Rapport de conception
Projet KERAAL - Groupe 10

Equipe: Baumstimler Philippe, Ma Ziqi, Aziz Ghaddab Mohamed

ENSTA

IP PARIS

Sommaire

1. Introduction

Le rapport de conception du projet KERAAL est un document qui explique et détaille la démarche et le cheminement de pensées qui ont amené aux solutions techniques et logiciels choisies.

Une première partie traite des réalisations faites par l'équipe informatique. Cela prend en compte la programmation du robot mais aussi la conception de l'algorithme d'analyse des écarts de détection de squelette dans une vidéo. Une deuxième partie est dédiée à la partie mécanique du projet, qui rassemble la phase d'assemblage du robot et des choix techniques réalisés, mais comprend également la phase de conception du système de changement de position par vérin électrique.

Ce rapport détaille également les difficultés rencontrées tout au long du projet, et les solutions choisies pour y remédier.

2. Partie Informatique

La partie informatique du projet KERAAL est consacrée à la programmation du robot mais aussi à l'analyse de l'algorithme de détection de squelette dans une vidéo. En effet, l'objectif de ce projet est d'adapter le robot POPPY à une utilisation pour la kinésithérapie. Pour cela, le robot doit être capable d'observer un patient, de déterminer ce qu'il réalise et de le corriger si nécessaire. Il existe de nombreux algorithmes capables de donner le squelette d'un homme dans une vidéo, et nous souhaitons ici analyser les écarts entre les algorithmes du marché par rapport à celui que nous utilisons, et qui se nomme Blaze Pose. Le travail du pôle informatique est donc essentiel à l'avancement du projet.

2.1. Programmation du robot

Le robot POPPY qui nous est fourni est codé en python. Cependant, les connaissances en python nécessaires pour programmer le robot ne sont pas très élevées puisque le système POPPY possède ses propres librairies, nous n'avons donc rien fait de particulier mis à part créer des fonctions python tests disponible sur le drive dans la section *"Informatique/Test_function"* qui permettent de vérifier le fonctionnement du robot. Pour plus de détails sur les fonctions tests et leurs résultats, allez lire directement le rapport de tests.

Pour bien comprendre comment programmer le robot, il faut se rendre sur le site POPPY et/ou dans le manuel d'utilisation qui décrit comment accéder au robot et quelles sont les fonctions principales POPPY et leur utilité. Voici le lien du site si le besoin se fait sentir : https://docs.poppy-project.org/en/installation/

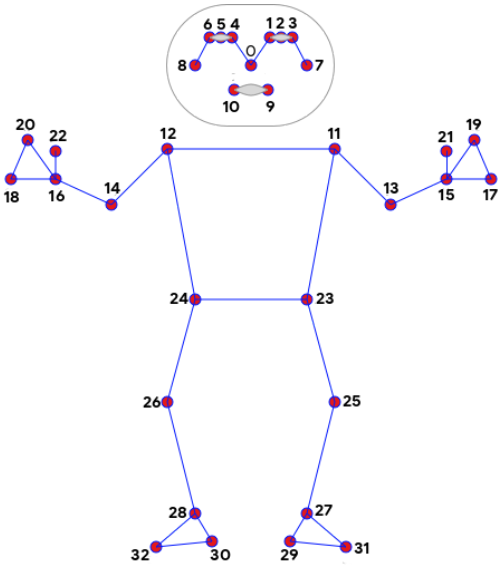
2.2. Code écart

Cette section est dédiée à la conception de l'algorithme d'analyse d'écart entre notre l'algorithme BlazePose que nous utilisons, et les squelettes générés par Kinect et OpenPose.

2.2.1. Traitement le squelette en utilisant le BlazePose

Commençons tout d'abord par l'analyse d'une vidéo par l'algorithme BlazePose. Nous avons à notre disposition une petite base de données fournie par Mme Nguyen qui contient des vidéos d'exercices de kinésithérapie, dans un environnement idéal. On utilise BlazePose pour obtenir le squelette associé dans chaque vidéo. L'algorithme est disponible dans annexe (5.1) et dans la section *"Informatique/Error/Code".* Un fois executé sur chacune des vidéos, les informations de l'articulation sont alors stocké dans la variable POSE_LANDMARK Chaque point de repère se compose des éléments suivants :

- x et y : coordonnées du point de repère normalisées à [0.0, 1.0] par la largeur et la hauteur de l'image respectivement.
- z : Représente la profondeur du point de repère avec la profondeur au milieu des hanches comme origine, et plus la valeur est petite, plus le point de repère est proche de la caméra. La magnitude de z utilise à peu près la même échelle que x.
- visibilité : une valeur dans [0.0, 1.0] indiquant la probabilité que le point de repère soit visible (présent et non occulté) dans l'image.
  L'ordre des articulations est indiqué dans le graphe ci-dessous:



Landmarks

0. Nose
1. Left_eye_inner
2. Left_eye
3. Left_eye_outer
4. Right_eye_inner
5. Right_eye
6. Right_eye_outer
7. Left_ear
8. Right_ear
9. Left_mouth
10. Right_mouth
11. Left_shoulder
12. Right_shoulder
13. Left_elbow
14. Right_elbow
15. Left_wrist
16. Right_wrist
17. Left_pinky
18. Right_pinky
19. Left_index
20. Right_index
21. Left_thumb
22. Right_thumb
23. Left_hip
24. Right_hip
25. Left_knee
26. Right_knee
27. Left_ankle
28. Right_ankle
29. Left_heel
30. Right_heel
31. Left_foot_index
32. Right_foot_index

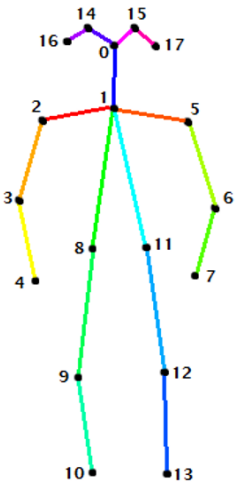
Pour chaque frame, on traite que les informations de x, y et z de l'articulation et on les met dans un dictionnaire:

dic["nom de landmark"] = (x,y,z)

Pour chaque vidéo, on met toutes les frames dans un grand dictionnaire, et on les stocke sous forme de '.json'. Et voilà, maintenant nous avons toutes les informations associées aux squelettes générés par BlazePose.

2.2.2. Comparaison du squelette de Blazepose, Openpose et Kinect

Maintenant, il nous faut comparer le squelette obtenu aux autres squelettes. Tout d'abord, squelette de Openpose nous est donné par l'encadrant et utilise le modèle suivant:

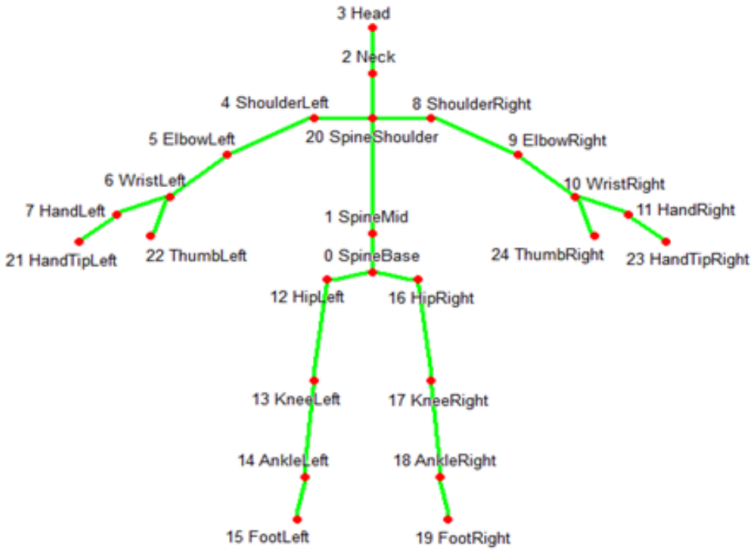


0. Head
1. mShoulder
2. rShoulder
3. rElbow
4. rWrist
5. lShoulder
6. lElbow
7. lWrist
8. rHip
9. rKnee
10.rAnkle
11.lHip
12.lKnee
13.lAnkle

Un point de repère possède les caractéristiques suivantes:

- x et y : coordonnées du point de repère normalisées à [0.0, 1.0] par la largeur et la hauteur de l'image respectivement.

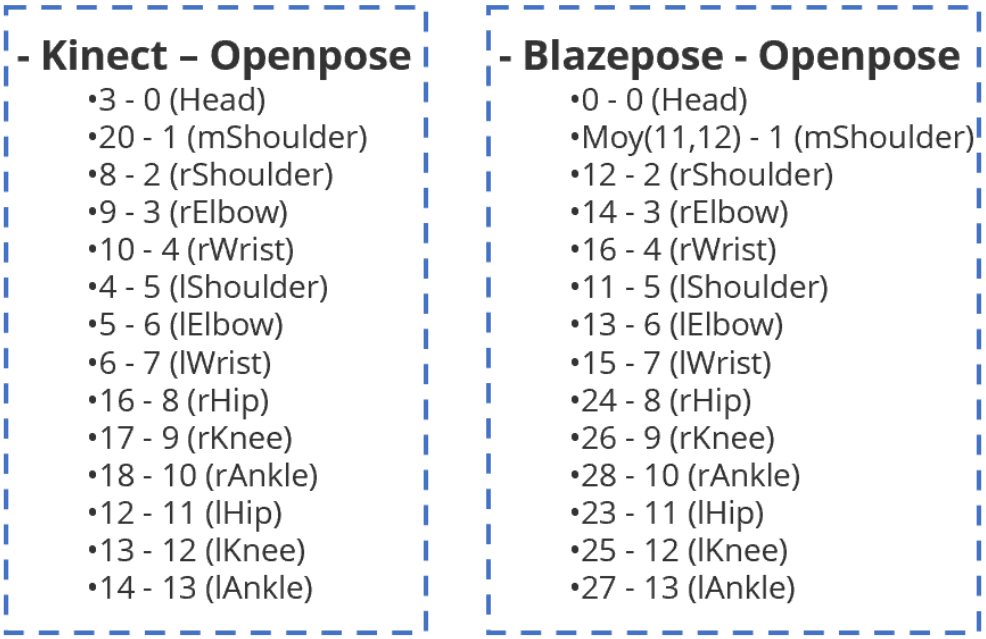
De même, le squelette de Kinect est donnée par l'encadrant et possède la forme suivante:



Chaque point de repère se compose des caractéristiques suivantes:

- x_pos, y_pos, z_pos, x_quat, y_quat, z_quat, w_quat, ce dont on a besoin, c'est le (x_pos, y_pos, z_pos). Comme le (x_pos, y_pos, z_pos) est dans le repère caméra, il faut unifier le repère.

Pour faciliter la comparaison, on transpose le repère de Blazepose et le repère de Kinect par rapport au repère de Openpose, et nous obtenons les relations correspondantes suivantes:



Ainsi par exemple, le point 3 du squelette Kinect correspond au point 0 du squelette OpenPose. De même, le barycentre des points 11 et 12 du squelette BlazePose correspond au point 1 du squelette OpenPose.

On souhaite maintenant réaliser la fonction d'écart entre les squelettes. Comme Blazepose et Openpose utilisent le repère de l'image, une simple relation de transfert sera suffisante pour la comparaison. L'écart se fait de manière immédiate par comparaison des positions sur l'image des points que l'on considère équivalent.

Cependant on observe un cas pathologique pour le squelette de la Kinect. En effet, les deux algorithmes ne sont pas basés sur le même, ce qui fait qu'on ne peut pas les comparer directement. On doit procéder à une transformation un peu plus élaborée.
Pour changer le repère de Kinect par rapport au repère OpenPose, on choisit le point mShoulder comme point de référence et la distance de 'Head-mShoulder' comme distance de référence dans chaque repère. Ainsi, afin de comparer la position de deux points équivalent des deux squelettes, on compare la distance de ces deux points par rapport au point de référence, et on normalise par rapport à la distance de référence. On obtient alors une valeur dans [0;1] que l'on peut comparer.

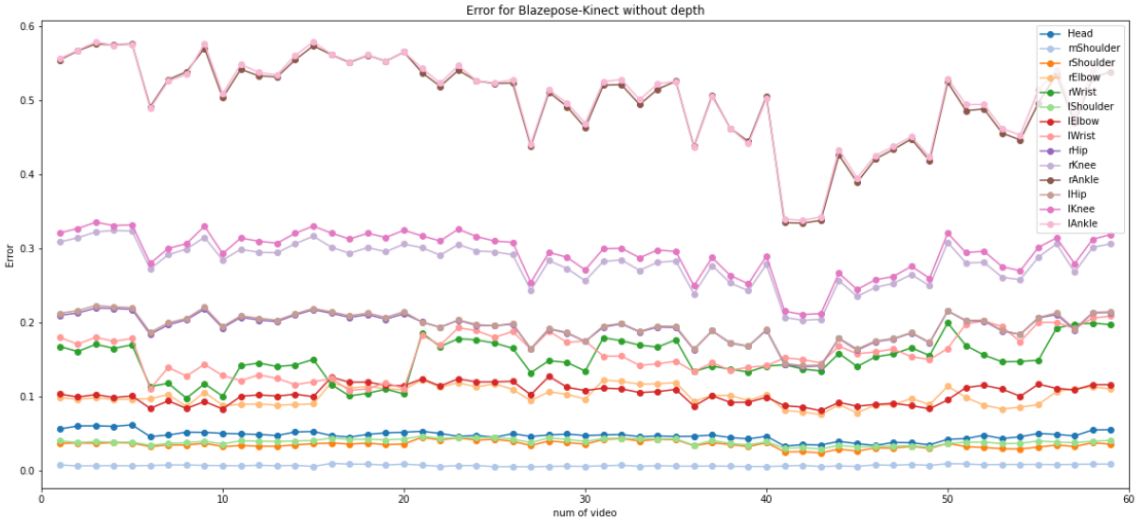
Il suffit alors, pour chaque articulation, de calculer l'écart moyen dans une vidéo selon les relations de transfert décrites précédemment, et de tracer les courbes de toutes les articulations pour toutes les vidéos dans un graphe.

