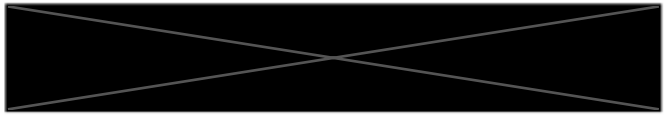
TRAITEMENT D'IMAGES SPORTIVES ET MÉCATRONIQUE AU SERVICE DE LA CÉCITÉ



TRAITEMENT D'IMAGES SPORTIVES ET MÉCATRONIQUE AU SERVICE DE LA CÉCITÉ

Accessibilité des matches pour les personnes malvoyantes

Audiodescription pour les spectateurs malvoyants (Coupe du Monde de rugby, 2023)
© Ilian Valet

Touch2See
© Touch2See

TRAITEMENT D'IMAGES SPORTIVES ET MÉCATRONIQUE AU SERVICE DE LA CÉCITÉ

Etude de la position d'une balle

Balle équipée d'une centrale inertielle (Coupe du Monde de football, 2022)

© Adidas

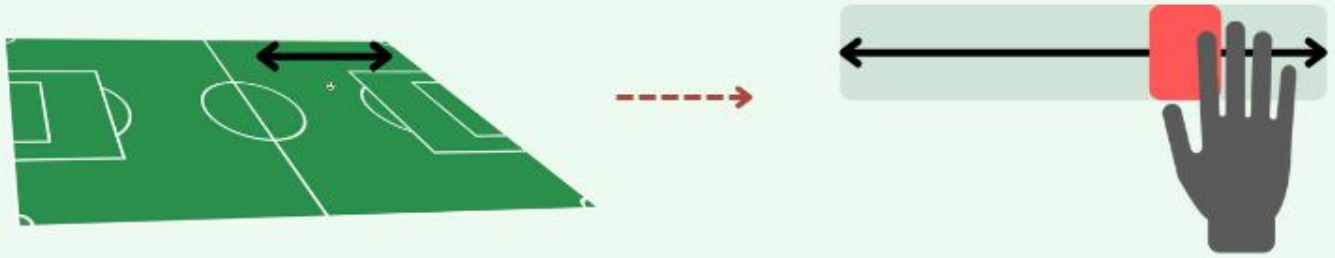
Caméras autour d'un terrain de football (Finale de la Ligue des Champions, 2023)

© Dan Chung

Enjeux :

— Etude d'un système visant à :

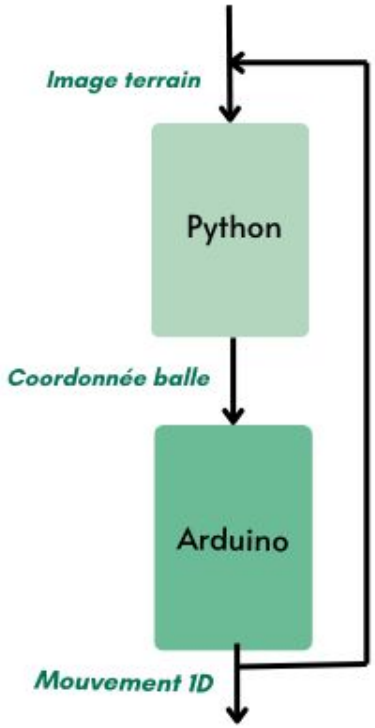
- Recueillir les positions de la balle sur un terrain de football à partir d'images
- Transmettre en temps réel l'information par le toucher



— Évaluation des performances

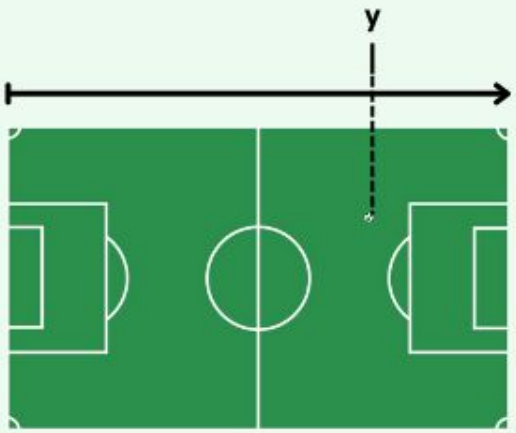
— Détermination de la validité d'une approche à une caméra

SOMMAIRE



I. TRAITEMENT DE L'INFORMATION VISUELLE

Python



Entrée :
vidéo = succession d'images

Sortie :
coordonnée y du ballon

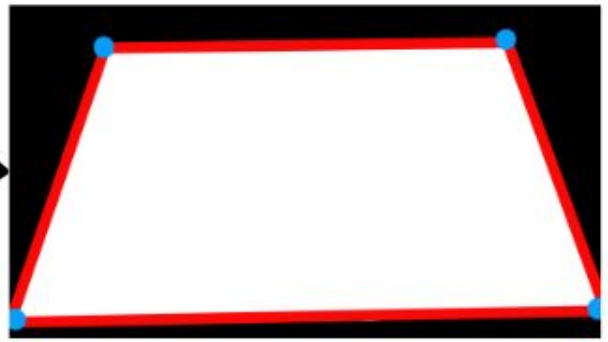
Hypothèses :

- (H1) Ballon au sol
- (H2) ∅ distorsion image
- (H3) 4 coins terrain filmés

A. PRINCIPE DE L'ALGORITHME

Image caméra

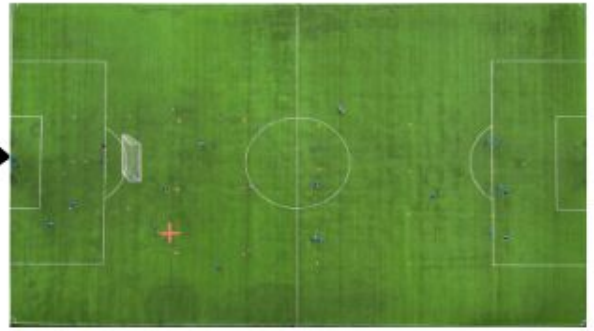
Segmentation couleur
Filtrage bruit



Contours, coins

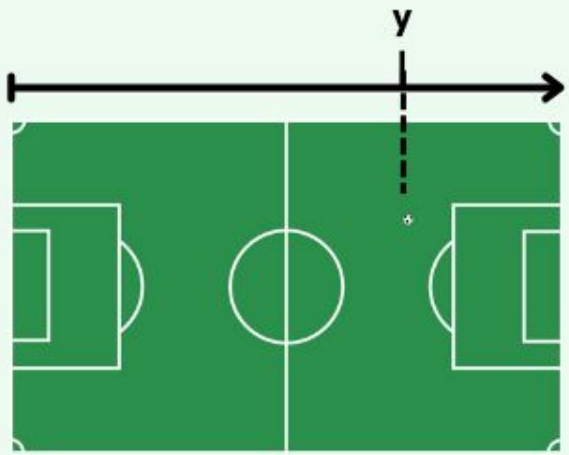
Matrice projection

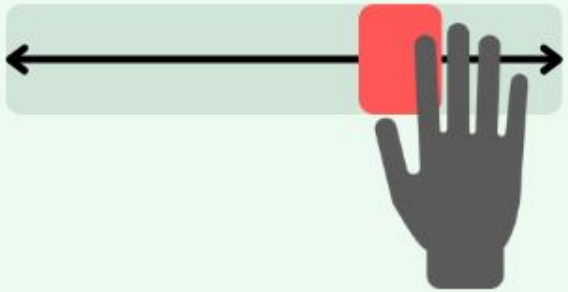
Coordonnées balle



Projection
Coordonnées balle corrigées

II. TRANSMISSION DE L'INFORMATION HAPTIQUE





Entrée : succession coordonnées y (Python)

Sortie : Mouvement translation 1D

Création du mouvement ID

- Moteurs linéaires
- Moteurs rotatifs

Moteurs à courant continu, synchrone, asynchrone

Pas à pas

Servomoteur

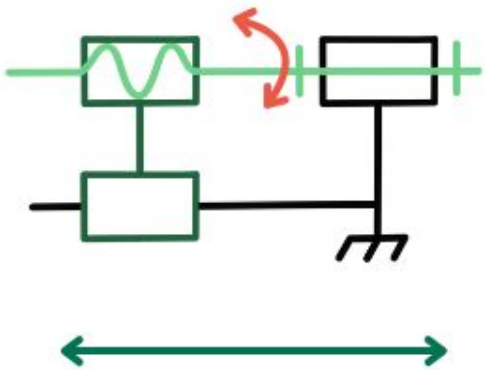
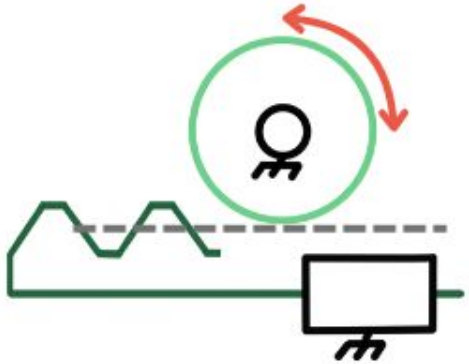
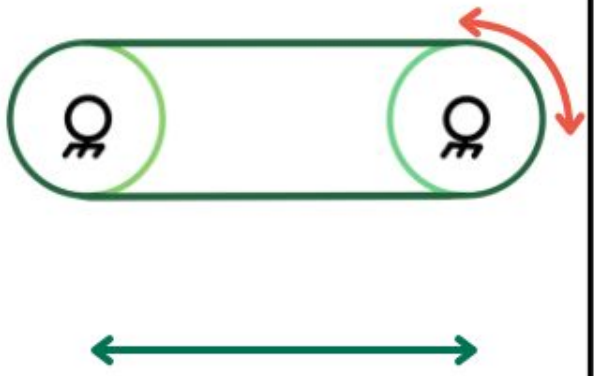
Position

Vitesse

Moteurs rotatifs et usages

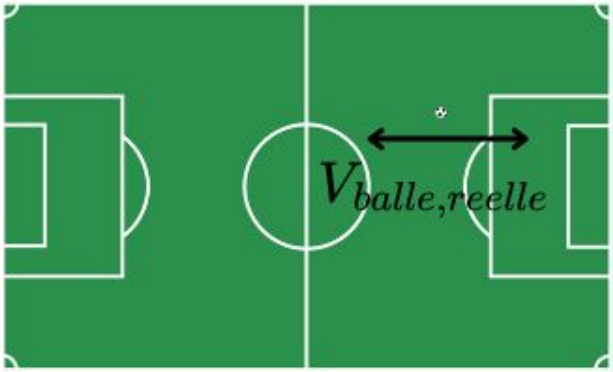
Transformation mouvement : rotation -> translation

Relation linéaire vitesses rotation / translation

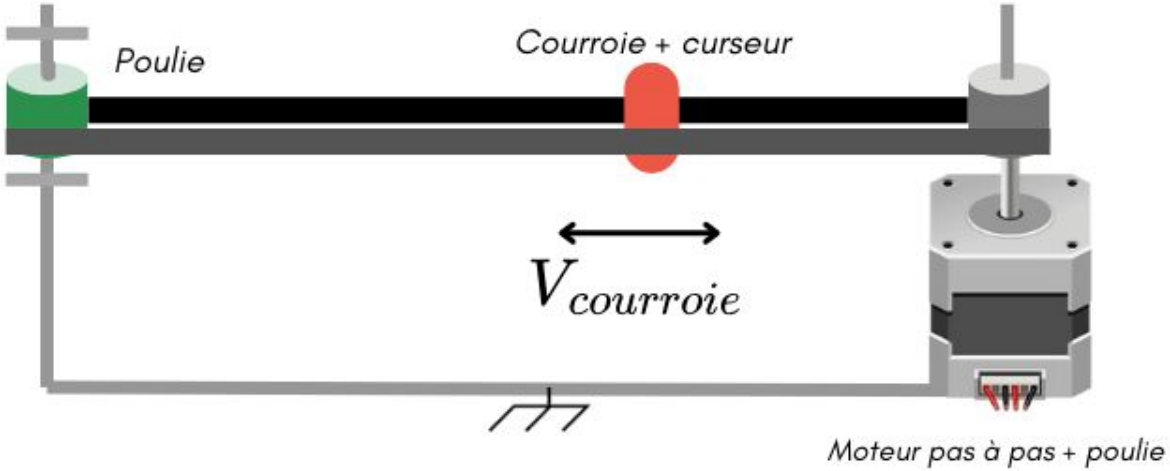
Vis écrou	Pignon crémaillère	**Poulie courroie**
		

- Réversibilité
- Ø usure
- Course ajustable

A. CHOIX DU MÉCANISME



$L_{terrain}$



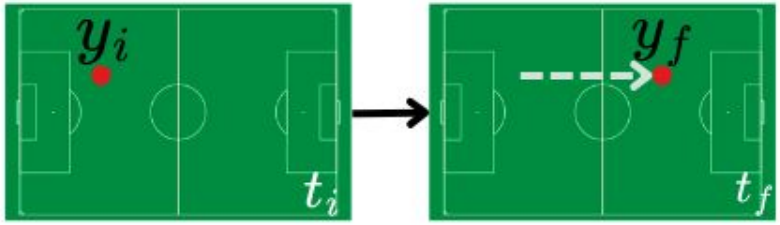
$L_{systeme}$

$$V_{courroie} = k \times V_{balle,reelle} \quad : (1) \qquad \text{où} \quad k = \frac{L_{systeme}}{L_{terrain}}$$

Le système doit vérifier :

$$V_{courroie} = k \times V_{balle,reelle} \quad : (1) \qquad \text{où} \quad k = \frac{L_{systeme}}{L_{terrain}}$$

I - Vitesse de rotation du moteur



$$V_{balle,reelle} = lim_{dt \to 0} \frac{dy_{balle,reelle}}{dt} \approx \frac{y_f - y_i}{t_f - t_i}$$

$$(1) \Leftrightarrow r\omega_{rot,rad} = k\frac{(y_f - y_i)}{(t_f - t_i)} \Leftrightarrow \omega_{rot,rad} = \frac{k\,(y_f - y_i)}{r\,(t_f - t_i)} \quad \text{en rad/s}$$

Arduino : vitesses en tour / min (rpm) :

$$\omega_{rot} = \lambda \frac{k\,(y_f - y_i)}{r\,(t_f - t_i)} \qquad \text{où} \qquad \lambda = \frac{60}{2\pi}$$

