TIPE : l'impact physique de la course à pied

Malaury DUTOUR

Épreuve de TIPE

Session 2023

Problématique

Quelle est l'influence du type de revêtement de sol et du chaussage sur le corps humain lors de la course à pied ?

Plan de l'exposé

Présentation du problème

Les sols dans nos villes sont souvent bétonnés et ne sembleraient donc pas optimaux pour les coureurs, est-ce vraiment le cas ? Quelle influence peut avoir le chaussage sur le confort du coureur ?

Nous allons le vérifier à l'aide d'un dispositif expérimental.

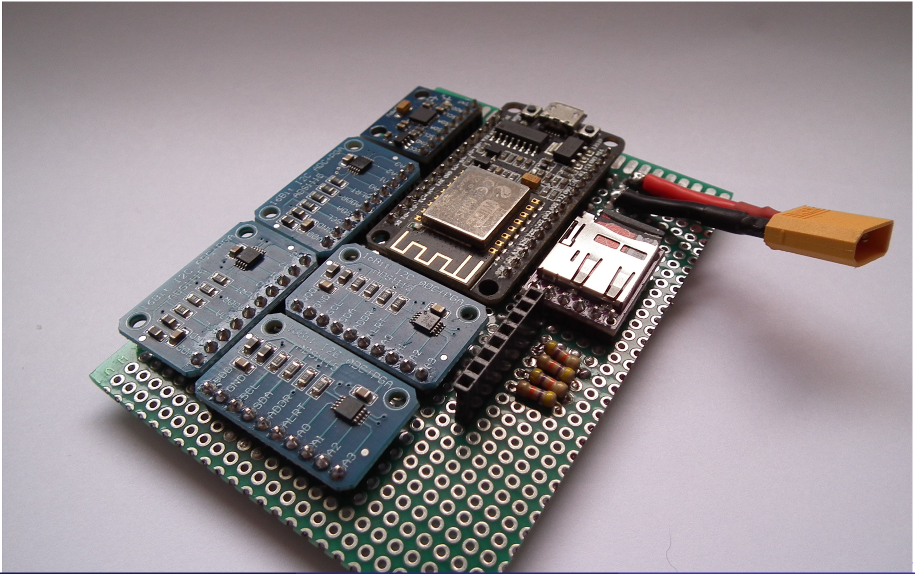
Présentation du dispositif expérimental

Afin de mesurer les chocs subis par le coureur nous avons réalisé une semelle particulière dotée de :

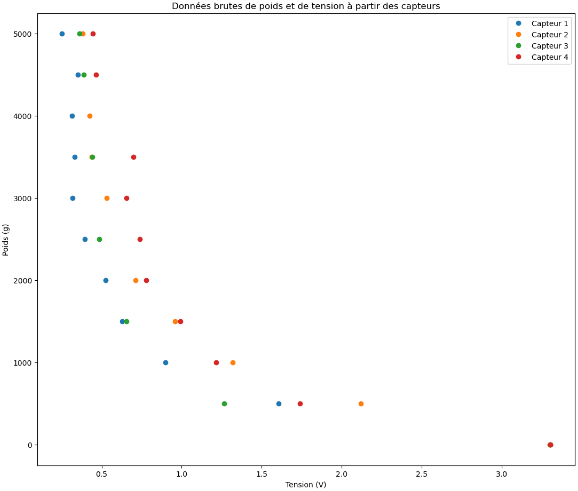
- 4 capteurs de pression
- Un lecteur de carte SD
- Un microcontrôleur

Carte de contrôle

Voici la carte avec le microcontrôleur, les convertisseurs analogiques-numériques et le lecteur de carte SD :



Le capteur de pression est une résistance variable de loi non linéaire.



Calibration - ajustement de courbe 1/2

Envisageons la fonction suivante : $f_{a,b,c}(x) = \frac{a}{x^b} + c$

Notons x_i les valeurs en tension correspondant à un poids mesuré y_i.

Il faut trouver a,b,c minimisant $\sum_{i=0}^{n}(f_{a,b,c}(x_i) - y_i)^2$

Calibration - ajustement de courbe 2/2

Nous utilisons la méthode de la descente de gradient pour trouver une valeur approchée de a,b,c minimisant $C(a,b,c) = \sum_{i=0}^{n}(f_{a,b,c}(x_i) - y_i)^2$.

