ISTITUTO STATALE SUPERIORE ERNESTO BALDUCCI

Istituto tecnico tecnologico Ernesto Balducci
Indirizzo Informatica e robotica

ELABORATO PER L'ESAME DI STATO

Candidato:
Vincenzo Marturano

Anno Scolastico 2020/2021

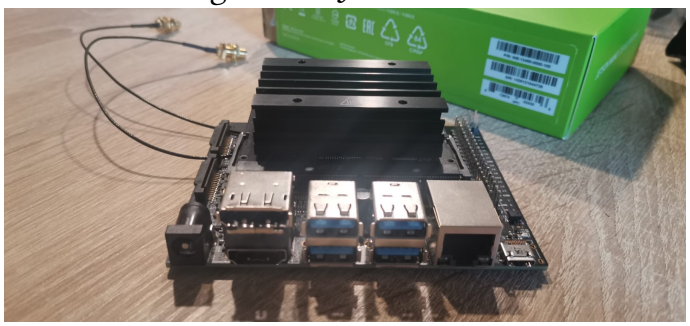
Contents

Chapter 1

Hardware Argos

1.1 Computer di bordo

Figure 1.1: Jetson nano



Il computer di bordo che ho scelto per la realizzazione del robot è il jetson nano della nvidia che è veramente un'ottimo sbc(Singol Board Computer) per applicazioni di robotica e intelligenza artificiale grazie alla sua GPU.
Le specifiche sono:

- GPU: 128-core Maxwell™ GPU
- CPU: quad-core ARM® Cortex®-A57 CPU
- RAM: 4GB 64-bit LPDDR4
- Memoria di massa: Micro SD
- Video:
  - Encode: 4K @ 30 (H.264/H.265)
  - Decode: 4K @ 60 (H.264/H.265)
- interfacce
  - Ethernet: 10/100/1000BASE-T
  - Camera: 2 slot MIPI-CSI2
  - USB: 4x USB 3.0, USB 2.0 (Micro USB)
- Altri: GPIO, I2C, I2S, SPI, UART

1.2 Sensori

Il robot ha a bordo dei sensori per consentirgli di avere una percezione dell'ambiente che lo circonda e in base a quello prendere delle decisioni, per esempio che movimenti compiere per evitare degli ostacoli o navigare all'interno di una mappa con dei punti di interesse.

1.2.1 Telecamera 3D

Figure 1.2: Realsense D435i

La Realsense D435i è una telecamera 3d sviluppata da Intel e fornisce due stream, uno RGB e uno per la profondita(Depth).
Lo Stream depth è come un'immagine, soltanto che ogni pixel ha un valore che rappresenta la distanza al posto di informazioni di colore.
Il funzionamento si basa su due telecamere per fornire una visione stereo e un proiettore IR per migliorare l'accuratezza del depth stream

1.2.2 IMU

IMU, acronimo di Inertial Measurement Unit è un sensore in grado di misurare accelerazione, velocità angolare e campo magnetico, ogni misura viene fatta su un sistema a 3 assi, quindi in totale avremmo 9 variabili che esprimono la posizione nello spazio.

1.3 Alimentazione

L'alimentazione avviene attraverso una batteria ai polimeri di litio costituita da due celle in serie, ciascuna da 4.2V per una tensione totale di 8.4V, visto che il computer di bordo necessita di una tensione di 5V ho impiegato un buck converter per abbassare la tensione da 8.4V a 5V.
Gli attuatori sono direttamente collegati alla batteria perché possono lavorare a 8.4V.
Ho scelto una batteria da due celle perché mi è sembrata la scelta migliore visto

che se avessi utilizzato una batteria con una tensione più elevata avrei dovuto usare due buck converter introducendo delle inefficienze nel sistema.

1.4 Attuatori

Figure 1.3: Servomotori a bus seriale lx-225

Come attuatori ho scelto dei servo a bus seriale, questi hanno molti vantaggi rispetto ai servomotori convenzionali.
Il primo vantaggio è che offrono della telemetria, quindi è possibile ottenere informazioni utili sullo stato del servomotore.
Inoltre offrono dei sistemi di protezione contro overheating e stallo del rotore.
Le informazioni ottenibili sono:

- tensione di alimentazione
- temperatura
- posizione
- errori

Un altro vantaggio è la possibilità di connetterli fra di loro a catena risparmiando sulla lunghezza dei cavi e ottenendo un cablaggio più pulito.

