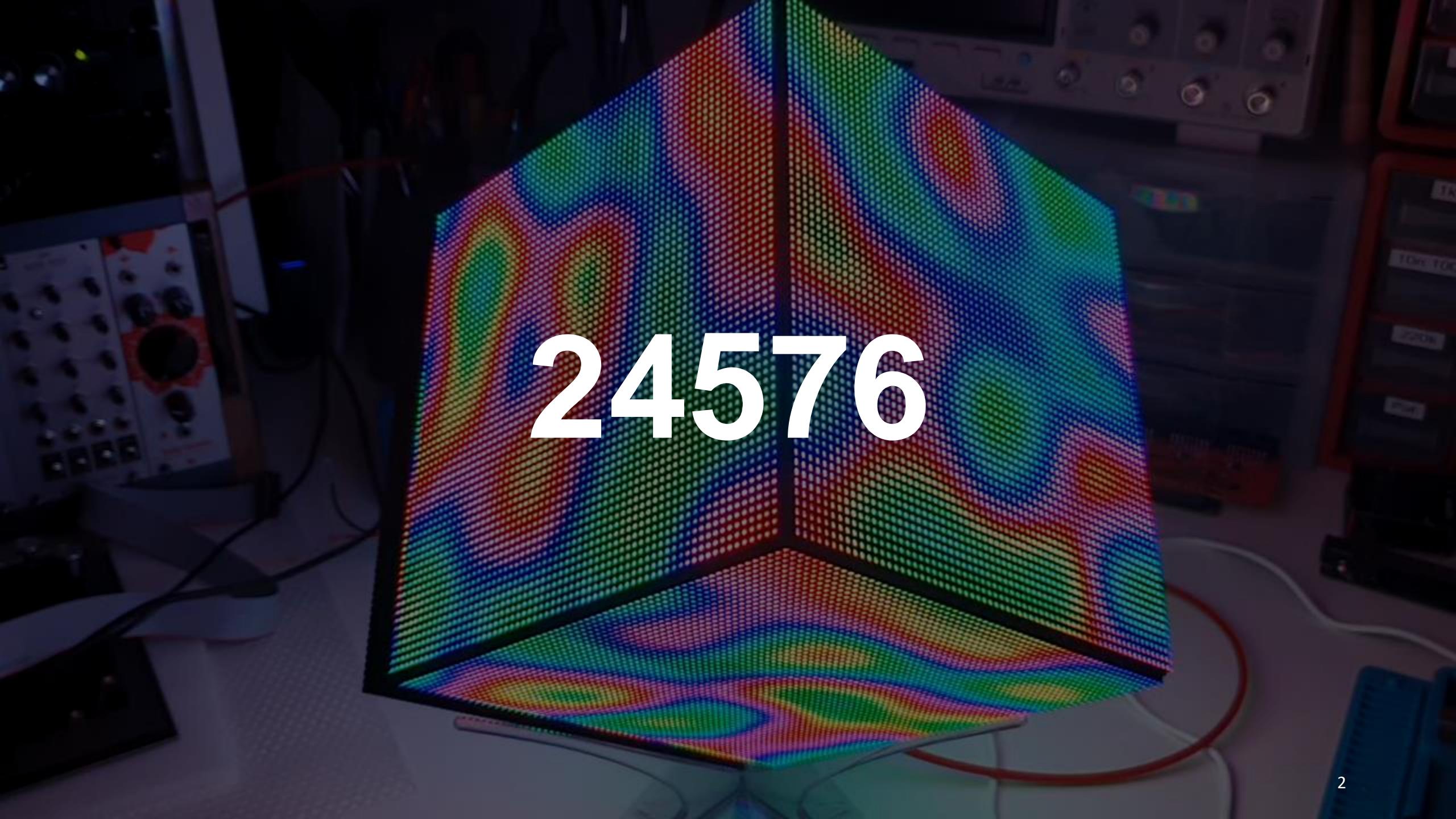


**24576**



24576

# Un LED Cube, pourquoi?

- Multidisciplinaire
- Tangible, visuel
- Educatif
- Esthétique
- Un défi
- Pour le plaisir

# C'est quoi un LED Cube ?

Un **outil** pour amplifier la créativité

Le peintre s'exprime sur un canvas plat, en 2 dimensions.

Le cube offre la 3ème dimensions et le pinceau prend alors la forme d'un algorithme.

# La recette pour fabriquer un LED Cube

## Ingrédients :

- 6 panneaux LED avec leurs câbles
- 1 Raspberry Pi 4B & sa carte mémoire
- 2 batteries 7.4V
- 3 convertisseurs DC-DC
- 1 accéléromètre et son câble
- 1m de câble électrique
- 5 paires de connecteurs électriques
- 2 connecteurs de distribution (Wago)
- 20 paires de petits aimants et de la colle
- 24 vis et quelques élastiques
- Une imprimante 3D
- Beaucoup de temps

## Recette :

1. Imprimez les supports
2. Assemblez les faces verticales avec les vis
3. Collez les aimants pour les faces supérieure et inférieure
4. Assemblez le reste du cube
5. Configurez l'OS et le logiciel sur la carte mémoire du RPi
6. Installez le RPi et les batteries
7. Connectez tous les câbles
8. Connectez les batteries
9. Admirez votre création

# Anatomie d'un panneau LED

Panneaux de toutes les tailles.

*Pitch* allant de moins de 2mm à plus de 10mm.

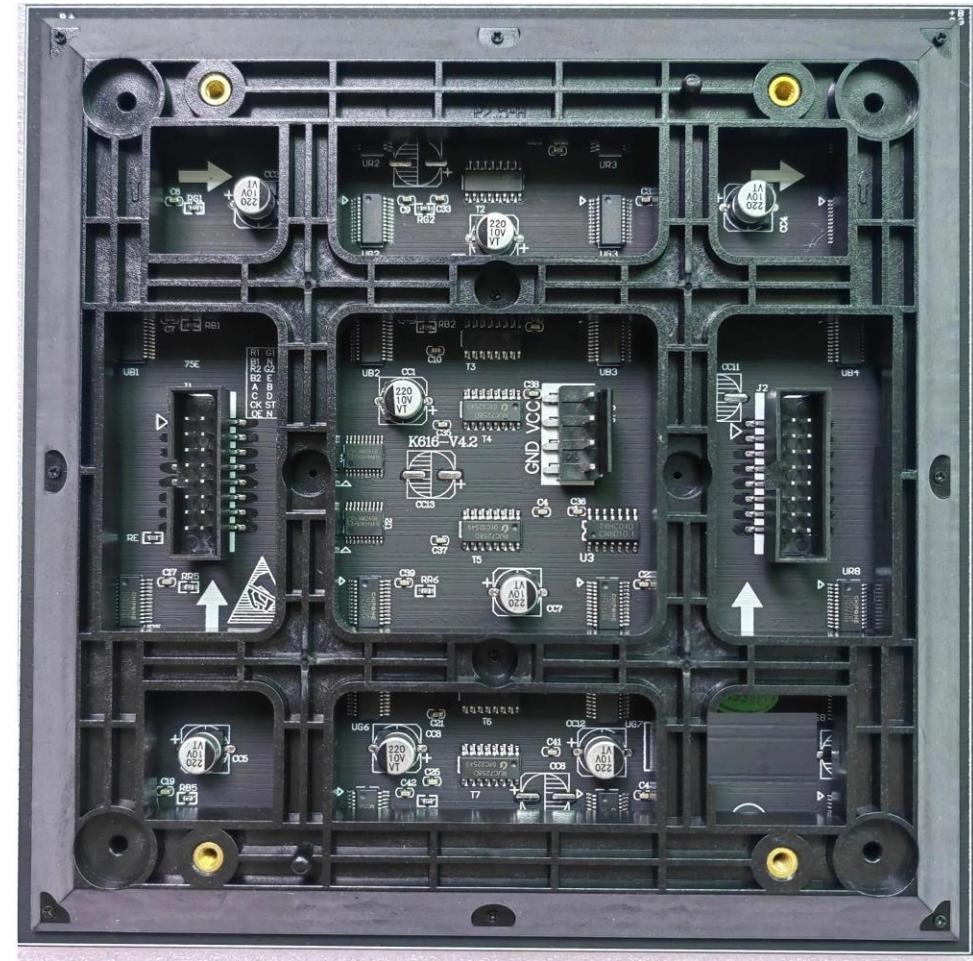
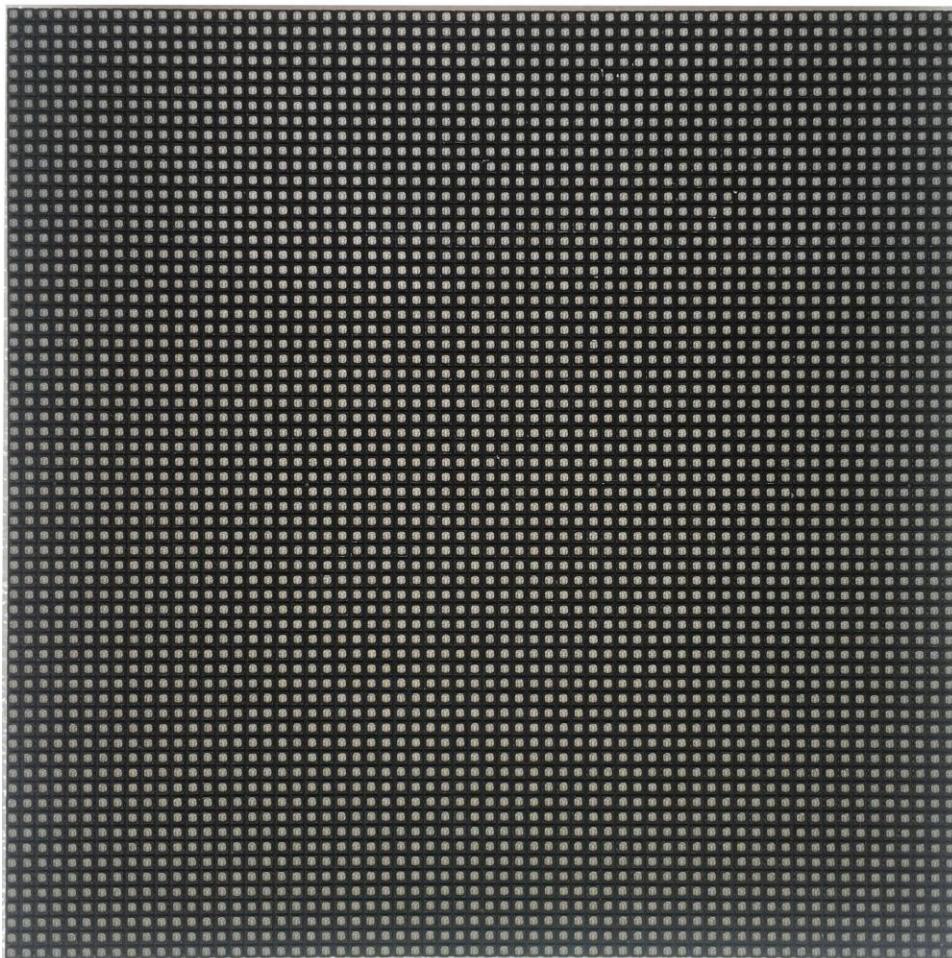
Standard HUB75.

Chaque pixel est une LED RGB :

- Pas de mémoire interne.
- Nécessite un rafraîchissement constant.
- Couleur binaire (on ou off). Gradation logicielle.
- Difficile à piloter par logiciel (facile avec un FPGA).



# Anatomie d'un panneau LED



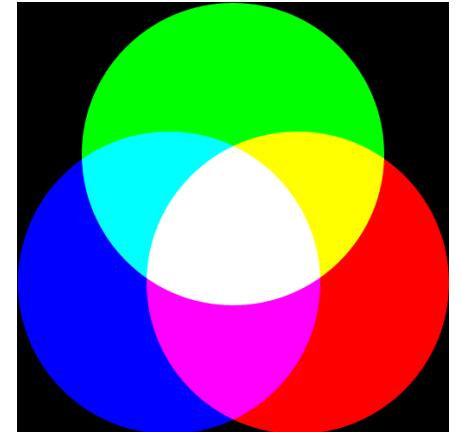
# Limites des panneaux LED

Synthèse additive Rouge, Vert, Bleu (RVB)

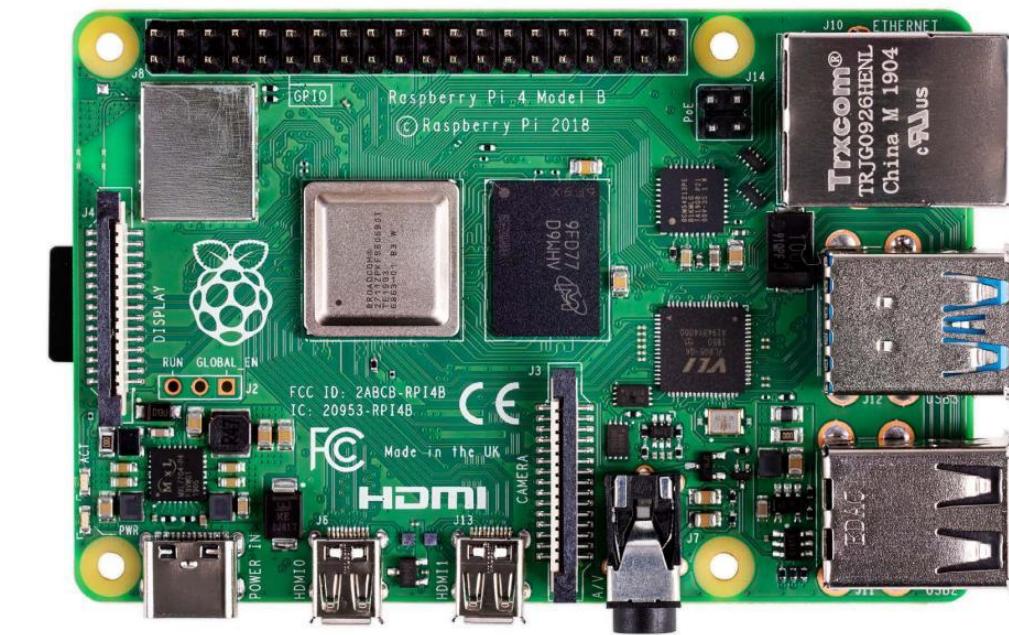
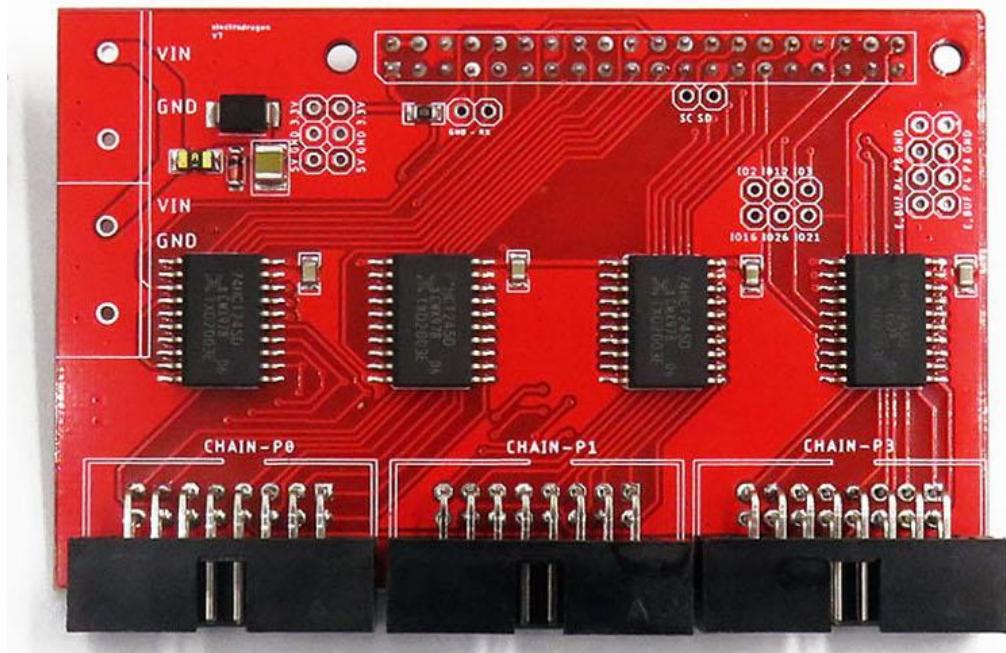
La synthèse additive ne peut reproduire que les couleurs de chromaticité moindre que ses primaires.