L'ideal est d'avoir $\frac{\partial C}{\partial a}(a,b,c) = 0$ $\frac{\partial C}{\partial b}(a,b,c) = 0$ $\frac{\partial C}{\partial c}(a,b,c) = 0$, soit $\nabla C = 0$.

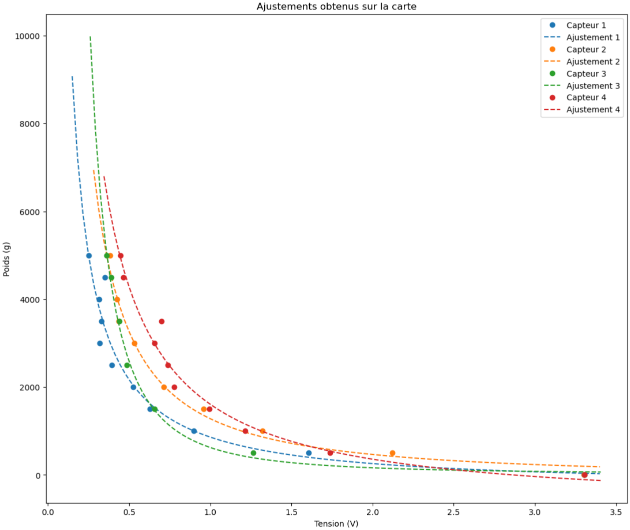
Pour cela fixons $\alpha \in \mathbb{R}^+$, le taux d'apprentissage et un seuil $\varepsilon \in \mathbb{R}^+$.

Nous construisons la suite $P_k = \begin{pmatrix} a_k \\ b_k \\ c_k \end{pmatrix}$ de la façon suivante :

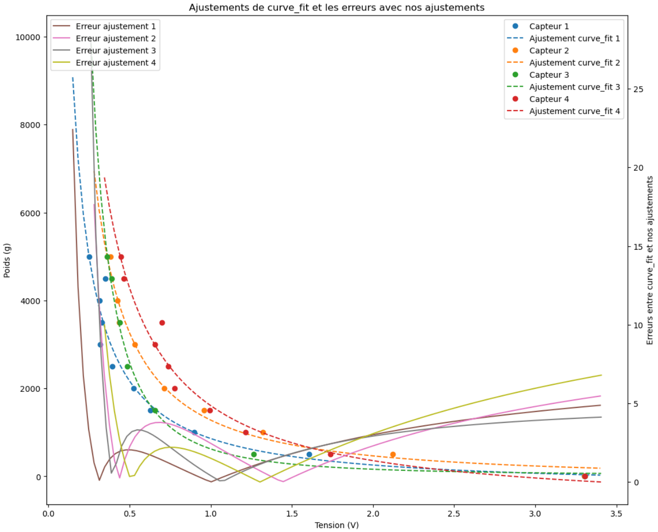
- P_0 fixé de manière arbitraire
- $P_{k+1} = P_k - \alpha \times \nabla C$