Chapter 2

Meccanica Argos

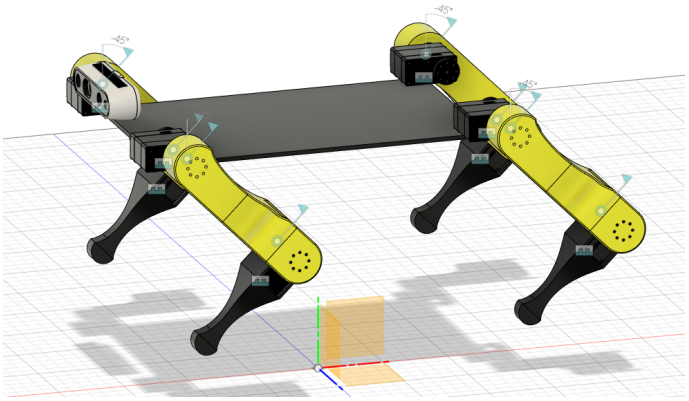
2.1 Realizzazione

La meccanica è stata sviluppata da zero con l'ausilio del CAD Autodesk Fusion360. Poi è stata realizzata con la stampa 3D in PLA.
Il robot ha 12Dof ovvero 12 giunti rotazionali, questo introduce complessità dal punto di vista del design ma permette al robot di fare dei movimenti
fluidi e complessi.
Il design meccanico è stato influenzato dalla fase successiva, ovvero la realizzazione del modello cinematico del robot, ho scelto di avere il femore e la tibia della gamba della stessa lunghezza, ovvero 10cm, in modo tale da semplificarmi i calcoli.

2.2 Primo prototipo

L'approccio che ho utilizzato per la realizzazione della meccanica è stato di tipo bottom-up, prima ho realizzato una gamba e poi a partire da quella ho disegnato il resto.

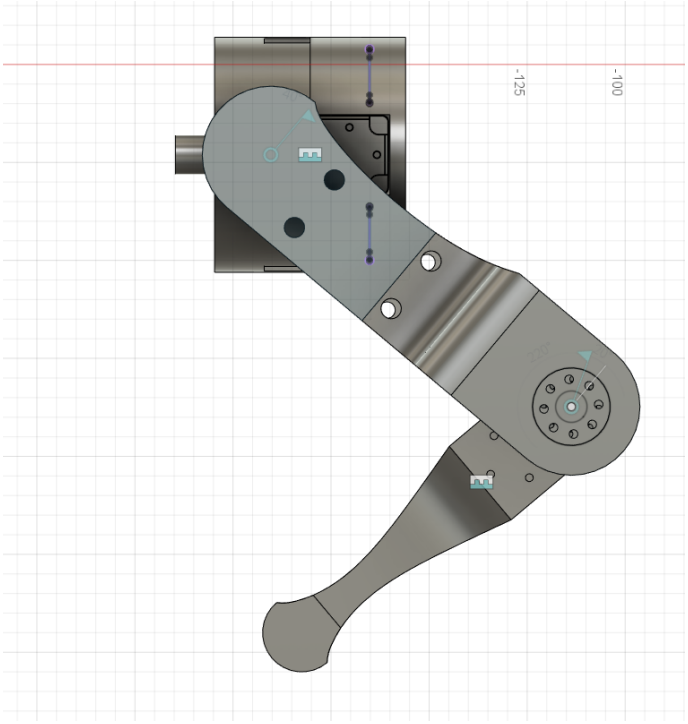
Figure 2.1: prima idea



2.2.1 Gamba

Una volta messi a fuoco i principali aspetti del design ho proceduto
alla realizzazione di una gamba che rispettasse le specifiche decise in precedenza.

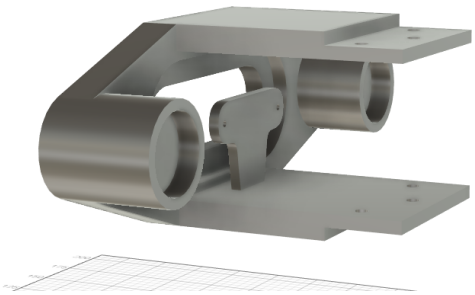
Figure 2.2: gamba



2.2.2 Parte frontale

La parte frontale è stata disegnata in modo da poter supportare la telecamera 3d ed alloggiare i cuscinetti che servono al robot per muoversi.

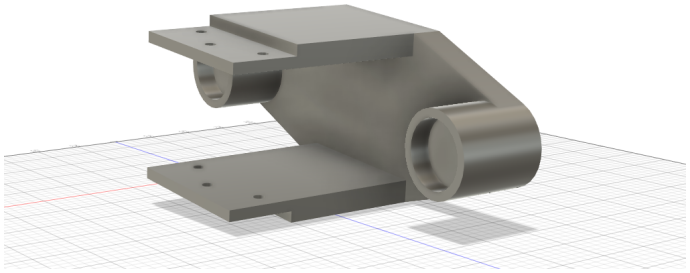
Figure 2.3: parte frontale



2.2.3 Parte dietro

Questo pezzo è identico a quello frontale soltanto che non ha l'alloggiamento per la telecamera

