# Raspberry Pi — Human Presence / Breathing ML Project

A complete project folder to collect data (GPIO or UART), extract features, train a small ML model (Random Forest), and run inference on Raspberry Pi in real time. Designed for digital-output radars (GPIO) and UART-capable radar modules (recommended).

---

## File tree

```
RaspberryPi_HumanDetection_ML_Project/
├─ README.md
├─ requirements.txt
├─ collectors/
│  ├─ data_collector_gpio.py
│  └─ data_collector_uart.py
├─ tools/
│  ├─ labeler.py
│  └─ feature_extractor.py
├─ training/
│  ├─ train_model.py
│  └─ evaluate_model.py
├─ deploy/
│  ├─ infer_pi.py
│  └─ model/ (output: model.pkl)
└─ examples/
   └─ sample_raw_dataset.csv
```

---

## README (quick start)

1. **Hardware**
2. UART (recommended): connect sensor TX -> Pi RX (GPIO15), GND -> GND, VCC -> 5V.

3. GPIO (digital-only): connect sensor OUT -> chosen BCM GPIO pin (e.g. 17), GND -> GND.

4. **Enable UART on Pi**

```
sudo raspi-config
-> Interface Options -> Serial Port -> Enable hardware, disable console
```

1. **Install dependencies**

```
sudo apt update
sudo apt install python3-pip -y
pip3 install -r requirements.txt
```

1. **Collect data**
2. For UART: `python3 collectors/data_collector_uart.py --duration 60 --out raw_uart.csv`
3. For GPIO: `python3 collectors/data_collector_gpio.py --duration 60 --out raw_gpio.csv`
4. **Label**
5. `python3 tools/labeler.py raw_uart.csv labeled_uart.csv --label breathing` (or use manual labeling in a spreadsheet)
6. **Feature extraction**
7. `python3 tools/feature_extractor.py labeled_uart.csv features.csv`
8. **Train**
9. `python3 training/train_model.py features.csv model/model.pkl`
10. **Deploy**
11. Copy `model/model.pkl` to `deploy/model/` on Pi and run `python3 deploy/infer_pi.py`.

---

## requirements.txt

```
RPi.GPIO==0.7.0
pyserial==3.5
pandas==2.1.1
numpy==1.25.2
scikit-learn==1.4.2
joblib==1.3.2
```

---

## collectors/data_collector_uart.py

```
#!/usr/bin/env python3
"""
Collects UART data from radar module and writes CSV with columns:
  timestamp, distance, move_energy, static_energy
Expected line format from sensor (example):
  DATA,12.3,45.2,3.4
"""

import serial
import time
import argparse
import csv

parser = argparse.ArgumentParser()
parser.add_argument('--port', default='/dev/serial0')
parser.add_argument('--baud', type=int, default=256000)
parser.add_argument('--duration', type=int, default=60)
parser.add_argument('--out', default='raw_uart.csv')
args = parser.parse_args()

ser = serial.Serial(args.port, args.baud, timeout=0.5)
end = time.time() + args.duration

with open(args.out, 'w', newline='') as f:
    w = csv.writer(f)
    w.writerow(['timestamp','distance','move_energy','static_energy'])
    while time.time() < end:
        line = ser.readline().decode(errors='ignore').strip()
        if not line:
            continue
        parts = [p.strip() for p in line.split(',')]
        # Basic parser, adapt to your sensor's UART format
        if parts[0].upper().startswith('DATA') and len(parts) >= 4:
            try:
                ts = time.time()
                dist = float(parts[1])
                move = float(parts[2])
                sta = float(parts[3])
                w.writerow([ts, dist, move, sta])
            except Exception:
                continue

print('Saved', args.out)
```

## collectors/data_collector_gpio.py

```
#!/usr/bin/env python3
"""
Collects digital GPIO signals (HIGH/LOW) with timestamps.
Outputs CSV: timestamp, digital_value
"""

import RPi.GPIO as GPIO
import time
import argparse
import csv

parser = argparse.ArgumentParser()
parser.add_argument('--pin', type=int, default=17, help='BCM pin number')
parser.add_argument('--duration', type=int, default=60)
parser.add_argument('--out', default='raw_gpio.csv')
args = parser.parse_args()

PIN = args.pin
GPIO.setmode(GPIO.BCM)
GPIO.setup(PIN, GPIO.IN)

end = time.time() + args.duration
with open(args.out, 'w', newline='') as f:
    w = csv.writer(f)
    w.writerow(['timestamp','digital_value'])
    while time.time() < end:
        ts = time.time()
        val = GPIO.input(PIN)
        w.writerow([ts, int(val)])
        time.sleep(0.01)  # 100 Hz sampling

GPIO.cleanup()
print('Saved', args.out)
```

