Tutorial: **Delta robot kinematics**

07-27-2009 02:13 PM
mzavatsky
Abacus

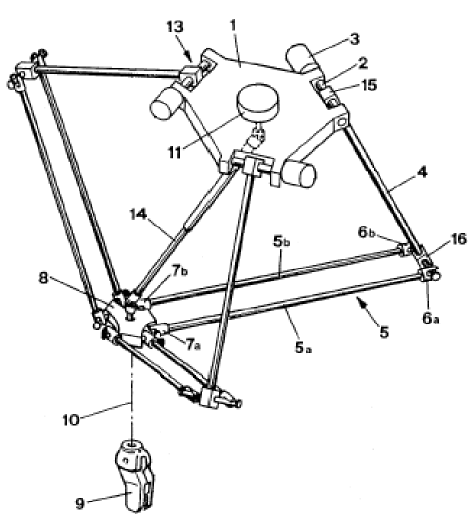
Category: Introduction
Views: 90,881
Replies: 30

Delta robot kinematics

Difficulty: Medium

Delta robot kinematics

When one talks about industrial robots, most of people imagine robotic arms, or articulated robots, which are doing painting, welding, moving something, etc. But there is another type of robots: so-called parrallel delta robot, which was invented in the early 80's in Switzerland by professor Reymond Clavel. Below the original technical drawing from U.S. Patent 4,976,582 is shown, and two real industrial delta robots, one from ABB, and one from Fanuc.



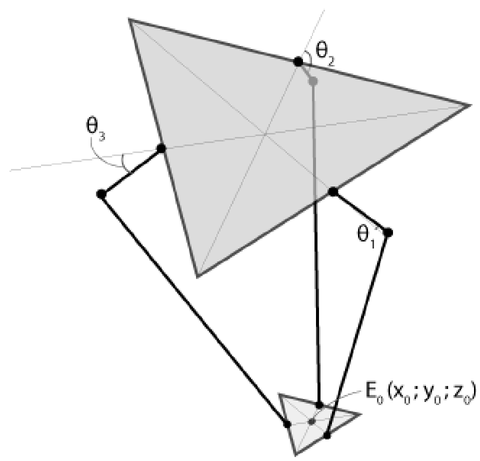
The delta robot consists of two platforms: the upper one (1) with three motors (3) mounted on it, and smaller one (8) with an end effector (9). The platforms are connected through three arms with parallelograms, the parallelograms restrain the orientation of the lower platform to be parallel to the working surface (table, conveyor belt and so on). The motors (3) set the position of the arms (4) and, thereby, the XYZ-position of the end effector, while the fourth motor (11) is used for rotation of the end effector. You can find more detailed description of delta robot design in the corresponding Wikipedia article.

The core advantage of delta robots is speed. When typical robot arm has to move not only payload, but also all servos in each joint, the only moving part of delta robot is its frame, which is usually made of lightweight composite materials. To get an evidence of delta robots outstanding abilities, take a look at this and this video. Due to its speed, delta robots are widely used in pick-n-place operations of relatively light objects (up to 1 kg).

Problem definition

If we want to build our own delta robot, we need to solve two problems. First, if we know the desired position of the end effector (for example, we want to catch pancake in the point with coordinates X,Y,Z), we need to determine the corresponding angles of each of three arms (joint angles) to set motors (and, thereby, the end effector) in proper position for picking. The process of such determining is known as **inverse kinematics.**
And, in the second place, if we know joint angles (for example, we've read the values of motor encoders), we need to determine the position of the end effector (e.g. to make some corrections of its current position). This is **forward kinematics** problem.

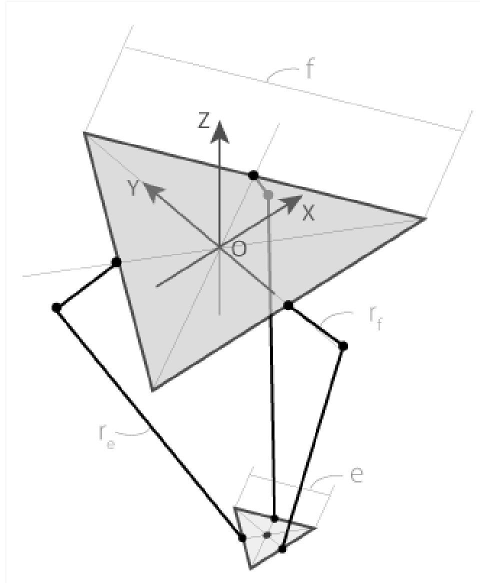
To be more formal, let's look at the kinematic scheme of delta robot. The platforms are two equilateral triangles: the fixed one with motors is green, and the moving one with the end effector is pink. Joint angles are *theta1*, *theta2* and *theta3*, and point *E0* is the end effector position with coordinates (*x0*, *y0*, *z0*). To solve inverse kinematics problem we have to create function with E0 coordinates (*x0*, *y0*, *z0*) as parameters which returns (*theta1*, *theta2*, *theta3*). Forward kinematics functions gets (*theta1*, *theta2*, *theta3*) and returns (*x0*, *y0*, *z0*).