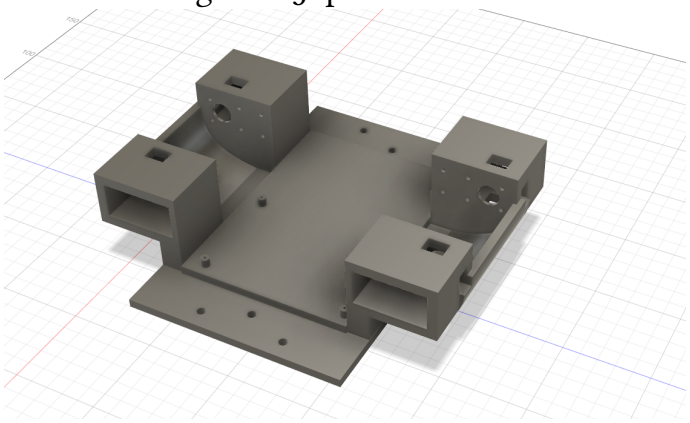
Figure 2.4: parte dietro



2.2.4 Parte centrale

Questo è il pezzo che collega tutto assieme ed è stato il più difficile sia da disegnare che da realizzare con la stampa 3D. Alloggia i quattro servomotori delle anche e i vari circuiti elettronici del robot.

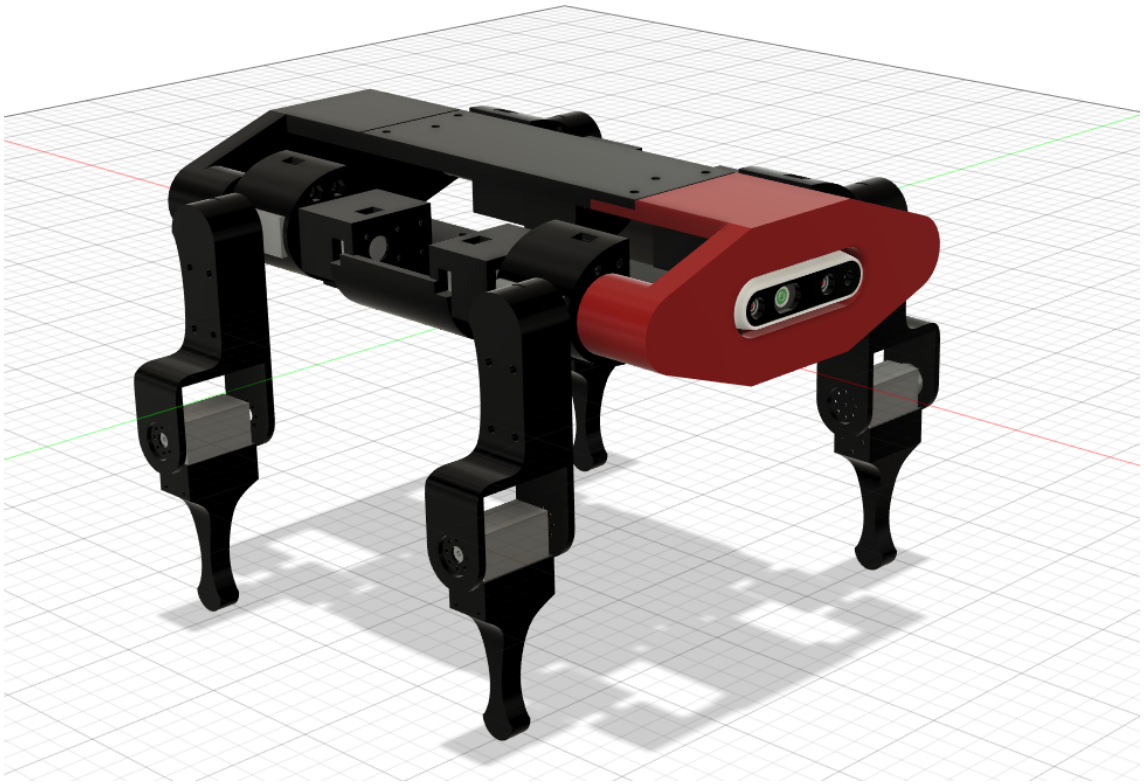
Figure 2.5: parte centrale



2.2.5 Risultato finale

Dopo diverse iterazioni del design questo è il risultato finale.

