sketch_22_3DOFPlanare

```
float q1=0;
float q2=0.0;
float q3=0.0;
float q1g=0.0;
float q2g=0.0;
float q3g=0.0;
float angolo1=0.0;
float angolo2=0.0;
float angolo3=0.0;
int link1=100;
int link2=100;
int link3=100;
float angolo1v=0.0;
float angolo2v=0.0;
float angolo3v=0.0;
float phi=0.0;

float x=0;
float y=0;
int target_r=255;
int target_g=255;
int target_b=255;
float K =0.1;
float gomito = 1.0;
float gomitoN=0.0;
float gomitoG=0.0;
float kk=0.00001;

void setup()
{
 size(1000, 1000);
 background(#45E6EA);
}

void draw()
```

```
{
 newton();
 gradiente();

 //Leggi di controllo
 angolo1v=angolo1v+K*(angolo1-angolo1v);
 angolo2v=angolo2v+K*(angolo2-angolo2v);
 angolo3v=angolo3v+K*(angolo3-angolo3v);

 background(#45E6EA);
 details();

 pushMatrix();
 translate(500, 500);
 fill(255);
 circle(0, 0, 2*(link1+link2+link3));

 pushMatrix();
 translate(x, y);
 //Punto da raggiungere
 target(50);
 popMatrix();

 treDOF(angolo1, angolo2, angolo3, 255, 183, 77, link1, link2, link3);
 popMatrix();
//Robot vero controllo proporzionale
 pushMatrix();
 translate(500, 500);
 treDOF(angolo1v, angolo2v, angolo3v, 0, 255, 0, link1, link2, link3);
 popMatrix();
//Robot che implementa l&apos;algoritmo di Newton
 pushMatrix();
 translate(500, 500);
 treDOF(q1-PI/2, q2, q3, 132, 255, 255, link1, link2, link3);
 popMatrix();
//Robot che implementa l&apos;algoritmo del gradiente
```

```
 pushMatrix();
 translate(500, 500);
 treDOF(q1g-PI/2, q2g, q3g, 88, 24, 69, link1, link2, link3);
 popMatrix();
}

void treDOF(float q1, float q2, float q3, int r, int g, int b, int l1, int l2, int l3)
{
 rotate(q1);
 link(50, 50, 100, r, g, b, l1);
 translate(0, 100);
 rotate(q2);
 link(50, 50, 100, r, g, b, l2);
 translate(0, 100);
 rotate(q3);
 link(50, 50, 100, r, g, b, l3);
}

void target(int R) {
 fill(target_r, target_g, target_b);
 circle(0, 0, R);
}

void link(int R, int larg, int lung, int r, int g, int b, int t)
{
 fill(r, g, b, t);
 circle(0, 0, R);
 circle(0, lung, R);
 rect(-R/2, 0, larg, lung);
}


void keyPressed()
{
 if (keyCode == &amp;apos;1&amp;apos;) angolo1 = angolo1+0.5;
 if (keyCode == &amp;apos;2&amp;apos;) angolo2 = angolo2+0.5;
```

```
 if (keyCode == &apos;3&apos;) angolo3 = angolo3+0.5;
 if (keyCode == &apos;7&apos;) angolo3 = angolo3-0.5;
 if (keyCode == &apos;8&apos;) angolo1 = angolo1-0.5;
 if (keyCode == &apos;9&apos;) angolo2 = angolo2-0.5;
 if (keyCode == ENTER) {
   angolo1=0;
   angolo2=0;
   angolo3=0;
 }
 if (keyCode == &apos;G&apos;) {
   gomito=-gomito;
   gomitoN++;
   gomitoG++;
   mousePressed();
 }
 if (keyCode == &apos;O&apos;) {
   phi+=0.523599;
   angolo1=0;
   angolo2=0;
   angolo3=0;
   mousePressed();
 }
 if (keyCode == &apos;P&apos;) {
   phi-=0.523599;
   angolo1=0;
   angolo2=0;
   angolo3=0;
   mousePressed();
 }
}

/*
details() è una funzione che permette la stampa di tutte le informazioni nella
finestra di esecuzione
*/
void details() {
```

```
String line="q1="+angolo1+"\nq2=+"+angolo2+"\nq3="+angolo3;
String lineV="q1v="+angolo1v+"\nq2v=+"+angolo2v+"\nq3v="+angolo3v;
String phiText="phi="+phi+"="+(angolo1+angolo2+angolo3);
String lineNewton="q1="+q1+"\nq2="+q2+"\nq3="+q3;