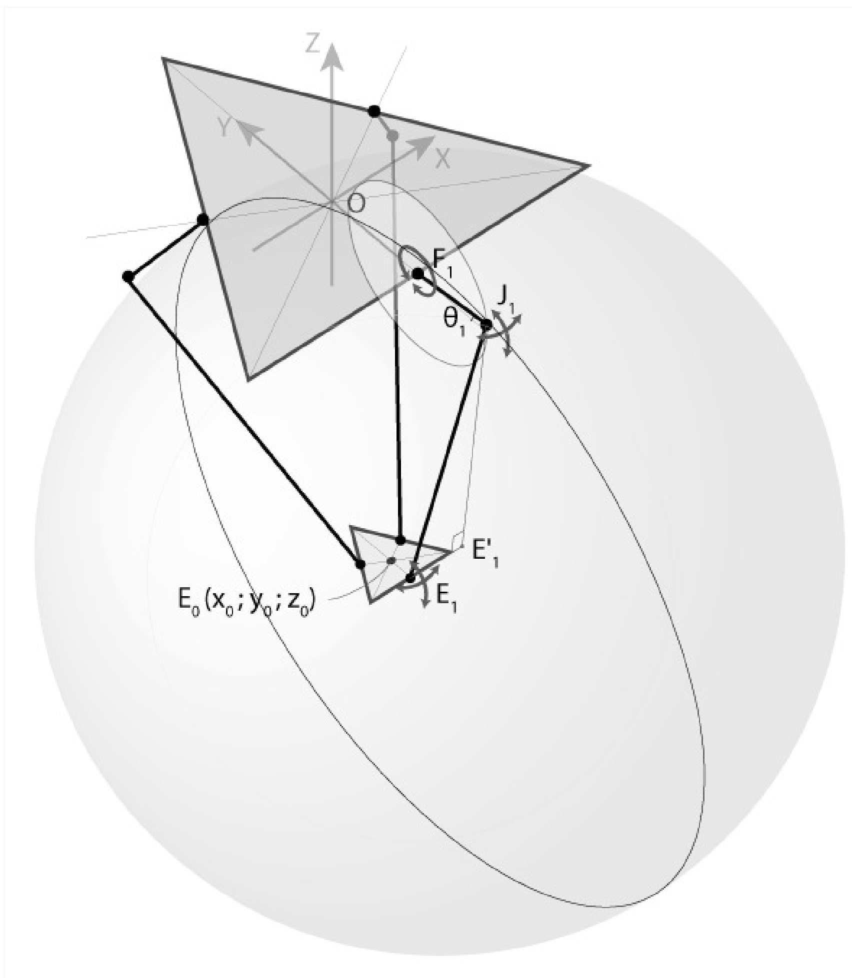
In the following two paragraphs will come the theoretical part of delta robots kinematics. Those who don't like mathematics and trigonometry may jump right to the practical part: sample programs written in C language. So, let's start from

Inverse Kinematics

First, let's determine some key parameters of our robot's geometry. Let's designate the side of the fixed triangle as **f**, the side of the end effector triangle as **e**, the length of the upper joint as **rf**, and the length of the parallelogram joint as **re**. These are physical parameters which are determined by design of your robot. The reference frame will be choosen with the origin at the center of symmetry of the fixed triangle, as shown below, so z-coordinate of the end effector will always be negative.