Figure 2.6: risultato finale



Chapter 3

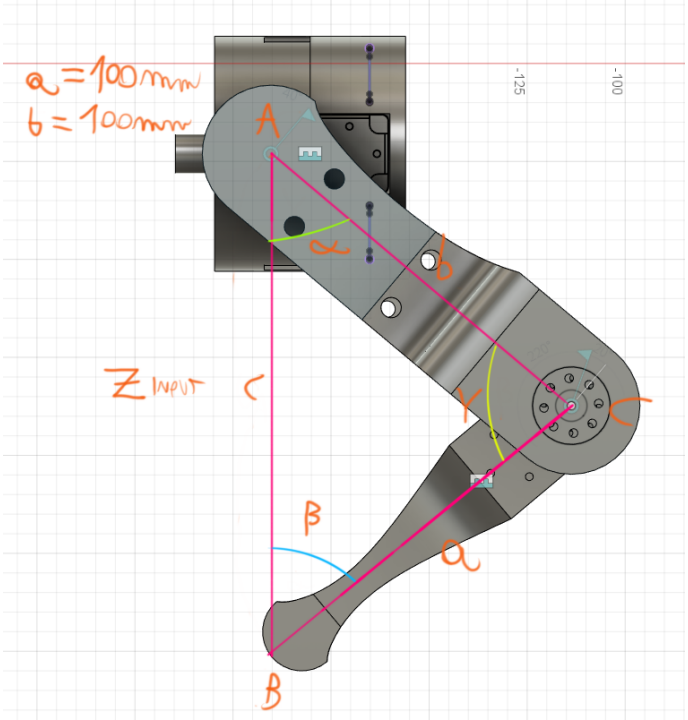
Modello Cinematico Argos

3.1 cosa é la cinematica inversa?

La cinematica inversa è un processo attraverso il quale si determinano gli angoli da dare ad un oggetto articolato, per soddisfare il raggiungimento di una posa desiderata.
Nel mio caso per la risoluzione ho utilizzato delle strategie algebriche che prevedono l'utilizzo di equazioni trigonometriche.

3.2 traslazione sull'asse Z

Figure 3.1: Gamba del robot con rappresentazione geometrica



L'obbiettivo è quello di calcolare gli angoli α β γ data una certa altezza Z in input e per fare ciò si utilizza il teorema del coseno

3.2.1 risoluzione dell'angolo α

$\cos\alpha = \frac{b^2+c^2-a^2}{2bc}$
da cui $\alpha = \arccos\frac{b^2+c^2-a^2}{2bc}$ facendo così si ottiene l'angolo α in radianti

3.2.2 risoluzione dell'angolo β

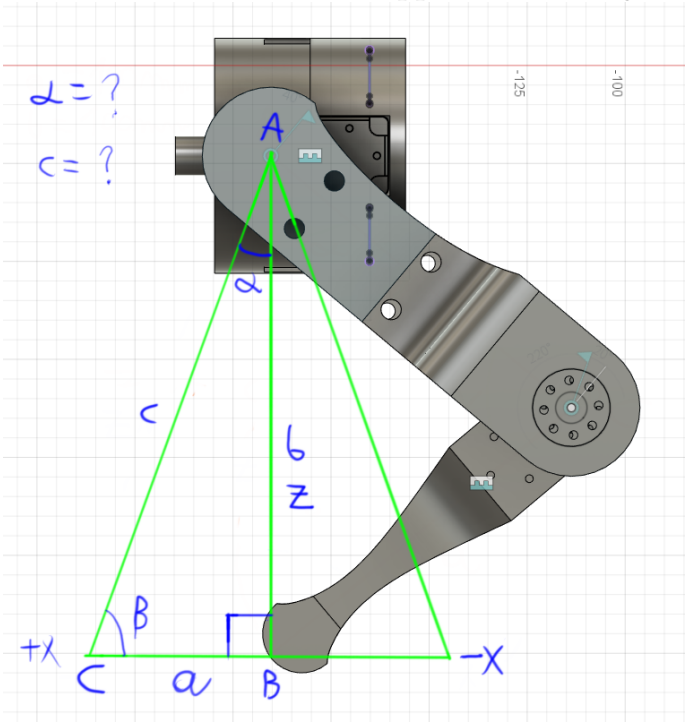
L'angolo β è uguale all'angolo α visto che i lati a e b del triangolo hanno la stessa dimensione.

3.2.3 risoluzione dell'angolo γ

Successivamente per calcolare l'angolo γ si deve semplicemente fare $\pi - (\alpha \cdot 2)$ visto che la somma degli angoli in un triangolo deve fare π
Facendo cosi si ottiene un movimento fluido con la punta del piede che sta sempre sotto la spalla.

3.3 traslazione sull'asse X

Figure 3.2: Gamba del robot con rappresentazione geometrica



L'obbiettivo è quello di calcolare l'angolo α e l'ipotenusa c del triangolo data una certa altezza Z e una distanza X in input.
Per fare ciò si utilizzano i teoremi trigonometrici sul triangolo rettangolo, in particolare ho utilizzato il terzo e il secondo.