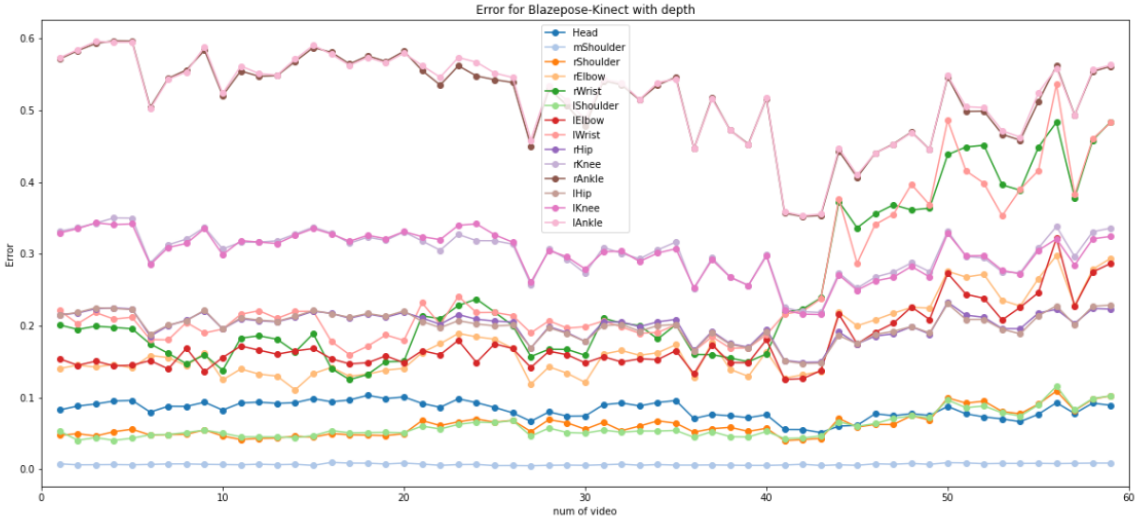
2.3. Analyse des écarts et interprétation

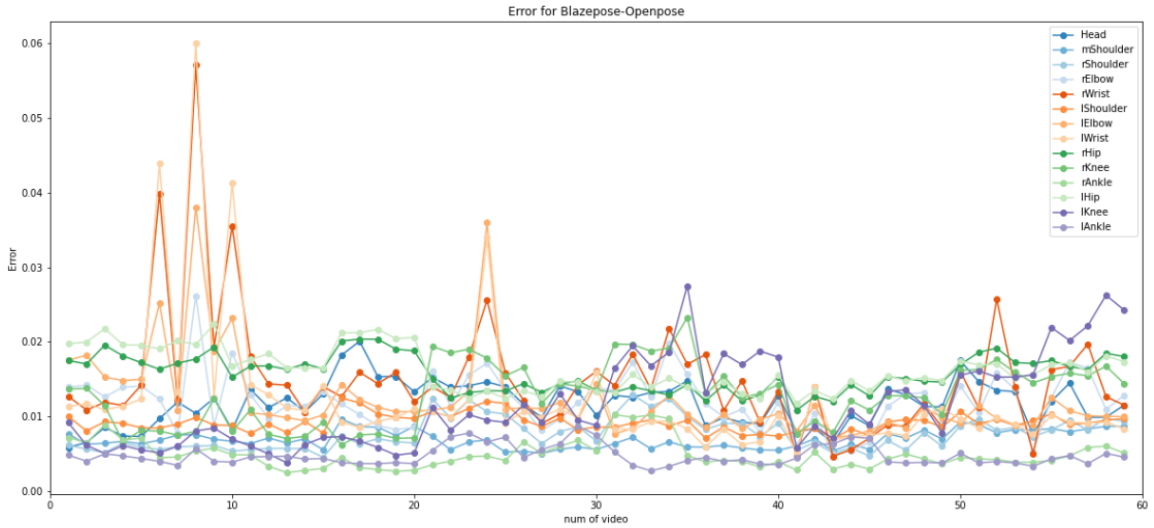
L'algorithme permettant de faire cette analyse est disponible sous la forme d'un Jupyter Notebook, disponible sur le lien suivant:
https://drive.google.com/file/d/1jSEXz2B-GkqkwDv5LRFSFRe41qBXP6F5/view?usp=sharing
Il est également disponible dans le dossier 'Error/Code'. Pour l'exécution du code, référez vous au 'Manuel d'utilisation'.

Nous avons effectué l'analyse des écarts sur trois types d'exercices: "cache tête", "étirement latéral" et "rotation du tronc". Les vidéos nous ont été fournies par Mme Nguyen. Pour chacun des exercices, nous avons plusieurs vidéos de deux catégories différentes : des exercices dit réussis par le patient et des exercices ratés par le patient. Nous ne nous intéressons pas particulièrement à cette distinction pour l'instant car nous souhaitons simplement obtenir l'écart entre les squelettes.

Voici les résultats obtenus, également disponibles sur le drive dans la section 'Error/Results'.







On observe que le squelette obtenu via Blazepose est plus proche du squelette réalisé par Openpose que celui de Kinect. Quand on compare l'écart entre Blazepose et Kinect, les écarts des points lAnkle, rAnkle, lKnee et rKnee sont très élevés comparé aux autres. On pense que cela est dû au fait que tous les mouvements soient assis, il est alors plus difficile de détecter les articulations en bas, car les mouvements en bas sont moins étirés. On observe aussi, quand on ajoute la profondeur, que les vidéos du troisième type d'exercice "rotation du tronc" ont plus d'écart qu'avant, car elles contiennent le mouvement de rotation, ce qui ajoute un écart supplémentaire selon la profondeur.

