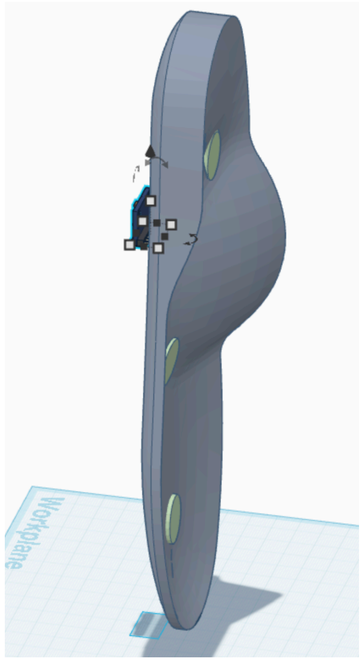
Smart Shoe

Monitor movement and alert uneven weight distribution and falling.

Github: https://github.com/patorsiang/smart-shoe

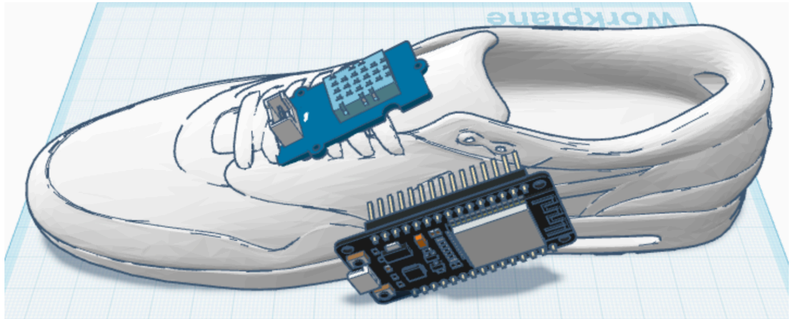
Napatchol Thaipanich (nt375)

The Evolution of an Idea



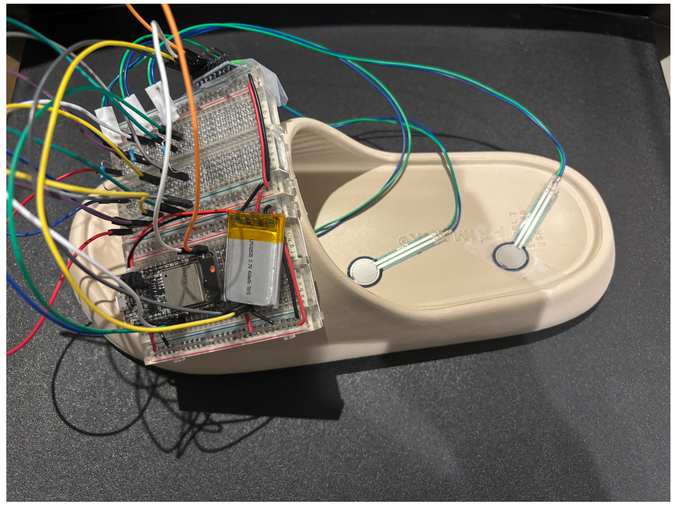
Insole

Original Model
Cr. https://grabcad.com/library/insole-plantar-faciitis-1



Sport Shoe

Original Model
Cr. https://www.tinkercad.com/things/fpsg9GW40K7-nike-shoe



Sandal Shoe

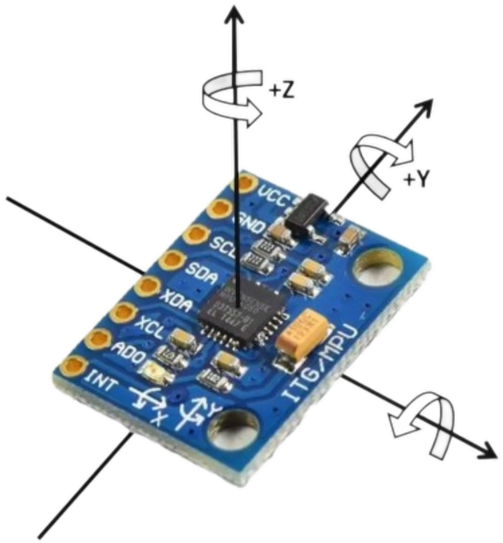
Values

Grandma at home alone

- For relatives or caregivers
  - Fall Detection Alert
- For elderly individuals
  - Checking weight balance distribution on a foot when standing
  - Keep the Step-counting
  - Alert Standing every one hour

https://www.flaticon.com/free-icon/house_619153
https://www.flaticon.com/free-icon/old-woman_3554027