3.3.1 risoluzione dell'angolo α

Utilizzando il terzo teorema ho ricavato dalla formula $a = b \cdot \tan \alpha$
$\tan \alpha = \frac{X}{Z}$

da cui $\alpha = \arctan \frac{X}{Z}$

3.3.2 risoluzione dell'ipotenusa c

Utilizzando il secondo teorema ho ricavato dalla formula $b = c \cdot \cos \alpha$
$c = \frac{Z}{\cos \alpha}$

3.4 traslazione sull'asse Y

Figure 3.3: Gamba del robot con rappresentazione geometrica

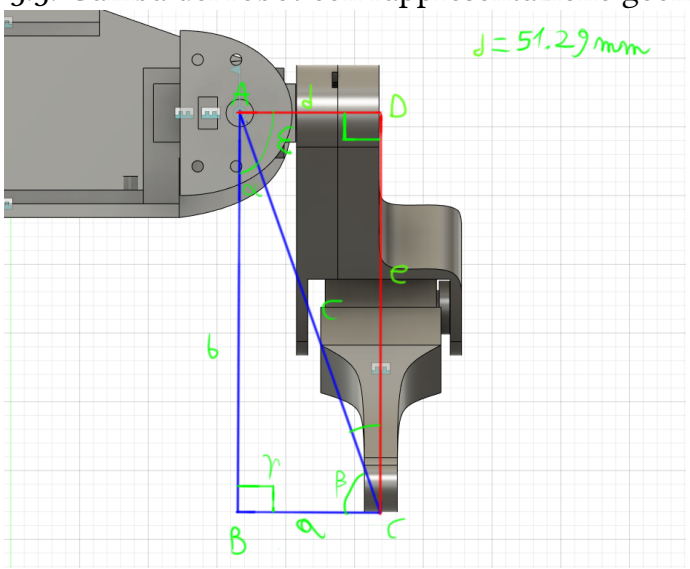
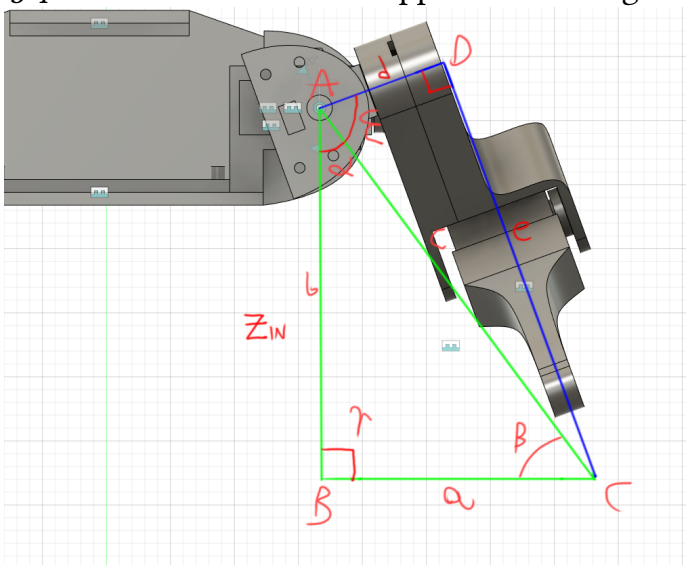


Figure 3.4: Gamba del robot con rappresentazione geometrica



L'obbiettivo è quello di calcolare gli angoli α ed ϵ e l'ipotenusa e (che diverra' la nuova estensione della gamba), data una certa altezza Z e una distanza Y in input.
Per fare ciò si ricorre dinuovo ai teoremi trigonometrici sul triangolo rettangolo.

3.4.1 risoluzione angolo α

Utilizzando il terzo teorema ho ricavato dalla formula $a = b \cdot \tan \alpha$
$\tan \alpha = \frac{a}{b}$
da cui $\alpha = \arctan \frac{a}{b}$

3.4.2 risoluzione dell'ipotenusa c

Utilizzando il secondo teorema ho ricavato dalla formula $b = c \cdot \cos \alpha$
$c = \frac{b}{\cos \alpha}$

3.4.3 risoluzione dell'angolo ϵ

Utilizzando il secondo teorema ho ricavato dalla formula $d = c \cdot \cos \epsilon$
$\cos \epsilon = \frac{d}{c}$
da cui $\epsilon = \arccos \frac{d}{c}$

3.4.4 risoluzione dell cateto e

Utilizzando il primo teorema
$e = c \cdot \sin \epsilon$

3.5 risultato finale

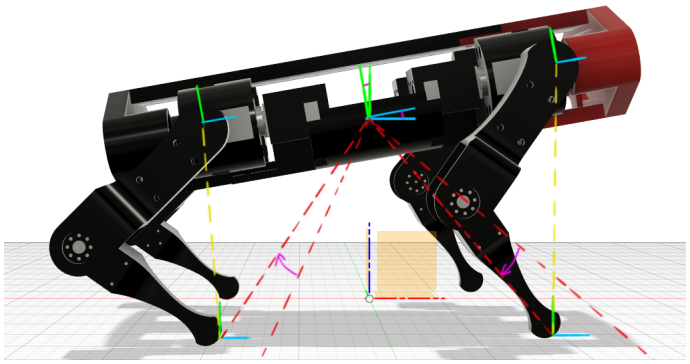
```
def computeLegIk(legid,x,y,z,roll,pitch,yaw):
    z*=-1
    x*=-1
    y*=-1
    #y axis first triangle
    xOff=0.05129

    if legid==1 or legid == 3:
        y=y*-1

    y=y+xOff


    alpha=math.atan(y/z)
    hyp=z/math.cos(alpha)
    #y axis second triangle
    beta=math.acos(matematica.checkDomain(xOff/hyp))
    z2=hyp*math.sin(beta)
    A=alpha+beta-1.5707963267948968
    #si sottrae l'angolo di quando il robot e'
    #a [0,0,0] rotazione [0,0,0] traslazione
    #x axis
    sh=math.atan(x/z2)
    z3=z2/math.cos(sh)
    #z axis
    cosA=math.pow(doggo_config.tightLength,2)+math.pow(z3,2)
    -math.pow(doggo_config.shankLength,2)
    k=2*doggo_config.tightLength*z3
    c=cosA/k
    B=math.acos(matematica.checkDomain(c))
    C=math.pi-(B*2)-math.pi
    B+=sh
    return A,B,C
```