3. Partie mécanique

La partie mécanique du projet KERAAL se concentre d'une part sur l'assemblage du robot POPPY et sa configuration, et d'autre part sur la conception d'un système de changement de position permettant de soutenir et maintenir le robot à tout instant, et de changer de position en fonction des exercices. Nous décrivons l'ensemble des choix de conception et technologique réalisés pour cette partie du projet.

3.1. Assemblage du robot

Le robot POPPY est un robot humanoïde pré-conçu, et dont nous souhaitons adapter les fonctions à la réalisation d'exercices de kinésithérapie. Ce dernier possède 25 moteurs et est capable de réaliser des mouvements précis et proches de ce que pourrait faire un humain. Entièrement programmable et opensource, il est donc le candidat idéal au projet KERAAL. La première étape a donc été d'assembler le robot que nous avons reçu en pièces détachées.

Pour l'assemblage du robot nous avons suivi les consignes présentes sur le site POPPY sur le lien suivant :
https://docs.poppy-project.org/en/assembly-guides/poppy-humanoid/
On retrouve sur ce site l'inventaire du matériel nécessaire à sa construction, la nomenclature de chacun des composants et les consignes d'assemblage. Chacun des membres sur site avait lu au préalable les consignes de montage, notamment les avertissements et les risques de mauvais montage possible. Nous nous sommes alors séparés les tâches de sorte à ce que chacun d'entre nous s'occupe d'une partie du robot pour avancer au plus vite. Pour gagner en précision nous avons également utilisé des vidéos disponibles sur youtube qui montrent l'assemblage du robot.

Le plus difficile a été de ne pas être en retard, car l'assemblage est plus fastidieux qu'il n'en a l'air. De plus, nous nous sommes retrouvés à deux très vite, réduisant nos capacités de moitié. Un facteur diminuant a été la multiplication des tutoriels et guides que nous devions utiliser, le guide du site n'étant pas adapté pour les dernières versions de POPPY, notamment pour la tête et les poignets.

Nous avons également dû faire face à de nombreux problèmes techniques et qui ont fortement ralentis notre avancement:

- Nous avions deux hanches gauches, il nous a fallu attendre la réception de la nouvelle hanche pour terminer l'assemblage.
- Nous avons eu des problèmes de place dans la tête du robot. Nous n'avions pas la fixation nécessaire pour fixer la carte électronique. Nous avons résolu le problème premièrement en utilisant du scotch fort, solution à court terme.

- Nous avons eu de nombreuses erreurs d'installation du logiciel Raspberry Pi sur la carte SD. Nous avions des cartes SD de rechange sur lesquelles on a réussi à installer le programme.
- L'écran LCD rendait le fonctionnement de la carte instable. Nous avons décidé de le retirer pour l'instant.
- Nous avons eu des erreurs d'installation logiciels, que nous avons réussi à régler après plusieurs tentatives.

La résolution des erreurs logiciels se trouve directement dans le manuel d'utilisation, ainsi que les détails d'assemblage. Il ressort de cette phase la difficulté de maintenir un système aussi complexe qu'un robot humanoïde.

3.2. Tests de bon fonctionnement du robot

La réalisation d'un tel robot implique bien évidemment la création de trames de tests afin de vérifier si l'assemblage s'est bien déroulé et si le robot fonctionne comme il le devrait. La description des tests se trouve dans le rapport de tests.

Ce qui est important à remarquer c'est que nous n'avons pas réalisé l'ensemble des tests que nous avions prévu, et nous n'avons pas nécessairement validé tous les tests, et c'est tout à fait normal. Le retard accumulé dû aux problèmes techniques cités plus haut ont engendré un retard généralisé sur l'ensemble du projet. Afin d'y remédier, nous avons opté pour la réalisation des tests qui assurent le bon fonctionnement du robot. Ces tests comprennent:

- les tests de fonctionnement de la connexion avec la Raspberry Pi
- les tests de lecture de la librairie POPPY
- les tests de détection de moteurs
- les tests de mouvements moteurs par script
- les tests d'enregistrement de mouvement et leur lecture

Il s'agit là des fonctions principales de POPPY, nécessaires à son utilisation pour la kinésithérapie.

3.3. Contrôle du vérin (analyse matériel et solution)

Le robot POPPY sur lequel le projet est fondé ne possède aucun moyen pour effectuer ses transitions sans risquer de chuter et de causer une casse matérielle.

Pour répondre à cela, nous souhaitons réaliser un système mécanique de changement de position qui permettra d'assurer la stabilité du robot à tout instant de la séance d'exercices, et qui assure la transition du robot aux positions : debout, assis (sur un tabouret), assis (sur le sol jambes tendues) et allongé dos au sol.

3.3.1. Etat des lieux

La conception du système de changement de position a été enclenchée par la précédente équipe, et a abouti à la conclusion que le système serait centré autour d'un vérin électrique qui permettrait le maintien du robot à une hauteur adéquate pour chacune de ces positions. Nous avons donc à notre disposition le vérin électrique acheté par l'ancienne équipe ainsi que des supports de fixation. Un modélisation 3D de la pièce du robot POPPY modifié pour la fixer au vérin nous est également fournie.