Couleurs moins intenses.

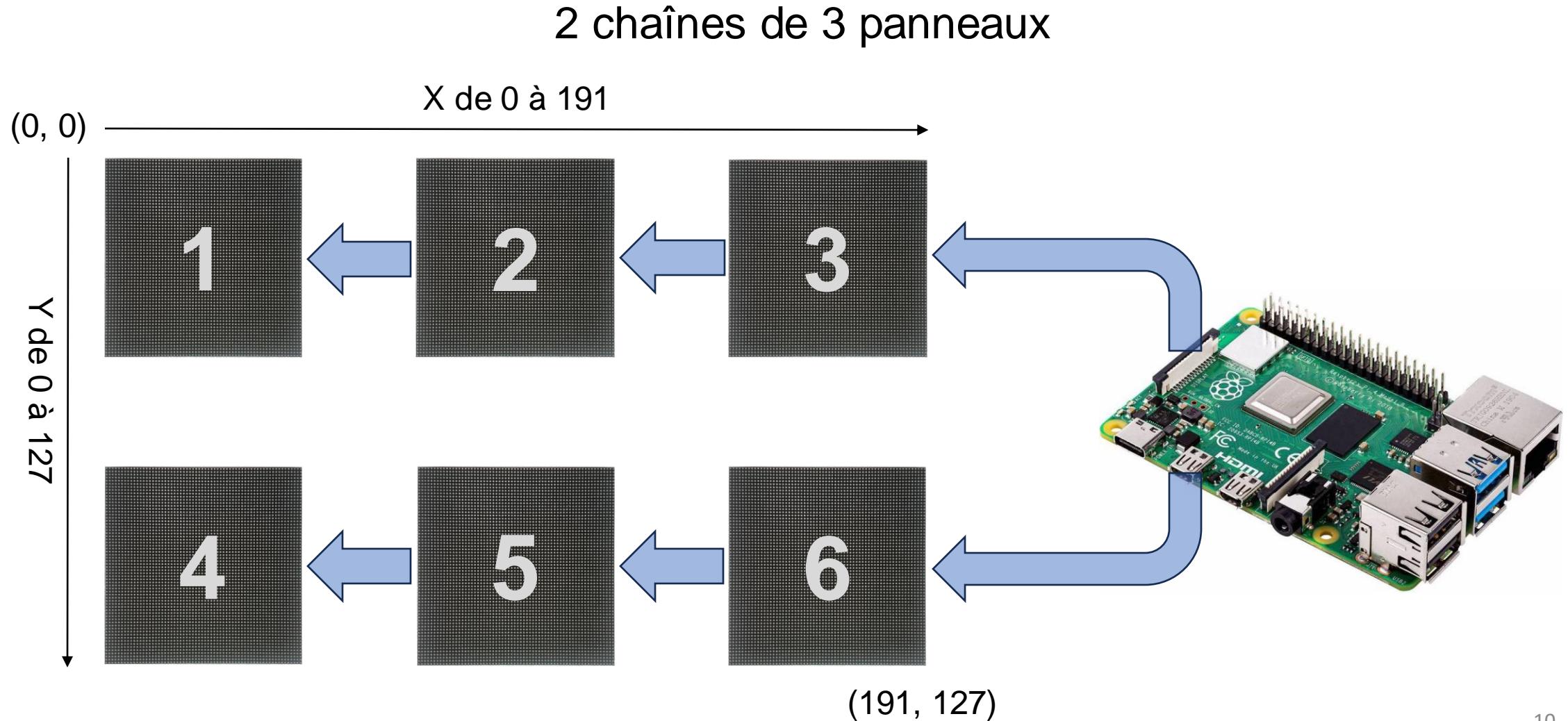
Correction des couleurs possibles.