Condition d'arrêt : $\nabla C < \varepsilon$

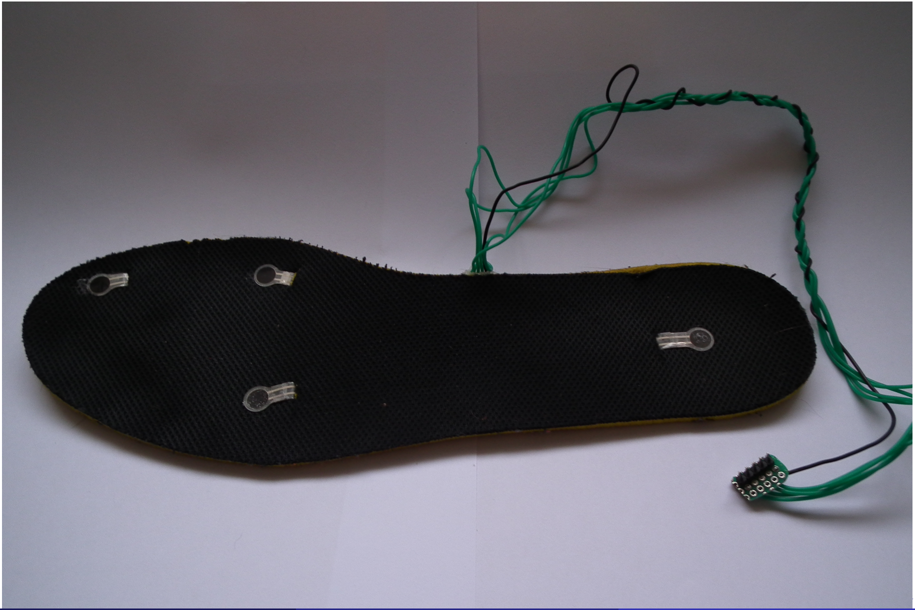
Résultats de la méthode des moindres carrés avec la descente de gradient effectuée sur la carte :



Comparaison avec la fonction `curve_fit` de `scipy` :



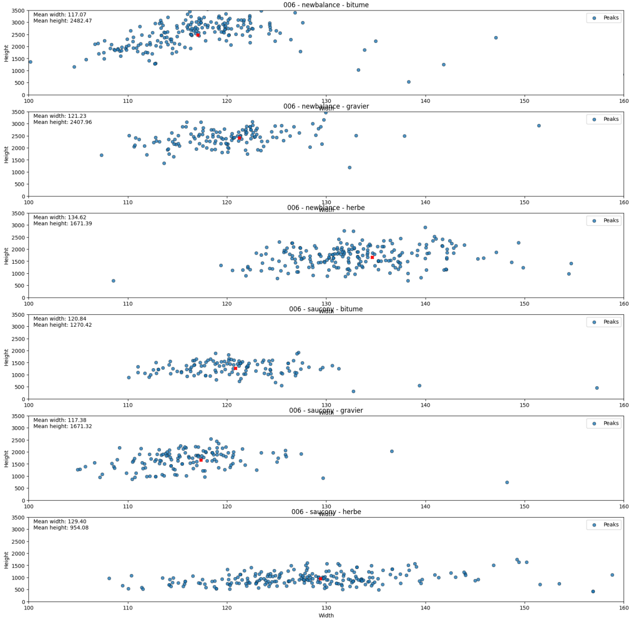
Voici les capteurs disposés sur la semelle :



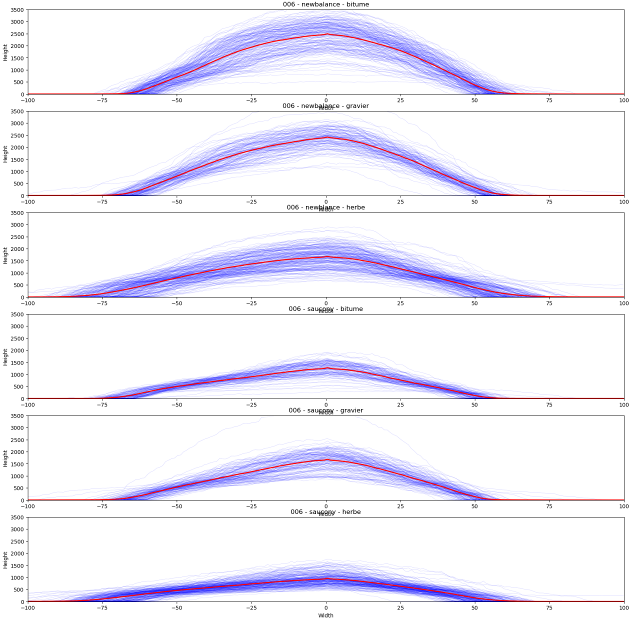
Les premiers résultats

Après avoir couru sur différents revêtements (béton, gravier, pelouse) avec des chaussures de ville et de course, nous obtenous :