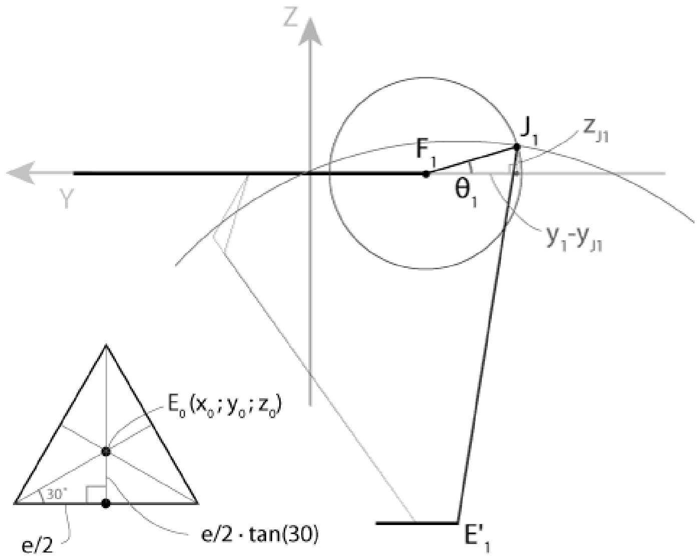


Because of robot's design joint *F1J1* (see fig. below) can only rotate in YZ plane, forming circle with center in point *F1* and radius *rf*. As opposed to *F1*, *J1* and *E1* are so-called universal joints, which means that *E1J1* can rotate freely relatively to *E1*, forming sphere with center in point *E1* and radius *re*.



Intersection of this sphere and YZ plane is a circle with center in point *E'1* and radius *E'1J1*, where *E'1* is the projection of the point *E1* on YZ plane. The point J1 can be found now as intersection of to circles of known radius with centers in *E'1* and *F1* (we should choose only one intersection point with smaller Y-coordinate). And if we know *J1*, we can calculate *theta1* angle.

Below you can find corresponding equations and the YZ plane view:



$$E(x_0, y_0, z_0)$$

$$EE_1 = \frac{e}{2}\tan 30^o = \frac{e}{2\sqrt{3}}$$

$$E_1(x_0, y_0 - \frac{e}{2\sqrt{3}}, z_0) \Rightarrow E'_1(0, y_0 - \frac{e}{2\sqrt{3}}, z_0)$$

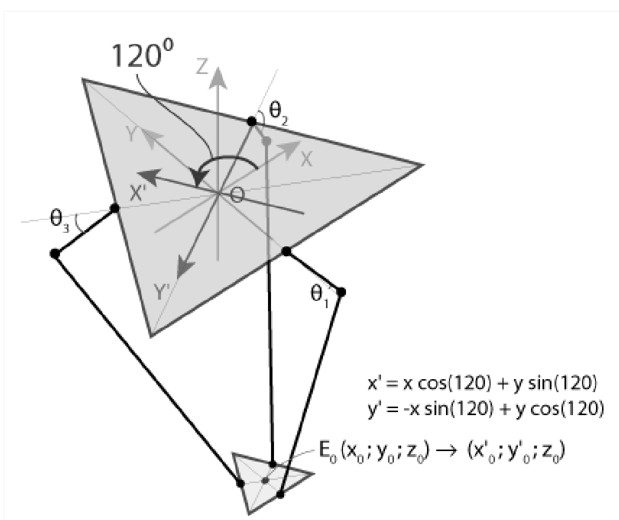
$$E_1E'_1 = x_0 \Rightarrow E'_1J_1 = \sqrt{E_1J_1^{\ 2} - E_1E'_1{}^{2}} = \sqrt{r_e^2 - x_0^2}$$

$$F_1(0, -\frac{f}{2\sqrt{3}}, 0)$$

$$\begin{cases} (y_{J1} - y_{F1})^2 + (z_{J1} - z_{F1})^2 = r_f^2 \\ (y_{J1} - y_{E'1})^2 + (z_{J1} - z_{E'1})^2 = r_e^2 - x_0^2 \end{cases} \Rightarrow \begin{cases} (y_{J1} + \frac{f}{2\sqrt{3}})^2 + z_{J1}^{\ 2} = r_f^2 \\ (y_{J1} - y_0 + \frac{e}{2\sqrt{3}})^2 + (z_{J1} - z_0)^2 = r_e^2 - x_0^2 \end{cases} \Rightarrow J_1(0, y_{J1}, z_{J1})$$

$$\theta_1 = \arctan\left(\frac{z_{J1}}{y_{F1} - y_{J1}}\right)$$

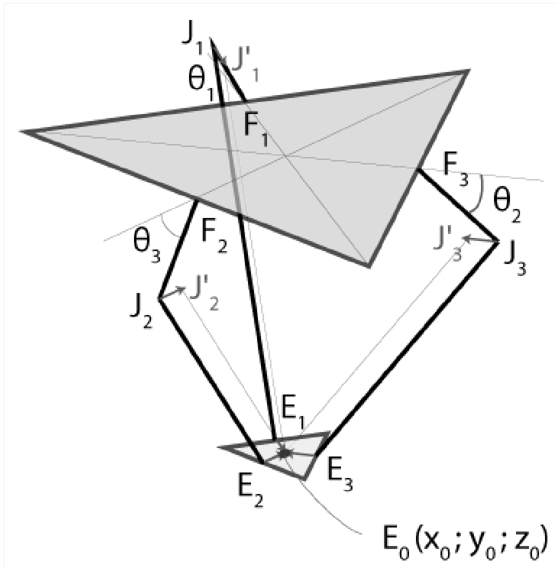
Such algebraic simplicity follows from good choice of reference frame: joint *F1J1* moving in YZ plane only, so we cat completely omit X coordinate. To take this advantage for the remaining angles *theta2* and *theta3*, we should use the symmetry of delta robot. First, let's rotate coordinate system in XY plane around Z-axis through angle of 120 degrees counterclockwise, as it is shown below.