# Cœur et interface pour les LEDs

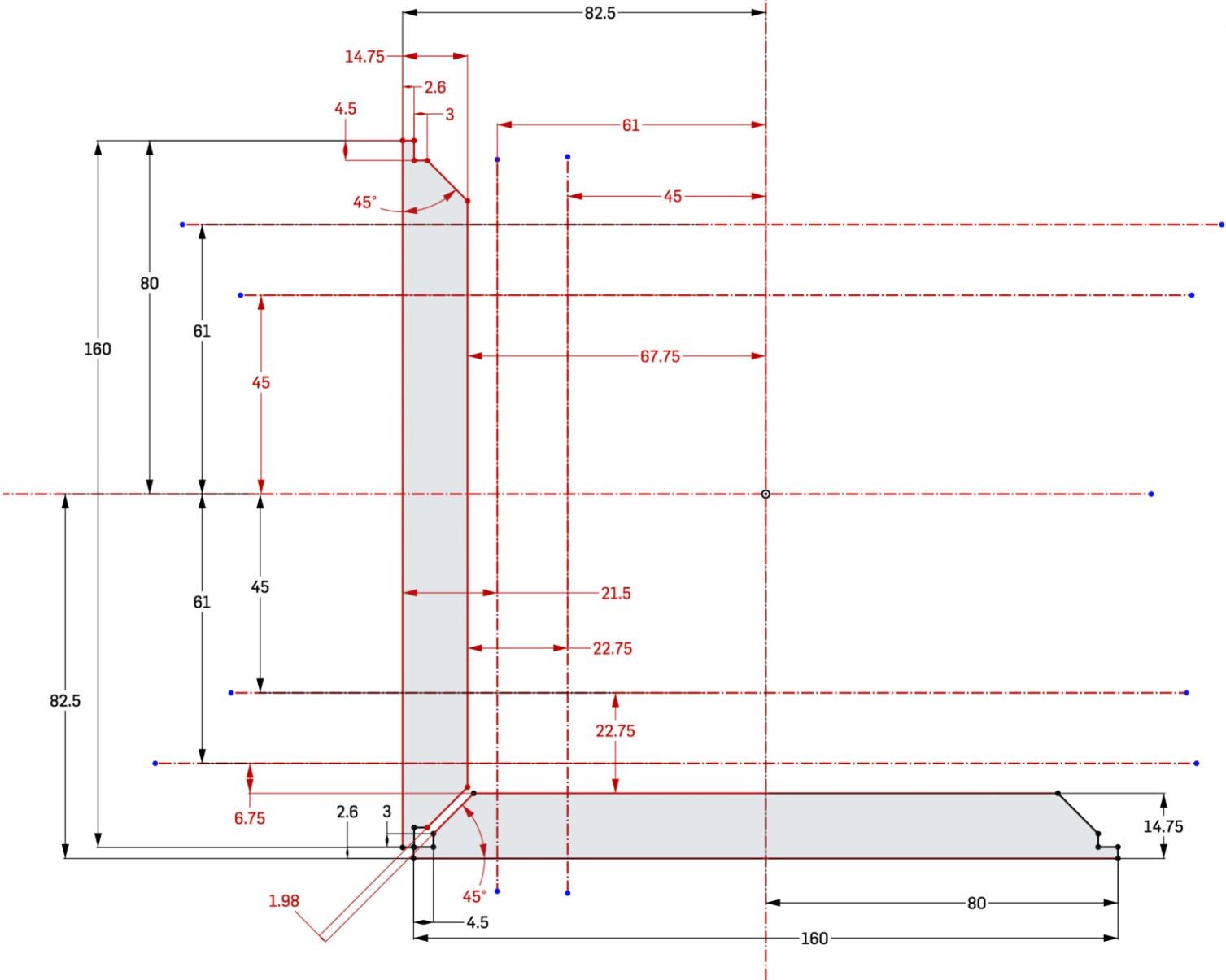


# Connexion des panneaux

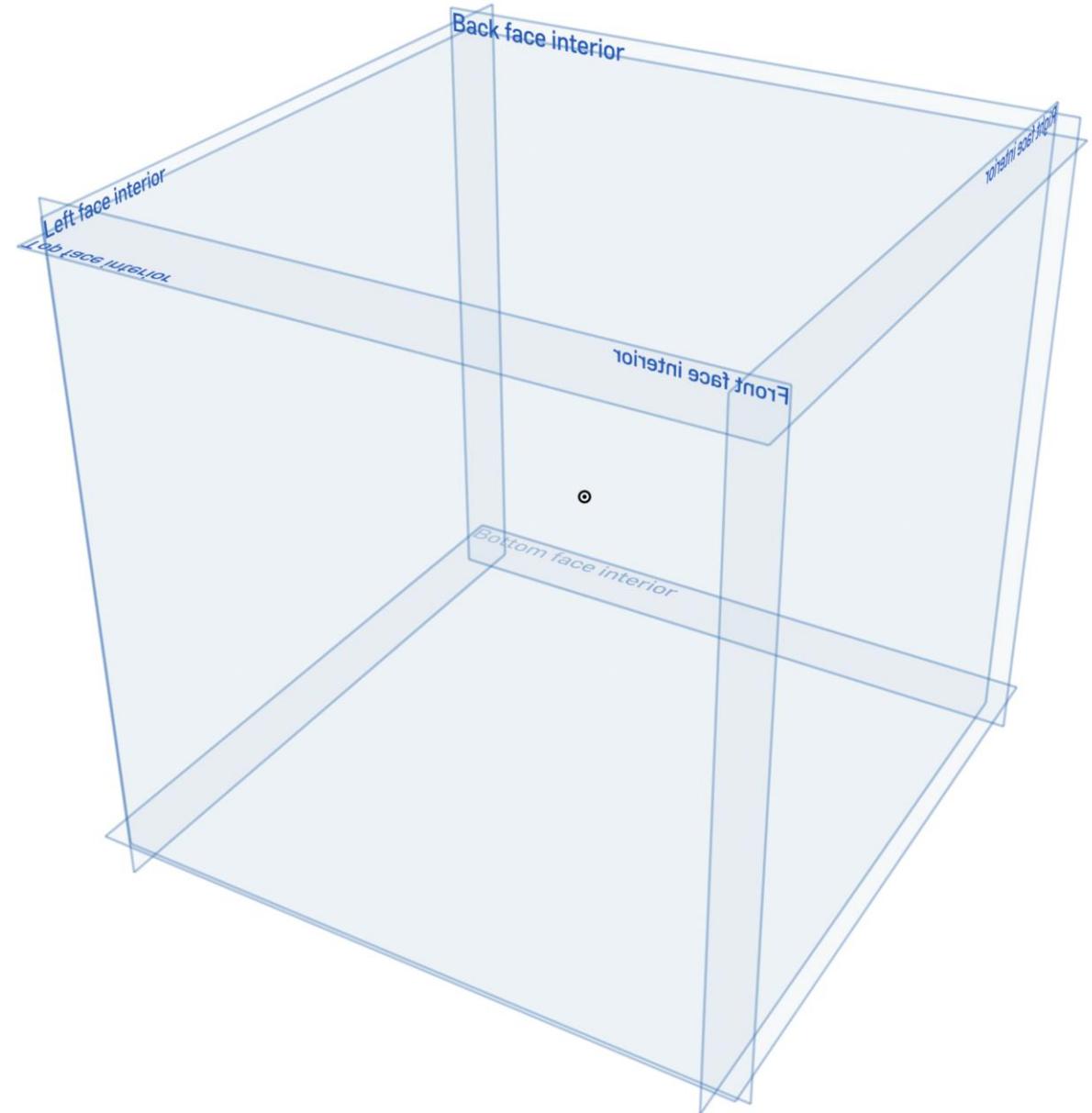


# Assemblage mécanique

## Jointure des panneaux

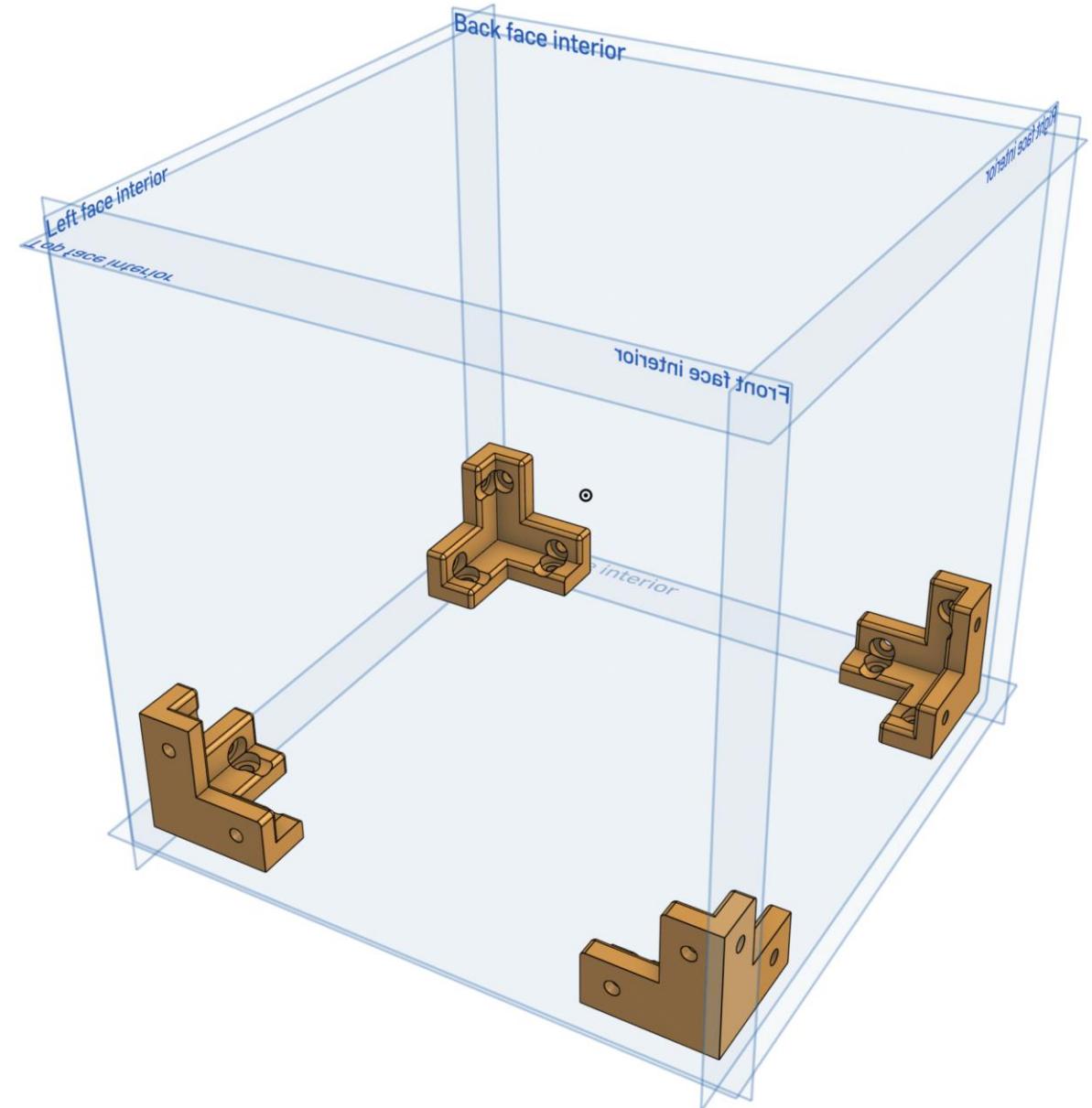


# Assemblage mécanique



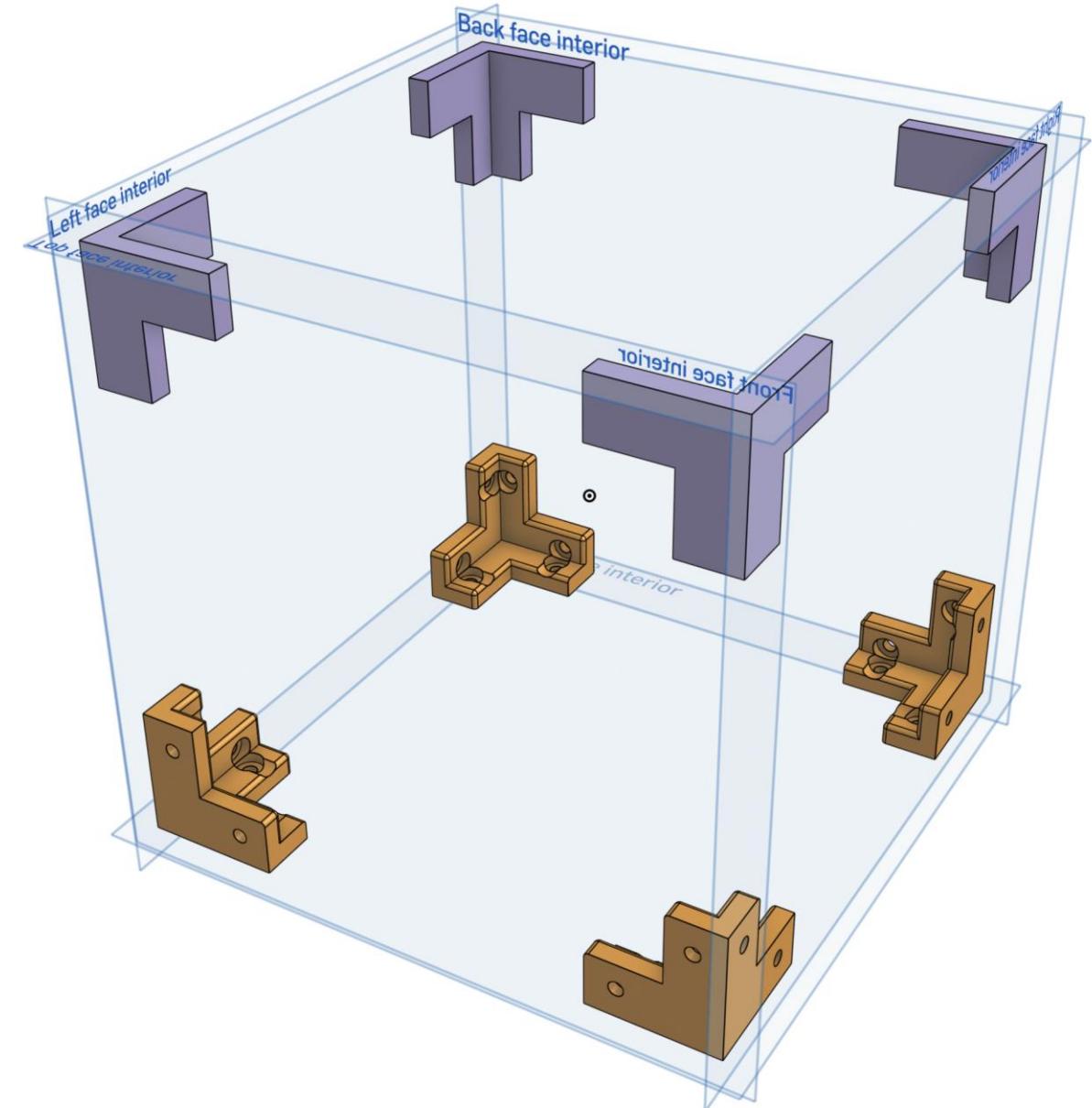
# Assemblage mécanique

## Liaison des panneaux



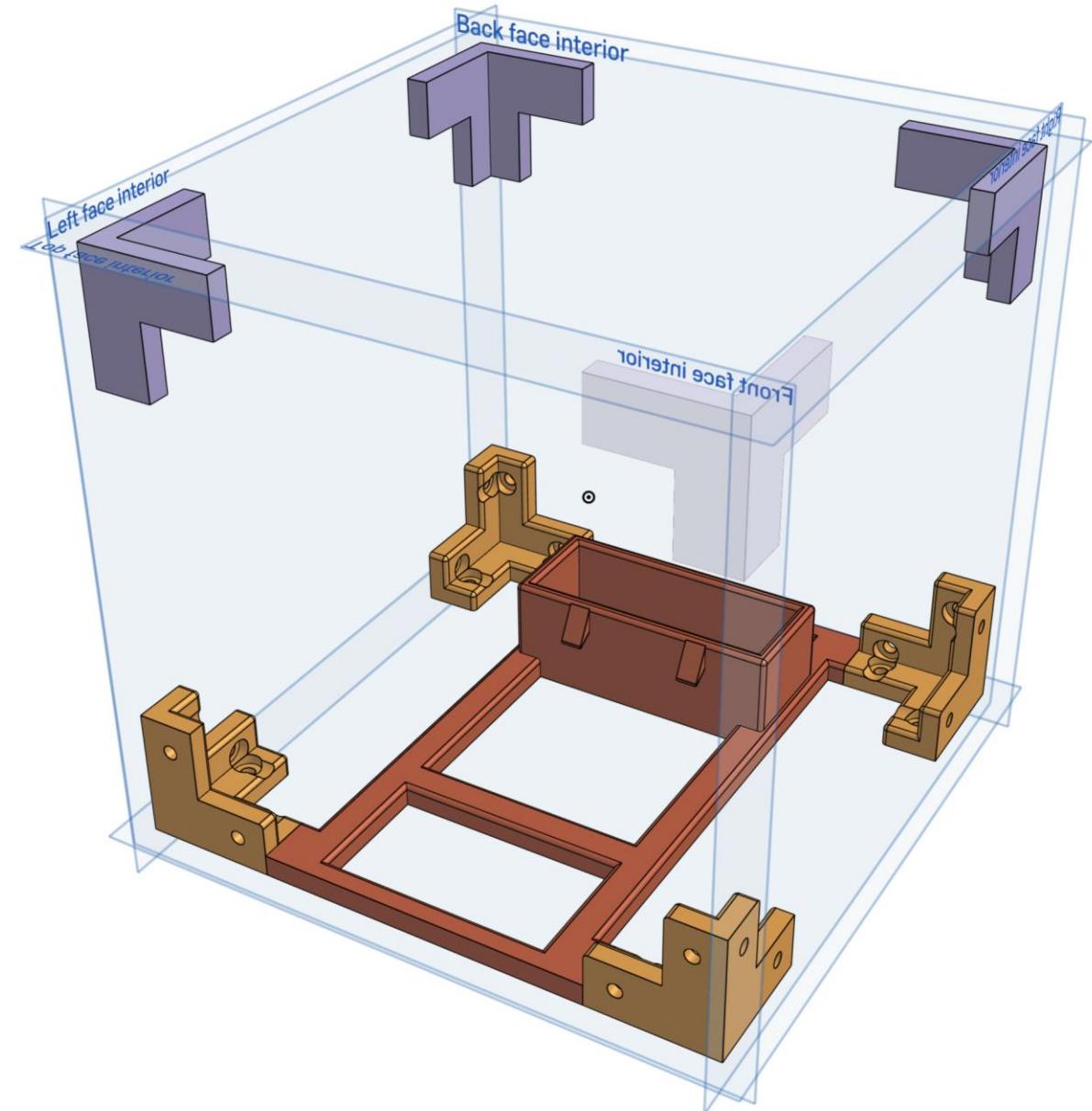
# Assemblage mécanique

## Liaison des panneaux



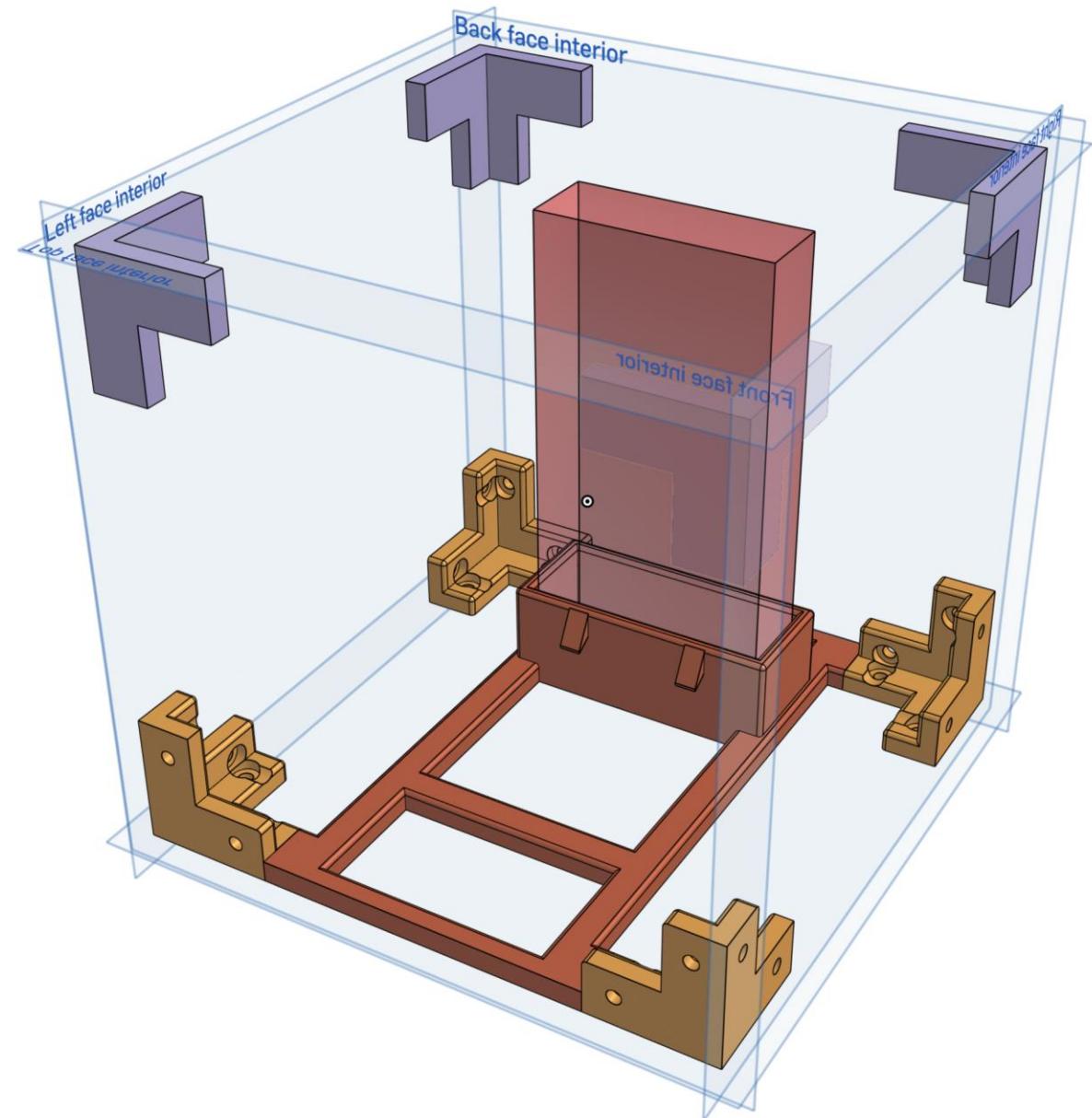