2 - Nombre de pas à réaliser

$$\omega_{rot,rad} = lim_{dt \to 0} \frac{d\theta_{poulie}}{dt} \approx \frac{(\theta_f - \theta_i)}{(t_f - t_i)}$$

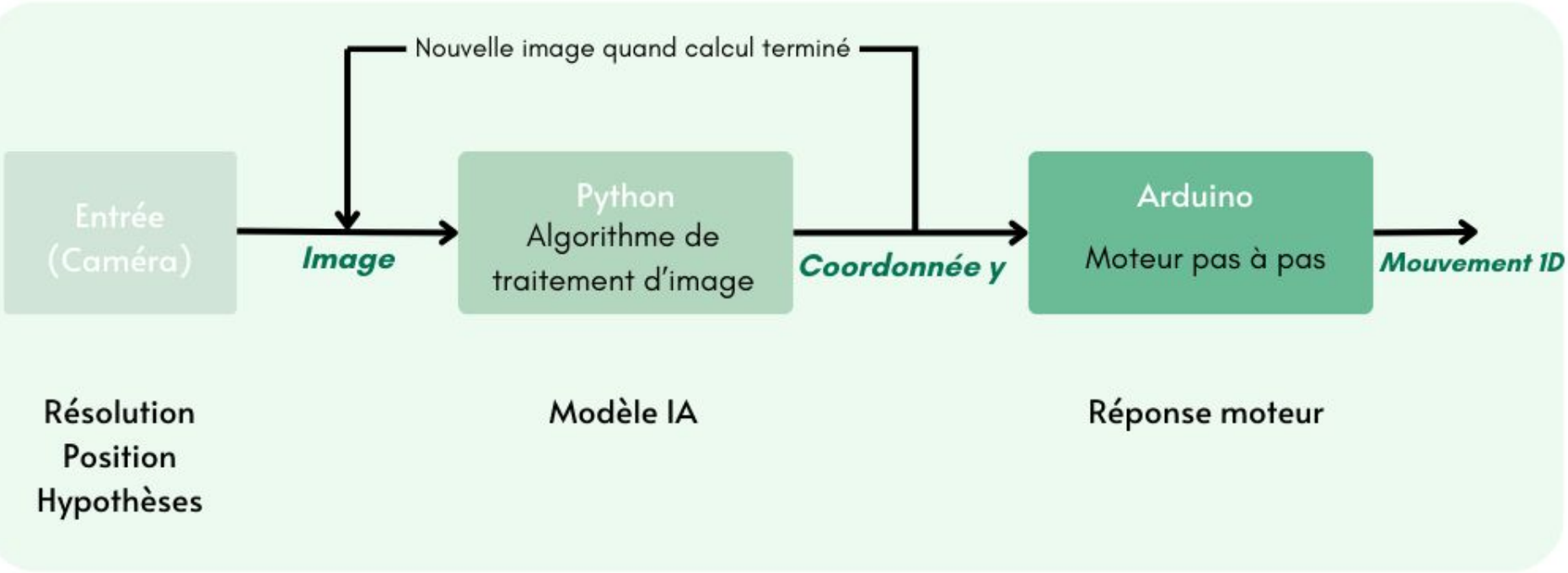
$$\Rightarrow \frac{\theta_f - \theta_i}{t_f - t_i} = \frac{k\,(y_f - y_i)}{r\,(t_f - t_i)} \quad \Rightarrow \theta_f - \theta_i = \frac{k\,(y_f - y_i)}{r}$$

Or $\theta_f - \theta_i = N_{pas}\theta_{1pas}$

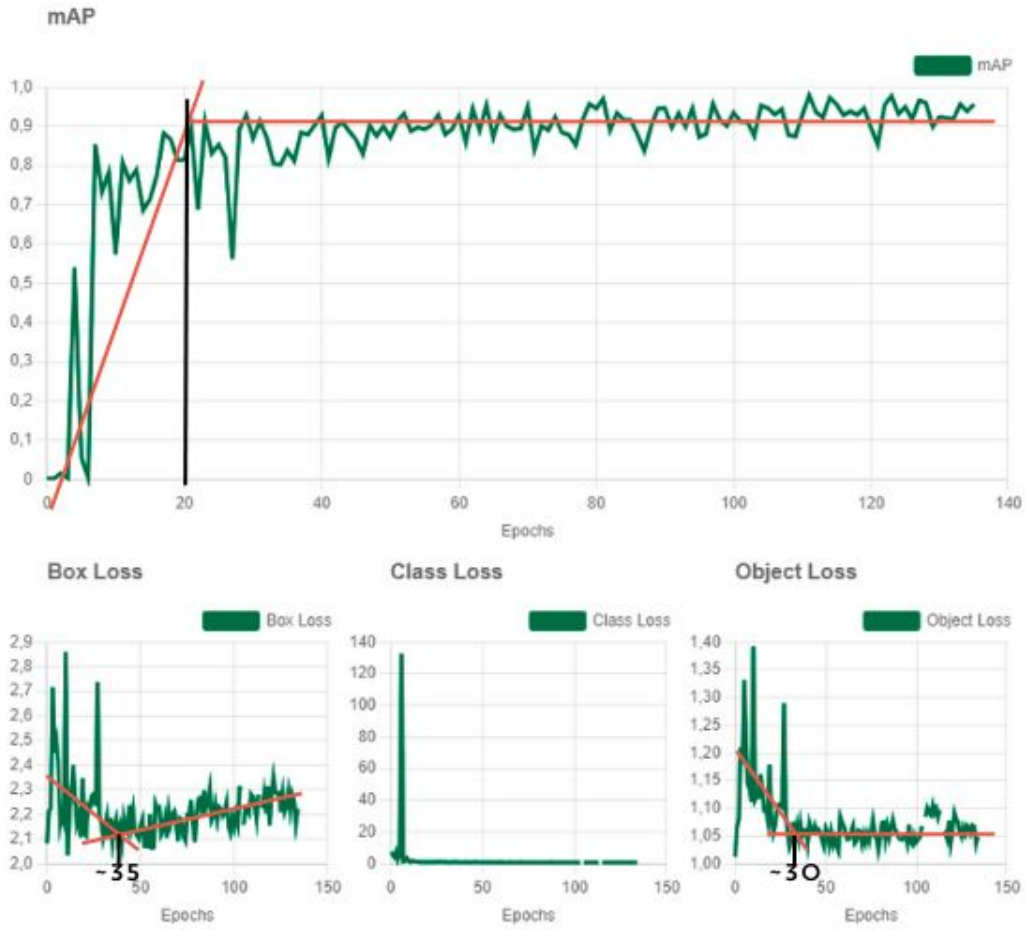
$$N_{pas} = \frac{k\,(y_f - y_i)}{r\theta_{1pas}}$$

III. PERFORMANCES DU SYSTÈME

Etude de la rapidité et de la précision



I - Modèle de l'IA



Courbes d'entraînement du modèle
(interface d'entraînement Roboflow)

35 epoch

2 - Paramètres d'entrée

Logiciel de CAO pour générer images terrains / ballons :

Paramètres fixes

Positions balle, caméra

–> Grande quantité de tests individuels avec les mêmes conditions expérimentales

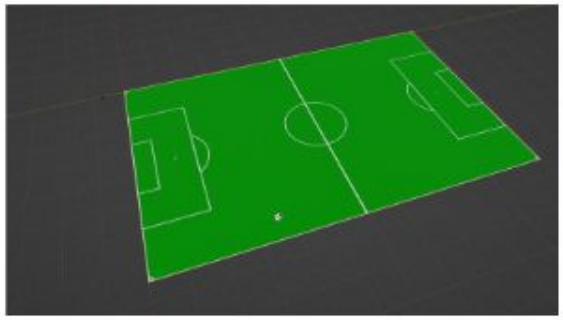
Ⓑ

2 - Paramètres d'entrée : Résolution

Blender
230 positions balle / résolution
Caméra fixée

↓

Python
Distance
Temps calcul
Moyennes/résolut°



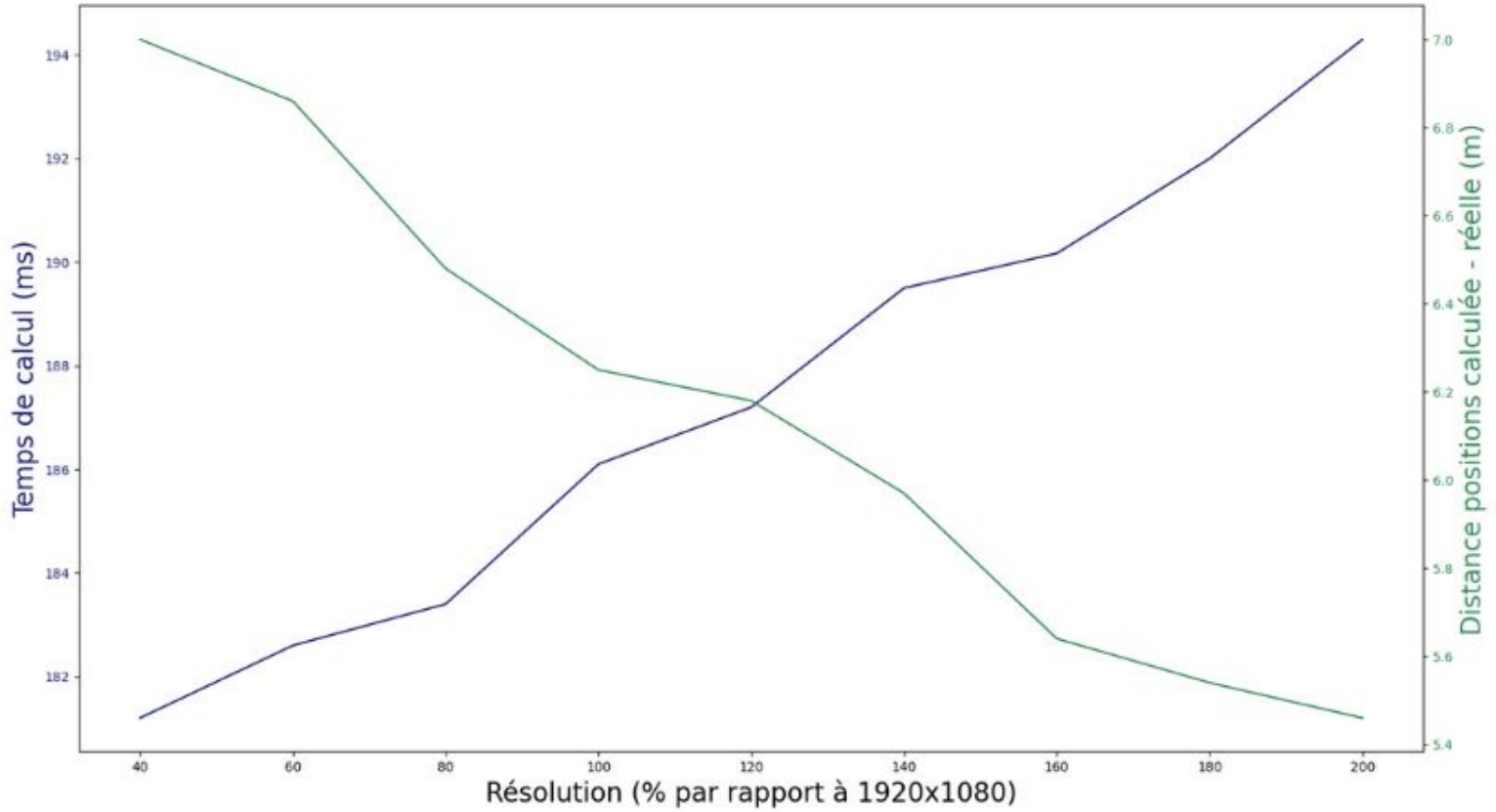
Exemples d'images générées

Distance (m) $\delta = |y_{exp} - y_{th}|$

Ⓑ

2 - Paramètres d'entrée : Résolution

-> Compromis vitesse précision

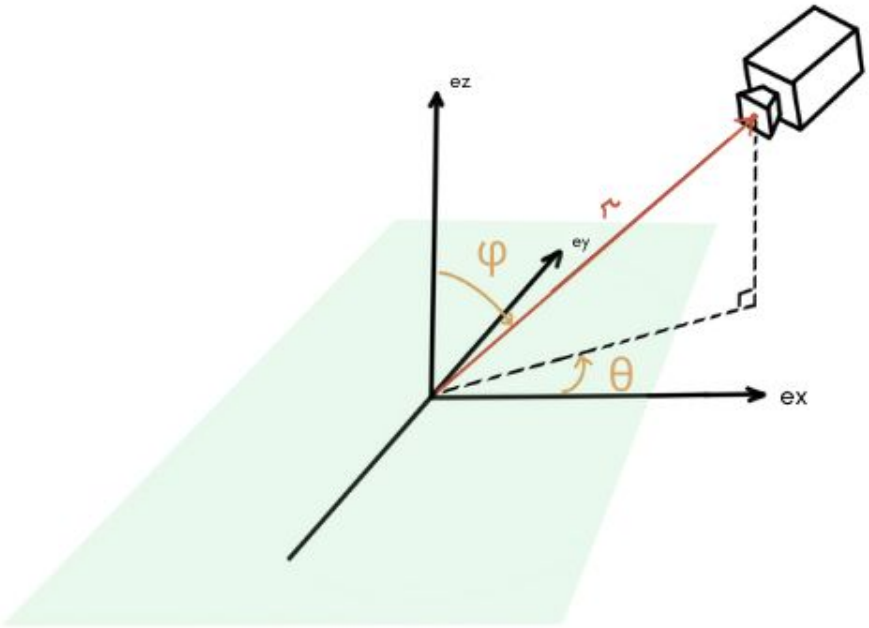


Temps de calcul et distance position calculée / réelle en fonction de la résolution

80 %

2 - Paramètres d'entrée : Position de caméra

- **Elévation (φ) et azimut (θ)**
- Coordonnées sphériques (r,θ,φ), r fixé
- origine repère : centre terrain