Quelques visualisations 1/3

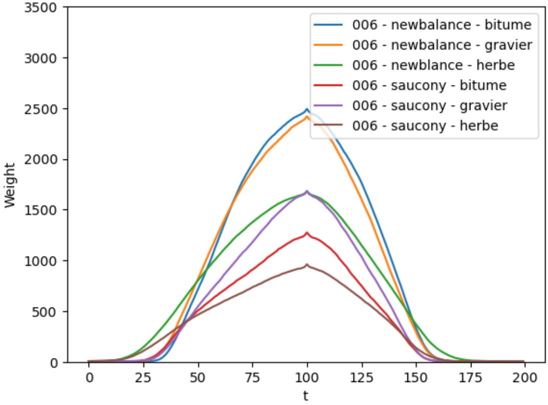


Quelques visualisations 2/3

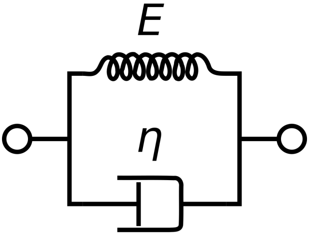


Quelques visualisations 3/3

Gardons ici les pics moyens pour chaque revêtement et type de chaussures :



L'os est un matériau viscoélastique, utilisons le modèle de Kelvin-Voigt :



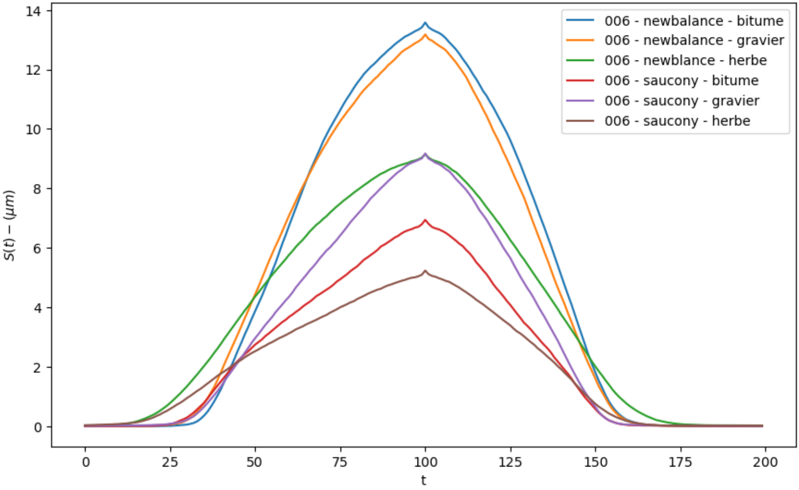
Nous établissons l'équation différentielle suivante :

$$\sigma = E \times \varepsilon + \eta \times \frac{d\varepsilon}{dt}$$

Et la fonction de transfert associée :

$$H(\omega) = \frac{1}{E + i\omega\eta} \quad avec \quad E = 18 \cdot 10^9 GPa \quad \eta = 218,4 Pa \cdot sec^{-1}$$

Puisque $\omega \ll 10^3 rad \cdot sec^{-1}$, nous avons $H(\omega) \approx \frac{1}{E}$ on obtient donc :



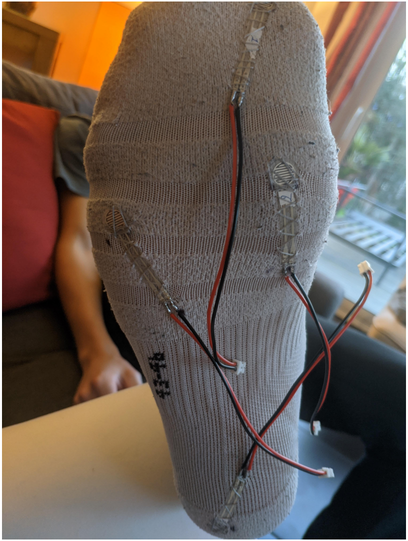
Conclusion

D'après les résulats expérimentaux et le modèle physique :