# Assemblage mécanique

## Support pour le Raspberry



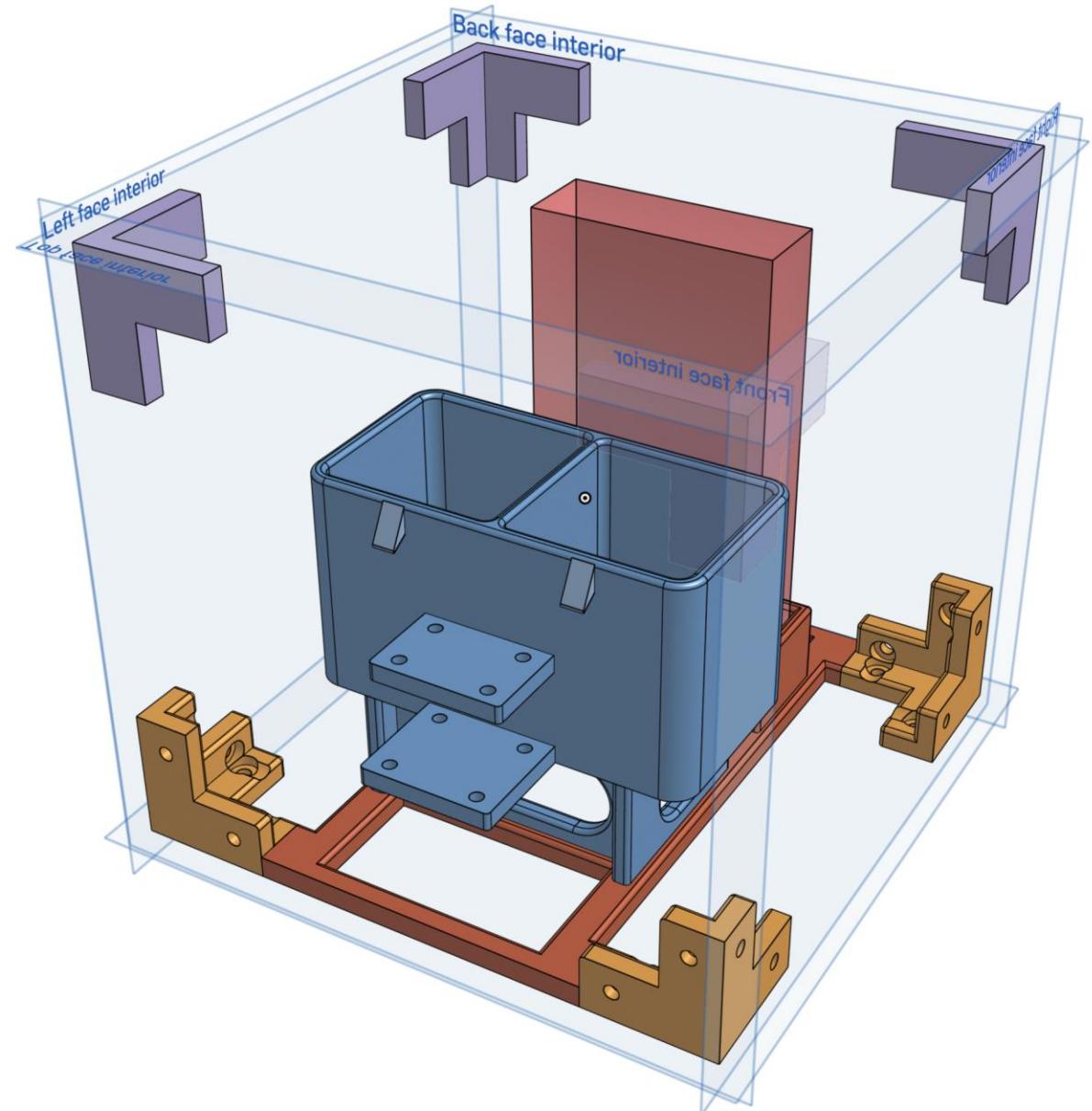
# Assemblage mécanique

## Support pour le Raspberry



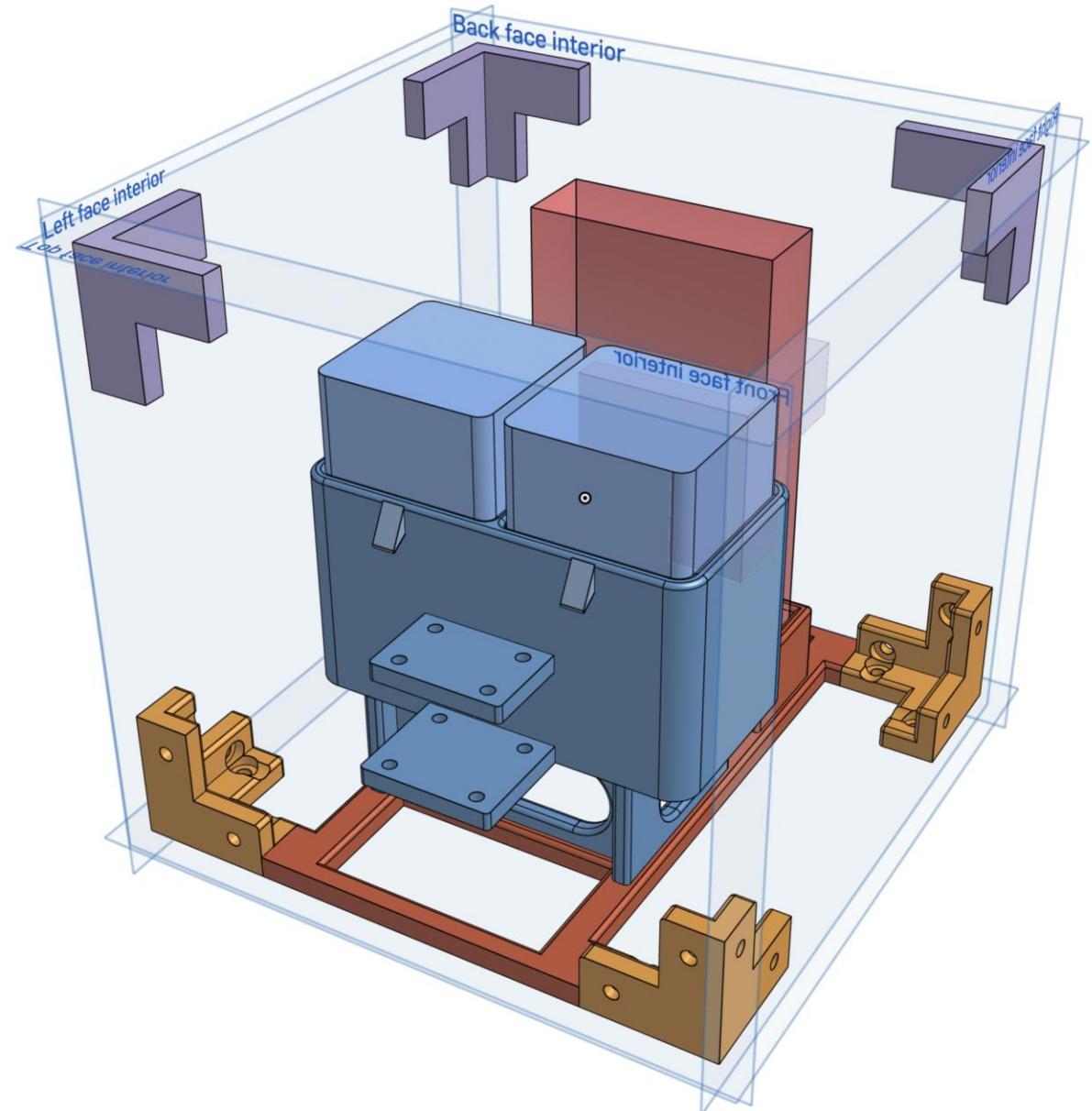
# Assemblage mécanique

Support pour  
les batteries



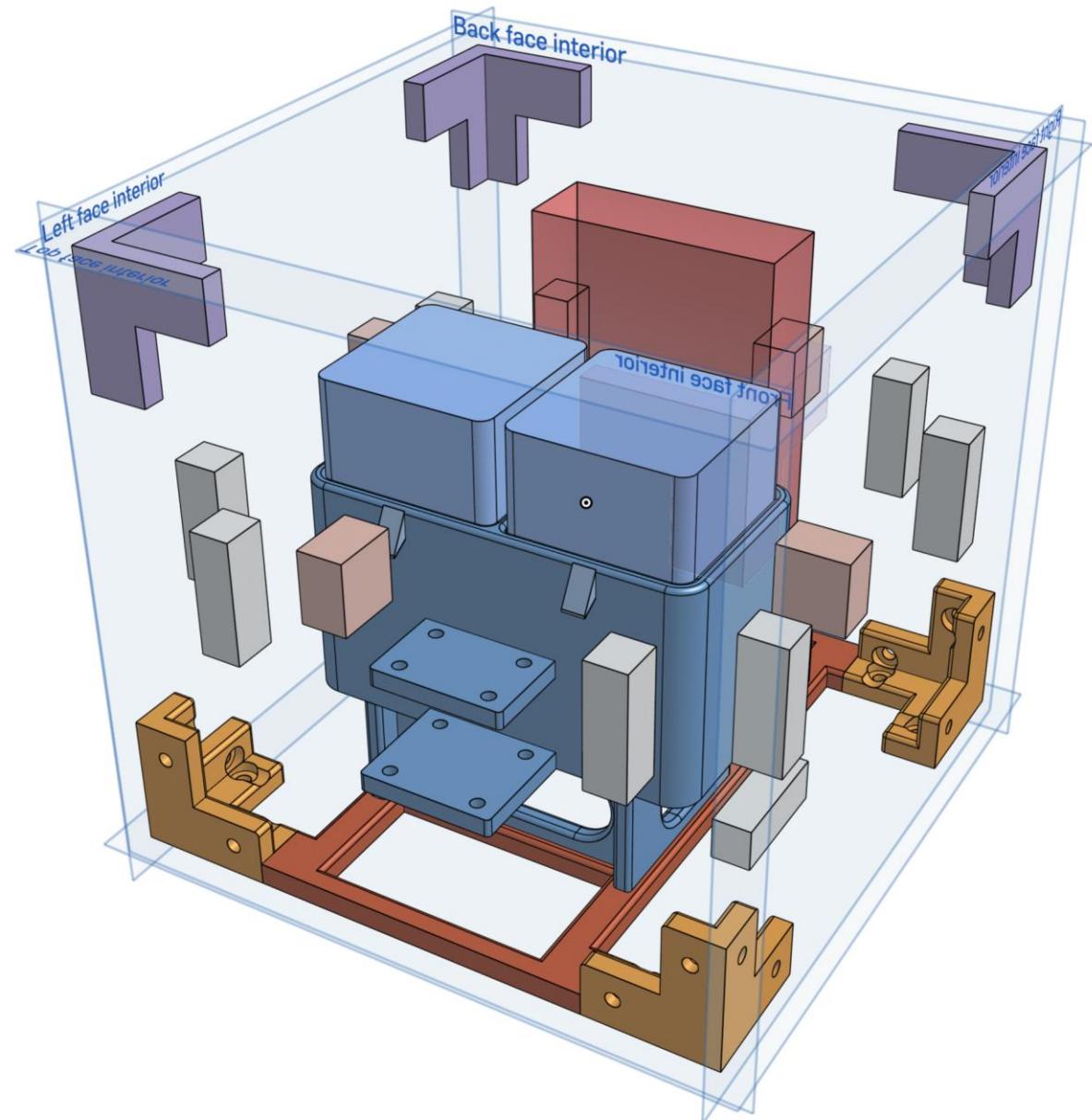
# Assemblage mécanique

Support pour  
les batteries

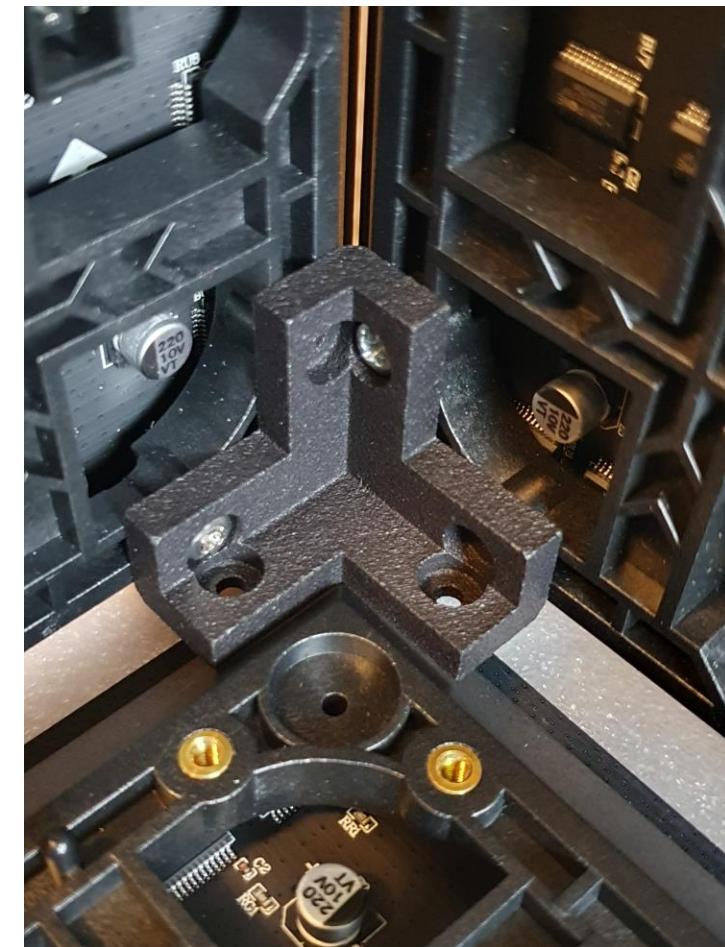
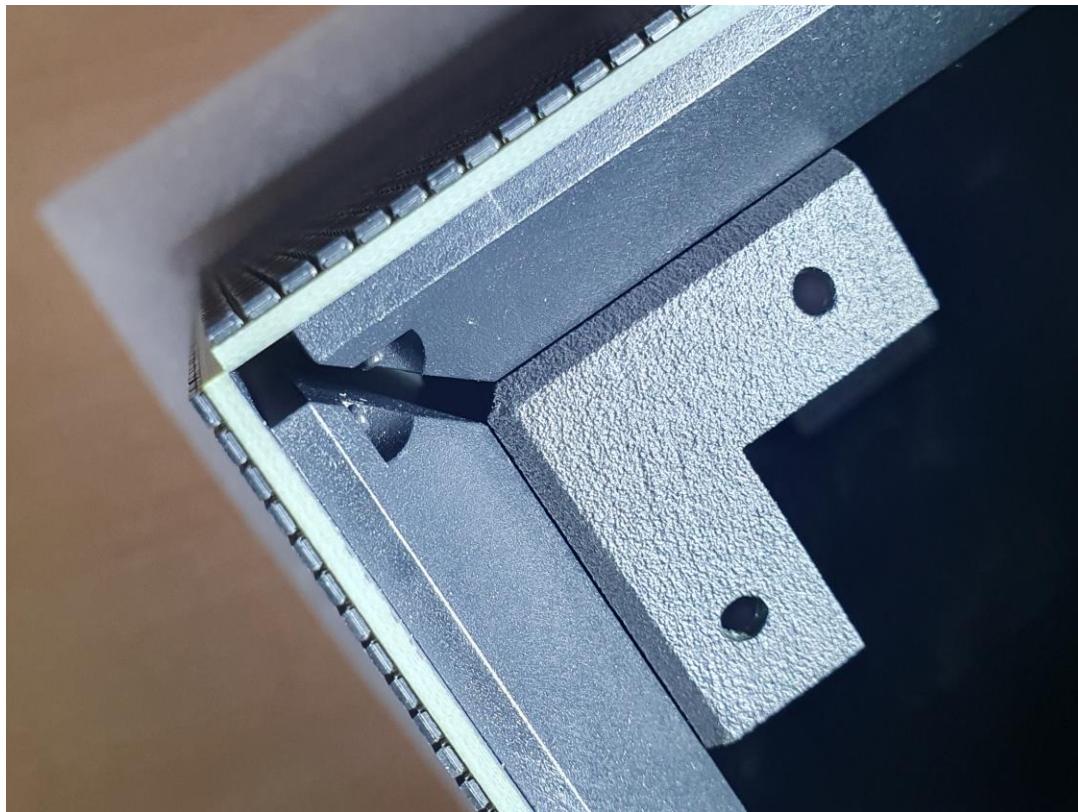