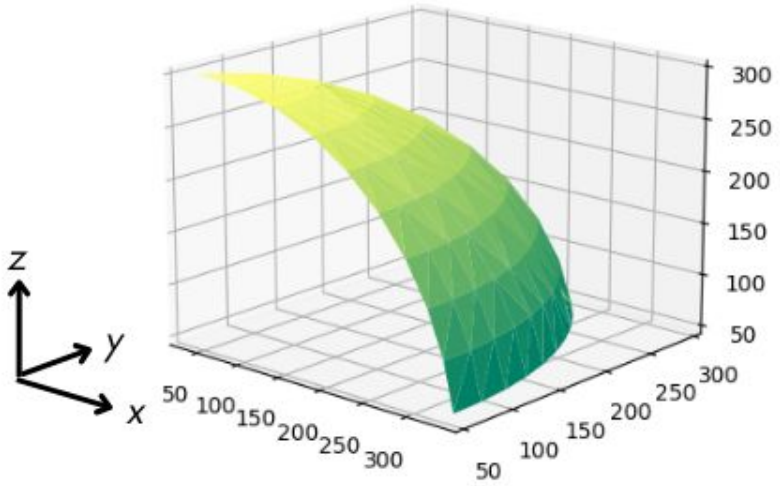


Positionnement d'une caméra dans l'espace

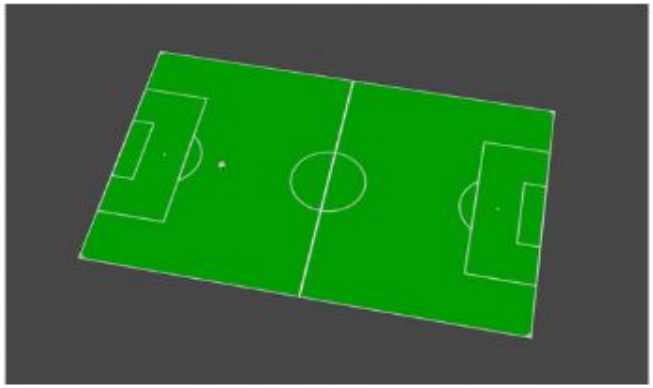
Dans la suite : étude des positions sur 1/4 du terrain (symétries)

Ⓒbis

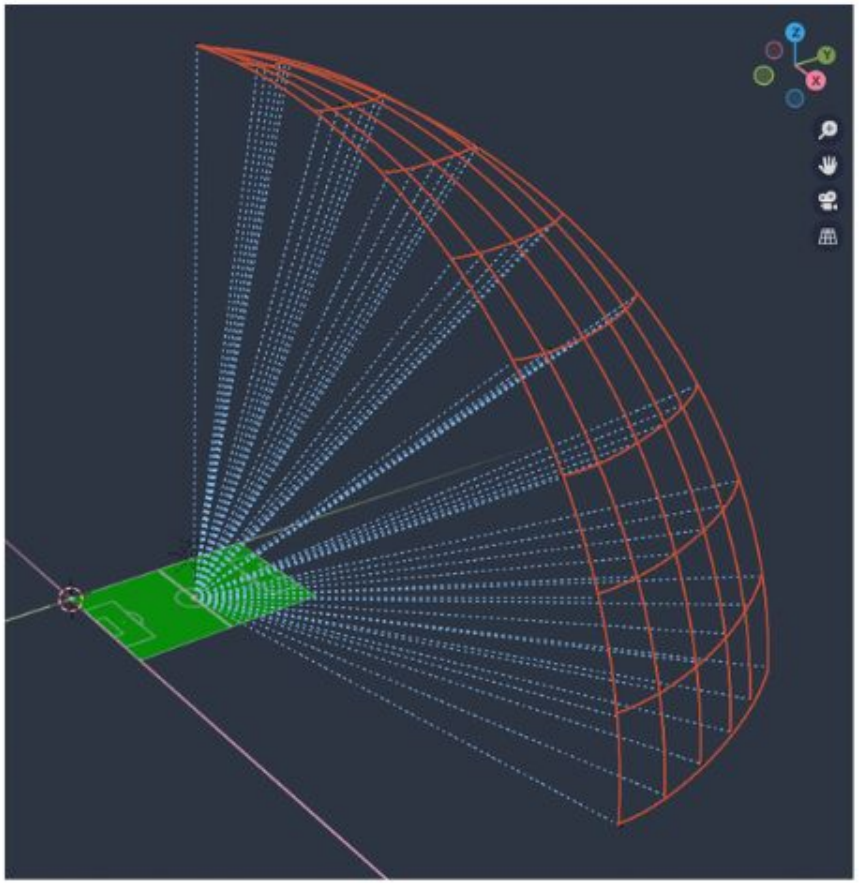
2 - Paramètres d'entrée : Position de caméra



Représentation visuelle des coordonnées générées sur Python (r=300m)

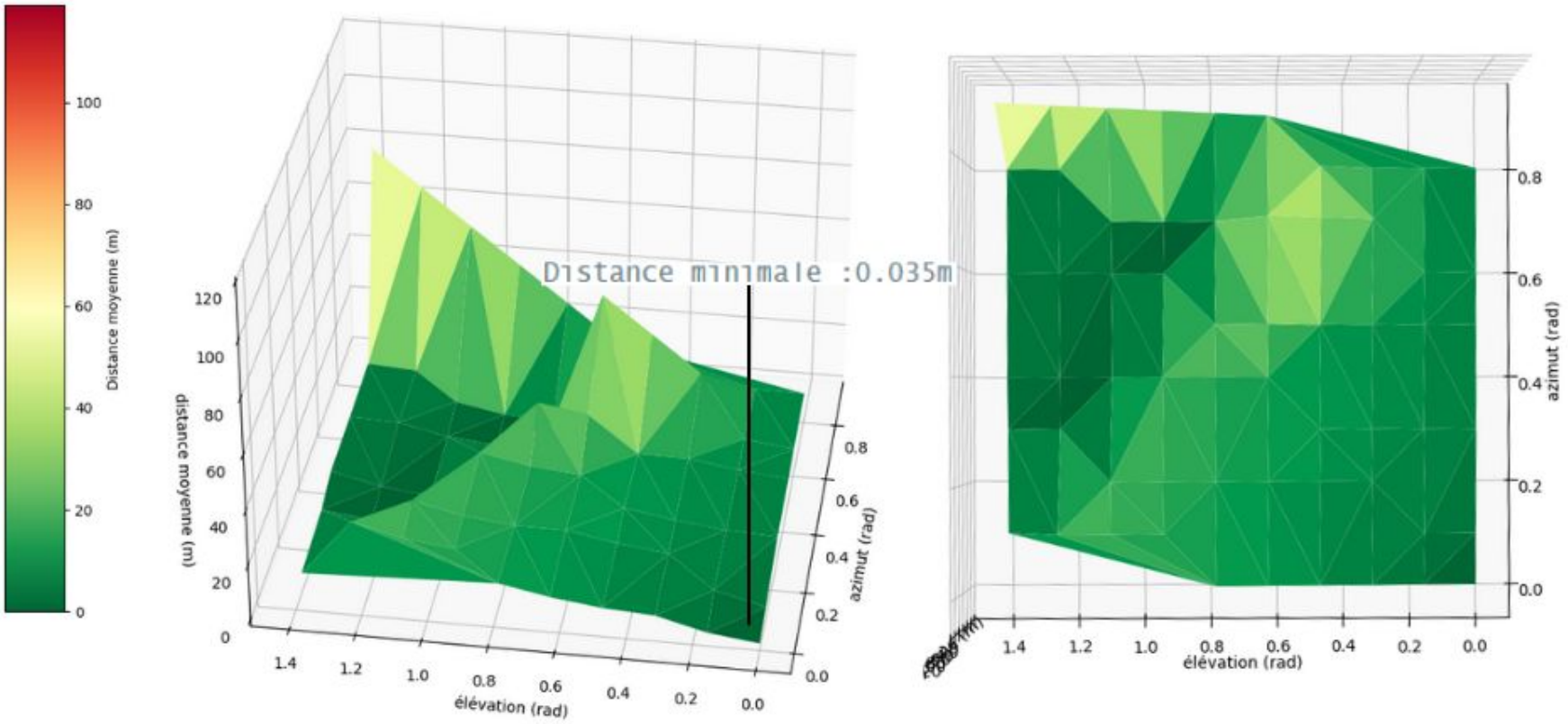


Exemple d'image générée



Interface Blender - 60 caméras placées aux coordonnées générées

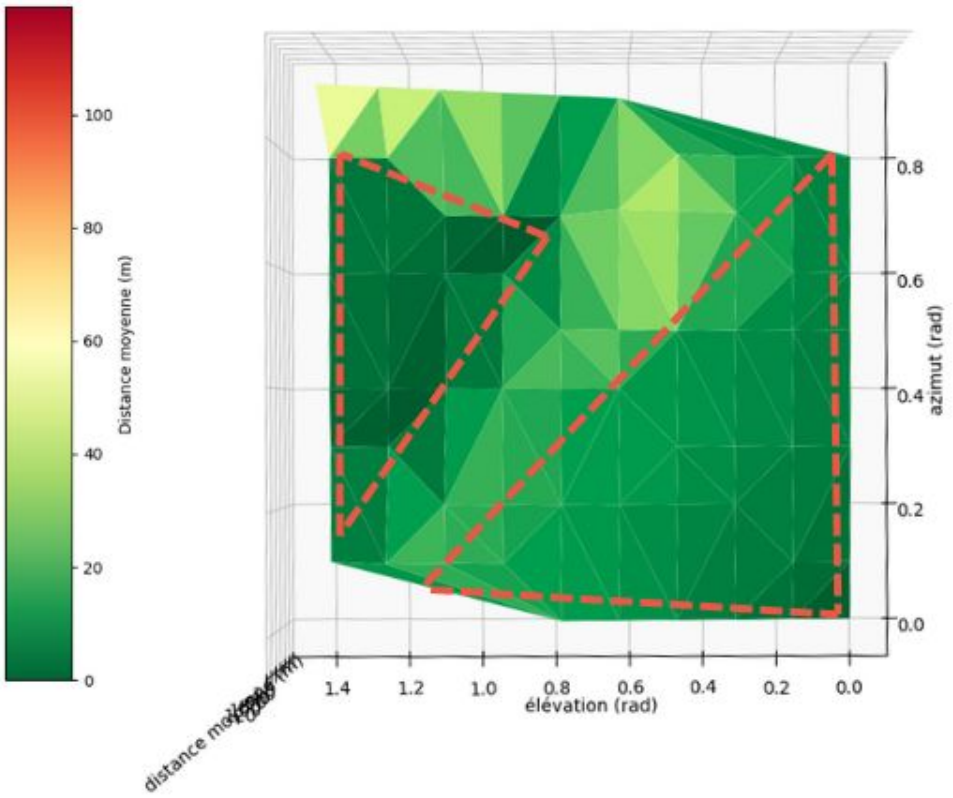
2 - Paramètres d'entrée : Position de caméra

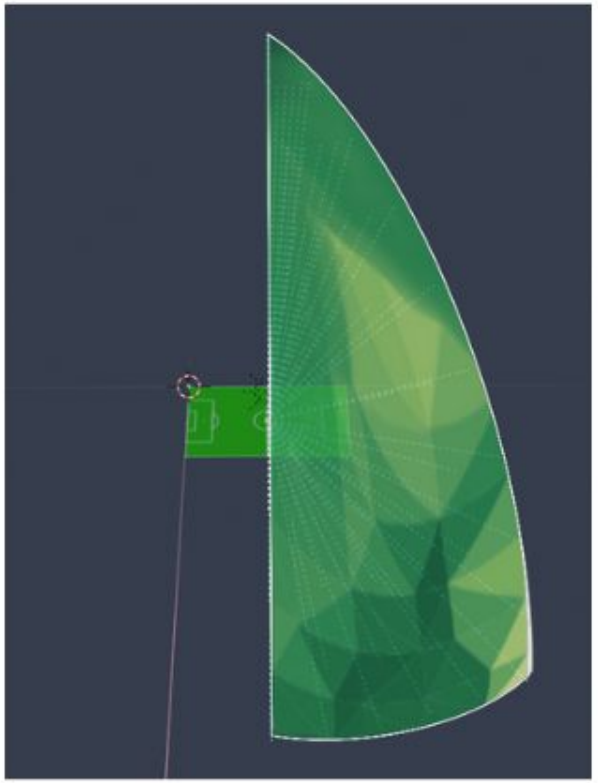


Distance moyenne positions calculée / réelle en fonction de l'azimut et l'élévation de la caméra

60 caméras, moyenne sur 9 positions de balle

2 - Paramètres d'entrée : Position de caméra





Distance moyenne positions calculée / réelle en fonction de l'azimut et l'élévation de la caméra

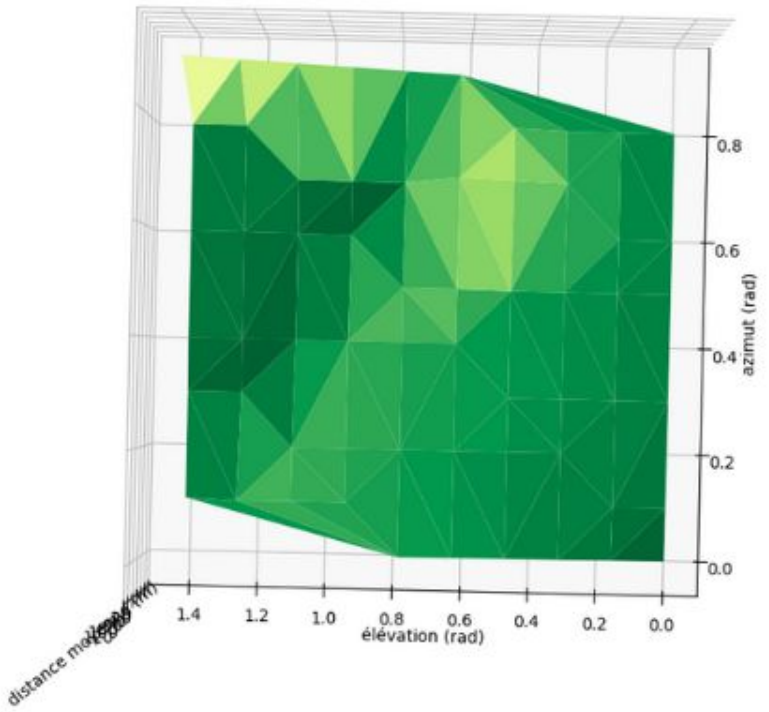
φ, θ -> 0,0 ou φ->90°

3 - Hypothèse : "Balle au sol"