Vérin électrique MPP-EC 90 Kg

Expédition en 48 heures

129,92 €

HT

Actionneur linéaire à moteur électrique et à vis ACME. Doté de capteur fin de course intégré non réglable au début et à la fin de la course qui permet l' arrêt de l' alimentation automatiquement.

Le changement de direction en inversant tout simplement la polarité de la tension appliquée. (interrupteur spéciaux disponible sur notre site)

La longueur de course peut être personnalisée sur demande ainsi que la vitesse et la force.

Toute augmentation de vitesse implique une perte de force. (inversement proportionnel)

Course	Force	Vitesse	Volts
1000 mm	900 N	8 mm/s	12v

https://www.motionproplus.com/verins-electrique-lineaire/10-175-verin-electrique-mpp-ec-90-kg.html#/35-course-1000_mm/37-force-900_n/44-vitesse-8_mm_s/48-volts-12v

Support de fixation MPP-SF2

Expedition sous 48 h

9,90 €

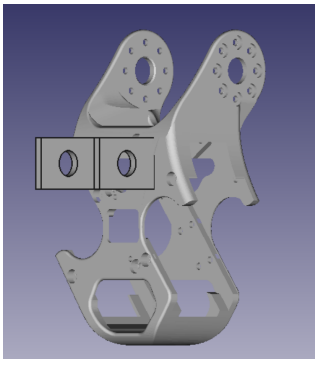
HT

Support de fixation pour vérins linéaire. Résiste à une pression de 1200 N soir 120 Kilos environ.

Compatible avec le modèle MPP-EC et Mpp-EC.

1 Ajouter au panier

https://www.motionproplus.com/supports-de-fixation/16-support-de-montage.html



L'actionneur et les fixations nous étant déjà fournis, notre équipe s'est concentrée sur la partie architecture électronique et contrôle du système.

3.3.2. Architecture électronique du système

Puisque le vérin doit être activé au bon moment et en harmonie avec les mouvements du robot, il est nécessaire de passer par une carte électronique pour effectuer la commande.

Le contrôle du vérin se fait nécessairement par le contrôle de l'intensité électrique en entrée de vérin. Le système de contrôle est donc un système automatique, connecté à l'interface pour avoir connaissance de la position actuelle, et de la position voulue.

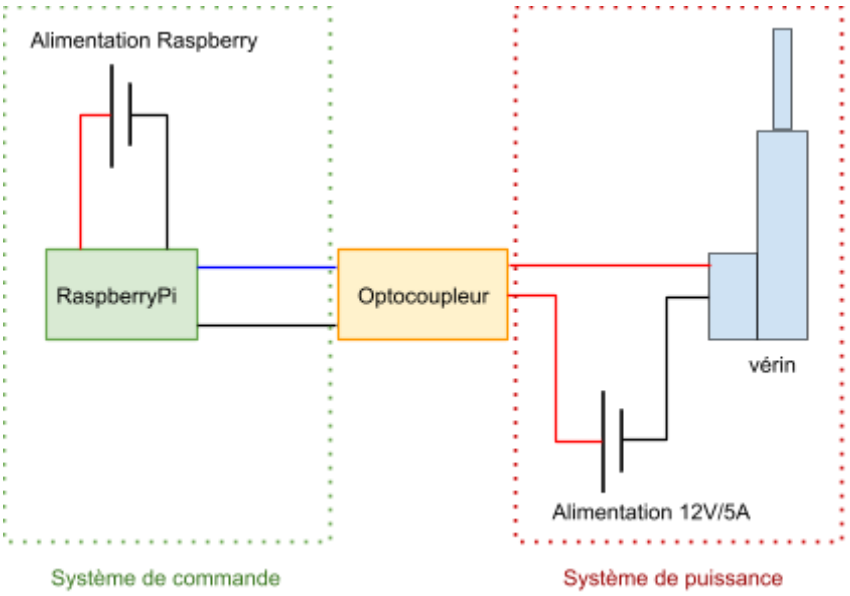
Le contrôle devra donc se faire par carte électronique. Deux possibilités s'offrent à nous :

- Connexion directe via la raspberry pi présente sur le robot POPPY : pas de surcoût lié à l'achat de nouveau matériel, aucune interface entre le système POPPY et le système de contrôle de vérin, moins encombrant. Cependant : risques liés à la surchauffe, peut engendrer des dégâts matériels sur le robot POPPY, il faut connaître la disponibilité des ports présents sur la raspberry pi, plus difficile à coder car il y a déjà du code implémenté dans la raspberry pi.

- Connexion via un système secondaire, arduino ou raspberry, s'occupant uniquement du contrôle du vérin. Moins de risques sur le système POPPY, plus facile d'accès pour de la maintenance, facilité de code, plus sûr car s'il y a un souci sur la carte de contrôle de vérin le système POPPY ne sera pas impacté. Cependant : ajoute un coût matériel, nécessite de réaliser une interface entre les deux cartes (sans-fil ou physique), plus encombrant.