# Assemblage mécanique

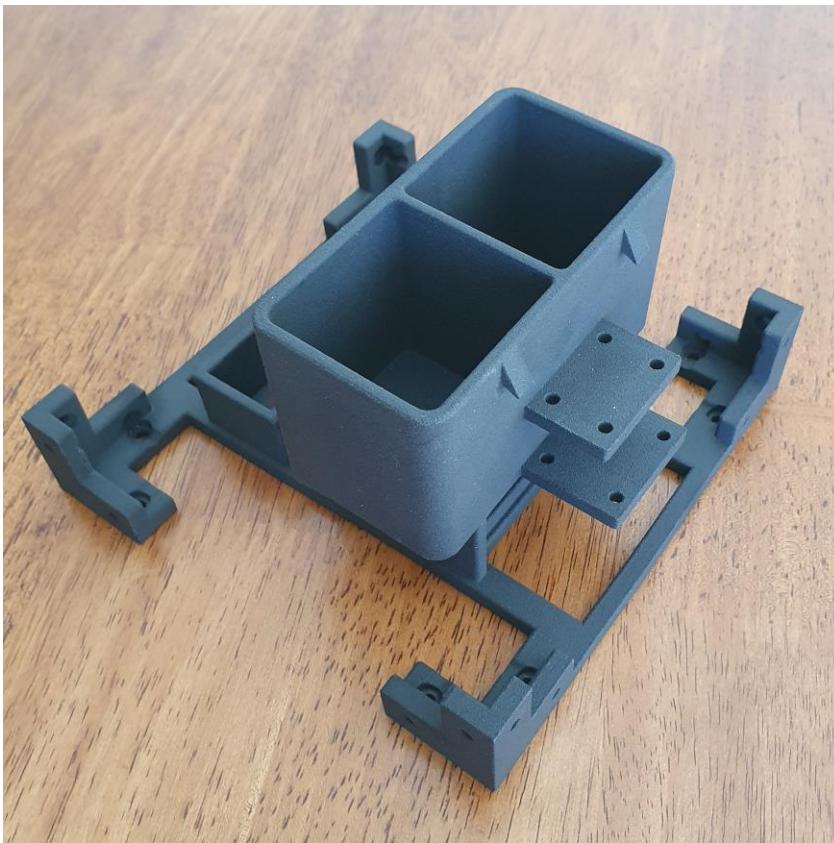
## Connecteurs sur les panneaux



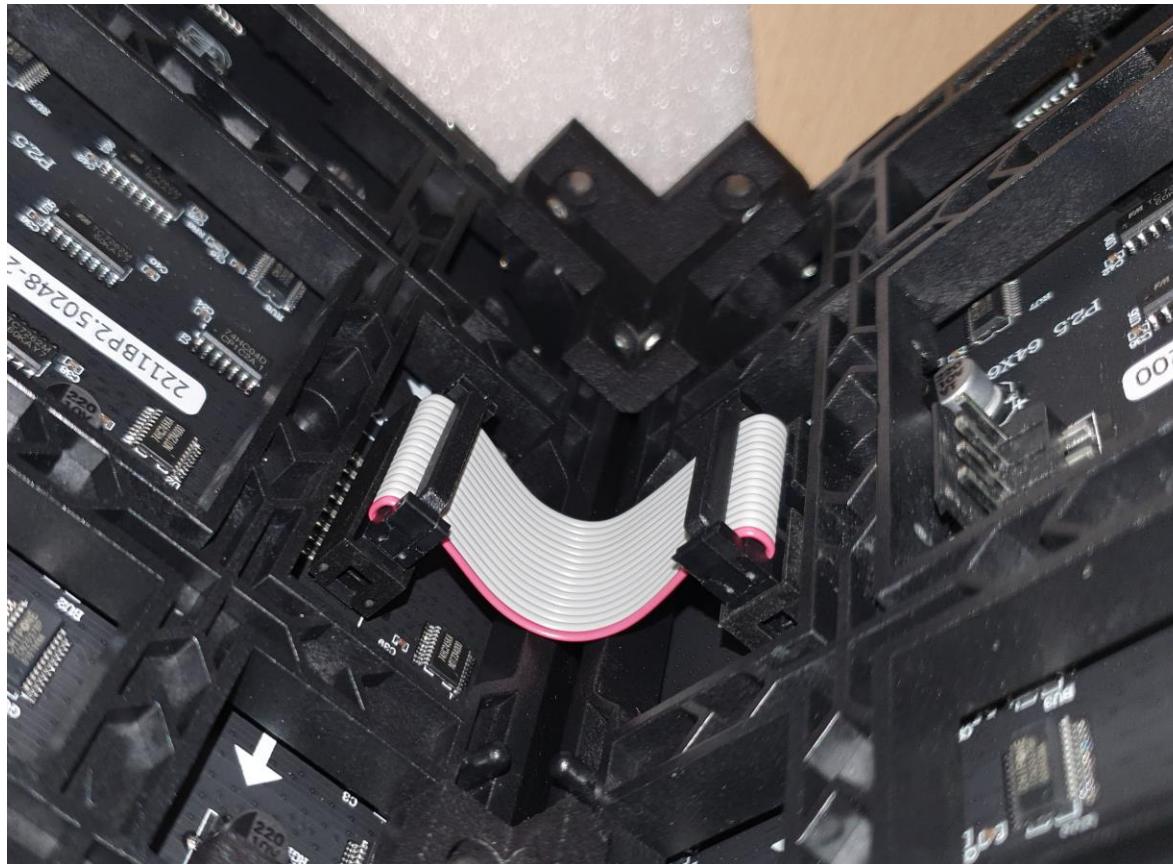
# L'assemblage mécanique



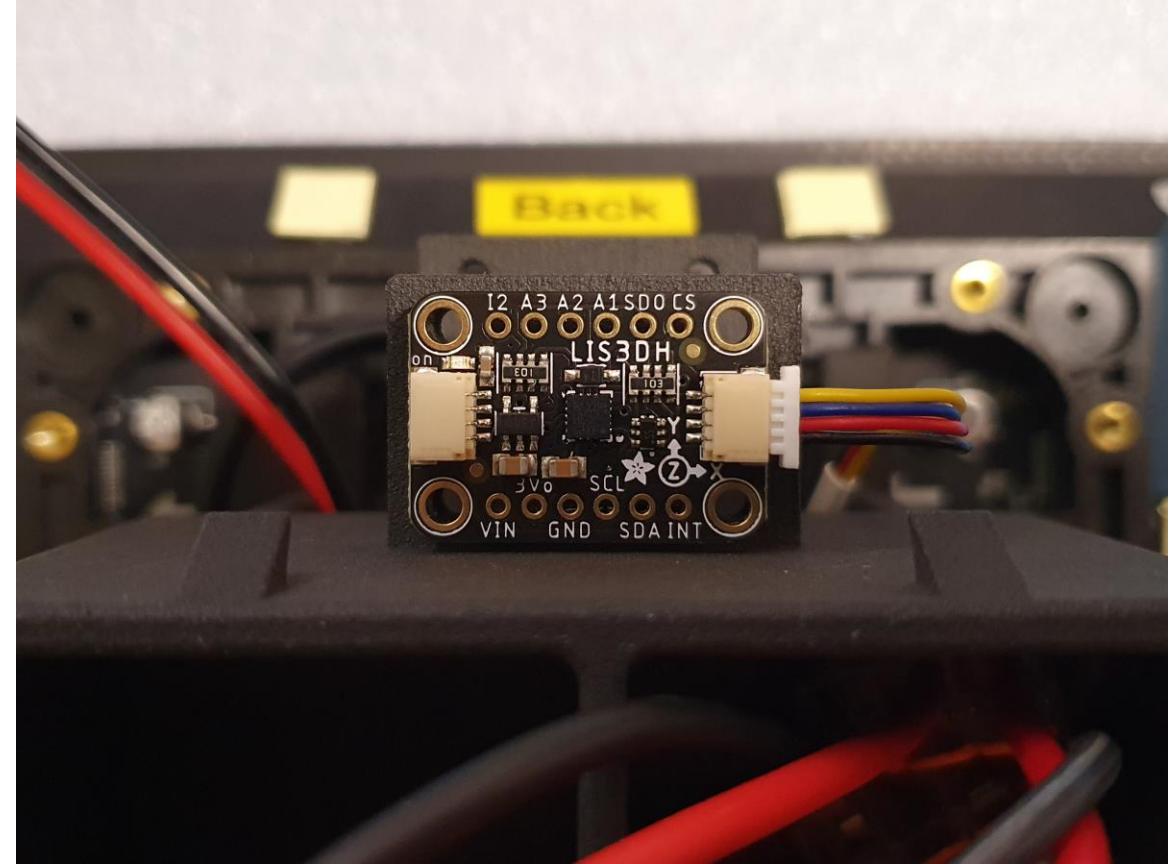
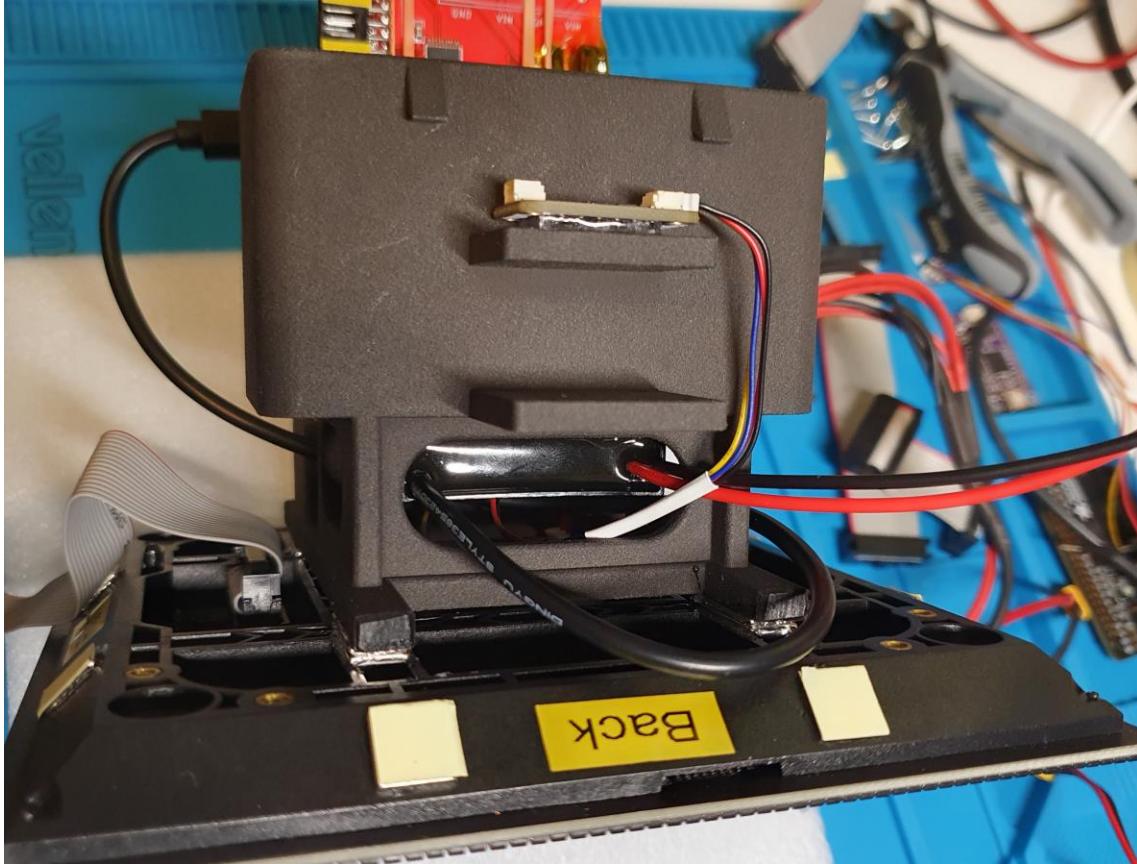
# Support pour le Raspberry et les batteries



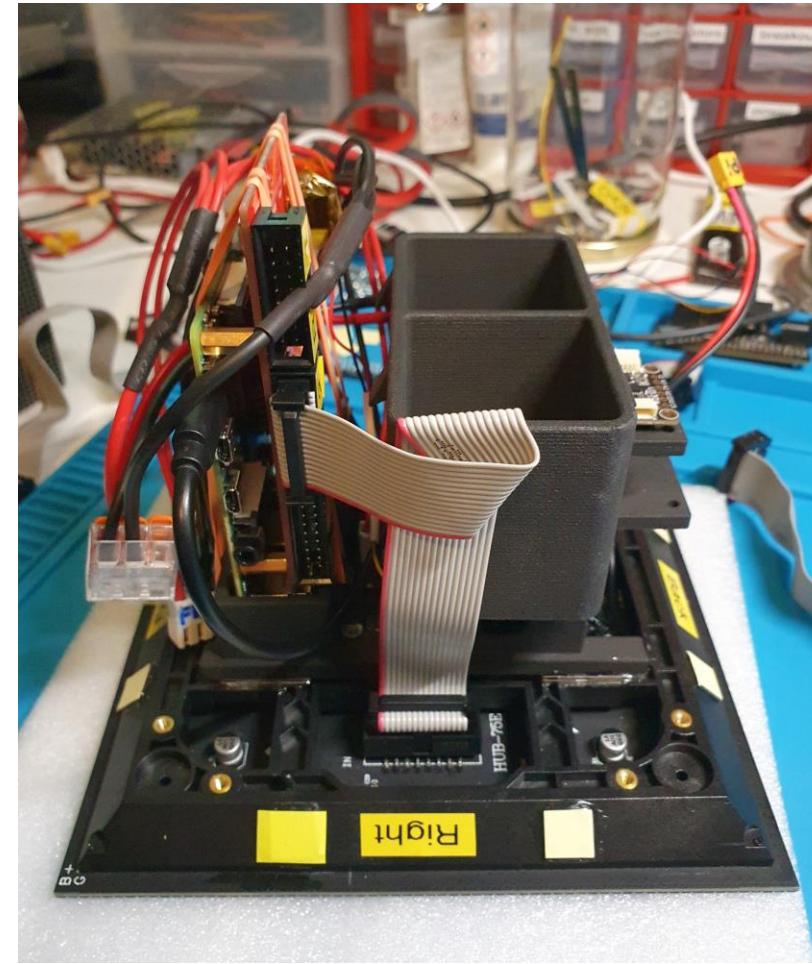
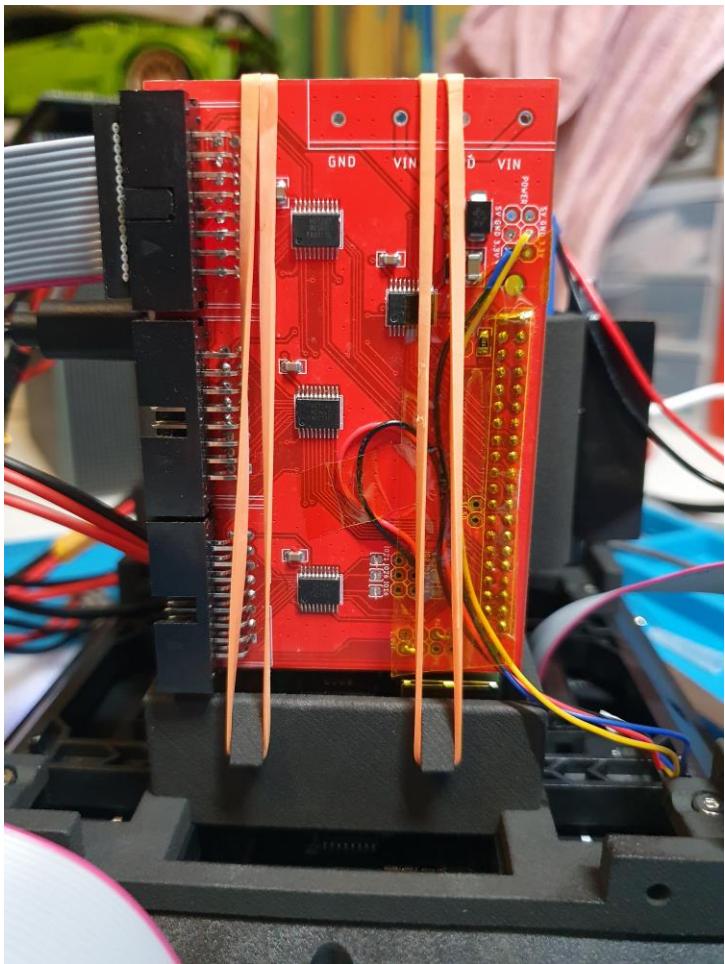
# Liaison électrique entre les panneaux



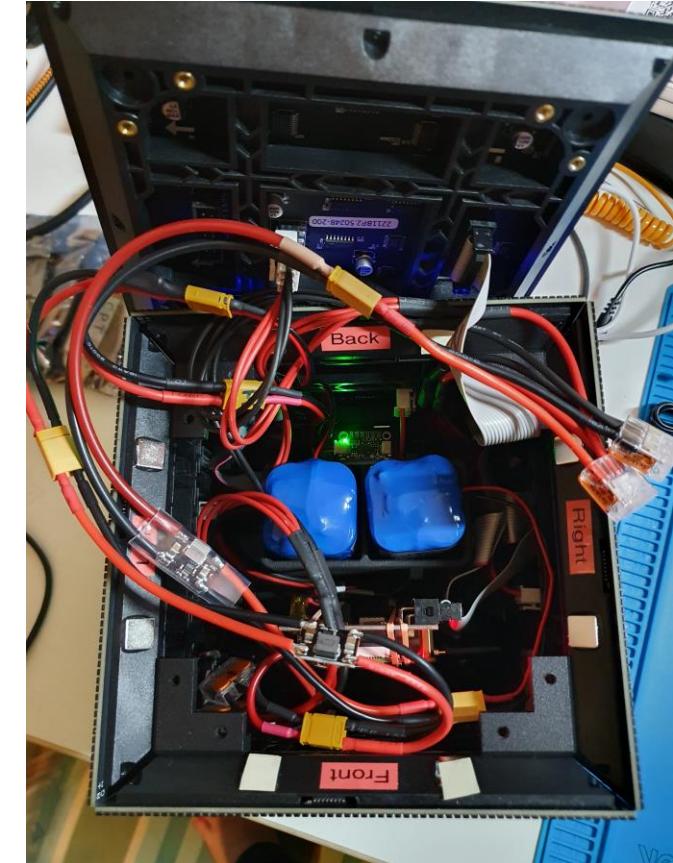
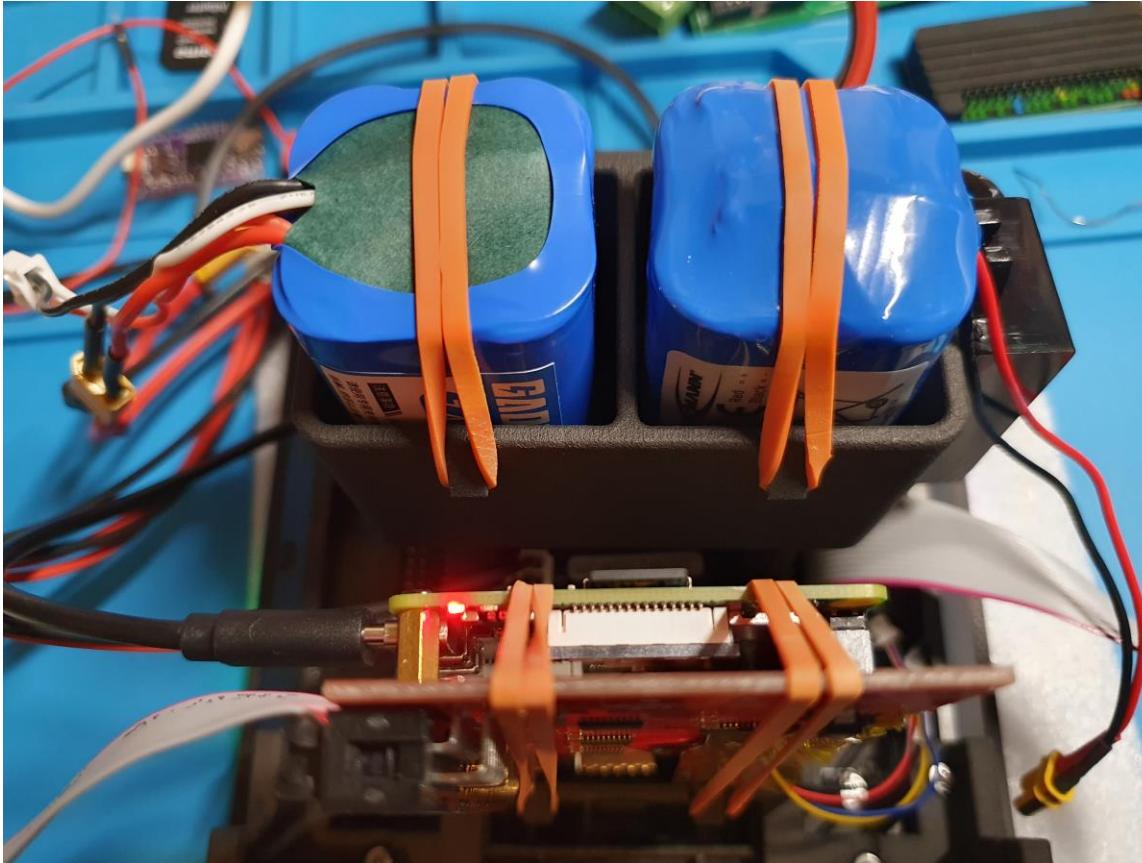
# Convertisseur DC-DC et accéléromètre