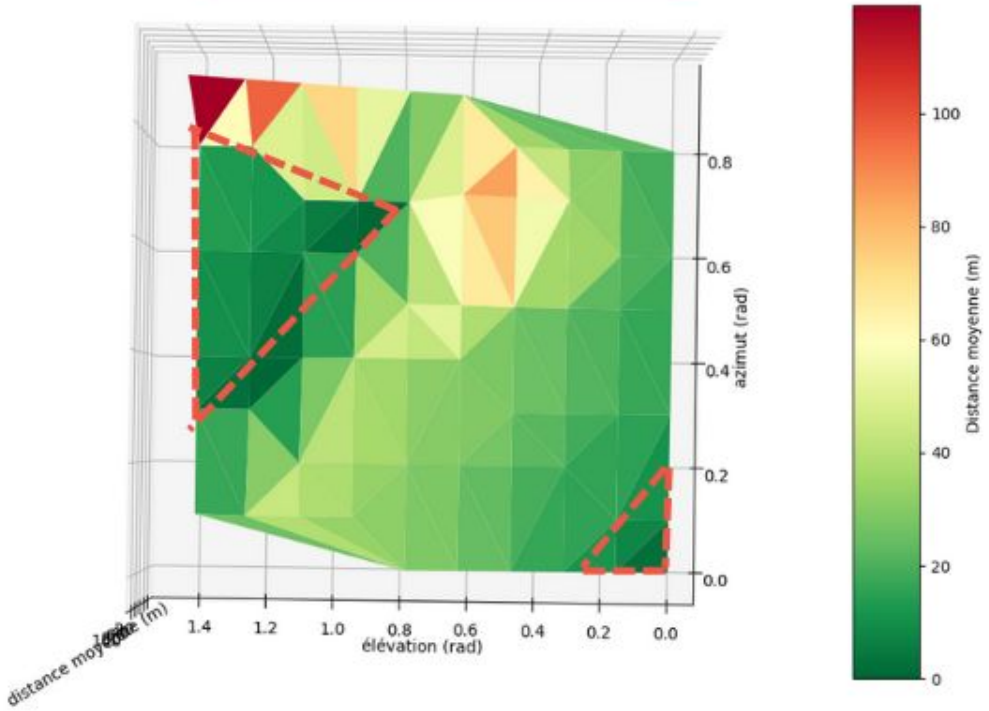
Erreur liée à la hauteur de la balle et le positionnement de la caméra

3 - Hypothèse : "Balle au sol"

Balle au sol

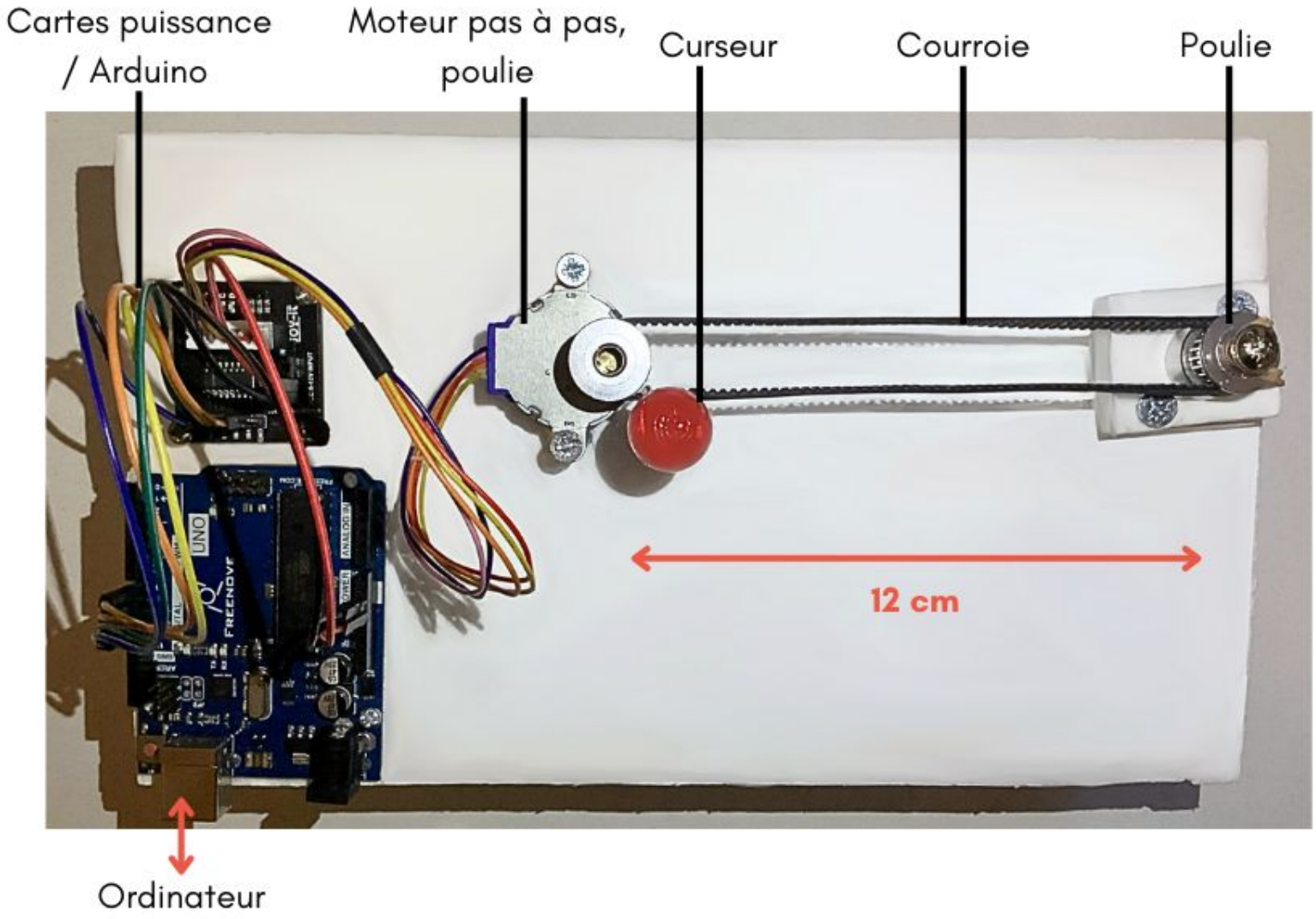


Hauteur balle variable



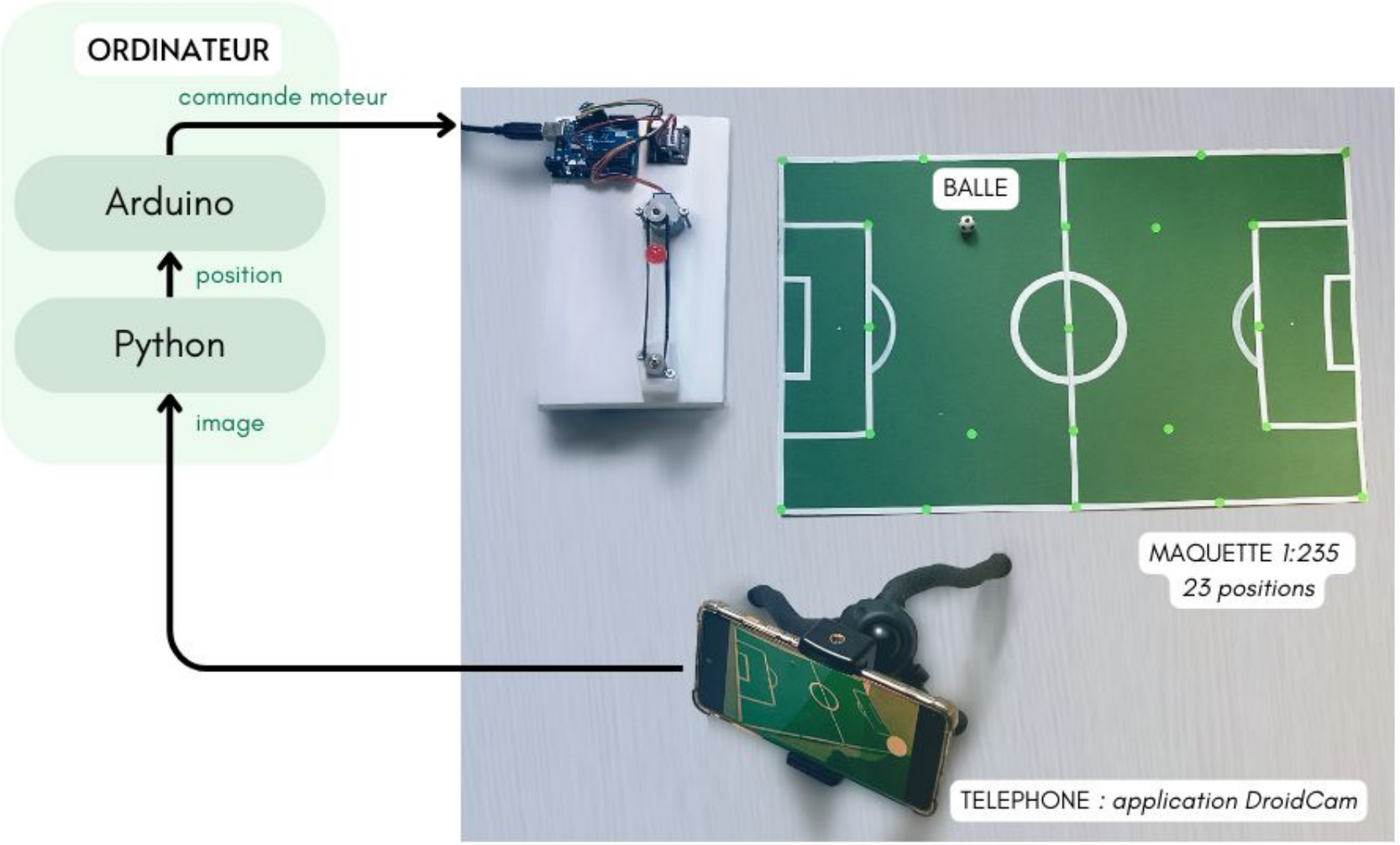
Comparaison : distances moyennes en fonction de l'azimut et l'élévation de la caméra

Hypothèse non nécessaire à certaines positions de caméra



Montage poulie-courroie

I - Précision (système entier)



Expérience de pointage

Ⓐ Ⓓ

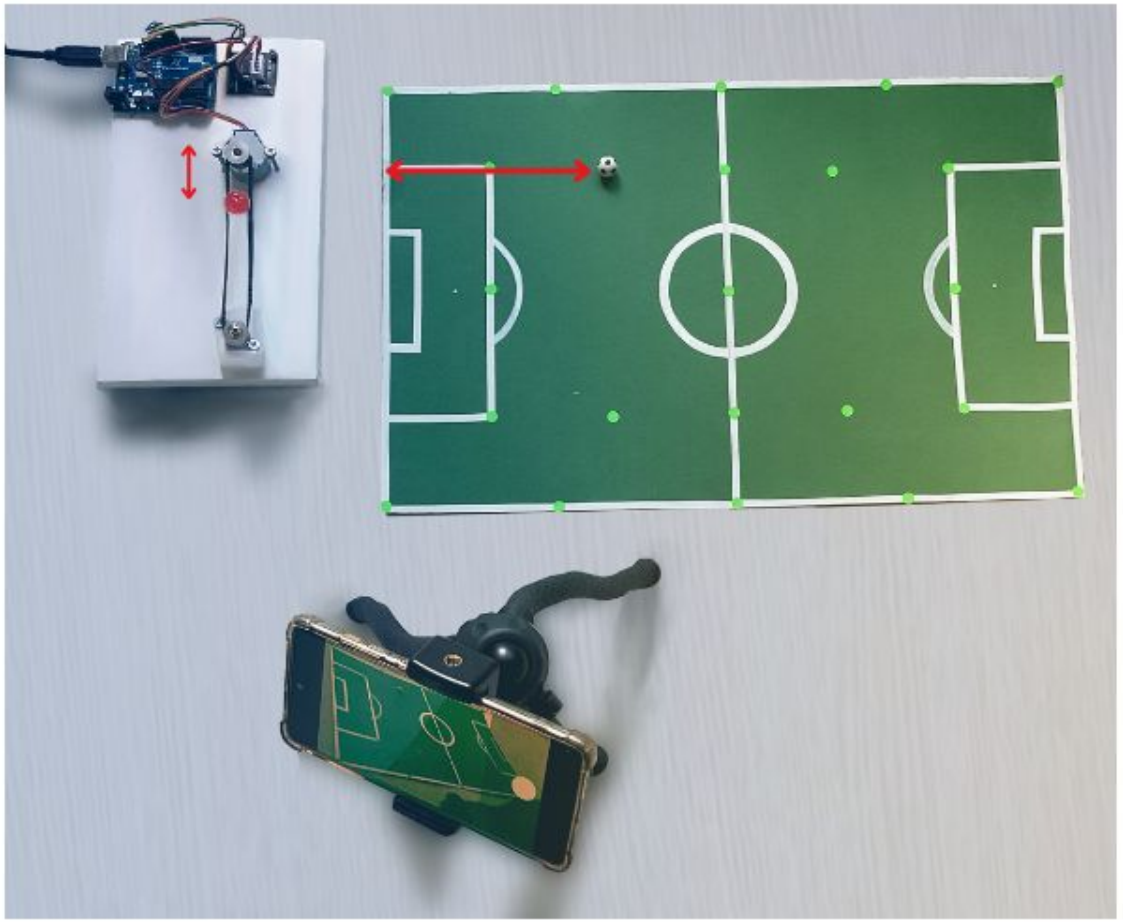
I - Précision

Erreur moyenne :

e = 1.6 ± 0.5 cm

Sources :

- Algorithme
- Poulie-courroie
- Mesures



II - Rapidité : Vitesse

Arduino : rotation vitesse variable

>Vitesse max

—> arrêt

Résultats :

ωmax = 31844 pas/min

ωmax ≈ 15,5 tour/min (2048 pas/tour)