String lineGradiente="q1="+q1g+"\nq2="+q2g+"\nq3="+q3g;

String phiTextNewton="phi="+phi+"="+(q1+q2+q3);
String phiTextGradiente="phi="+phi+"="+(q1g+q2g+q3g);

textSize(20);

fill(0);
textLeading(20);
text(line, 5, 100);
textLeading(20);
fill(0);
text(lineV, 200, 100);
textLeading(20);
text(phiText, 5, 20);
text("Newton:"+phiTextNewton, 5, 40);
text("Gradiente:"+phiTextGradiente, 5, 60);

text("Newton", 400, 80);
text("Gradiente", 600, 80);
textLeading(20);

text(lineNewton, 400, 100);
textLeading(20);

text(lineGradiente, 600, 100);
}

/*
 La funzione newton() implementa l&apos;algoritmo di Newton che permette di stimare in maniera asintotica i parametri
 di giunto e di conseguenza risolvendo il problema di cinematica
```

diretta/inversa di un qualsiasi robot. Si basa
sulla conoscenza della matrice Jacobiana J(q) ottenuto dalle variabili x,y,phi.
VANTAGGI:
- Convergenza esponenziale -> convergenza molto veloce;
- Poco affetto da rumori ed incertezze;
SVANTAGGI:
- In vicinaza alle singolarità, il problema diventa malcondizionato poiché questo algoritmo necessita la conscenza
dell'inversa di J e quindi il suo determinante è prossimo a 0.
- In punti non appartenenti allo spazio di lavoro, il robot procederà a scatti non raggiungendo la soluzione.
Al fine di ridurre questa problematica si corregge il det(J) per valori prossimi a 0.

OSS.:
Nel caso del robot 3DOF al fine di ottenere il cambio gomito del robor si devono rispettare due specifiche:
- mantenere il robot lontano dalle singolairtà;
- portare il robot verso la seconda soluzione;
Quindi, per portare il robot verso la zona "attrattiva" della seconda soluzione evitando la singolarità, al cambio gomito
sin(q2)->-sin(q2)=sin(q2+PI).
A questo punto l'algoritmo di Newton convergerà normalmente alla soluzione richiesta,ottenendo così il risultato richiesto.
*/

```
void newton() {


 float kN=0.1;
 if (gomitoN%2!=0) {
   q2=q2+gomito*PI;

   gomitoN++;
 }
 float s_2=sin(q2);
```

```
if (abs(s_2)<0.01) {
  println("Correzione Newton");
  s_2=gomito*0.01;
}
float s_1=sin(q1);
float c_2=cos(q2);
float c_1=cos(q1);
float c_3=cos(q3);
float s_3=sin(q3);
float s_12=c_1*s_2 + c_2*s_1;
float c_12=c_1*c_2-s_1*s_2;
float s_23=c_2*s_3+s_2*c_3;
float c_23=c_2*c_3-s_2*s_3;
float
Q1=(s_12*y+c_12*x+link3*s_3*phi+((-q2-q1)*link3-link3*q3)*s_3-link3*c_
3-link1*c_2-link2)/(link1*s_2);
float
Q2=-((link2*s_12+link1*s_1)*y+(link2*c_12+link1*c_1)*x+(link1*link3*s_2
3+link2*link3*s_3)*phi+((-link1*q2-link1*q1)*link3-link1*link3*q3)*s_23-li
nk1*link3*c_23+((-link2*q2-q1*link2)*link3-link2*link3*q3)*s_3-link2*link
3*c_3-2*link1*link2*c_2-link2*link2-link1*link1)/(link1*link2*s_2);
float
Q3=(s_1*y+c_1*x+(link3*s_23+link2*s_2)*phi+((-q2-q1)*link3-link3*q3)*s
_23-link3*c_23+(-link2*q3-link2*q2-q1*link2)*s_2-link2*c_2-link1)/(link2*s
_2);
q1=q1+kN*Q1;

q2=q2+kN*Q2;

q3=q3+kN*Q3;
}

/*
La funzione gradiente() implementa l&apos;algoritmo del gradiente. Questo
algoritmo si basa, come l&apos;algoritmo di Newton
sulla conoscenza della matrice Jacobiana J(q), ma per effettuare la stima dei
```

parametri si utilizza la sua trasposta
 negata.