Sensors



MPU6050

https://github.com/ElectronicCats/mpu6050

- Data Collection:
  - 3-axis accelerometer
  - 3-axis gyroscope
  - temperature
- Calibrate

Force Sensor

- Data Collection:
  - Force Value (number)
- Filter: medianFilter

Force Sensor

```
int sensorPins[SENSOR_COUNT] = {A2, A3, A4};
int readings[SENSOR_COUNT][3];

void setupForceSensors()
{
  for (int i = 0; i < SENSOR_COUNT; i++)
  {
    pinMode(sensorPins[i], INPUT);
    // esp_sleep_enable_ext0_wakeup((gpio_num_t)sensorPins[i], 1); // Wake on HIGH
  }
  // set the ADC attenuation to 11 dB (up to ~3.3V input)
  analogSetAttenuation(ADC_11db);
  Serial.println("Set up Sensors");

  Serial.println("Configuring wake-up source...");

  esp_sleep_enable_ext0_wakeup((gpio_num_t)sensorPins[0], 1); // Wake on HIGH
  // esp_sleep_enable_gpio_wakeup();
  // gpio_wakeup_enable((gpio_num_t)sensorPins[0], GPIO_INTR_HIGH_LEVEL);
  esp_sleep_wakeup_cause_t wakeup_reason = esp_sleep_get_wakeup_cause();
  Serial.print("Wakeup reason: ");
  Serial.println(wakeup_reason);
}
```

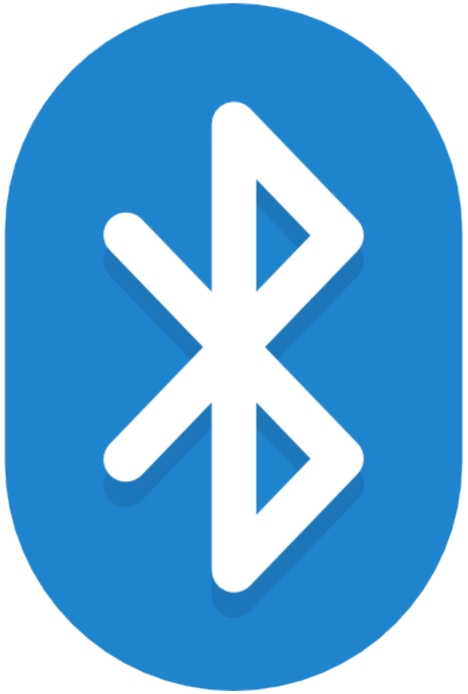
- **Technique**: Median Filtered Force Sensing with Sleep Mode Triggering
- **Input**: Analog readings from 3 force sensors connected to pins A2, A3, and A4 (representing different foot zones: front, mid, heel).
- **Preprocessing**:
  - Each sensor keeps a history of its last 3 readings.
  - A median filter is applied to smooth out short-term fluctuations or noise from raw analog values.
- **Evaluation**:
  - The filtered force values are used for real-time monitoring and are broadcasted via Bluetooth (via forceChar->notify()).
  - Data is also uploaded via MQTT in JSON format.
- **Detection Logic (Sleep Trigger)**:
  - Compute the average of the most recent filtered readings across all sensors.
  - If the average is above the force threshold, update lastActiveTime.
  - If no significant activity is detected for more than 3 minutes, the system flags that it should enter sleep mode (shouldSleep() returns true)