≈ 1.62 rad/s

Or $V_{balleMax} = \frac{\omega_{rot} \times r}{k}$

$$V_{balleMax} = 31.2 kmh^{-1}$$

II - Rapidité : Vitesse

$$V_{balleMax} = 31.2 kmh^{-1}$$

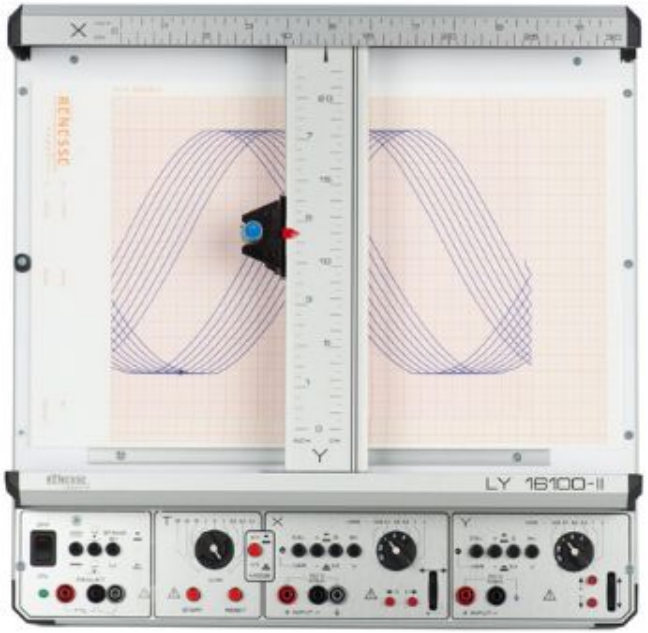
Résultat insuffisant :

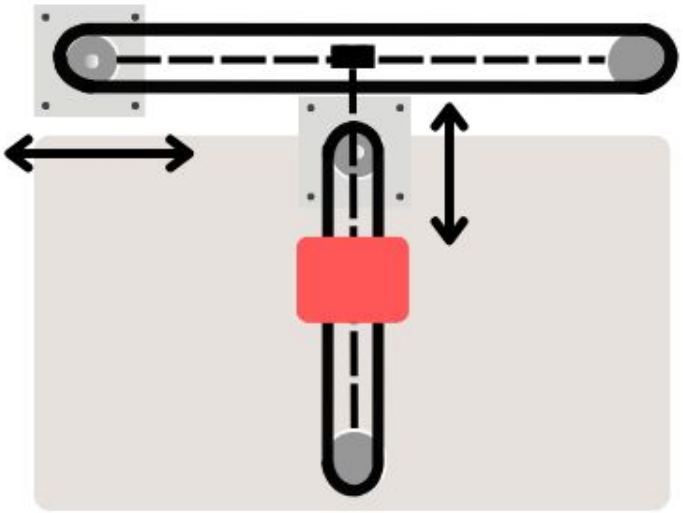
$$V_{balleMoy} = 85 kmh^{-1}$$

Solutions :

- Système plus petit
- Moteur plus performant (ex NEMA17)

Transmettre un mouvement 2D

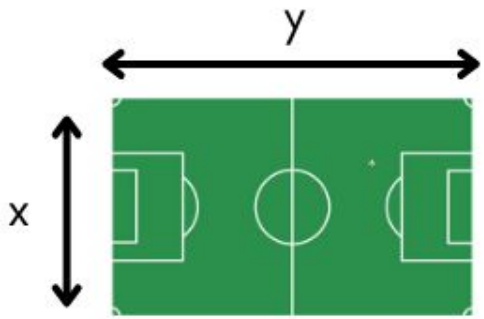


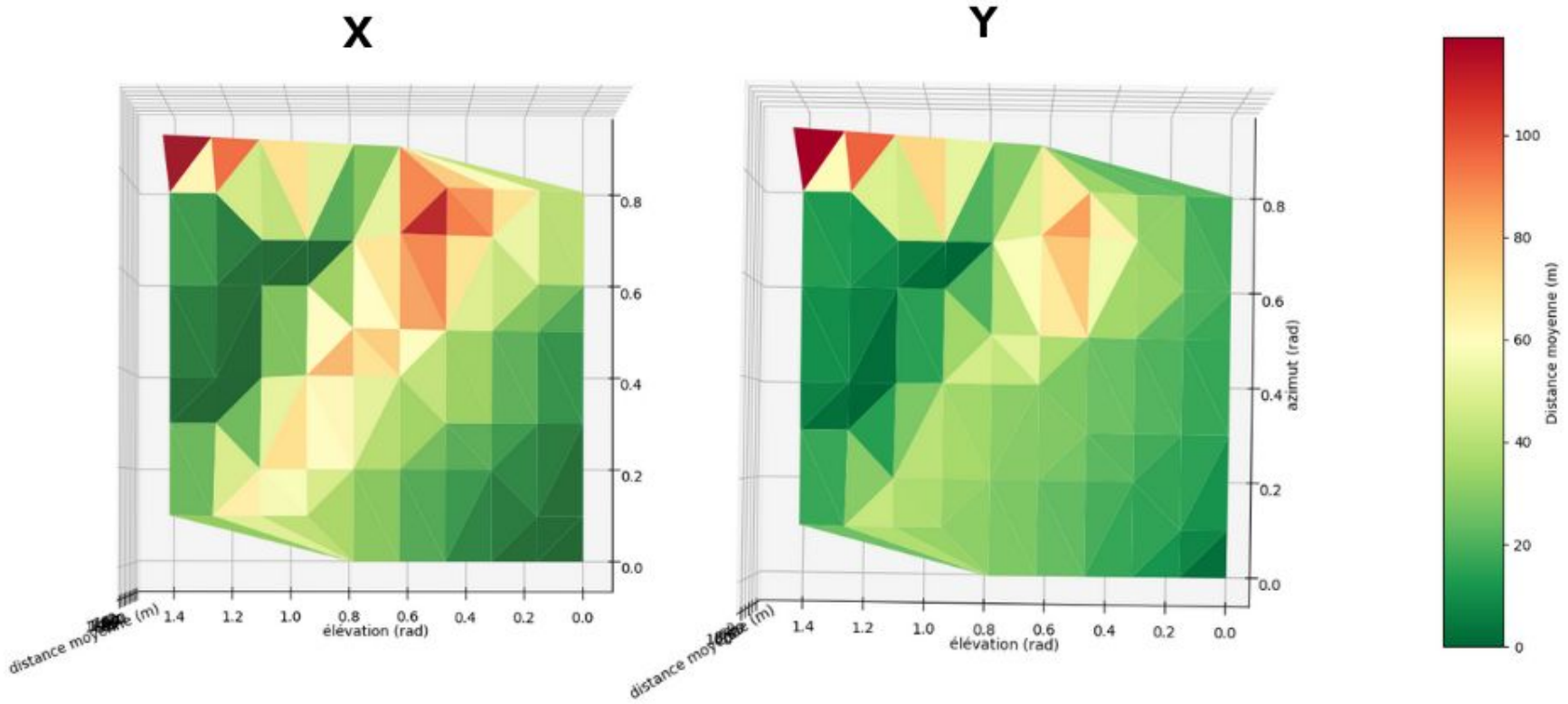


Principe d'une table traçante

photo : © kp8.fr

Limite de l'algorithme à une caméra





Distance moyenne en fonction de l'azimut et l'élévation de la caméra

IV. CONCLUSION

	Modèle IA	**Résolution (entrée)**	**Position caméra**	**Améliorations**
Rapidité	/	⊖ vidéo < 5 fps	/	Moteur plus performant
Précision	35 epoch	⊕	θ,φ --> 0°,0° φ->90°	2+ caméras (stéréovision)
Stabilité	/	/	/	/

ANNEXE

ALGORITHMES- PYTHON BOUCLE PRINCIPALE

```python
## Modules et données

import cv2
import numpy as np

# Modèle de détection
from inference.models.utils import get_roboflow_model
model_name = "balles-de-football"
model_version = "1"
model = get_roboflow_model(model_id="{}/{}".format(model_name, model_version),api_key="WuPJ2uBzqVxnodwmTxCu")

# Accès à la caméra
video = cv2.VideoCapture(0)
assert video.isOpened(), "image inaccessible"


# Segment de couleur
vert1 = np.array([0, 0, 100])
vert2 = np.array([100, 255, 255])


# Coordonnées des points pour la projection
destination = np.array([(0, 0), (0, 119), (75, 0), (75, 119)], dtype=np.float32)

# Données pour la détection du terrain
nb_coins = 4
qualite = 0.1
aire_min = 100000

## Connexion avec Arduino

import serial
arduino = serial.Serial(port='COM3', baudrate=115200, timeout=.1)

def envoi_coordonnee(x):
    arduino.write(bytes(x, 'utf-8'))


## Boucle principale : vraie tant qu'il y a une image en entrée; jusqu'à Echap

while True and cv2.waitKey(1) != 27 :

    # Lecture de chaque image de la vidéo
    ret, frame = video.read()
    if not ret:
        break

    # 1er filtre : segmentation par couleur
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    masque = cv2.inRange(hsv, vert1, vert2)
    img_masque = cv2.bitwise_and(frame, frame, mask=masque)

    # 2eme filtre : bruit
    img_gris = cv2.cvtColor(img_masque, cv2.COLOR_BGR2GRAY)
    noyau = np.ones((10, 10), np.uint8) # = matrice de convolution
```

```
    seuil = cv2.threshold(img_gris, 127, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    seuil = cv2.morphologyEx(seuil, cv2.MORPH_CLOSE, noyau) #dilatation - erosion (fonction "closing")


    #Recherche TERRAIN

    # Image binarisée des contours (algorithme Canny)
    bords = cv2.Canny(seuil, 100, 200)

    # Listes des vecteurs et des hierarchies (liens parents-enfants) des contours dessinés
    # Vecteur = liste de points (x,y)
    contours, hierarchie = cv2.findContours(bords, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    c = []

    for contour in contours:
        approx = cv2.approxPolyDP(contour, 3, True) #lissage du contour (algorithme Ramer Douglas Peucker)
        rectangle = cv2.boundingRect(approx) #cadre rectangulaire autour du contour -> approximation aire (=> filtre de taille)
        L,h = rectangle [2], rectangle [3]
        aire = L * h

        if aire > aire_min:

            enveloppe = cv2.convexHull(contour) #enveloppe convexe : plus petite enveloppe contenant tous les points du contour
            cv2.polylines(frame, [enveloppe], True, (0, 0, 255), 2) #tracé de l'enveloppe

            hauteur, largeur = bords.shape[:2]
            enveloppe_img = np.zeros((hauteur, largeur), dtype=np.uint8)
            cv2.drawContours(enveloppe_img, [enveloppe], 0, 255, 2) #tracé de l'enveloppe convexe

            # détection des coins (algorithme Shi Tomasi)
            distance_min = int(max(hauteur, largeur) / nb_coins)
            coins = cv2.goodFeaturesToTrack(enveloppe_img, nb_coins, qualite, distance_min)
            coins = np.intp(coins)

            for coin in coins:
                x, y = coin.ravel()
                c.append((x, y))
```

```
    if len(c) == nb_coins: #=4

        #tri des points d'intérêt par y croissants puis x croissants par paires (correspondance pour projection)
        coins_tri1 = sorted(c, key = lambda x: x[0])
        coins = sorted(coins_tri1[:2], key = lambda x: x[1]) + sorted(coins_tri1[2:4], key=lambda x: x[1])


        # PROJECTION

        source = np.array(coins,dtype=np.float32)
        # calcul de la matrice de projection H (3x3)
        H, _ = cv2.findHomography(source, destination)


        #DETECTION BALLE

        results = model.infer(frame) #renvoie predictions 'confiance' décroissante -> premier set de coordonnées. Je suppose qu'il y a toujours une prédiction tant que la projection est possible

        #(indiçage dépend de structure résultat)
        r = str(results[0])
        r2 = r.split(" ")

        coord = str(r2[5:9])

        xywh = coord[39:-1]
        coo_boite = xywh.split(',')
        x = float(coo_boite[0][2:])
        y = float(coo_boite[1][2:])
        w = float(coo_boite[2][6:])
        h = float(coo_boite[3][7:])

        coo_balle_img = np.array([x + w / 2, y + h, 1])

        coo_balle = np.dot(H, coo_balle_img)


        #ENVOI A ARDUINO

        envoi_coordonnee(coo_balle[0])

        return coo_balle[0]
    # aucune coordonnée retournée si projection a échoué

# fermeture des fenetres
video.release()
cv2.destroyAllWindows()
```

```python
## Modules et données

import matplotlib.pyplot as plt
import cv2
import time
import numpy as np

#Modèle de détection
from inference.models.utils import get_roboflow_model
model_name = "balles-de-football"
model_version = "1"
model = get_roboflow_model(model_id="{}/{}".format(model_name, model_version),api_key="WuPJ2uBzqVxnodwmTxCu")


# Segment de couleur
vert1 = np.array([0, 0, 100])
vert2 = np.array([100, 255, 255])

# Coordonnées des points pour la projection
destination = np.array([(0, 0), (0, 119), (75, 0), (75, 119)], dtype=np.float32)

# Données pour la détection des coins
nb_coins = 4
qualite = 0.1
aire_min = 100000

pas_de_detection = 0

## Programmes

#génération des chemins de chaque dossier contenu dans le dossier principal à partir de la liste de leur noms
def nom_dossier(chemin,ldossiers):
    return [chemin + str(dossier) for dossier in ldossiers]

#reprise de l'algorithme principal
def position_x(cimg):

    global pas_de_detection
    pas_de_detection = 0

    frame=plt.imread(cimg)

    # 1er filtre : segmentation par couleur
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    masque = cv2.inRange(hsv, vert1, vert2)
    img_masque = cv2.bitwise_and(frame, frame, mask=masque)

    # 2eme filtre : bruit
    img_gris = cv2.cvtColor(img_masque, cv2.COLOR_BGR2GRAY)
    noyau = np.ones((10, 10), np.uint8) # = matrice de convolution
    seuil = cv2.threshold(img_gris, 127, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    seuil = cv2.morphologyEx(seuil, cv2.MORPH_CLOSE, noyau)#dilatation - erosion (fonction "closing")