 L'idea è quella di percorrere la direzione del gradiente in cerca di minimi locali. Vi sono numerose implementazioni di
 questo algoritmo (ADAM,RMSProp,Antigradiente,AdaGrad etc), in questo robot è stato implementato l'algoritmo della discesa
 del gradiente in cui si sceglie la direzione massima di discesa del gradiente per ottenere il minimo locale della funzione
 del parametro.
 VANTAGGI:
  - In vicinanza delle singolarità non occorre correggere la stima;
  - Convergenza asintotica al valore vero;
 SVANTAGGI:
  - Forte dipendenza con il learning rate kk: per valori troppo alti, la stima dei parametri non converge; per valori
  troppo bassi, la convergenza diventa molto lenta.
  - Affetto da rumori;
  - La stima può essere costante nel caso in qui la derivata di q è nulla
*/
```
void gradiente() {
 if (gomitoG%2!=0) {
   q2g=q2g+gomito*PI;
   gomitoG++;
 }
 float s_2=sin(q2g);
 if(s_2>-0.9) s_2+=-0.01;
float phig =phi;
 float s_1=sin(q1g);
 float c_2=cos(q2g);
 float c_1=cos(q1g);
 float c_3=cos(q3g);
 float s_3=sin(q3g);
 float c_123=c_1*c_2*c_3-s_1*s_2*c_3-s_1*c_2*s_3-c_1*s_2*s_3;
 float s_123=-s_1*s_2*s_3+s_1* c_2*c_3+c_1*s_2*c_3+c_1*c_2*s_3;
 float s_12=c_1*s_2 + c_2*s_1;
 float c_12=c_1*c_2-s_1*s_2;
```

```
 float s_23=c_2*s_3+s_2*c_3;
 float L_3=link3;
 float L_2=link2;
 float L_1=link1;
 float
Q1g=((L_3*c_123+L_2*c_12+L_1*c_1)*y+(-L_3*s_123-L_2*s_12-L_1*s_1
)*x+phi-q3g-q2g-q1g);
 float
Q2g=((L_3*c_123+L_2*c_12)*y+(-L_3*s_123-L_2*s_12)*x+L_1*L_3*s_23
+L_1*L_2*s_2+phi-q3g-q2g-q1g);
 float
Q3g=(L_3*c_123*y-L_3*s_123*x+phi+L_1*L_3*s_23+L_2*L_3*s_3-q3g-q
2g-q1g);

   q1g=q1g+kk*Q1g;

   q2g=q2g+kk*Q2g;
   phig=phig+0.00000000000000001*Q3g;
   q3g=phig-q1g-q2g;

}

/*
 Implementazione della cinematica del 3DOF
*/
void mousePressed() {
 x=mouseX-500;
 y=mouseY-500;
 println(x, y);

 float xpolso= x-cos(phi)*link3;
 float ypolso= y-sin(phi)*link3;
 println("XPOLSO="+xpolso+"YPOLSO="+ypolso);
 if (sqrt(abs(pow(xpolso, 2)+pow(ypolso, 2)))<(link2+link1) &&
sqrt(abs(pow(xpolso, 2)+pow(ypolso, 2)))>abs(link2-link1)) {
   target_b=0;
```

```
  target_r=0;
  target_g=255;
  println("Punto raggiungibile");
 } else {
  target_b=0;
  target_r=255;
  target_g=0;
  println("Punto non raggiungibile");
 }

 float
a=(xpolso*xpolso+ypolso*ypolso-link1*link1-link2*link2)/(2*link1*link2);
 float c2=a;
 float s2=gomito*sqrt(abs(1-a*a));
 float b1=link1+c2*link2;
 float b2=link2*s2;
 float c1=b1*xpolso+b2*ypolso;
 float s1=-b2*xpolso+b1*ypolso;
 angolo1 = atan2(s1, c1)-PI/2;
 angolo2 = atan2(s2, c2);
 angolo3=phi-(angolo2+angolo1)-PI/2;
 println("Angolo1="+angolo1+"\tAngolo2:"+angolo2+"\tAngolo3:"+angolo3);
}
```