3.6 modello cinematico

L'obbiettivo è quello di muovere il corpo del robot in relazione ai piedi, nel robot ci sono quattro coordinate di riferimento per ogni gamba più una che fa da riferimento per tutte, per ogni gamba le più importanti sono la prima, che sarebbe quella dell'anca e l'ultima che è la punta del piede. Con la cinematica inversa possiamo localizzare la posizione del piede nello spazio 3D rispetto alla coordinata di riferimento dell'anca. il procedimento è il seguente:

- si ruota e trasla il corpo attraverso quattro punti(le anche) per metterlo nella posa che si desidera
- si prende il vettore dal centro del corpo alla punta del piede
- si ricava il vettore necessario per la cinematica inversa sottraendo il vettore centro-punta del piede dal vettore centro-anca
- si rimuove la trasformazione al vettore della cinematica inversa affinché i piedi stiano per terra, visto che il vettore dell'anca é trasformato dopo la sottrazione, anche il vettore della cinematica inversa subisce la stessa trasformazione.

Figure 3.5: rotazione su asse y



```
import matematica
from ik import computeLegIk
import numpy as np
class kinematic:
    def __init__(self):
        self.length=0.0348
        self.width=0.076
        self.heigth=0.10
        self.xoffCoxa=0.05129
        self.bodyToRFcoxa=
        np.array([0.09965,-(self.width/2+self.xoffCoxa),0])
```

```
        self.bodyToLFcoxa=
        np.array([0.09965,self.width/2+self.xoffCoxa,0])
        self.bodyToRHcoxa=
        np.array([-0.09965,-(self.width/2+self.xoffCoxa),0])
        self.bodyToLHcoxa=
        np.array([-0.09965,self.width/2+self.xoffCoxa,0])
        self.bodyToFootRF=
        np.array([0.09965,-(self.width/2+self.xoffCoxa),self.heigth])
        self.bodyToFootLF=
        np.array([0.09965,self.width/2+self.xoffCoxa,self.heigth])
        self.bodyToFootRH=
        np.array([-0.09965,-(self.width/2+self.xoffCoxa),self.heigth])
        self.bodyToFootLH=
        np.array([-0.09965,self.width/2+self.xoffCoxa,self.heigth])

    def solve(self,orn,pos,bodytoFeet):
        bodyToFootRF =
        np.asarray([bodytoFeet[0,0],bodytoFeet[0,1],bodytoFeet[0,2]])
        bodyToFootLF =
        np.asarray([bodytoFeet[1,0],bodytoFeet[1,1],bodytoFeet[1,2]])
        bodyToFootRH =
        np.asarray([bodytoFeet[2,0],bodytoFeet[2,1],bodytoFeet[2,2]])
        bodyToFootLH =
        np.asarray([bodytoFeet[3,0],bodytoFeet[3,1],bodytoFeet[3,2]])
        orn[0]*=-1
        orn[2]*=-1
        _bodyToRFcoxa=matematica.trasforma(self.bodyToRFcoxa,orn,pos)
        _bodyToLFcoxa=matematica.trasforma(self.bodyToLFcoxa,orn,pos)
        _bodyToRHcoxa=matematica.trasforma(self.bodyToRHcoxa,orn,pos)
        _bodyToLHcoxa=matematica.trasforma(self.bodyToLHcoxa,orn,pos)

        RFcord=bodyToFootRF-_bodyToRFcoxa
        LFcord=bodyToFootLF-_bodyToLFcoxa
        RHcord=bodyToFootRH-_bodyToRHcoxa
        LHcord=bodyToFootLH-_bodyToLHcoxa

        undoOrn= -orn
        undoPos= -pos

        _RFcord=matematica.trasforma(RFcord,undoOrn,undoPos)
        _LFcord=matematica.trasforma(LFcord,undoOrn,undoPos)
        _RHcord=matematica.trasforma(RHcord,undoOrn,undoPos)
        _LHcord=matematica.trasforma(LHcord,undoOrn,undoPos)
        #print(RFcord)
        A,B,C=computeLegIk(0,_RFcord[0],_RFcord[1],_RFcord[2],0,0,0)
        D,E,F=computeLegIk(1,_LFcord[0],_LFcord[1],_LFcord[2],0,0,0)
        G,H,I=computeLegIk(2,_RHcord[0],_RHcord[1],_RHcord[2],0,0,0)
```

```
L,M,N=computeLegIk(3,_LHcord[0],_LHcord[1],_LHcord[2],0,0,0)

_bodytoFeetRF=_bodyToRFcoxa+_RFcord
_bodyTofeetLF=_bodyToLFcoxa+_LFcord
_bodytoFeetRH=_bodyToRHcoxa+_RHcord
_bodyTofeetLH=_bodyToLHcoxa+_LHcord
_bodyTofeet=np.matrix([
    _bodytoFeetRF,
    _bodyTofeetLF,
    _bodytoFeetRH,
    _bodyTofeetLH
])
return np.array([A,B,C,D,E,F,G,H,I,L,M,N]),_bodyTofeet
```