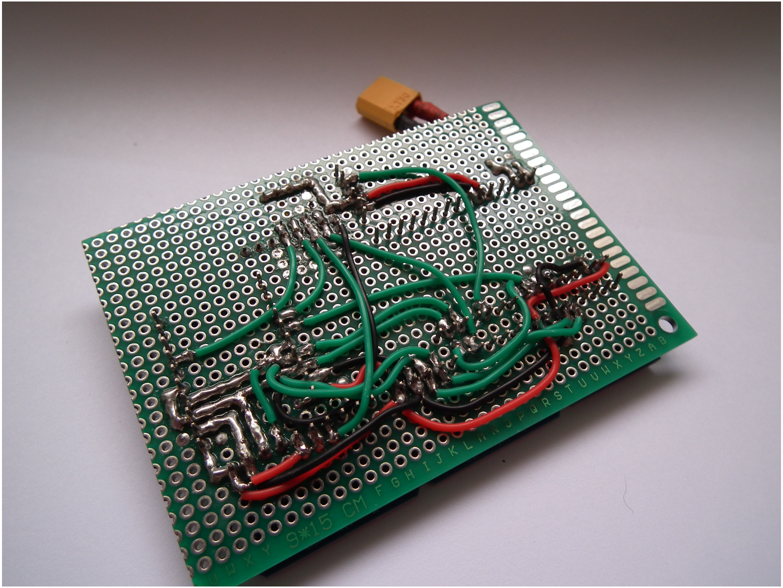
- Un chaussage adapté est primordial pour atténuer les chocs
- Le revêtement le moins traumatisant est le gazon

Dès lors, la bétonisation des sols est traumatisante pour les coureurs.