Nous choisissons l'option d'utiliser la raspberry pi située dans la tête du robot POPPY, pour des questions de coûts et éviter d'augmenter la complexité de l'architecture électronique en multipliant le nombre de cartes électroniques, qui d'ailleurs ont une empreinte carbone forte que l'on souhaite réduire. Il nous faut donc relier le système de commande du robot au système de puissance du vérin électrique. Cependant, pour des questions de sécurité de matériel, il est primordial que les alimentations des deux systèmes soient isolées l'une de l'autre. En effet, la carte ne pourra pas supporter l'alimentation du vérin de 12V/5A.

Il nous faut passer par un système relais permettant de séparer le circuit d'alimentation du vérin du circuit d'information, que l'on appelle optocoupleurs. Schéma de l'architecture électrique du système:



Pour le circuit du vérin on a deux possibilités:

-Utiliser une batterie

- Meilleur mobilité, facilité d'installation
- Coût environnemental, de recyclage et de maintenance

-Utiliser le courant du secteur avec un adaptateur AC/DC

- Coût environnemental plus faible, pas de question de recyclage à court terme
- Moins de mobilité, nécessite une prise électrique

Notre choix se porte finalement sur un adaptateur secteur AC/DC 12V/5A car le robot POPPY est également branché sur secteur, la mobilité est donc déjà réduite, et le coût sera moindre.

L'optocoupleur n'est cependant qu'un "interrupteur", dans cette architecture le vérin ne sera donc capable d'aller que dans un seul sens. Afin de remédier à cela, plusieurs possibilités peuvent être imaginées, comme l'utilisation d'un pont en H contrôlé par la Raspberry Pi, ou l'utilisation d'un second optocoupleur branché dans le sens inverse sur la source d'alimentation du vérin. Malheureusement, le retard accumulé ne nous a pas permis de prendre une décision sur ce point.

Nous avons fait le choix de ne pas produire un prototype, pour ne pas laisser un montage inachevé à la prochaine équipe au vu du retard que nous avons pris. Mais afin de réduire le coût budgétaire et de dépenser une partie du budget qui nous est alloué, nous avons tout de même décidé de commander le matériel adapté à notre modèle.

3.3.3. Liste du matériel

Voici la liste du matériel commandé issu de la conception de l'architecture électronique du système de changement de position, extrait du rapport final dans la section budget.

Matériel	Fournisseur	Quantités	Prix (sans frais de port)	Etat livraison
Adaptateur AC/DC 12V/5A	Amazon	x1	25,99€	non commandé par l'ENSTA Paris
Fil électrique	Amazon	5m - 1.5mm²	4,19€	livré et réceptionné
Adaptateur connectique LED Terminal	Amazon	x5 mâles x5 femelles	7,99€	livré et réceptionné
Optocoupleur 12V	GoTronic	x1	6,60€	livré et réceptionné

L'adaptateur AC/DC non livré peut être remplacé par l'alimentation de rechange du robot POPPY, qui possède les mêmes caractéristiques.

3.3.4. Tests vérin

Nous avons cependant pris le soin de tester le fonctionnement du vérin, description disponible dans le rapport de tests. Il est intéressant également de noter que la connectique d'alimentation du vérin électrique ne ressemblait à aucune norme usuel de branchement, nous avons donc décidé de modifier cette connectique en réalisant une soudure à deux fils électriques, de sorte à pouvoir alimenter le vérin en distinguant un fil de charge positive et un fil de charge négative. La connexion à l'adaptateur AC/DC se fait alors grâce aux adaptateur connectique LED Terminal qui réalise cette transition.

3.4. Table de soutien

Le changement apporté aux exigences du projet décrit dans le rapport final joue un rôle important dans le projet. En effet, il était initialement prévu de réaliser un système de changement de position pour le robot POPPY, de sorte à ce qu'il puisse être placé sur une table. Mais nous souhaitions rendre cela le plus clair possible en clarifiant les exigences. Le but n'est pas de réaliser un système qui se pose sur une table quelconque . Chaque table

possède une hauteur et une taille différente, et il serait difficile de prendre cela en compte dans le projet au vu de la taille du vérin.

Ce qui est en réalité souhaité c'est de réaliser la table elle-même en harmonie avec le système de changement de position, de l'adapter en quelque sortes à un son propre standard. Au vu de notre retard et de l'absence de prototype, nous laissons le soin à la prochaine équipe de penser au soutien de sorte à l'adapter au système qu'elle aura imaginé.

4. Conclusion

En somme, nous avons au travers de cette année réalisé un algorithme capable de réaliser des écarts entre le squelette BlazePose et les squelettes Kinect et OpenPose, grâce à une étude approfondie des méthodes de représentation propre à chacun. Nous avons également réalisé l'assemblage du robot POPPY, à présent fonctionnel, mais dont certaines fonctionnalités n'ont pas encore été testé. Le matériel propre au système de changement de position est lui aussi fonctionnel, mais nous laissons la liberté au prochain groupe de réaliser une conception différente plutôt que de fournir un prototype mi-achevé à cause du retard que nous avons accumulé, décrit dans le rapport final.