MPU6050

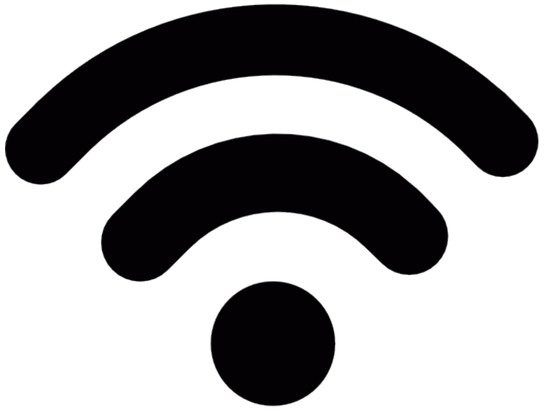
```
Adafruit_MPU6050 mpu;

sensors_event_t a, g, temp;

float gyroX, gyroY, gyroZ;
float accX, accY, accZ;
float temperature;

float accXOffset = 0.0, accYOffset = 0.0, accZOffset = 0.0;
float gyroXerror = 0.0, gyroYerror = 0.0, gyroZerror = 0.0;

void calibrateMPU(int samples)
{
  float gx = 0, gy = 0, gz = 0;
  float ax = 0, ay = 0, az = 0;

  Serial.println("Calibrating MPU... Please keep the device still.");
  for (int i = 0; i < samples; i++)
  {
    updateMPUEvents();
    gx += g.gyro.x;
    gy += g.gyro.y;
    gz += g.gyro.z;

    ax += a.acceleration.x;
    ay += a.acceleration.y;
    az += a.acceleration.z;
    delay(5);
  }

  // store average offsets
  gyroXerror = gx / samples;
  gyroYerror = gy / samples;
  gyroZerror = gz / samples;

  accXOffset = ax / samples - 9.81;
  accYOffset = ay / samples;
  accZOffset = az / samples;

  Serial.println("Calibration complete.");
}
```

- **Technique**: IMU-Based Motion and Orientation Sensing with Offset Calibration and Low-Pass Filtering
- **Input**: Sensor data from the MPU6050 – including:
  - Accelerometer: accX, accY, accZ
  - Gyroscope: gyroX, gyroY, gyroZ
  - Temperature: temp
- **Preprocessing**:
  - Calibration:
    - Take multiple samples while stationary.
    - Compute average values to determine offsets (baseline error).
    - Apply corrections for gravity (e.g., subtracting ~9.81 m/s² from vertical axis).
  - Filtering:
  - Apply a simple low-pass filter on gyroscope data to smooth out noise:
    filtered = previous × 0.9 + new × 0.1
- Detection Logic:
  - Each time data is read:
  - Acceleration and gyroscope values are corrected using calibration offsets.
  - Data is then notified over Bluetooth and uploaded to a server at regular intervals.

Network connectivity



- BLE
  - Update data in realtime if connected



- WiFi with MQTT
  - periodically update data
  - Notify users

Save Energy By

- Enter light sleep after 5 minutes of inactivity
- Wake up when the front force sensor is pressed