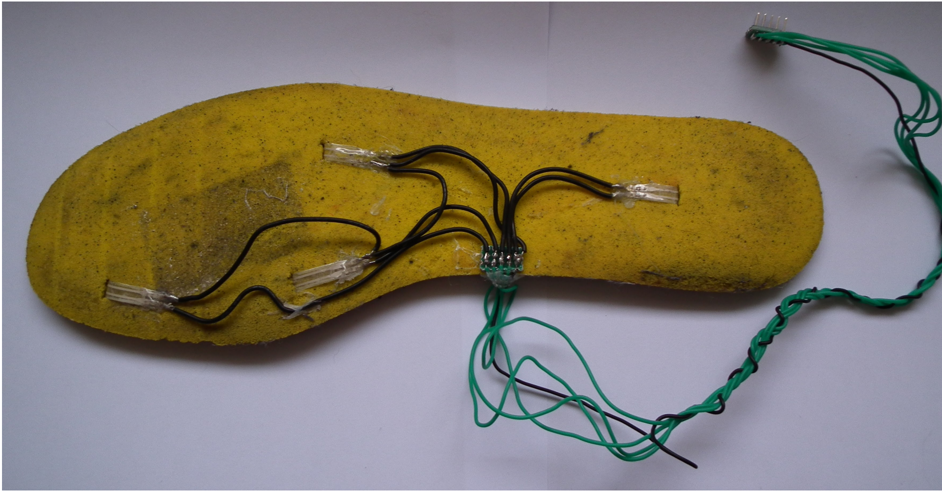
Annexe 1



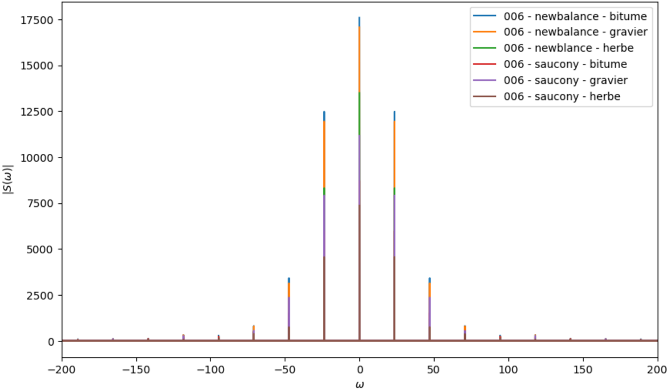
Annexe 2



Annexe 3

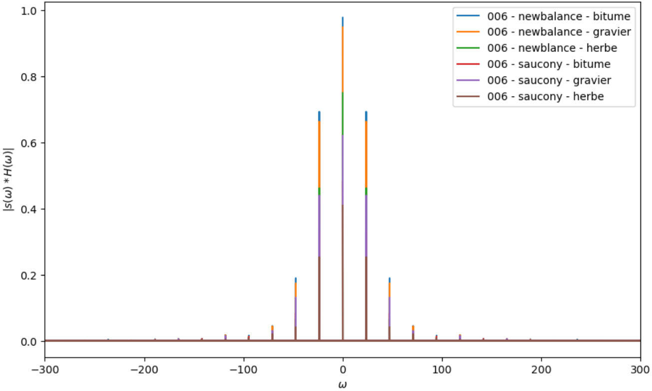


Annexe 4



17500
15000
12500
10000
7500
5000
2500
0
$|S(\omega)|$
−200
−150
−100
−50
0
50
100
150
200
ω
006 - newbalance - bitume
006 - newbalance - gravier
006 - newblance - herbe
006 - saucony - bitume
006 - saucony - gravier
006 - saucony - herbe

Annexe 5



006 - newbalance - bitume
006 - newbalance - gravier
006 - newblance - herbe
006 - saucony - bitume
006 - saucony - gravier
006 - saucony - herbe
$|s(\omega) * H(\omega)|$
ω
1.0
0.8
0.6
0.4
0.2
0.0
−300
−200
−100
0
100
200
300

Annexe 6

```
template <typename T, std::size_t params_count>
T GradientDescFactory<T, params_count>::cost_func(
    DataSet<T> &data, std::array<T, params_count> params) {
    T cost = data.accumulate([&](DataPoint<T> dp) {
        return std::pow(func(dp.x, params) - dp.y, 2);
    });
    return cost / (2 * data.size());
}
template <typename T, std::size_t params_count>
std::array<T, params_count>
GradientDescFactory<T, params_count>::grad_cost_func(
    DataSet<T> &data, std::array<T, params_count> params) {
    std::array<T, params_count> grad;
    std::array<T, params_count> deriv_eps;
    std::fill(deriv_eps.begin(), deriv_eps.end(), T());
    T h = std::pow(std::numeric_limits<T>::epsilon(), T(1.0 / 3.0));
    for (std::size_t i = 0; i < params_count; i++) {
        deriv_eps[i] = h;
        grad[i] = (cost_func(data, add_arr(params, deriv_eps)) -
                   cost_func(data, add_arr(params, deriv_eps, T(-1.0)))) /
                  (2 * h);

        deriv_eps[i] = T();
    }
    return grad;
}
```

Annexe 7

```
template <typename T, std::size_t params_count>
std::unique_ptr<FittingResult<T>>
GradientDescFactory<T, params_count>::calculateFitting(
    DataSet<T> &data_nonnorm) {
    std::array<T, params_count> params;
    auto data = data_nonnorm.normalize();
    auto initial_params = settings.getInitialParams();
    std::copy(initial_params.begin(), initial_params.end(), params.begin());

    T learning_rate = settings.getLearningRate();
    long iterations = 0;
    GradientDescCompletion completion = GRADIENT_DESC_CONVERGED;
    while (true) {
        auto grad = grad_cost_func(data, params);
        for (std::size_t i = 0; i < params_count; i++) {
            params[i] -= learning_rate * grad[i];
        }

        T norm_grad = 0;
        for (std::size_t i = 0; i < params_count; i++) {
            norm_grad += std::pow(grad[i], 2);
        }

        if (norm_grad < settings.getTolerance()) {
            break;
        }
        if (iterations > settings.getMaxIterations()) {
            completion = GRADIENT_DESC_MAX_ITERATIONS;
            break;
        }
```

Annexe 8

```
        if (iterations % 100 == 0) {
            ESP.wdtFeed();
            Serial.println("Iteration: " + String(iterations) +
                          " Norm: " + String(norm_grad, 8));
        }
        iterations++;
    }
    T cost = cost_func(data, params);

    T rmse = std::sqrt(cost / data.size());
    T sse = cost;
    T r2 = 1 - (sse / (data.size() * std::pow(data.std(), 2)));

    last_stats = GradientDescStats<T>(completion, sse, r2, rmse, iterations);

    return std::make_unique<GradientDescResult<T, params_count>>(
        func, params, data_nonnorm.mean(), data_nonnorm.std());
};
```