```

```
54
55      #Recherche TERRAIN
56
57      # Image binarisée des contours (algorithme Canny)
58      bords = cv2.Canny(seuil, 100, 200)
59
60      # Listes des vecteurs et des hierarchies (liens parents-enfants) des contours dessinés
61      # Vecteur = liste de points (x,y)
62      contours, hierarchie = cv2.findContours(bords, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
63
64      c = []
65
66      for contour in contours:
67          approx = cv2.approxPolyDP(contour, 3, True) #lissage du contour (algorithme Ramer Douglas Peucker)
68          rectangle = cv2.boundingRect(approx) #cadre rectangulaire autour du contour -> approximation aire (=> filtre de taille)
69          L,h = rectangle [2], rectangle [3]
70          aire = L * h
71
72          if aire > aire_min:
73
74              enveloppe = cv2.convexHull(contour) #enveloppe convexe : plus petite enveloppe contenant tous les points du contour
75              cv2.polylines(frame, [enveloppe], True, (0, 0, 255), 2) #tracé de l'enveloppe
76
77              hauteur, largeur = bords.shape[:2]
78              enveloppe_img = np.zeros((hauteur, largeur), dtype=np.uint8)
79              cv2.drawContours(enveloppe_img, [enveloppe], 0, 255, 2) #tracé de l'enveloppe convexe
80
81              # détection des coins (algorithme Shi Tomasi)
82              distance_min = int(max(hauteur, largeur) / nb_coins)
83              coins = cv2.goodFeaturesToTrack(enveloppe_img, nb_coins, qualite, distance_min)
84              coins = np.intp(coins)
85
86              for coin in coins:
87                  x, y = coin.ravel()
88                  c.append((x, y))
89
90      if len(c) == nb_coins: #=4
91
92          #tri des points d'intérêt par y croissants puis x croissants par paires (correspondance pour projection)
93
94          coins_tri1 = sorted(c, key = lambda x: x[0])
95
96          coins = sorted(coins_tri1[:2], key = lambda x: x[1]) + sorted(coins_tri1[2:4], key = lambda x: x[1])
97
```

```
98
99          # PROJECTION
100
101         source = np.array(coins,dtype=np.float32)
102         # calcul de la matrice de projection H (3x3)
103         H, _ = cv2.findHomography(source, destination)
104
105         results = model.infer(cimg) #renvoie predictions 'confiance' décroissante -> premier set de coordonnées. Je suppose qu'
    a toujours une prédiction tant que la projection est possible
106
107         #(indiçage dépend de structure résultat)
108         r = str(results[0])
109         r2 = r.split(" ")
110
111         coord = str(r2[5:9])
112
113         xywh = coord[39:-1]
114         coo_boite = xywh.split(',')
115         x = float(coo_boite[0][2:])
116         y = float(coo_boite[1][2:])
117         w = float(coo_boite[2][6:])
118         h = float(coo_boite[3][7:])
119
120         coo_balle_img = np.array([x + w / 2, y + h, 1])
121
122         coo_balle = np.dot(H, coo_balle_img)
123
124         return coo_balle[0]
125         #return coo_balle[1] pour la coordonnée selon la largeur
126     pas_de_detection = 1
127
128
129 # mesure du temps de calcul d'une position dans une image
130 def tps_position(cimg):
131     t0 = time.time()
132     position_x(cimg)
133     t = time.time()
134     return t - t0
135
136 # calcul de distance
137 def precision(cimg): #on ne traite pas le cas sans détection car dans ce cas fonction non appelée par moyenne_dossier (L.156)
138
139     n = len(cimg)
140     num_image = int(cimg[n-8:n-4])
141     for i in range(3):
142         if cimg[n-8+i] == '0':
143             num_image = int(cimg[n-8+i:n-4]) #permet de retrouver la coordonée attendue
144
145     #position calculée
146     x = position_x(cimg)
147
148     #position attendue, l'ordonnée de la balle reste la même pour 21 images
149     t = (num_image // 21) * 11.9
150
151     d = abs (x-t)
152     return d
```

```python


#calcul du temps de traitement et de la precision moyens à une résolution fixée
def moyenne_dossier(cdossier,frames):

    tps = 0 #somme des durées de traitement
    d = 0 #somme des distances
    n = 0 #nombre d'images prises en compte dans la moyenne

    for i in range(1,frames):

        #noms de fichiers de la forme "000i","00ij","0ijk"...
        cimg = cdossier + '\\' + (4 - len(str(int(i)))) * '0' + str(int(i)) + ".jpg"

        x = position_x(cimg)

        if pas_de_detection == 0:
          tps += tps_position(cimg)
          d += precision(cimg)
          n += 1

    if n != 0:
        return tps/n,d/n
    return 0,0



#tracé d'un graphe à deux ordonnées (erreur, rapidité) à abscisse commune (résolution)
def graphe_RTP(x,y,z):

    fig, axe1 = plt.subplots()
    axe1.set_xlabel('Résolution (% par rapport à 1920x1080)')

    axe1.set_ylabel('Temps de calcul (ms)', color='midnightblue')
    axe1.plot(x, y, color='midnightblue')
    axe1.tick_params(axis='y', labelcolor='midnightblue')

    axe2 = axe1.twinx()  # second d'axe d'ordonnées qui partage un même d'axe d'abscisses

    axe2.set_ylabel('Distance positions calculée - réelle (m)', color='seagreen')
    axe2.plot(x, z, color='seagreen')
    axe2.tick_params(axis='y', labelcolor='seagreen')

    fig.tight_layout()

    plt.show()

#TESTS :
chemin_defaut = "F:\TIPE_resolution\\"
resolutions = [40,60,80,100,120,140,160,180,200]# Résolution de référence : 1920x1080. 20=>20% de cette résolution, etc
nb_img = 230
chemins = nom_dossier(chemin_defaut,resolutions) # liste des chemins des dossiers associés à diff. résolutions
graphe_RTP(resolutions,moyenne_dossier(chemins,nb_img)[0],moyenne_dossier(chemins,nb_img)[1])


```

ALGORITHMES- PYTHON ERREUR POSITIONS CAMÉRA

```python
def distance(cimg):

    n = len(cimg)
    num_image = int(cimg[n-5]) #numéro de l'image étudiée pour position balle théorique

    x = position_x(cimg)

    if pas_de_detection == 0:
        positions_attendues={0:0,1:59.5,2:119,3:29.75,4:59.5,5:89.25,6:0,7:59.5,8:119} #positions clefs utilisées pour tests
        #positions_attendues={0:0,1:0,2:0,3:37.5,4:37.5,5:37.5,6:75,7:75,8:75} selon largeur
        t = positions_attendues[num_image]
        d = abs (x-t)
        return d
    return 119 #on ne "favorise" pas les positions qui ne détectent pas de terrain : distance max sur un terrain


def distance_moyenne_dossier(cdossier,frames,nb_position_cam):

    l_distance = []
    for i in range(1,nb_position_cam):

        d = 0

        for j in range(frames):

            cimg = cdossier + (3 - len(str(i))) * '0'+ str(i) + '\\050' + str(int(j)) + ".jpg"
            d += distance(cimg)
            l_distance.append(d/frames)

    return l_distance

##Tracé graphe
x = [i/10 for i in range(10) for j in range(6) ] #positionnement de la caméra à partir du numéro de dossier : azimut
y = [int(str(no_dossier)[-1])*pi/20 for no_dossier in range(60)] #élévation
z = distance_moyenne_dossier("F:\TIPE_camera_pos\\",9,61)

fig = plt.figure()
ax = fig.add_subplot(projection='3d')

norme = Normalize(vmin=0, vmax=119)

triang = ax.plot_trisurf(x, y, z, cmap='RdYlGn_r', norm=norme)

ax.set_title("Distance moyenne entre les positions théoriques et issues du calcul en fonction de l'azimut et de l'élévation de la caméra")
ax.set_xlabel('azimut (rad)')
ax.set_ylabel('élévation (rad)')
ax.set_zlabel('distance moyenne (m)')

cbar = fig.colorbar(triang, ax=ax, shrink=0.8, aspect=10)
cbar.set_label('Distance moyenne (m)')

plt.show()
print("Distance minimale :"+str(round(min(z),3))+"m")
```

```python
#Génération de positions de caméras pour la simulation 3d -> conversion sphériques à cartésiennes

from math import sin, cos, pi
import matplotlib.pyplot as plt
import numpy as np

def coo_sph_cart(r,t,p): #r,theta,phi
    x = 37.5 + r * sin(p) * cos(t) #centre du terrain situé en (37.5,59.5,0)
    y = 59.5 + r * sin(p) * sin(t)
    z = r * cos(p)
    return round(x,1), round(y,1), round(z,1) #on arrondit à un chiffre après virgule

n = 10
p = 10

#on fixe r
r = 300

#theta,phi en radian pour cos, sin
theta = [j/p for j in range(p)] # angle max condition tri points : 1 rad
phi = [i*(pi/2)/n for i in range(n)] #détection impossible pour t=pi/2 (=au sol)

x=[]
y=[]
z=[]

for i in range(len(theta)) :
    for j in range(len(phi)) :
        x.append(coo_sph_cart(r,theta[i],phi[j])[0])
        y.append(coo_sph_cart(r,theta[i],phi[j])[1])
        z.append(coo_sph_cart(r,theta[i],phi[j])[2])

#Représentation graphique

fig = plt.figure()

ax = plt.axes(projection='3d')

surf = ax.plot_trisurf(np.array(x), np.array(y), np.array(z), linewidth=0, antialiased=True, cmap='summer')
ax.set_title('Représentation visuelle des coordonnées générées')
plt.show()
```

ALGORITHMES- ARDUINO BOUCLE PRINCIPALE

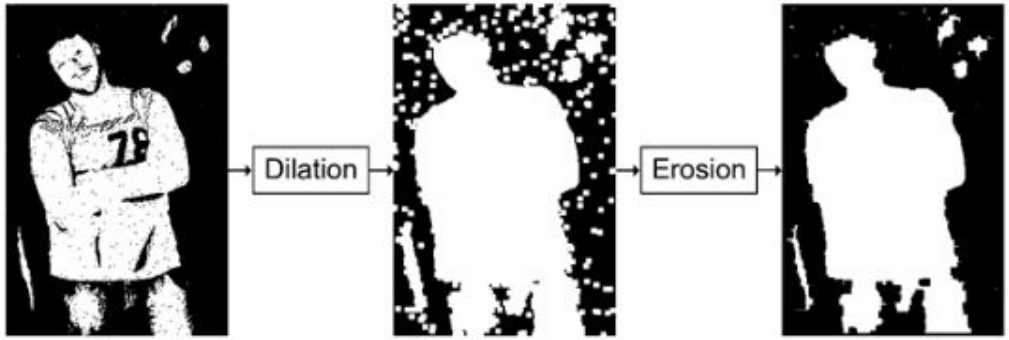
```
#include <Stepper.h>
//constantes : 'const int' -> lecture uniquement (valeur non modifiable dans la boucle)
const int pi = 3.14159;
const int r = 0,005;              // rayon de la poulie (m)
const int k = 0.00101;            // coefficient proportionnalité terrain -> courroie
const int nbPasTour = 2048;
const int a = 2 * pi / nbPasTour;  // angle d'un pas (rad)
const int wmax = 15.54;           // vitesse de rotation maximale (tour/min), déterminée dans le test de précision

int xi = 0;        // initialisation position initiale
int xf;

int w;             // vitesse de rotation

int Ti=0;          // initialisation instant initial
int Tf;
int dt;            // intervalle de temps, Tf-Ti défini dans boucle. théoriquement : dt = 1/fps

int N;             // nombre de pas à réaliser pour une coordonnée, calculé dans la boucle
int PasTot = 0;    // = - nb pas à réaliser pour revenir à la position initale (ou position actuelle en pas)

Stepper Moteur(nbPasTour,8,9,10,11);

void setup() {
  Serial.begin(115200);
  Serial.setTimeout(1);
}

```

```
void loop() {
  while(! Serial.available()) {

    Serial.setTimeout(1000);

    xf = Serial.readString().toInt();
    Tf = millis()*0.001;        // ms -> s
    dt = Tf-Ti;

    if(xf < 120 && xf>=0){      //longueur terrain = 119m
      w = k*(xf-xi)/(r*dt);     //vitesse
      if(w<wmax){
        Moteur.setSpeed(w);
        N = k * (xf-xi) / (r * a);  //nombre pas à faire
        Moteur.step(N);
      }
      else{
        Moteur.setSpeed(wmax);
        N = k * (xf-xi) / (r * a);
        Moteur.step(N);
      }
      PasTot+=N;
      xi=xf;
      Ti=Tf;
    }
  }

//retour à la position 0 si communication coupée et arrêt du programme Arduino
Moteur.setSpeed(1)
Moteur.step(-PasTot)
break
}

```

ALGORITHMES- ARDUINO VITESSE MAX

```
1   //détermination de la vitesse de rotation max
2   #include <Stepper.h>
3
4   const int nbpastour = 1; //en réalité : 2048 pas / tour
5   int i = 0 ;
6   int w;
7
8   Stepper Moteur(nbpastour, 8, 9, 10, 11);
9
10  void setup() {
11    Serial.begin(115200);
12    Serial.setTimeout(1);
13  }
14
15  void loop() {
16    while(i<100000){
17      w = 20480+i; //en tour/min, ici 1 tour = 1 pas donc w en pas/min (+20480 pas car vitesse max > 10 tours / min)
18      Moteur.setSpeed(w);
19      Moteur.step(100); //en pas
20
21      Serial.print(i);
22      Serial.print() ;
23      i+=1;
24      delay(200);
25     }
26  }
27  }
28
```

Dilatation, érosion



Fonction closing

Binarisation de l'image (seuillage)

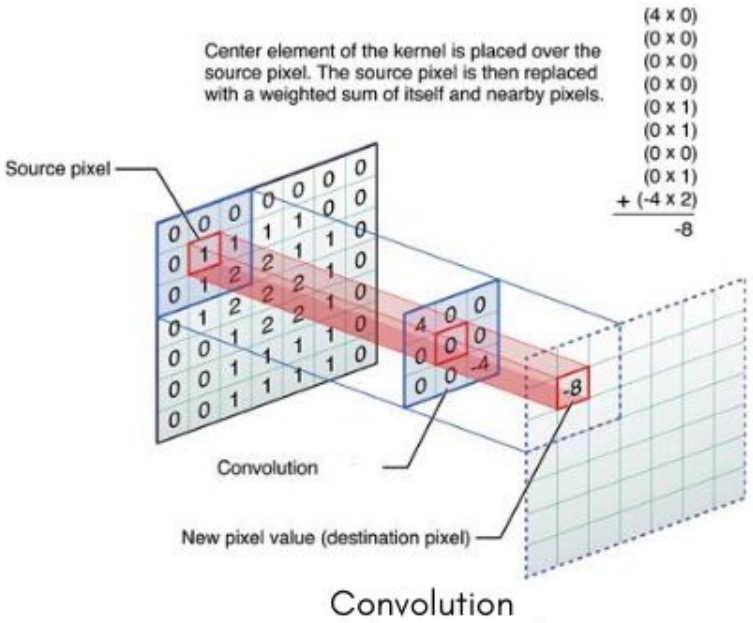
$$I_{ij} = \frac{(r + v + b)}{3}$$ intensité du pixel p(i,j) — gris