Uneven weights distribution

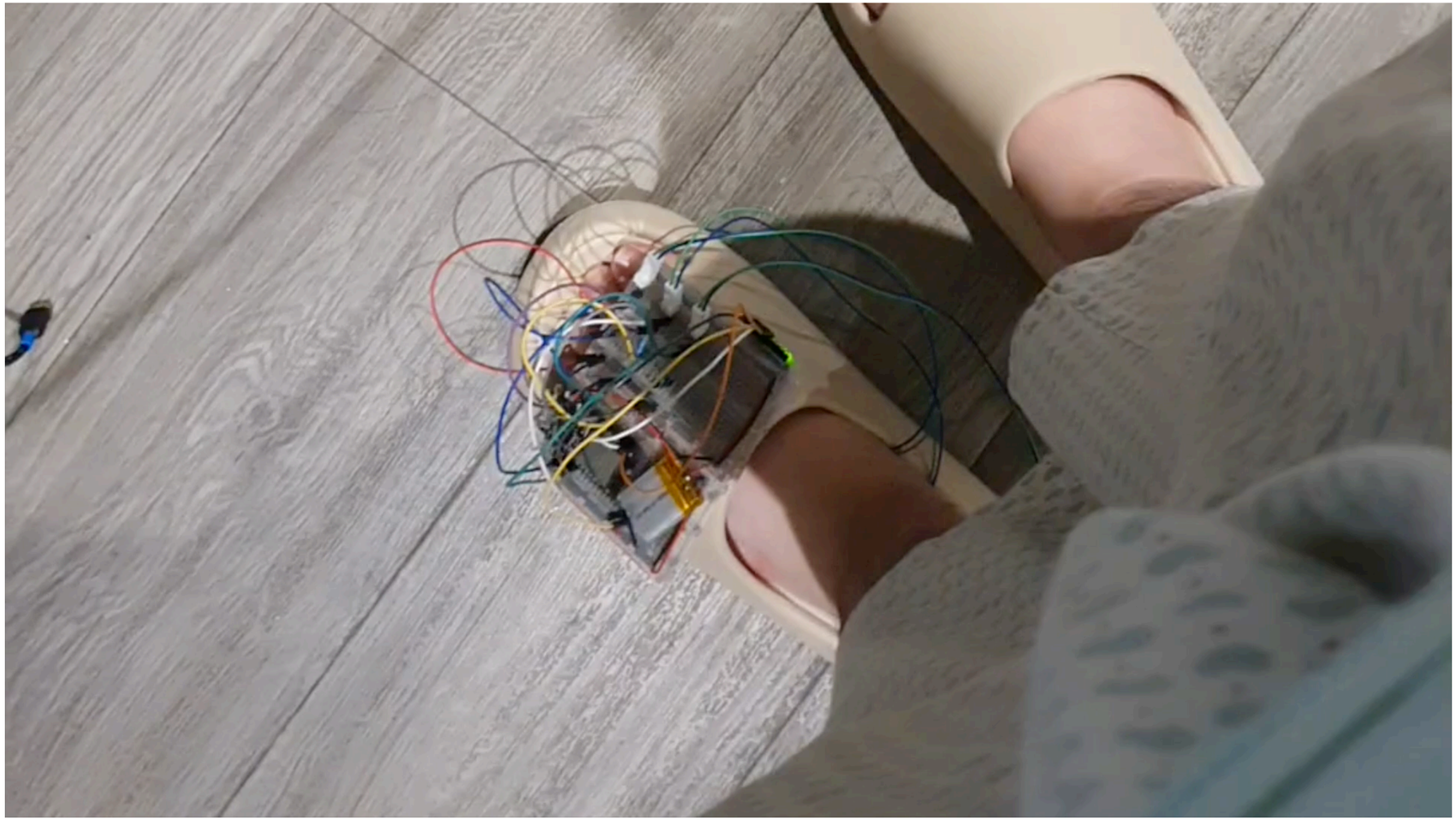
Feature & Execution

```
#define BAD_POSTURE_DETECTION_NUM 150

int badPostureCount = 0;

void detectPostureImbalance()
{
  Serial.println("Checking your posture...");
  float total = (float)getTotalForce();
  // Calculate percentage contribution
  float frontPct = ((double)getForce(0) / total);
  float midPct = ((double)getForce(1) / total);
  float heelPct = ((double)getForce(2) / total);

  // Define acceptable ranges
  const float minFront = 0.25, maxFront = 0.5;
  const float minMid = 0.15, maxMid = 0.4;
  const float minHeel = 0.25, maxHeel = 0.5;

  bool unbalanced =
      (frontPct < minFront || frontPct > maxFront) ||
      (midPct < minMid || midPct > maxMid) ||
      (heelPct < minHeel || heelPct > maxHeel);

  if (unbalanced)
  {
    badPostureCount++;
  }
  else
  {
    badPostureCount = 0;
  }

  if (badPostureCount >= BAD_POSTURE_DETECTION_NUM)
  {
    readingsJSONInfo["title"] = "⚠️ Warning: Unbalanced posture detecte   ;
    readingsJSONInfo["msg"] = "Front: " + String(frontPct) + "%, Mid: " + String(midPct) + "%, Heel: " +
String(heelPct) + "%";
    readingsJSONInfo["type"] = "warning";

    lastUploadUB = upload(INFO_TOPIC, JSON.stringify(readingsJSONInfo), lastUploadUB, uploadIntervalUB);
  }
}
```

- **Input**: Force readings from three zones (front, mid, heel).
- **Normalization**: Each zone's force is turned into a percentage of the total force.
- **Evaluation**: Check whether these percentages are within acceptable ranges (thresholds).
- **Detection Logic**: If any zone's percentage is outside the range repeatedly (badPostureCount >= N), trigger a warning.