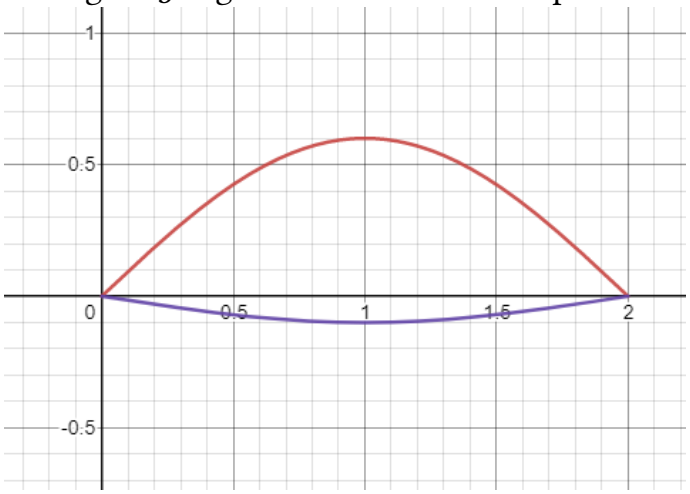
3.7 Come fa il robot a camminare?

Con l'ausilio del modello cinematico si può creare un sistema che varia nel tempo la posizione dei piedi, descrivendo delle traiettorie che permettono al robot di camminare.

3.7.1 Traiettoria di un passo

Per descrivere la traiettoria di un passo ho usato due sinusoidi, una delle quali è sfasata di mezzo periodo rispetto all'altra e di ampiezza minore.
La traiettoria di un passo è divisa in due fasi una di sollevamento e una di appoggio, La fase di sollevamento è descritta dall'equazione $p \sin(\frac{\pi}{k}x)\{0 < x < k\}$ la fase di appoggio è descritta dall'equazione $n \sin(\frac{\pi}{k}x - \pi)\{0 < x < k\}$
dove k è la lunghezza del passo e p,n sono rispettivamente ampiezza della fase di sollevamento e ampiezza della fase di appoggio.

Figure 3.6: grafico traiettoria di un passo



```
import numpy as np
import math
def stepTrajectorySwing(phi,sa,ss):
    x=(phi*ss)
    z=sa*np.sin((math.pi/ss)*x)
    return np.asarray([x,0,z])
def stepTrajectoryStance(phi,sa,ss):
    phi=1-phi#si sottrae 1 perche' deve partire
    #dall'ultimo punto della traiettoria
    x=(phi*ss)
    z=sa*np.sin(math.pi/(ss)*x-np.pi)
    return np.asarray([x,0,z])

def stepTrajectory(phi,ss,sa,stepOff):
    cord=np.zeros([0,3])
    if phi > 1:
        phi=phi-1
    if phi<= stepOff:
        phiStance = phi/stepOff
        cord=stepTrajectoryStance(phiStance,sa,ss)
    else:
        phiSwing=(phi-stepOff)/(1-stepOff)

        cord=stepTrajectorySwing(phiSwing,sa,ss)
    return cord
```