# Interface HUB75 et Raspberry



# Batteries et installation finale



# L'électronique

Batteries : Lithium-Ion 7.4V 5200mAh

Raspberry Pi : 5V  $\pm 5\%$

Panneaux LED : 4.5..5V

Max 4A par convertisseur DC-DC

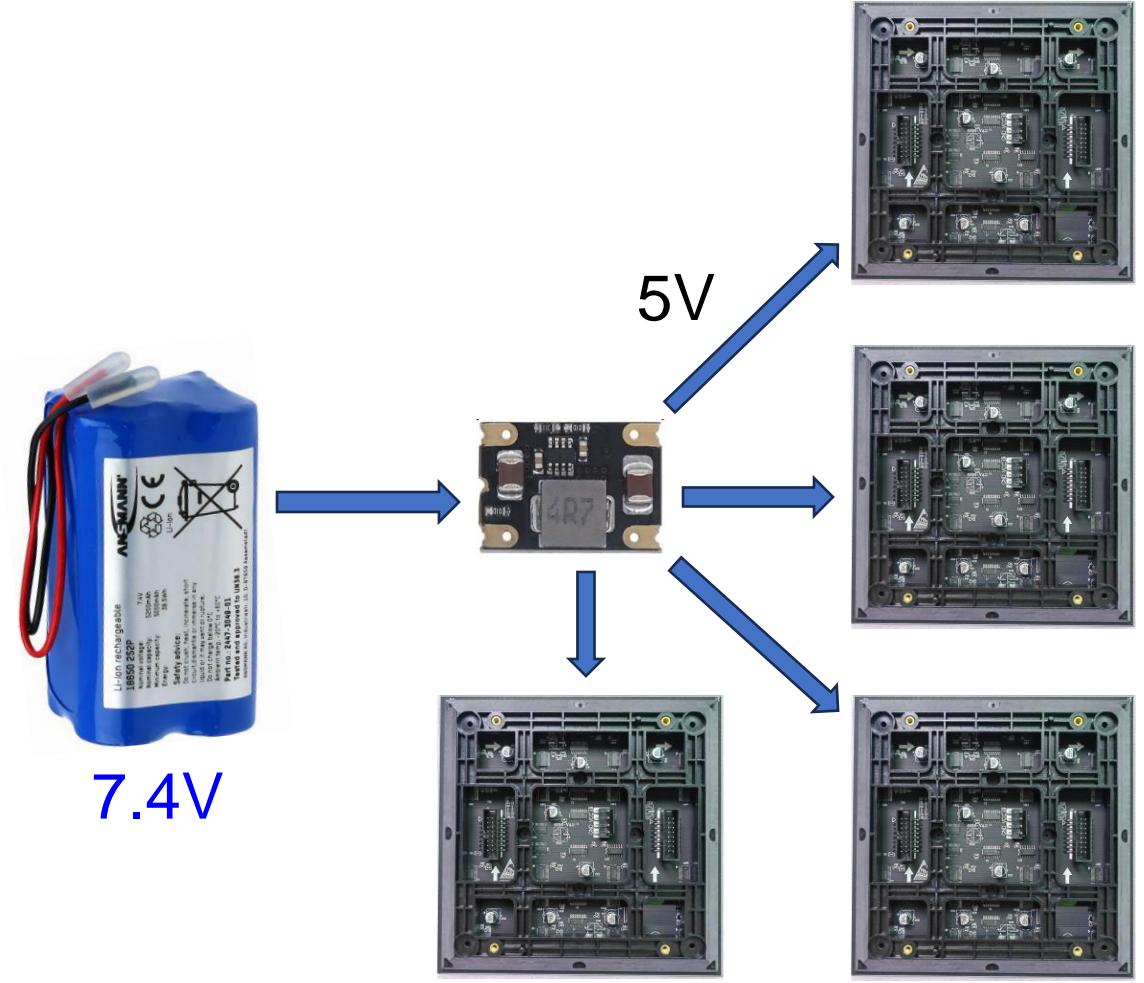
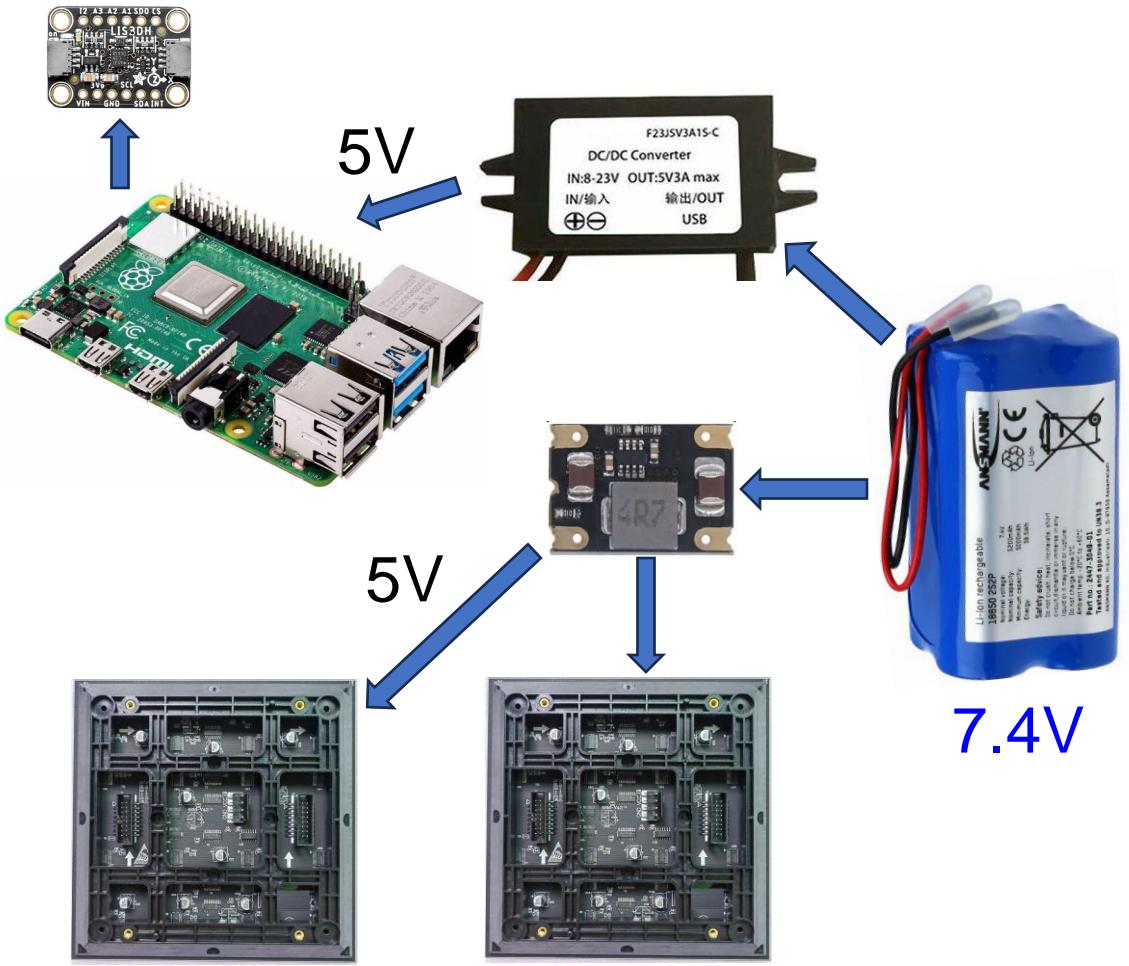
Batterie no 1:

- 1 convertisseur DC-DC pour alimenter le RPi et l'accéléromètre
- 1 convertisseur DC-DC pour alimenter 2 panneaux

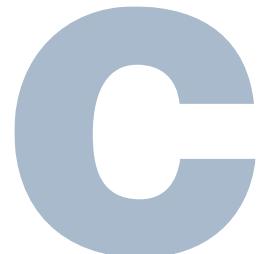
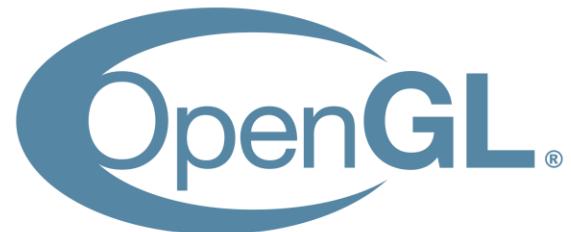
Batterie no 2:

- 1 convertisseur DC-DC pour alimenter 4 panneaux

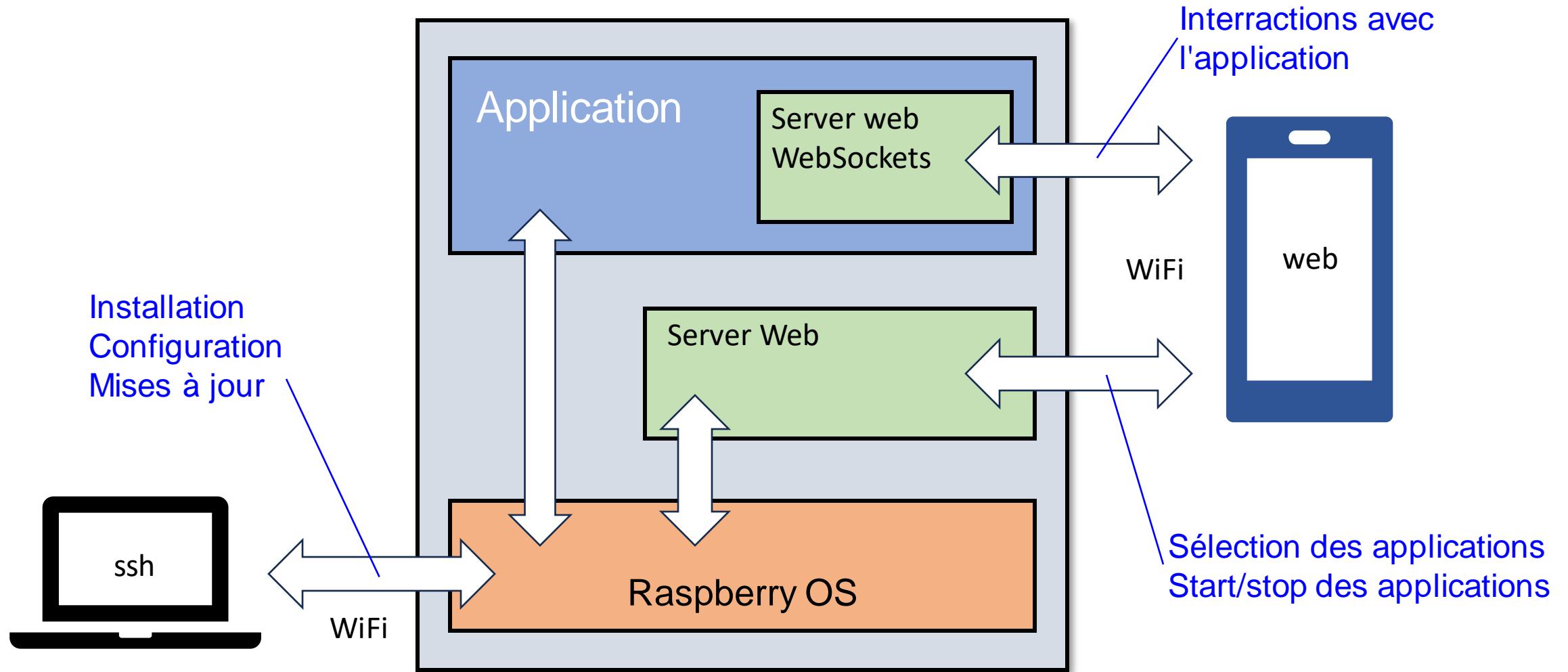
# L'alimentation en énergie



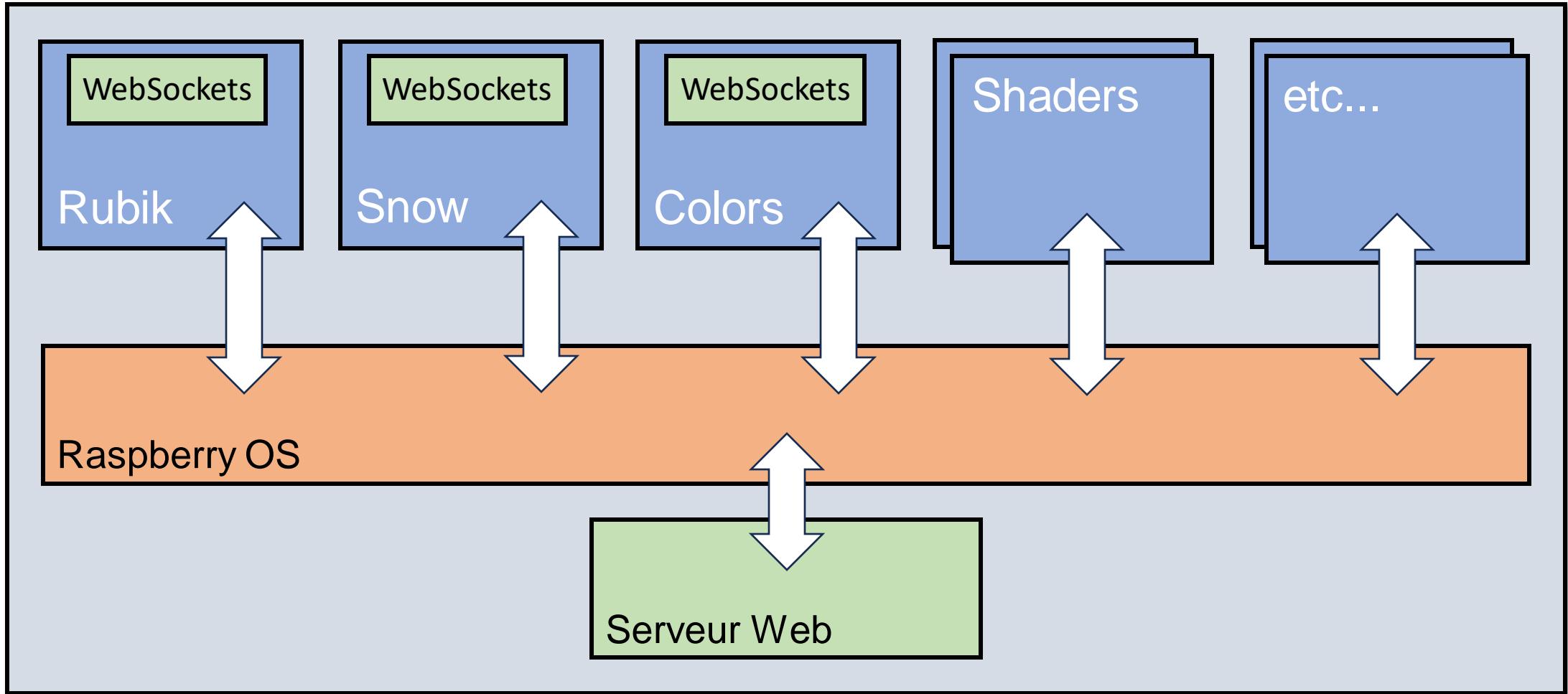
# Le logiciel – langages utilisés



# Le logiciel – une application

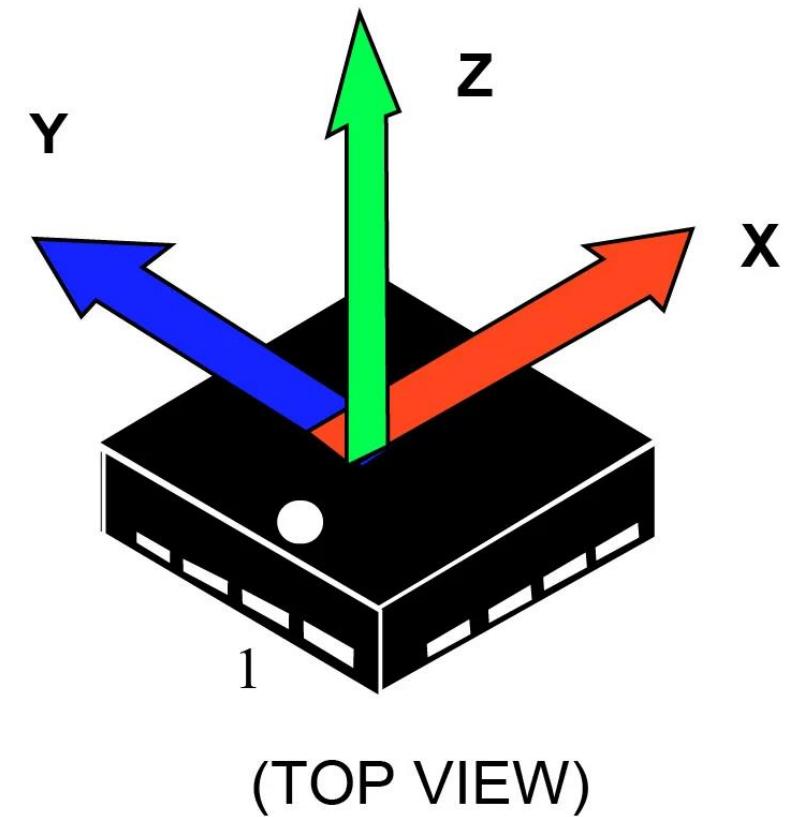
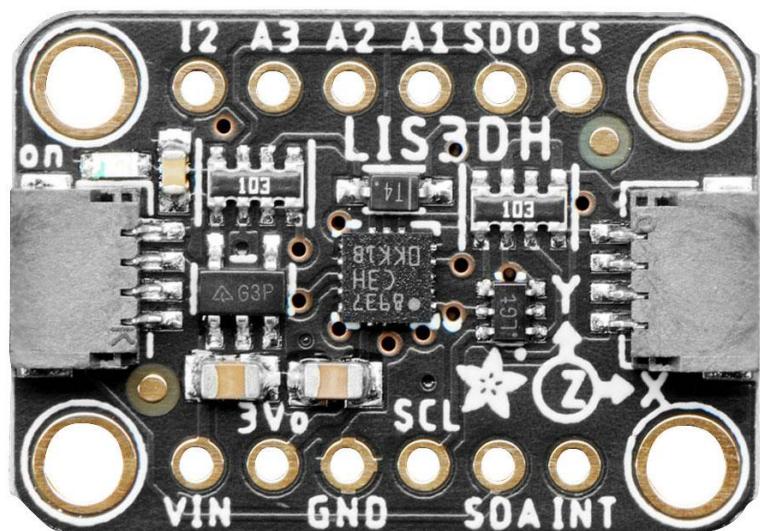