5. Annexes

5.1. Algorithme BlazePose

```
import cv2
import mediapipe as mp
import os
import numpy as np
import json
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_pose = mp.solutions.pose

#Process all Video, note the position x,y,z
method = "../Skeleton_Blazepose"
if not os.path.exists(method):
    os.mkdir(method)
videosAnon = os.listdir("../VideosAnon")
for action in videosAnon:
    action_path_read = os.path.join("../VideosAnon",action)
    action_path_write = os.path.join(method,action)
    if not os.path.exists(action_path_write):
        os.mkdir(action_path_write)
    trail_dir = os.listdir(action_path_read)
    for trail in trail_dir:
        trail_path_read = os.path.join(action_path_read,trail)
        trail_path_write = os.path.join(action_path_write,trail)
        if not os.path.exists(trail_path_write):
            os.mkdir(trail_path_write)
        video_dir = os.listdir(trail_path_read)
        for video in video_dir:
            file_path_read = os.path.join(trail_path_read,video)
            file_path_write =
os.path.join(trail_path_write,video.split('.')[0]+'.txt')
            frame_len = 1
            dic = {'positions':{}}
            print(file_path_read)
            print(file_path_write)
            # file_to_write=open(file_path_write,"w+")
            cap = cv2.VideoCapture(file_path_read)
            with mp_pose.Pose(
                min_detection_confidence=0.5,
                min_tracking_confidence=0.5) as pose:
                while cap.isOpened():
```

```
                    success, image = cap.read()
                    if not success:
                        print("Ignoring empty camera frame.")
                        # If loading a video, use 'break' instead of
'continue'.
                        break
                    # Flip the image horizontally for a later
selfie-view display, and convert
                  # the BGR image to RGB.
                    image = cv2.cvtColor(cv2.flip(image, 1),
cv2.COLOR_BGR2RGB)
                  # To improve performance, optionally mark the image
as not writeable to
                  # pass by reference.
                    image.flags.writeable = False
                    results = pose.process(image)

                    if not results.pose_landmarks:
                        continue
                    # keypoints = [0 for i in range(33*3)]
                    # print(results.pose_landmarks.landmark)
                    len_fin = 3 #we stoke the x,y,z dimension.
                    frame = {
                            "Nose":[0 for i in range(len_fin)],
                            "Left_eye_inner":[0 for i in
range(len_fin)],"Left_eye":[0 for i in
range(len_fin)],"Left_eye_outer":[0 for i in range(len_fin)],
                            "Right_eye_inner":[0 for i in
range(len_fin)],"Right_eye":[0 for i in
range(len_fin)],"Right_eye_outer":[0 for i in range(len_fin)],
                            "Left_ear":[0 for i in
range(len_fin)],"Right_ear":[0 for i in range(len_fin)],"Mouth_left":[0
for i in range(len_fin)],"Mouth_right":[0 for i in range(len_fin)],
                            "Left_shoulder":[0 for i in
range(len_fin)],"Right_shoulder":[0 for i in
range(len_fin)],"Left_elbow":[0 for i in
range(len_fin)],"Right_elbow":[0 for i in range(len_fin)],
                            "Left_wrist":[0 for i in
range(len_fin)],"Right_wrist":[0 for i in
range(len_fin)],"Left_pinky":[0 for i in
range(len_fin)],"Right_pinky":[0 for i in range(len_fin)],
                            "Left_index":[0 for i in
range(len_fin)],"Right_index":[0 for i in
```

```
range(len_fin)],"Left_thumb":[0 for i in
range(len_fin)],"Right_thumb":[0 for i in range(len_fin)],
                            "Left_hip":[0 for i in
range(len_fin)],"Right_hip":[0 for i in range(len_fin)],"Left_knee":[0
for i in range(len_fin)],"Right_knee":[0 for i in range(len_fin)],
                            "Left_ankle":[0 for i in
range(len_fin)],"Right_ankle":[0 for i in
range(len_fin)],"Left_heel":[0 for i in range(len_fin)],"Right_heel":[0
for i in range(len_fin)],
                            "Left_foot_index":[0 for i in
range(len_fin)],"Right_foot_index":[0 for i in range(len_fin)]
                            }
                    j=0 #Avoid appends
                    skeleton = list(frame.keys())
                    # print(skeleton)
                    for data_point in results.pose_landmarks.landmark:
#Keypoints contain all of the landmarks
                        frame[skeleton[j]][0]=data_point.x
                        frame[skeleton[j]][1]=data_point.y
                        frame[skeleton[j]][2]=data_point.z
                        j+=1
                    dic["positions"][str(frame_len)] = frame
                    frame_len += 1
                # print(dic)
                with open(file_path_write, 'w') as json_file:
                    json.dump(dic, json_file)


                cap.release()

                json_file.close()
```