Nouvelle image [p'(n,k)] :

$p'_{ij} = 255$ si $I_{ij} \geqslant s$ seuil

$p'_{ij} = 0$ sinon

Convolution



Convolution

1) Filtrage **anti-bruit**
2) Produit de **convolution** avec :

$$G_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad ; \quad G_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

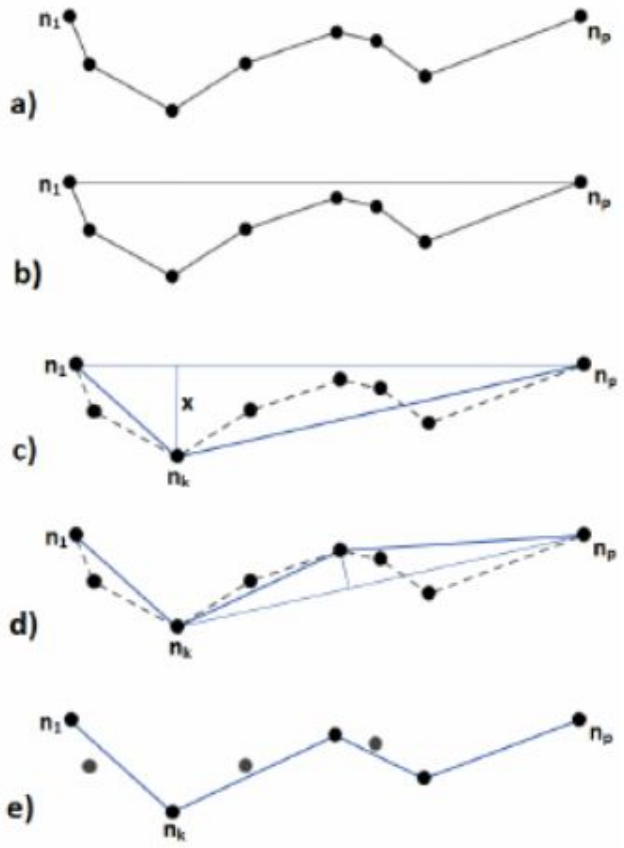
Gradient : vecteur de norme

$$|G| = \sqrt{G_x^2 + G_y^2}$$

\+ => forte probabilité de présence d'un contour

3) Recherche de **maxima** locaux (seuillage)

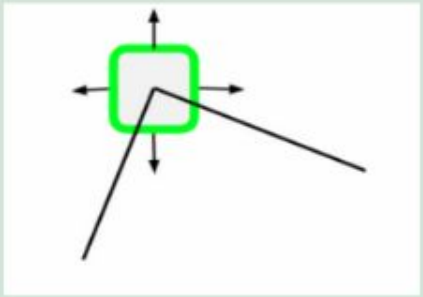
Algorithme de Ramer Douglas Peucker



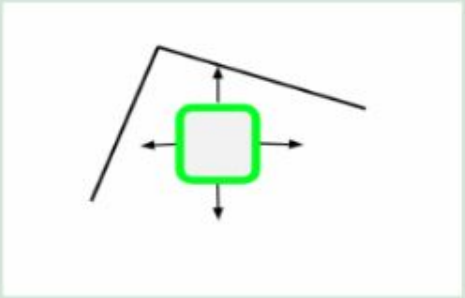
Ramer Douglas Peucker

PRINCIPE DE L'ALGORITHME - DÉTECTION DE COINS (SHI TOMASI)

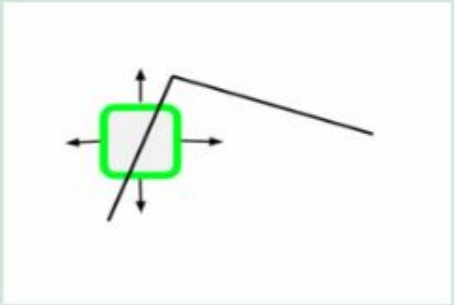
Balayage d'une "fenêtre"



Coin : changement d'apparence dans **2+ directions**



Zone plate



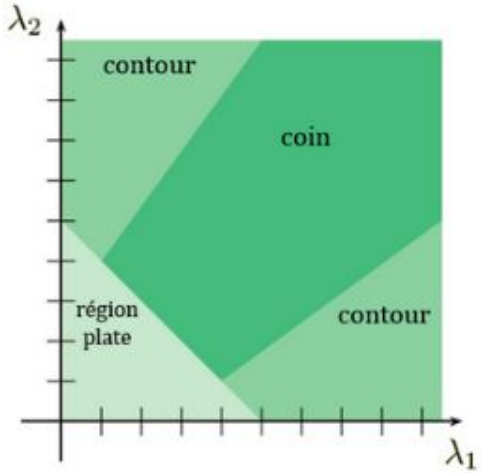
Bord : **selon 1 direction**

Recherche de coins

© Stacklima.com

1) **Gradient d'intensité** selon x et y
2) **Matrices de second moment**, Mij
3) **Valeurs propres** de Mij λ1,λ2 (-> courbure)

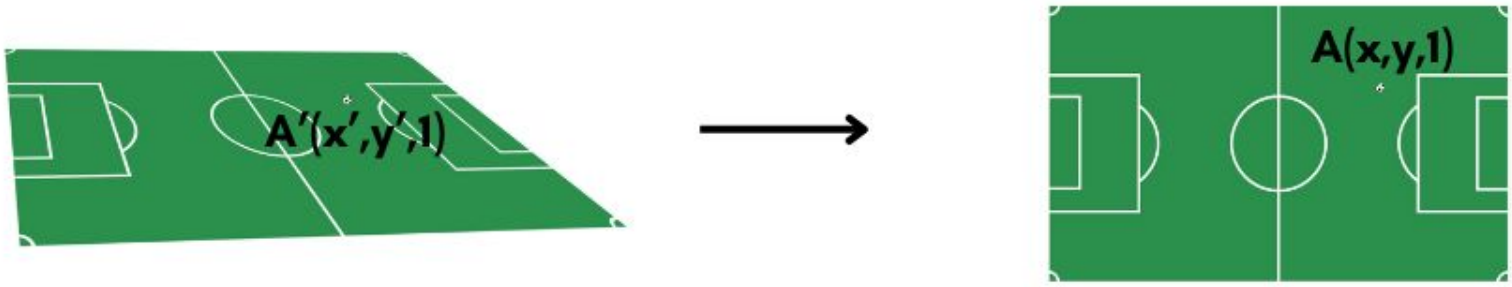
$$Mij = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$



Caractérisation du pixel

© Pascal Monasse

Transformation projective plane



Si A' et A se correspondent dans les deux plans, alors il existe H définie à un scalaire près telle que :

$$H.A' = A \Leftrightarrow \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

8 ddl

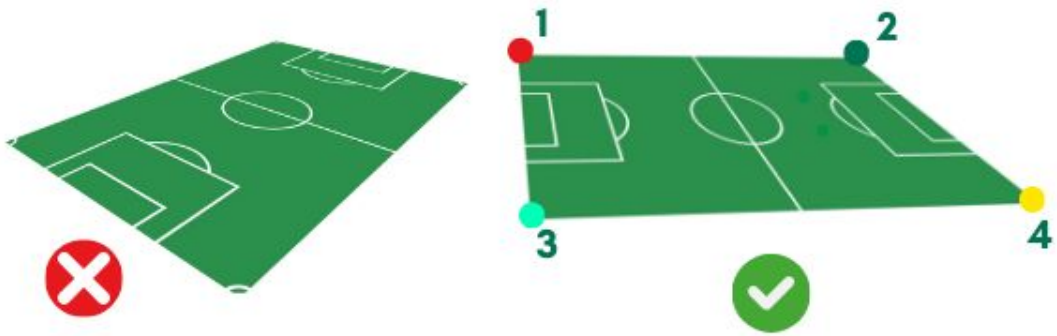
A'(x',y') et A(x,y) se correspondent dans les deux plans

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x'h_{11} + y'h_{12} + h_{13} \\ x'h_{21} + y'h_{22} + h_{23} \\ x'h_{31} + y'h_{32} + 1 \end{bmatrix}$$

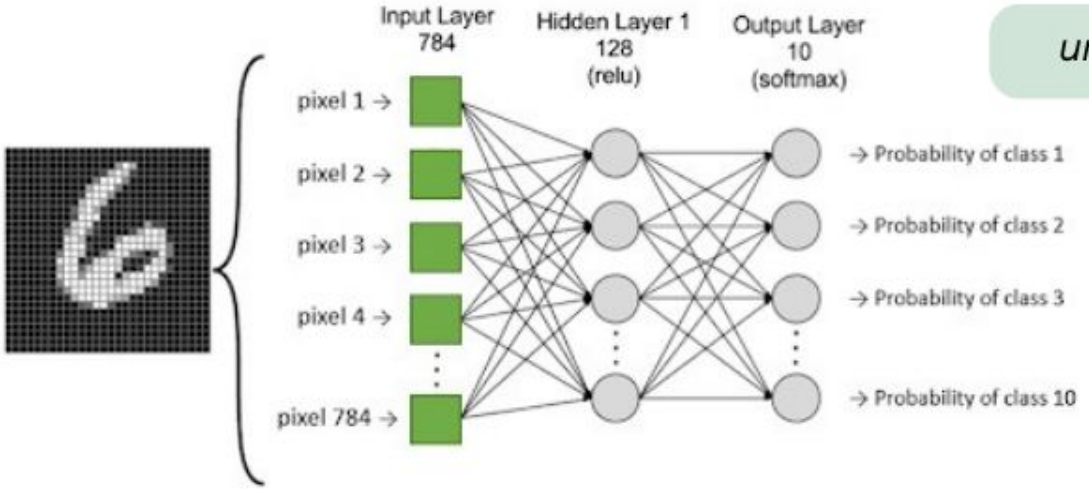
$$\Rightarrow \begin{cases} x = \dfrac{x'h_{11} + y'h_{12} + h_{13}}{x'h_{31} + y'h_{32} + 1} \\ y = \dfrac{x'h_{21} + y'h_{22} + h_{23}}{x'h_{31} + y'h_{32} + 1} \end{cases}$$

2 équations / couple de points
*8 inconnues : **4 couples** pour H*

- (H4) Terrain filmé tel que les sommets de la longueur opposée soient "au dessus" des autres



Détection d'objet = Classification + Localisation



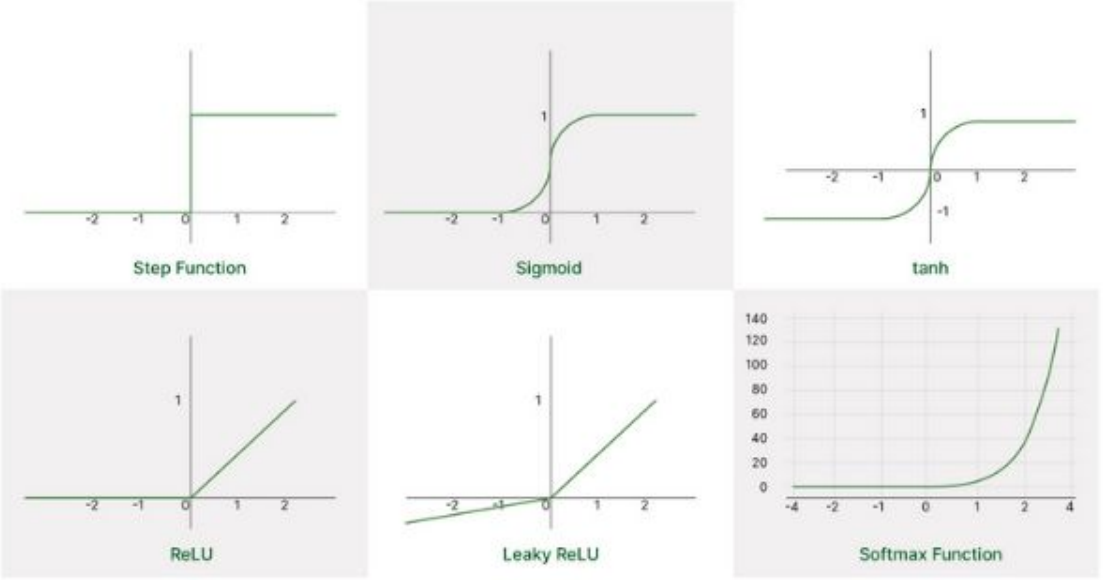
underfitting - overfitting

Réseau de neurones

© Datascientest.com

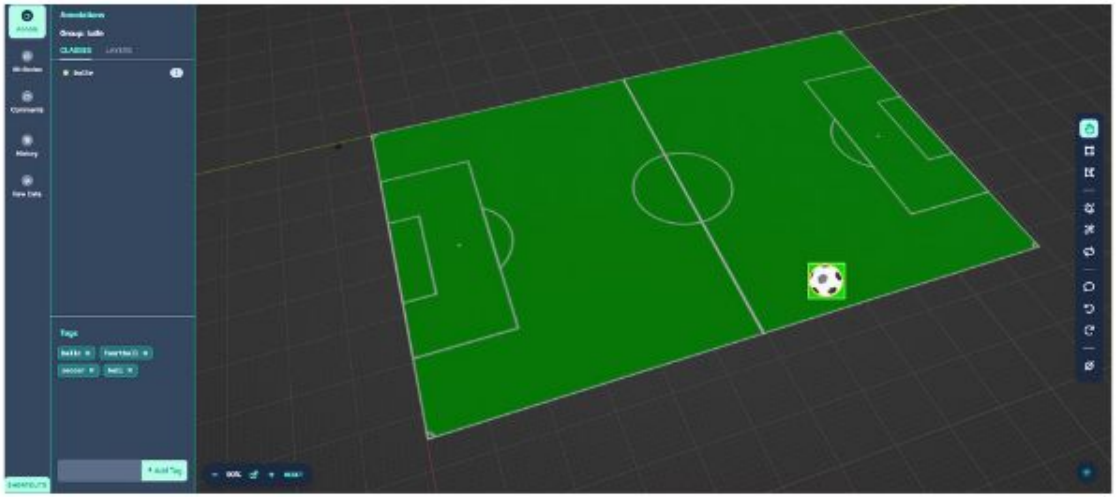
Feed Forward -->

Back propagation <--



Fonctions d'activation

© Shiksha.com



Interface d'annotation *Roboflow*

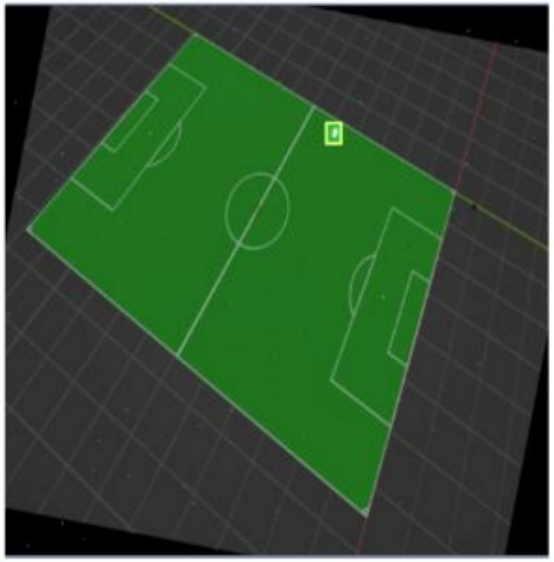


Image "augmentée"

base de données :