# Le logiciel – serveurs et applications



# L'accéléromètre 3D

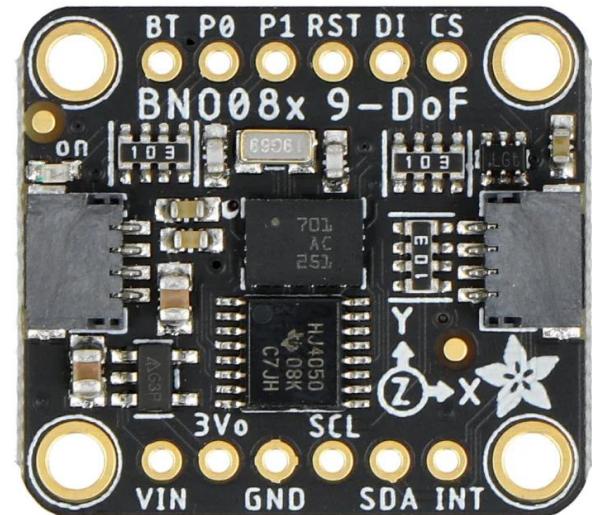
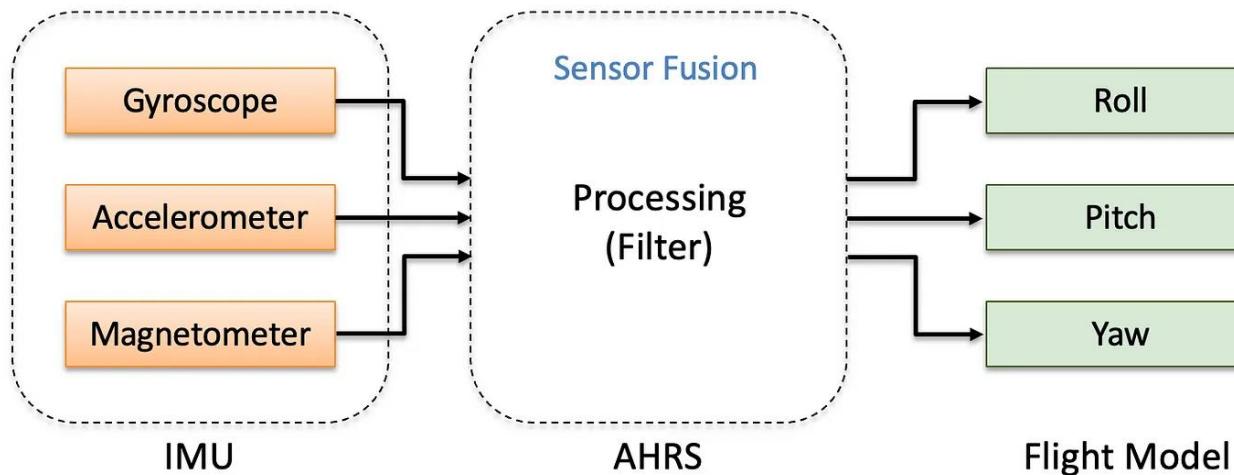
Accélération linéaire sur 3 axes X Y Z  
jusqu'à +/- 16G.



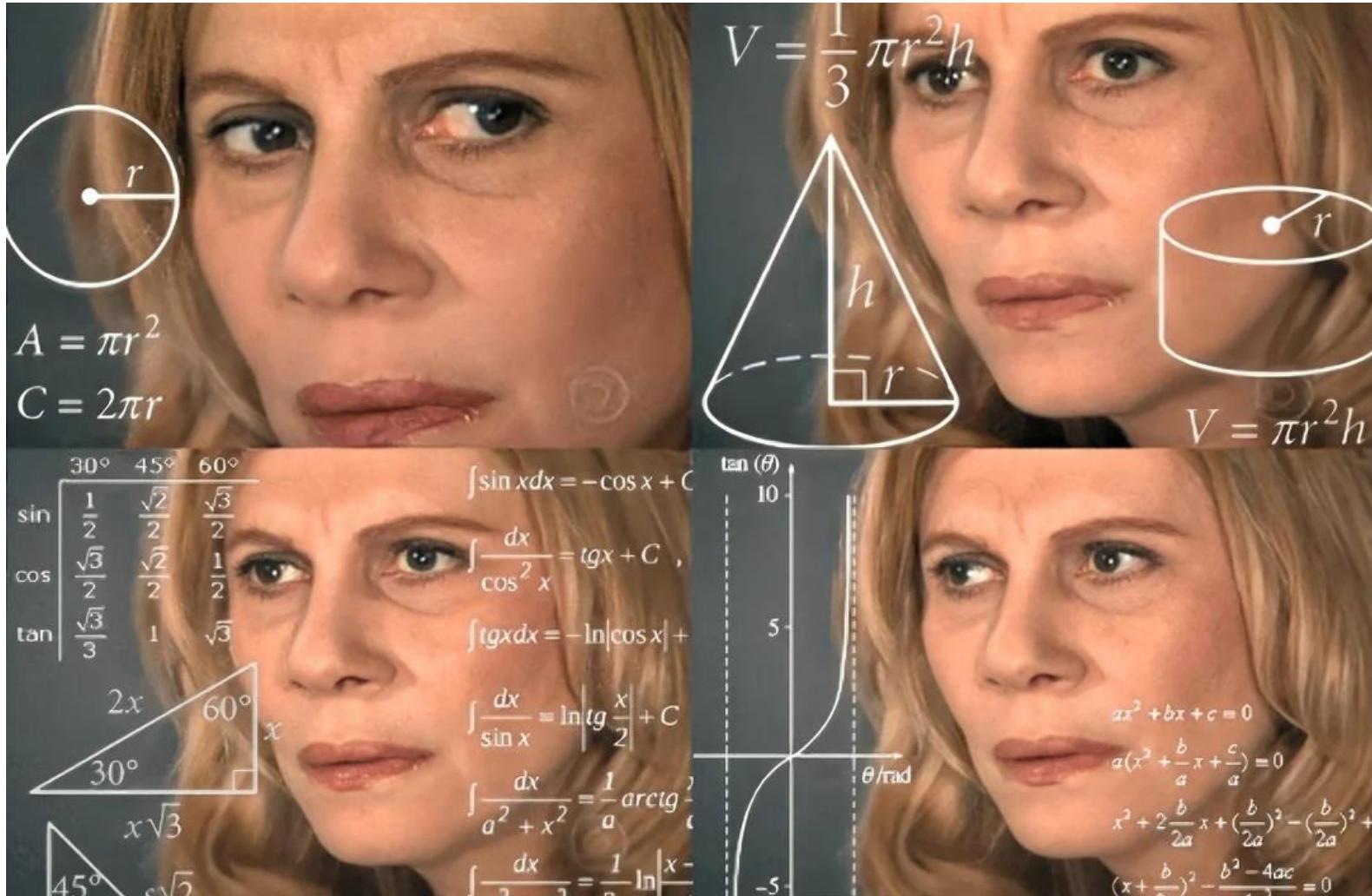
# L'accéléromètre 3D

Futur : 9 degrés

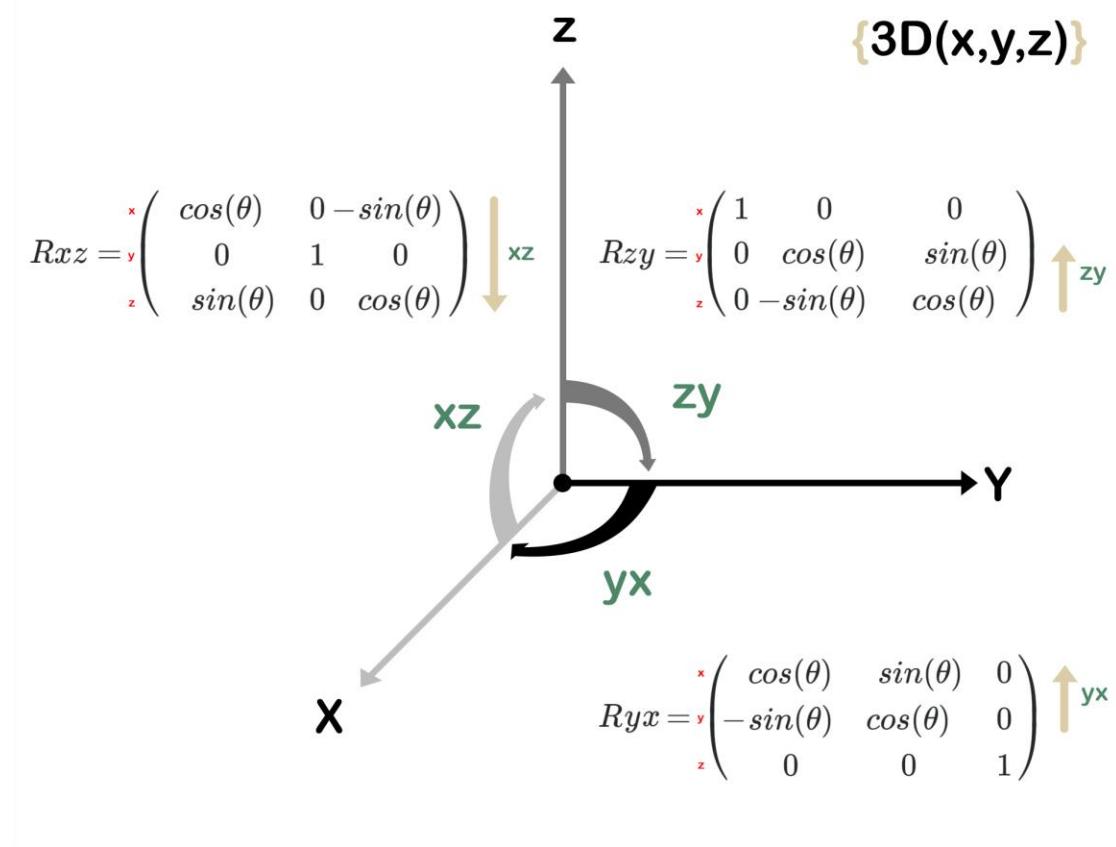
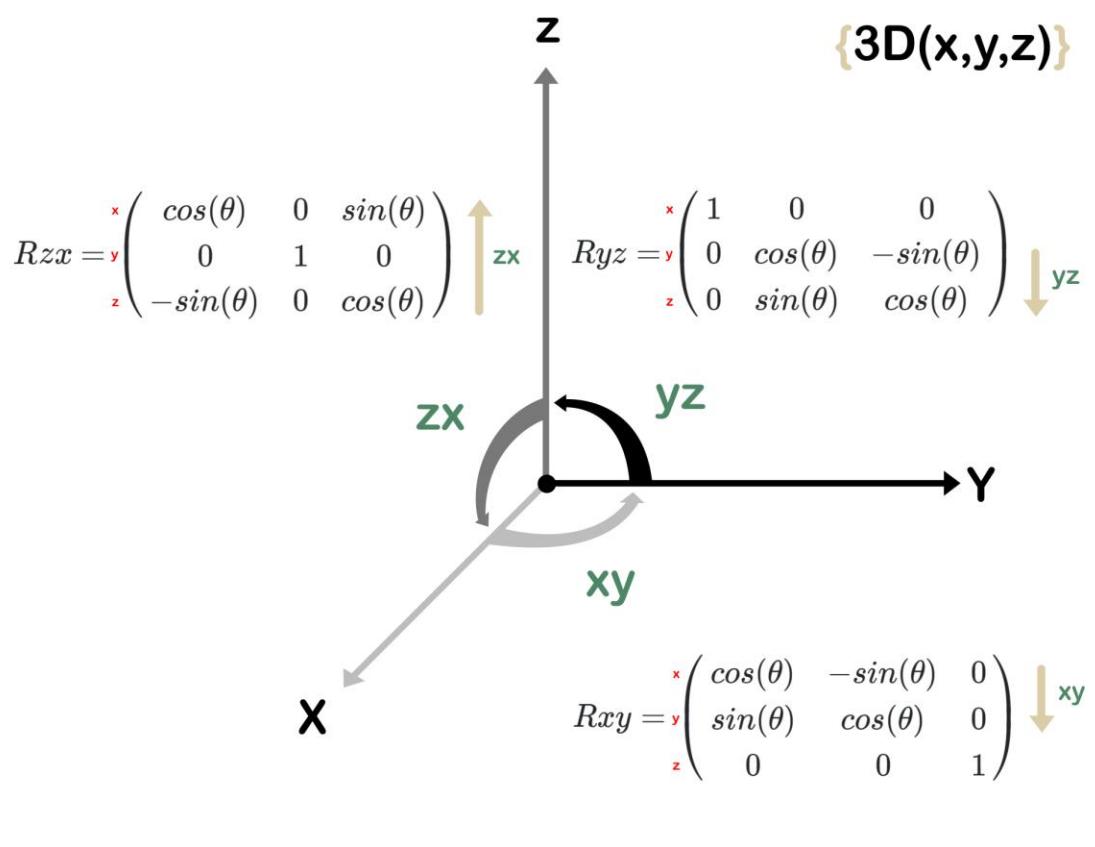
- **Accéléromètre** : gravité XYZ
- **Gyroscope** : vitesse de rotation XYZ
- **Magnétomètre** : champ magnétique XYZ