---

## tools/labeler.py

```
#!/usr/bin/env python3
"""
Simple labeler: append a label column for entire file or open CSV in editor for
fine-grained labels.
Usage: python3 tools/labeler.py raw.csv labeled.csv --label breathing
```

```
"""

import pandas as pd
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('infile')
parser.add_argument('outfile')
parser.add_argument('--label', default='breathing')
args = parser.parse_args()

df = pd.read_csv(args.infile)
df['label'] = args.label

df.to_csv(args.outfile, index=False)
print('Saved', args.outfile)
```

---

## tools/feature_extractor.py

```
#!/usr/bin/env python3
"""
Extract features from raw UART or GPIO labeled data.
For UART: compute per-window features (e.g. 5s window) using distance/move/
static.
For GPIO: compute pulses count, avg_interval, min_interval, max_interval,
variance in each window.
"""

import pandas as pd
import numpy as np
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('infile')
parser.add_argument('outfile')
parser.add_argument('--window', type=float, default=5.0)
args = parser.parse_args()

df = pd.read_csv(args.infile)
window = args.window

# Detect if UART (has move_energy) or GPIO
is_uart = 'move_energy' in df.columns or 'move' in df.columns
```

```
rows = []
start_ts = df['timestamp'].iloc[0]
end_ts = df['timestamp'].iloc[-1]
win_start = start_ts

while win_start + window <= end_ts:
    win_end = win_start + window
    win = df[(df['timestamp'] >= win_start) & (df['timestamp'] < win_end)]
    if is_uart:
        dist = win['distance'].values
        move = win['move_energy'].values if 'move_energy' in win.columns else
win['move'].values
        sta = win['static_energy'].values if 'static_energy' in win.columns else
win['sta'].values
        row = {
            'start': win_start,
            'count': len(win),
            'dist_mean': np.nanmean(dist) if len(dist) else 0,
            'move_mean': np.nanmean(move) if len(move) else 0,
            'move_max': np.nanmax(move) if len(move) else 0,
            'sta_mean': np.nanmean(sta) if len(sta) else 0,
        }
    else:
        # GPIO pulses: compute intervals between edges
        vals = win['digital_value'].values
        ts = win['timestamp'].values
        # find edges
        edges = []
        for i in range(1, len(vals)):
            if vals[i] != vals[i-1]:
                edges.append(ts[i])
        intervals = np.diff(edges) if len(edges) >= 2 else np.array([])
        row = {
            'start': win_start,
            'count': len(edges),
            'avg_interval': np.nanmean(intervals) if len(intervals) else 0,
            'min_interval': np.nanmin(intervals) if len(intervals) else 0,
            'max_interval': np.nanmax(intervals) if len(intervals) else 0,
            'var_interval': np.nanvar(intervals) if len(intervals) else 0,
        }
    # keep label if present
    if 'label' in win.columns and len(win):
        row['label'] = win['label'].mode()[0]
    rows.append(row)
    win_start += window

out = pd.DataFrame(rows)
```

```
out.to_csv(args.outfile, index=False)
print('Saved features to', args.outfile)
```

---

## training/train_model.py

```
#!/usr/bin/env python3
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import joblib
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('features_csv')
parser.add_argument('out_model')
args = parser.parse_args()

df = pd.read_csv(args.features_csv)
labels = df['label'] if 'label' in df.columns else None
X = df.drop(columns=[c for c in ['label','start'] if c in df.columns])

X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2,
random_state=42)

clf = RandomForestClassifier(n_estimators=200, random_state=42)
clf.fit(X_train, y_train)

pred = clf.predict(X_test)
print('Accuracy', accuracy_score(y_test, pred))
print(classification_report(y_test, pred))

joblib.dump(clf, args.out_model)
print('Saved model to', args.out_model)
```