- Images générées par CAO (blender)
- Photos

1. Annotation images

2. Augmentation de la base de données :

- Rotation
- Exposition, Saturation
- Flou, bruit

3. Entraînement

mAP : mean Average Precision

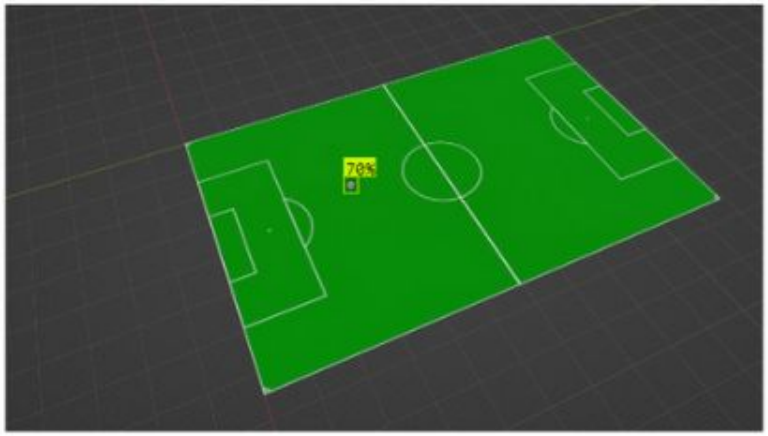
$$precision = \frac{VP}{VP + FP}$$

Boxloss localisation + dimensionnement

Classloss classe

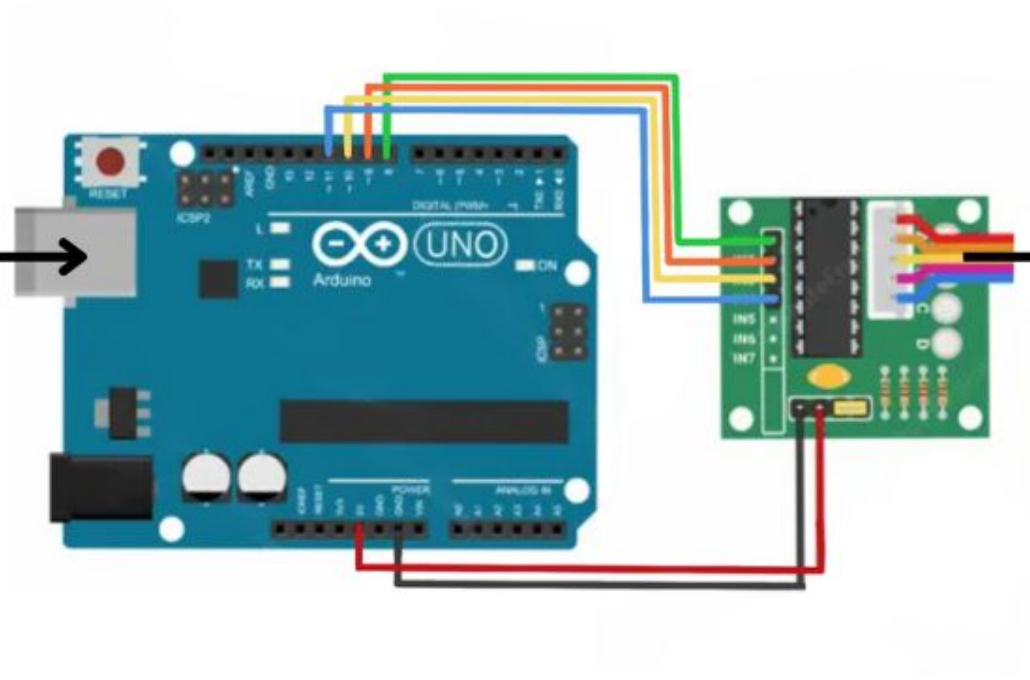
Objectloss confiance

Exemple de détection

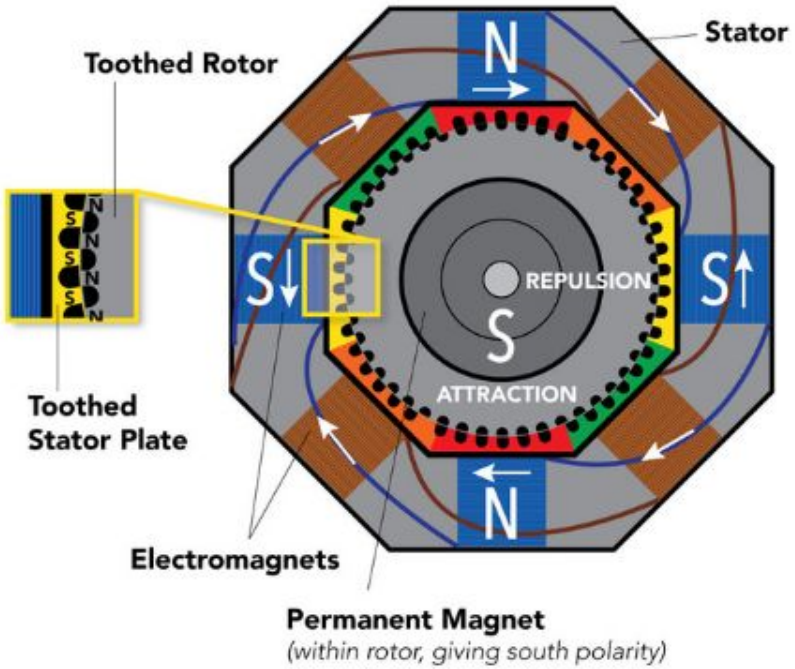



```
{
  "predictions": [
    {
      "x": 233.5,
      "y": 121.5,
      "width": 9,
      "height": 11,
      "confidence": 0.703,
      "class": "balle",
      "class_id": 0,
      "detection_id": "2b3972
    }
  ]
}
```

Ordinateur →

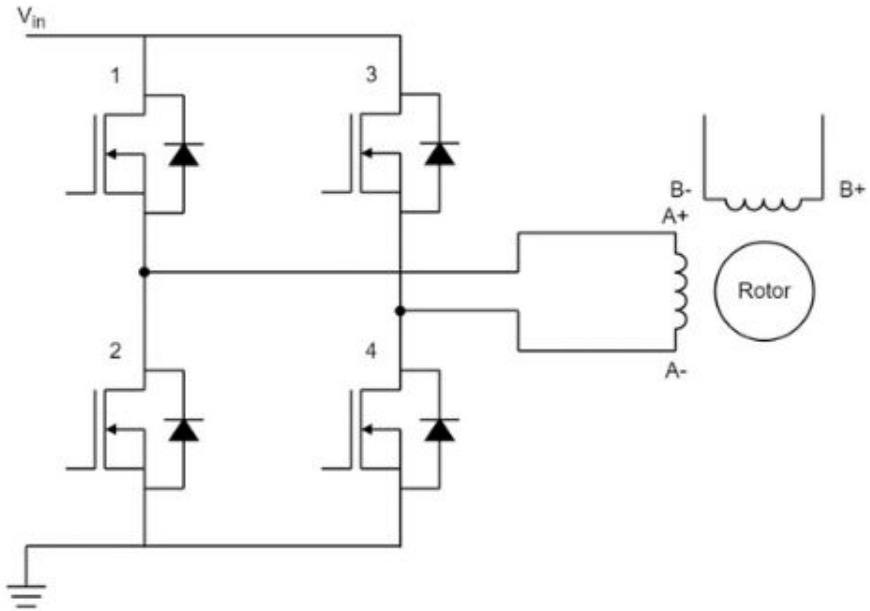


→ Moteur pas à pas



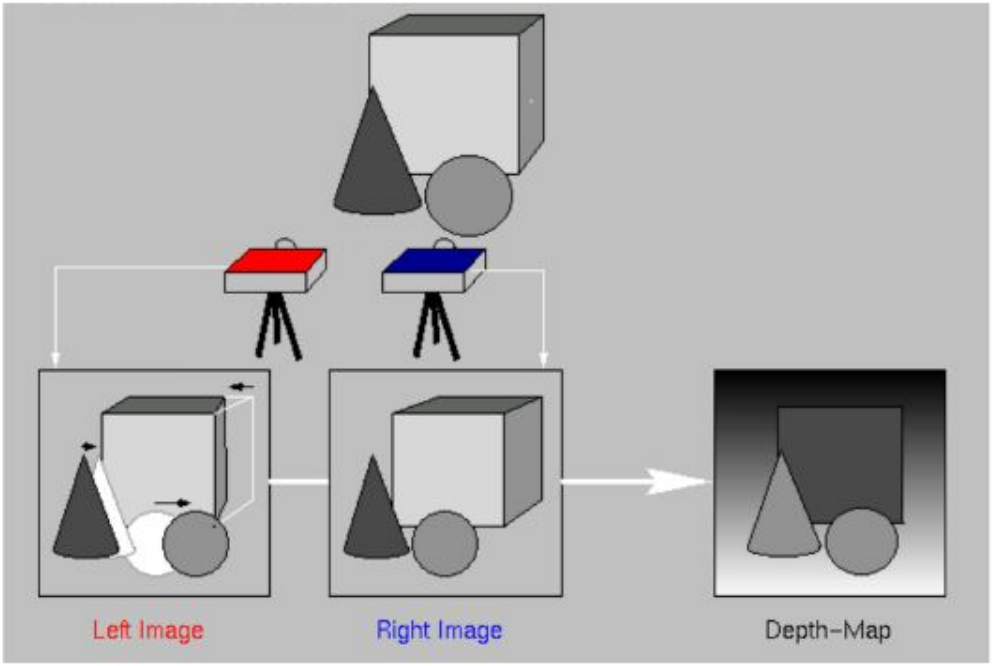
Moteur pas à pas

© Clippard.com



Moteur pas à pas

© Monolithic Power Systems



Stéréovision

© Rolf Henkel

Caméra Veo

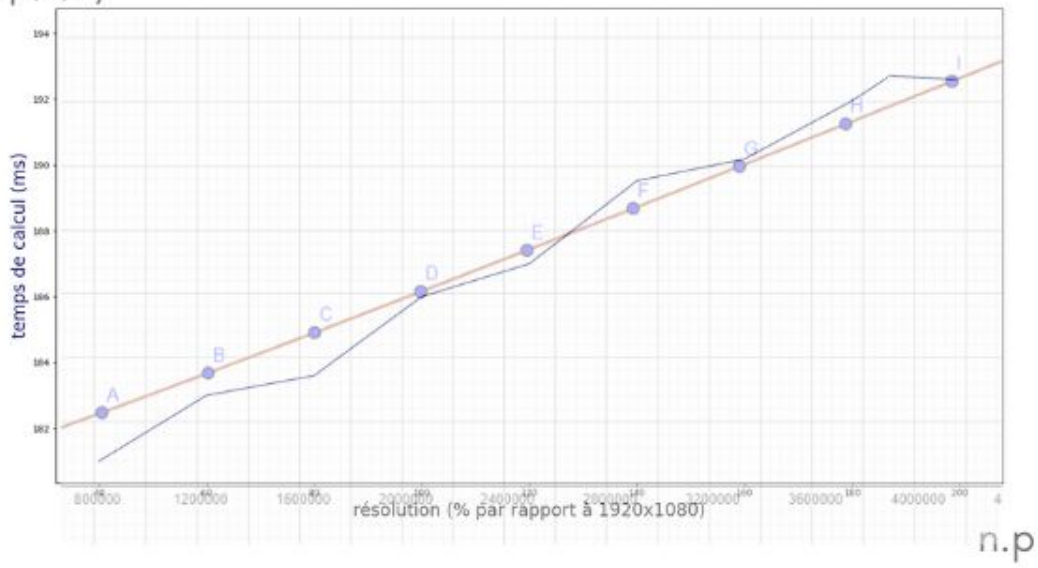
© Veo

Etude de la fonction position x(image) :

o(n.p[log(n.p)+k]+k.c.log(c))

- n.p : taille image
- k : nombre contours détectés (à priori 1)
- c : nombre de points dans les contours

f(n.p,k,c)



Superposition courbes vitesse expérimentale (Python) et fonction complexité (Python / GeoGebra)

[1] **FIFA : The incredible technology in World Cup ball** : https://www.fifa.com/fifaplus/en/watch/weFbPlyOvE2WX4b-1hJzPw, à partir de 42s, consulté le 17/10/23

[2] MANON KOK, JEROEN D. HOL, THOMAS B. SCHÖN : **Using Inertial Sensors for Position and Orientation Estimation : Foundations and Trends in Signal Processing** : Vol. 11: No. 1-2 (2017), p.70-71

[3] TAKEHIRO YAMAZAKI, MASAO SHIMIZU : **A Measuring Method for 3D Trajectory using Straightforward Triangulation** : College of Science and Technology, Nihon University : Conference draft program (2018), p.293

[4] ASSOCIATION FOR ADVANCING AUTOMATION : **Usages pour les moteurs pas-à-pas** : https://www.automate.org/blogs/what-kinds-of-applications-are-best-for-stepper-motors, consulté le 21/10/23

[5] ACADÉMIE DE LIMOGES : **Systèmes de transformation de rotation en translation** : http://pedagogie.ac-limoges.fr/sti_si/accueil/FichesConnaissances/Sequence3SSi/co/S3B22_Association_modele_composant_8.html, consulté le 23/10/23

[6] IBM : **Apprentissage supervisé ou non supervisé** : https://www.ibm.com/blog/supervised-vs- unsupervised-learning/ consulté le 15/11/23