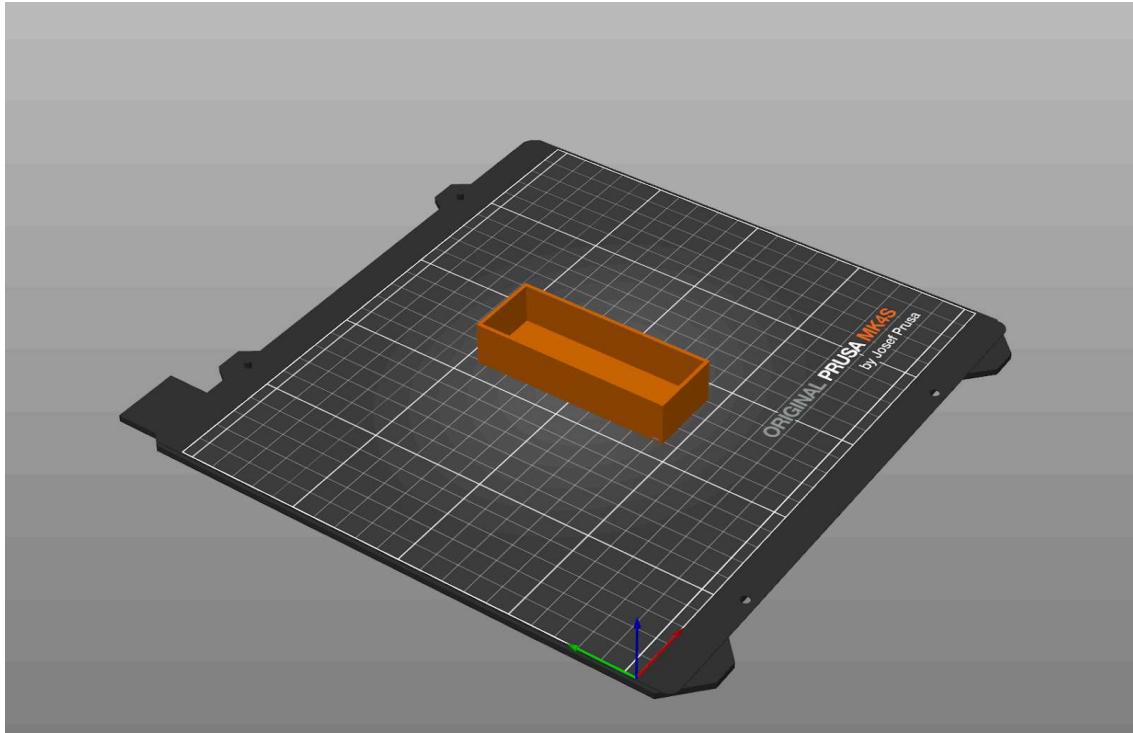
6.2 Printen van de Behuizing

Na het exporteren van het bestand uit Fusion360, moet de PrusaSlicer-applicatie worden geopend. Als je deze voor het eerst opent, moet je de printer selecteren die je gaat gebruiken. In dit geval selecteer je de Prusa Mini met een 0,4mm nozzle.



De overige instellingen kunnen op de standaardwaarden blijven staan, tenzij je specifieke wijzigingen moet aanbrengen voor de printkwaliteit.

Importeer vervolgens het .fbx-bestand in PrusaSlicer en controleer of de afmetingen correct zijn geïmporteerd. Het kan zijn dat de komma's in de afmetingen niet goed worden ingelezen, wat kan leiden tot verkeerde schaalinstellingen (bijvoorbeeld 12.3 wordt 123 of 1.23). Zorg ervoor dat de afmetingen correct zijn voordat je verder gaat met printen.



6.3 Kiezen van het Filament

Voor het printen hebben wij gekozen voor PETG-filament (Polyethyleentereftalaat Glycol), vanwege zijn duurzaamheid, temperatuurbestendigheid en lichte flexibiliteit. PETG is uitstekend geschikt voor het beschermen van elektronica omdat het zowel stevig als flexibel genoeg is om de interne componenten goed te beschermen. vergeleken met PLA biedt PETG betere slagvastheid en een hogere temperatuurweerstand, wat belangrijk is tijdens continu gebruik.

PrusaSlicer-instellingen:

Zorg ervoor dat je de instellingen in PrusaSlicer op de standaardwaarden houdt om een goede printkwaliteit te waarborgen. De print wordt uitgevoerd op een Prusa Original Mini printer en de printkwaliteit was uitstekend, wat resulteerde in een stevige en goed hechtende structuur.

Ontwerpdetails van de Behuizing

De behuizing is op maat ontworpen om de volgende componenten te huisvesten:

- Lolin32 ontwikkelbord (met ESP32)
- MPU6050 IMU-sensor
- Jumper wires voor connectiviteit

De lengte van de behuizing is precies afgestemd op het Lolin32-bord en de IMU-sensor, terwijl de breedte voldoende ruimte biedt voor de jumper wires, zodat ze netjes en zonder buigen kunnen worden geplaatst. Aan de onderkant van de behuizing zijn twee uitsparingen toegevoegd: één voor de aansluiting van de batterij en een andere voor de Micro-USB-poort,

waarmee het apparaat kan worden opgeladen of gegevens kan worden overgedragen zonder de behuizing te verwijderen.

Gebruiksgemak en Testfases

Het bovenste deksel is niet permanent bevestigd, zodat het eenvoudig kan worden verwijderd tijdens de test- en herprogrammeerfasen. Dit biedt gemakkelijke toegang tot de resetknop van de Lolin32, wat essentieel is voor debugging en herprogrammering van het systeem.

Deze op maat gemaakte behuizing zorgt niet alleen voor bescherming van de interne componenten, maar vergemakkelijkt ook de draagbaarheid van het systeem, waardoor het apparaat handig kan worden ingezet als draagbare oplossing voor de real-time Freezing of Gait (FoG)-detectie.