---

## deploy/infer_pi.py

```
#!/usr/bin/env python3
"""
Real-time inference on Raspberry Pi. Reads UART or polls GPIO and uses trained
model to predict.
```

```
If model predicts "breathing" or "movement" consistently, prints FINAL HUMAN
DETECTED (5 detections in 5s exact behavior optional).
"""

import argparse
import time
import joblib
import serial
import pandas as pd
import RPi.GPIO as GPIO

parser = argparse.ArgumentParser()
parser.add_argument('--model', default='model/model.pkl')
parser.add_argument('--method', choices=['uart','gpio'], default='uart')
parser.add_argument('--port', default='/dev/serial0')
parser.add_argument('--pin', type=int, default=17)
args = parser.parse_args()

clf = joblib.load(args.model)
print('Model loaded')

# Simple window-based detector using model predictions
WINDOW = 5.0
COUNT_THRESHOLD = 5

if args.method == 'uart':
    ser = serial.Serial(args.port, 256000, timeout=0.5)
    buffer = []
    t0 = time.time()
    while True:
        line = ser.readline().decode(errors='ignore').strip()
        if not line:
            continue
        parts = [p.strip() for p in line.split(',')]
        if parts[0].upper().startswith('DATA') and len(parts) >= 4:
            try:
                ts = time.time()
                dist = float(parts[1])
                move = float(parts[2])
                sta = float(parts[3])
                buffer.append((ts, dist, move, sta))
            except:
                continue
        # window check
        if time.time() - t0 >= WINDOW:
            df = pd.DataFrame(buffer,
columns=['timestamp','distance','move','static'])
            # extract features same as training
```

```
            feat = {
                'count': len(df),
                'dist_mean': df['distance'].mean() if len(df) else 0,
                'move_mean': df['move'].mean() if len(df) else 0,
                'move_max': df['move'].max() if len(df) else 0,
                'sta_mean': df['static'].mean() if len(df) else 0,
            }
            X = pd.DataFrame([feat])
            pred = clf.predict(X)[0]
            # simple counting logic: if pred indicates human states, increment
            if pred in ['breathing','movement','human']:
                detected = 1
            else:
                detected = 0
            # For simplicity: print final if single-window detection meets
threshold logic
            # (For exact 5-in-5 behavior, maintain sub-window counters as
before.)
            if detected:
                print('FINAL HUMAN DETECTED')
            buffer = []
            t0 = time.time()

else:
    # GPIO path: sample edges and extract intervals then features
    pin = args.pin
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(pin, GPIO.IN)
    edges = []
    t0 = time.time()
    while True:
        val = GPIO.input(pin)
        ts = time.time()
        # simple edge detection via polling (fine for 100Hz)
        # append timestamp of change
        if not hasattr(infer_pi, 'last_val'):
            infer_pi.last_val = val
        if val != infer_pi.last_val:
            edges.append(ts)
            infer_pi.last_val = val
        if time.time() - t0 >= WINDOW:
            if len(edges) >= 2:
                intervals = [edges[i+1]-edges[i] for i in range(len(edges)-1)]
                feat = {
                    'count': len(edges),
                    'avg_interval': sum(intervals)/len(intervals),
                    'min_interval': min(intervals),
                    'max_interval': max(intervals),
```

```
                'var_interval': pd.Series(intervals).var()
            }
            X = pd.DataFrame([feat])
            pred = clf.predict(X)[0]
            if pred in ['breathing','movement','human']:
                print('FINAL HUMAN DETECTED')
        edges = []
        t0 = time.time()
```

---

## Notes, tips and accuracy guidance

- Use UART raw values if possible: move_energy and static_energy give much better features for ML.
- Collect at least 20–30 min of labeled data across 5+ people to reach >90% accuracy for common indoor scenarios.
- Ensure diversity: clothing, positions (lying, sitting), distances (0.5–3 m), and occlusions (thin rubble) if relevant.
- Use randomness when splitting train/test: `train_test_split(..., stratify=y)`.
- If model prints too often, add cooldown (e.g., suppress repeats for 10s).

---

## Example CSV (examples/sample_raw_dataset.csv)

```
# For UART
timestamp,distance,move_energy,static_energy
1700000000.234,1.23,45.2,3.4
1700000000.254,1.23,46.1,3.5

# For GPIO
timestamp,digital_value
1700000100.001,1
1700000100.011,1
1700000100.021,0
```

---

If you want, I can now: - generate all files as separate downloadable artifacts, or - produce a ready-to-run zip on my side, or - create a simplified Edge Impulse workflow.

Tell me which one you want next and I will produce it immediately.