3.7.2 Gait Planner

Per fare camminare il robot è necessario fare quattro loop che vanno da 0 a 1 e definire gli offset fra le gambe e fra la fase in cui il piede viene alzato e quella che spinge per andare in avanti.

```
import numpy as np
from gait_controller import stepTrajectory
from time import time
from doggo_config import defaultPose
class gait_planner:
    def __init__(self):
        self.phi=0
        self.lastTime = 0
        self.bodytoFeet=defaultPose.copy()
        self.prevPose=defaultPose.copy()
        self.contact=np.zeros(4)
        print(self.contact)
    def loop(self,bodytoFeet_,sa,ss,T,offset,rot):
        if T <= 0.01:
            T = 0.01
```

```
if(self.phi>0.99):
    self.lastTime= time()
self.phi = (time()-self.lastTime)/T
stepcoord=stepTrajectory(self.phi+offset[0],sa,ss,0.75)
stepcoord[0]=stepcoord[0]*np.cos(rot)-stepcoord[1]*np.sin(rot)
stepcoord[1]=stepcoord[0]*np.sin(rot)-stepcoord[1]*np.cos(rot)
self.bodytoFeet[0,0] =  defaultPose[0,0] + stepcoord[0]
self.bodytoFeet[0,1] =  defaultPose[0,1] + stepcoord[1]
self.bodytoFeet[0,2] =  defaultPose[0,2] + stepcoord[2]

stepcoord=stepTrajectory(self.phi+offset[1],sa,ss,0.75)
stepcoord[0]=stepcoord[0]*np.cos(rot)-stepcoord[1]*np.sin(rot)
stepcoord[1]=stepcoord[0]*np.sin(rot)-stepcoord[1]*np.cos(rot)
self.bodytoFeet[1,0] =  self.prevPose[1,0] + stepcoord[0]
self.bodytoFeet[1,1] =  self.prevPose[1,1] + stepcoord[1]
self.bodytoFeet[1,2] =  self.prevPose[1,2] + stepcoord[2]

stepcoord=stepTrajectory(self.phi+offset[2],sa,ss,0.75)
stepcoord[0]=stepcoord[0]*np.cos(rot)-stepcoord[1]*np.sin(rot)
stepcoord[1]=stepcoord[0]*np.sin(rot)-stepcoord[1]*np.cos(rot)
self.bodytoFeet[2,0] =  self.prevPose[2,0] + stepcoord[0]
self.bodytoFeet[2,1] =  self.prevPose[2,1] + stepcoord[1]
self.bodytoFeet[2,2] =  self.prevPose[2,2] + stepcoord[2]

stepcoord=stepTrajectory(self.phi+offset[3],sa,ss,0.75)
stepcoord[0]=stepcoord[0]*np.cos(rot)-stepcoord[1]*np.sin(rot)
stepcoord[1]=stepcoord[0]*np.sin(rot)-stepcoord[1]*np.cos(rot)
self.bodytoFeet[3,0] =  self.prevPose[3,0] + stepcoord[0]
self.bodytoFeet[3,1] =  self.prevPose[3,1] + stepcoord[1]
self.bodytoFeet[3,2] =  self.prevPose[3,2] + stepcoord[2]
if(self.phi<0.5):
    y=(-0.003*ss)*(self.phi)
    self.contact=np.asarray([0,1,1,0])
else:
    y=(0.003*ss)*(self.phi)
    self.contact=np.asarray([1,0,0,1])
trans=np.asarray([y,y,0])
print(self.contact)
return self.bodytoFeet,trans
```

3.8 Miglioramenti

Un robot a quattro gambe è un sistema altamente dinamico e instabile, per migliorare la camminata andrebbe fatto un sistema che tenga traccia del centro di massa e che garantisca che la sua proiezione sia dentro il poligono di supporto descritto dalle zampe che toccano terra se sono più di due, mentre se sono due deve garantire che la sua proiezione sia sulla retta descritta dai due punti nella quale le gambe toccano terra.

Chapter 4

Software Argos

4.1 ROS

Per implementare il software del robot ho utilizzato ROS, che è ormai diventato uno standard in robotica.
ROS è un framework per lo sviluppo di applicazioni robotiche.
E' un sistema distribuito quindi ogni nodo può essere eseguito su host diversi.

4.2 nodi

4.2.1 Hardware Handler

Questo è il nodo di ros che funziona come layer di astrazione per interfacciarsi con l'hardware.
Utilizza due thread per comunicare, attraverso una seriale USB con i controller dei servomotori.
Riceve in input gli angoli da dare ai servomotori dal nodo argos Controller.

4.2.2 Argos Controller

Questo nodo riceve in input la posa del robot e, utilizzando il modello cinematico del robot, calcola gli angoli da dare ai giunti rotazionali e li pubblica.

4.2.3 Argos Keyboard Teleop

Questo nodo riceve input dalla tastiera e li invia all'Argos Controller.

4.3 Connessione Wifi

Il robot è in grado di connettersi ad una rete Wi-Fi esistente o di crearne una nuova cosi da poter permettere la connessione diretta fra un PC e lui stesso.
Setup della rete Wi-Fi generata dal robot.
nmcli con add type wifi ifname wlano mode ap con-name ARGOSNET ssid ARGOS
nmcli con modify ARGOSNET 802-11-wireless.band bg
nmcli con modify ARGOSNET 802-11-wireless.channel 1
nmcli con modify ARGOSNET 802-11-wireless-security.key-mgmt wpa-psk
nmcli con modify ARGOSNET 802-11-wireless-security.proto rsn
nmcli con modify ARGOSNET 802-11-wireless-security.group ccmp
nmcli con modify ARGOSNET 802-11-wireless-security.pairwise ccmp
nmcli con modify ARGOSNET 802-11-wireless-security.psk argosnet
nmcli con modify ARGOSNET ipv4.method shared
nmcli con up ARGOSNET