# Les maths en 3D



# Les maths en 3D



# Les maths en 3D

```
a, b, c = imu.acceleration
x, y, z = normalize((a, b, c))
pitch = atan2(-x, sqrt(y**2 + z**2))
roll = atan2(y, z)
yaw = 0.0

mat3 mz = mat3(
    cos(rz), sin(rz), 0.0,
    -sin(rz), cos(rz), 0.0,
    0.0, 0.0, 1.0 );
mat3 rotation = mx*my*mz*mat3(accel);
```

```
def get_orthonormal_vectors(z):
    u = (1.0, 0.0, 0.0)
    if abs(dot_product(u, z)) < 0.0001:
        u = (0.0, 1.0, 0.0)
    x = normalize(cross_product(z, u))
    y = normalize(cross_product(z, x))
    return x, y

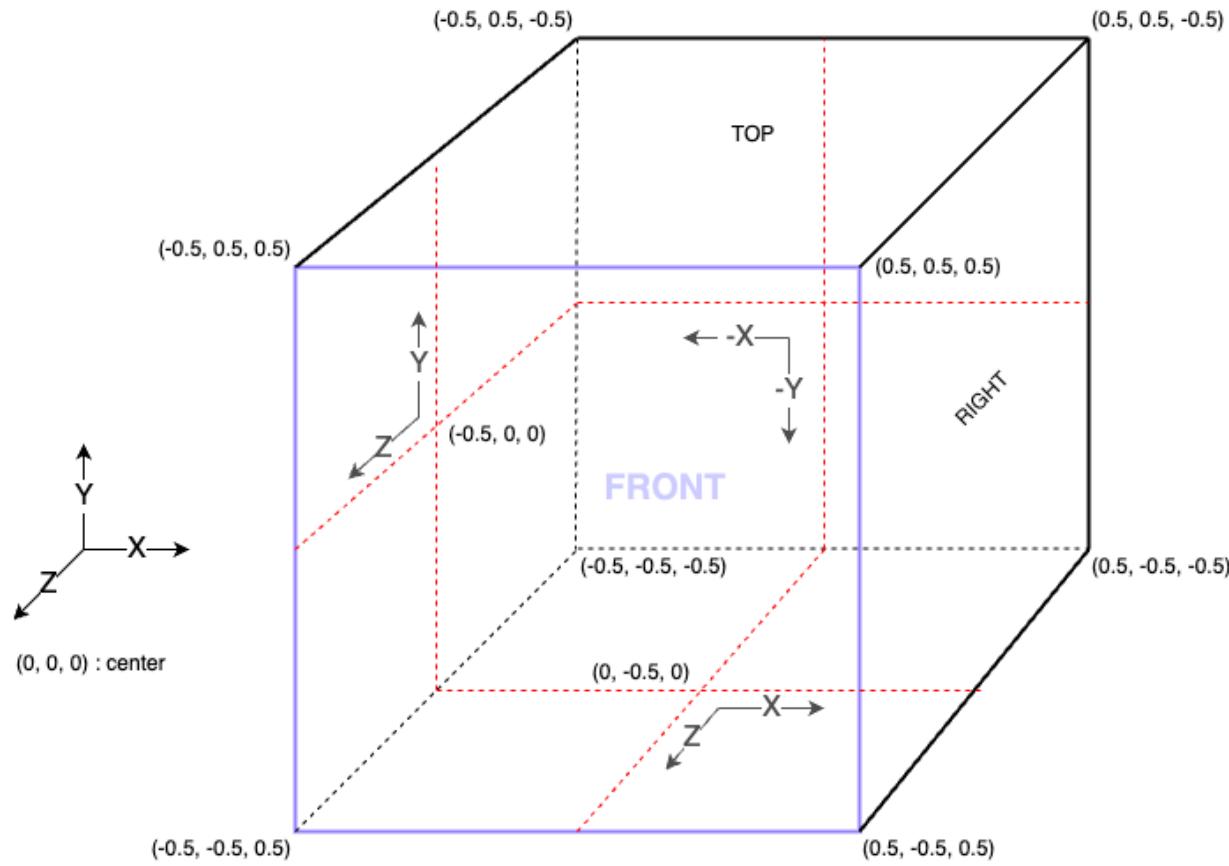
x, y, z = normalize(imu.acceleration)
vx, vy = get_orthonormal_vectors((x, y, z))
draw_plane((x, y, z))
draw_plane(vx)
draw_plane(vy)
```

# Le système de coordonnées du Cube

- Choix *left-hand, right-hand*
- Choix Z vertical ou Y vertical
- Inversion Y sur les panneaux
- *Mappers*
  - Glsl (glsl)
  - Shaders (python)
  - Cube (C++)
- Cubes en papier

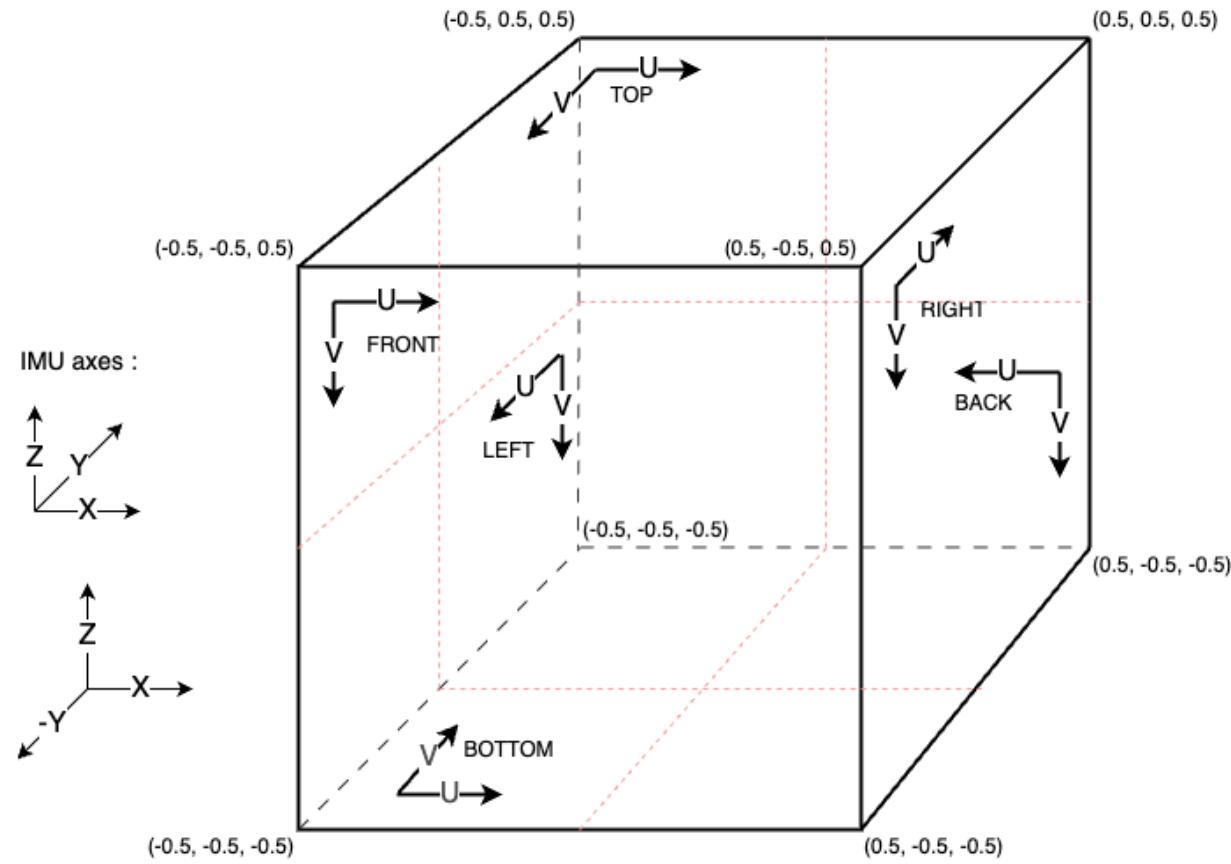
# Le système de coordonnées du Cube

Y vertical

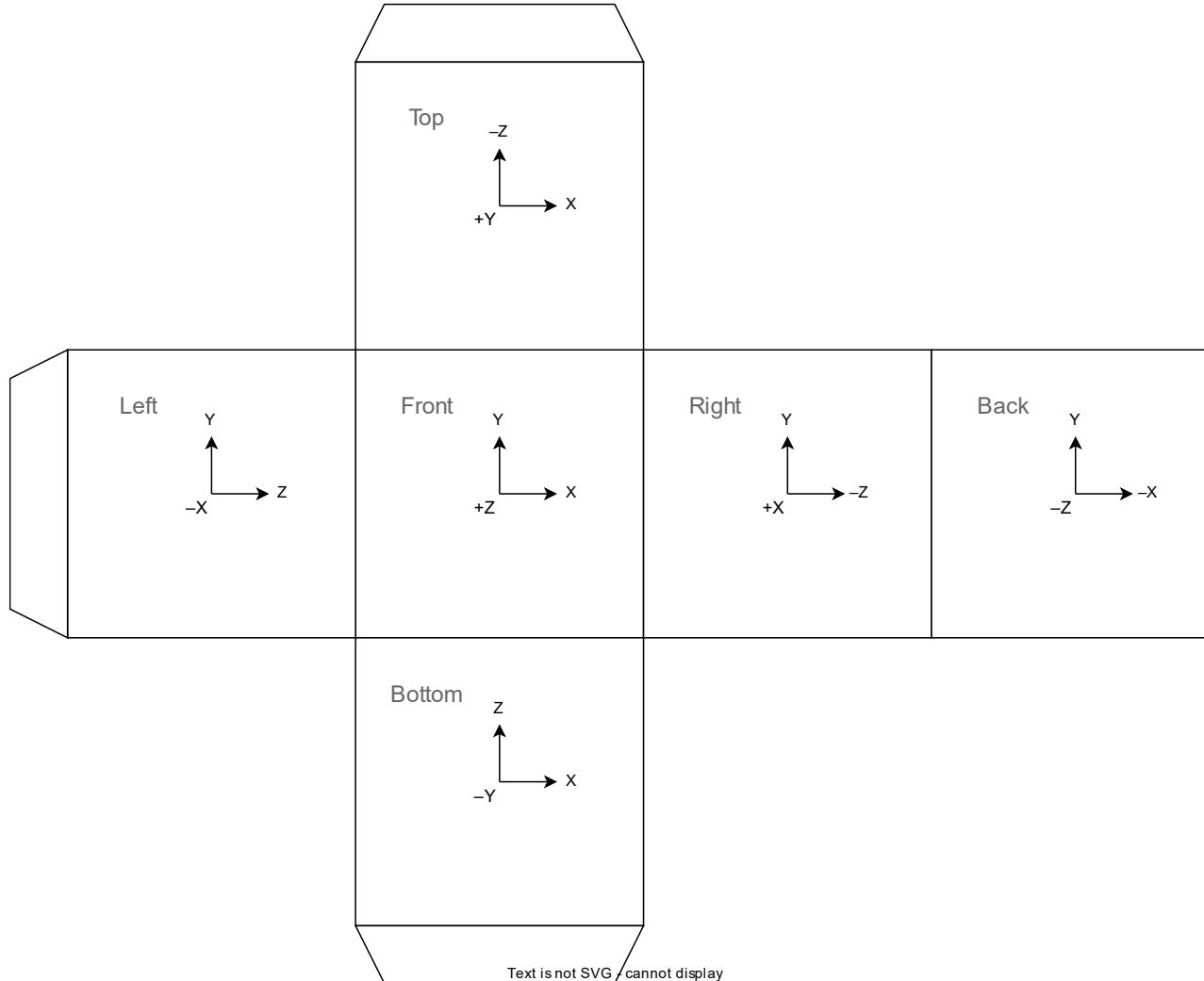


# Le système de coordonnées du Cube

Z vertical



# Le système de coordonnées du Cube



# La programmation

Technique du *double-buffering* :

```
t = 0
while True:
    canvas.clear()
    f = (math.sin(t / 7) + 1) / 2
    g = (math.cos(t / 11) + 1) / 2
    for u in range(64):
        for v in range(64):
            canvas.SetPixel(u, v, u*4*f, v*4*g, 100)
    canvas = matrix.SwapOnVSync(canvas)          # double-buffering
    t = t + 1
```

# La programmation

*Mapping 2D (canvas) vers 3D (cube) :*

```
while True:  
    canvas.Clear()  
    for v in range(canvas.height):  
        for u in range(canvas.width):  
            x, y, z = canvas_to_3d(u, v)      # mapping 2D --> 3D  
            r = x + 0.5  
            g = y + 0.5  
            b = z + 0.5  
            canvas.setPixel(u, v, r*255, g*255, b*255)  
    canvas = matrix.SwapOnVSync(canvas)
```

# Animations avec des Shaders

$$\text{couleur} = f(\text{position}, \text{temps})$$

$$(r, g, b) = f(x, y, z, t)$$

Les prochaines versions intégreront aussi les données de l'accéléromètre.

# Animations avec des Shaders

```
/* RGB */
void mainCube(out vec4 fragColor, in vec3 fragCoord) {
    fragColor.rgb = fragCoord.rgb + .5;
}

/* plasma */
const float speed = 0.1;
void mainCube(out vec4 fragColor, in vec3 fragCoord) {
    float h = mod(perlin_noise(fragCoord * 3 + iTIME*speed) + iTIME*speed, 1);
    fragColor = vec4(hsvToRGB(vec3(h, 1, 1)), 1.0);
}
```

# Jouons au Rubik's Cube

- Algorithme : <https://github.com/hkociemba/RubiksCube-TwophaseSolver>
- *[...] sufficiently fast to solve random cubes in less than 20 moves on average on slow hardware like the Raspberry Pi3 within a few seconds.*

# Il neige des pixels

- Basé sur le code de Adafruit  
PixelDust [https://github.com/adafruit/Adafruit\\_PixelDust](https://github.com/adafruit/Adafruit_PixelDust)
- Encore en développement
- A faire si possible : trier les pixels par "hauteur" pour avoir une animation plus réaliste
- Performances sensibles aux perturbations

# Outils pour le développement

- Simulateur de matrice LED
  - simulateur de Cube
- glslEditor, glslViewer
  - <http://editor.thebookofshaders.com/>
- Jupyter
  - <https://jupyter.org/>
- ChatGPT ?

# Les problèmes rencontrés

- Assemblage mécanique
  - Dimensions exactes des panneaux non fournies par le vendeur.
  - Peu de place pour l'utilisation d'outils.
- Impression 3D
- Alimentation électrique
- Autonomie
- Performances
- Géométrie 3D
- Prendre en photo ou filmer le cube sans artefacts

# Et l'I.A. dans tout ça?

- ChatGPT 4
- Conversion de code Python vers C++
  - Remarquablement bon.
  - Code utilisable presque immédiatement.
- Aide pour les rotations dans l'espace
  - Une aide pratique pour se rafraîchir la mémoire.
  - En comparant avec des ouvrages de références, toutes les formules données par ChatGPT se sont révélées exactes.

# Prochaines étapes

Reste à faire indispensable :

- Plans pour impression 3D
- Surveillance de la tension des batteries
- Bluetooth comme alternative au WiFi
- Installation des apps depuis le web

Presque indispensable :

- Support pour le *Live Coding*

Améliorations :

- IMU 9 axes
- Son (capture et génération)
- Support MIDI / OSC
- Assemblage 100% sans vis
- FPGA pour le pilotage des LEDs
- Raspberry Pi 5
- Recharge avec boucle d'induction

# En cours de développement



# Le Cube en chiffres

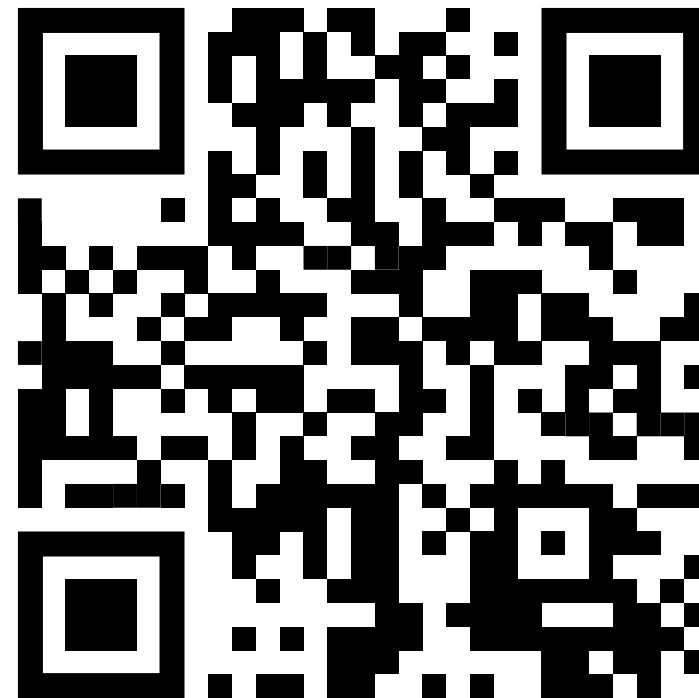
- **24576** : le nombre total de LEDs ( $6 \times 64 \times 64$ )
- **75W** : la puissance maximale consommée
- **3h** : l'autonomie minimale espérée avec 2 batteries
- **45°C** : la température à l'intérieur du cube
- **16 cm** : dimension du cube,  $4096 \text{ cm}^3 \approx 4 \text{ litres}$
- **2.1 kg** : poids total. Dont 2x200g pour les batteries
- **8 kg** : la force de retenue des aimants pour une face
- **550.-** : le coût du matériel (CHF)
- **>200h** : heures consacrées à la réalisation

# Remerciements

- Pyxis
- <https://github.com/hzeller/rpi-rgb-led-matrix>
- <https://github.com/ty-porter/RGBMatrixEmulator>
- <https://github.com/hkociemba/RubiksCube-TwophaseSolver>
- <https://github.com/kbob/shaderboy>
- <https://learn.adafruit.com/>

# Open Source

<https://github.com/francoisgeorgy/led-cube>



# Le futur reste à inventer

Et vous, qu'aimeriez-vous faire avec  
le LED Cube ?

Venez partager vos idées lors des  
prochains ateliers de programmation  
qui auront lieux ici-même dans  
quelques semaines.

# Démonstrations !