Uneven weights distribution

Demo



Step-counting

Feature & Execution

```
void detectStep()
{
  int front = getForce(0);
  int mid = getForce(1);
  int heel = getForce(2);

  double ax = (double)readingsJSONAcc["x"];
  double deltaAx = abs(ax - previousAx);
  previousAx = ax;

  unsigned long now = millis();

  if (deltaAx > ACCEL_THRESHOLD && heel > FORCE_THRESHOLD && mid > FORCE_THRESHOLD && front > FORCE_THRESHOLD && !stepDetected && (now - lastStepTime > DEBOUNCE_TIME))
  {
    stepDetected = true;
    lastStepTime = now;
    stepCount++;

    stepChar->setValue(String(stepCount));
    stepChar->notify();
  }

  if (heel < FORCE_THRESHOLD && mid < FORCE_THRESHOLD && front < FORCE_THRESHOLD)
  {
    stepDetected = false;
  }
  lastUploadStep = upload(STEP_TOPIC, String(stepCount), lastUploadStep, uploadIntervalStep);
}
```

- **Input**:
  Force readings from three zones (front, mid, heel) and the X-axis acceleration value (ax) from an accelerometer.

- **Preprocessing**:
  Compute the difference between the current and previous X-axis acceleration values (deltaAx = |ax - previousAx|).

- **Evaluation**:

  - Check whether all of the following conditions are satisfied:

  - deltaAx exceeds a predefined acceleration threshold.

  - Force values at all three zones are above their respective force thresholds.

  - A step hasn't been recently detected (!stepDetected), and debouncing time has passed (now - lastStepTime > DEBOUNCE_TIME).

- **Detection Logic**:
  If all conditions are met, register a new step (stepCount++), update the timestamp, and notify the connected device or server.
  If all force readings drop below the threshold, reset stepDetected to allow the next step detection.

Step-counting

Demo

Falling detection

Feature & Execution

```
static double lastAccMagnitude = 0;

void detectFalling()
{
  double accX = (double)readingsJSONAcc["x"];
  double accY = (double)readingsJSONAcc["y"];
  double accZ = (double)readingsJSONAcc["z"];
  double accMagnitude = sqrt(accX * accX + accY * accY + accZ * accZ);

  double gyroX = (double)readingsJSONGyro["x"];
  double gyroY = (double)readingsJSONGyro["y"];
  double gyroZ = (double)readingsJSONGyro["z"];
  double rotationMagnitude = sqrt(gyroX * gyroX + gyroY * gyroY + gyroZ * gyroZ);

  upload(String(FALL_TOPIC) + "_log", debug, 0, 0);
  if ((accMagnitude > FALLING_ACCEL_DETECTION_THRESHOLD_HIGH || accMagnitude <
FALLING_ACCEL_DETECTION_THRESHOLD_LOW) && rotationMagnitude >
FALLING_ROTATION_DETECTION_THRESHOLD)
  {
    lastAccMagnitude = accMagnitude;
    readingsJSONInfo["title"] = "⚠️ Warning: Possible Fall Detectio   ;
    // readingsJSONInfo["msg"] =nüebu
g;  readingsJSONInfo["msg"] =
"A possible fall has been detected. Please check on the elder under your care.";
    readingsJSONInfo["type"] = "warning";
    lastUploadFall = upload(INFO_TOPIC, JSON.stringify(readingsJSONInfo), lastUploadFall,
uploadIntervalFall);
  }
  else
  {
    lastAccMagnitude = 0;
  }
}
```

- **Technique**: Fall Detection Using Accelerometer and Gyroscope Magnitude Thresholding
- **Input**: Acceleration readings (accX, accY, accZ) and gyroscopic readings (gyroX, gyroY, gyroZ) from an IMU sensor.
- **Preprocessing**:
  - Compute: Acceleration Magnitude, Rotation Magnitude
- **Evaluation**:
  Check for a potential fall if:
  - Acceleration magnitude is either too high or too low (outside normal range), and rotational magnitude is above a certain threshold.
- **Detection Logic**:
  If both conditions are satisfied, trigger a fall alert, update the warning message, and upload it to a specified topic. Otherwise, reset the last acceleration magnitude to 0.

Falling detection

Demo