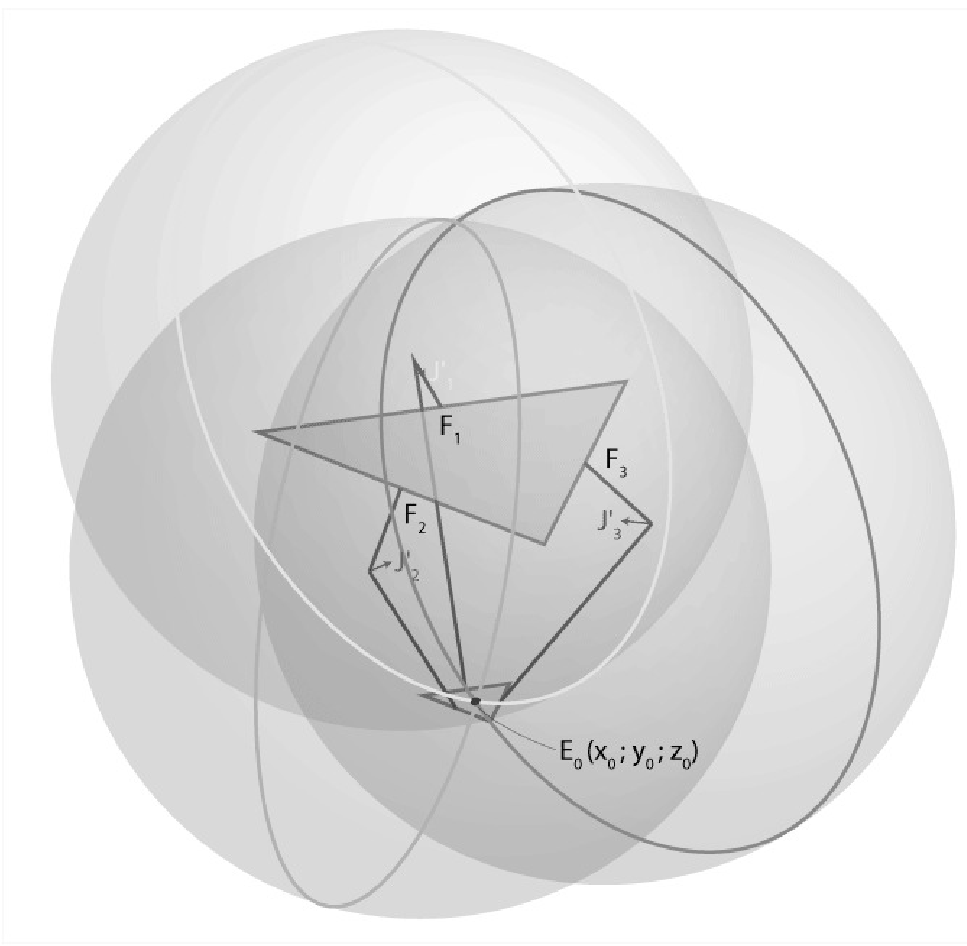
We've got a new reference frame X'Y'Z', and it this frame we can find angle *theta2* using the same algorithm that we used to find *theta1*. The only change is that we need to determine new coordinates *x'0* and *y'0* for the point *E0*, which can be easily done using corresponding rotation matrix. To find angle *theta3* we have to rotate reference frame clockwise. This idea is used in the coded example below: I have one function which calculates angle theta for YZ plane only, and call this function three times for each angle and each reference frame.

Forward kinematics

Now the three joint angles *theta1*, *theta2* and *theta3* are given, and we need to find the coordinates (*x0*, *y0*, *z0*) of end effector point *E0*. As we know angles theta, we can easily find coordinates of points *J1*, *J2* and *J3* (see fig. below). Joints *J1E1*, *J2E2* and *J3E3* can freely rotate around points *J1*, *J2* and *J3* respectively, forming three spheres with radius *re*.

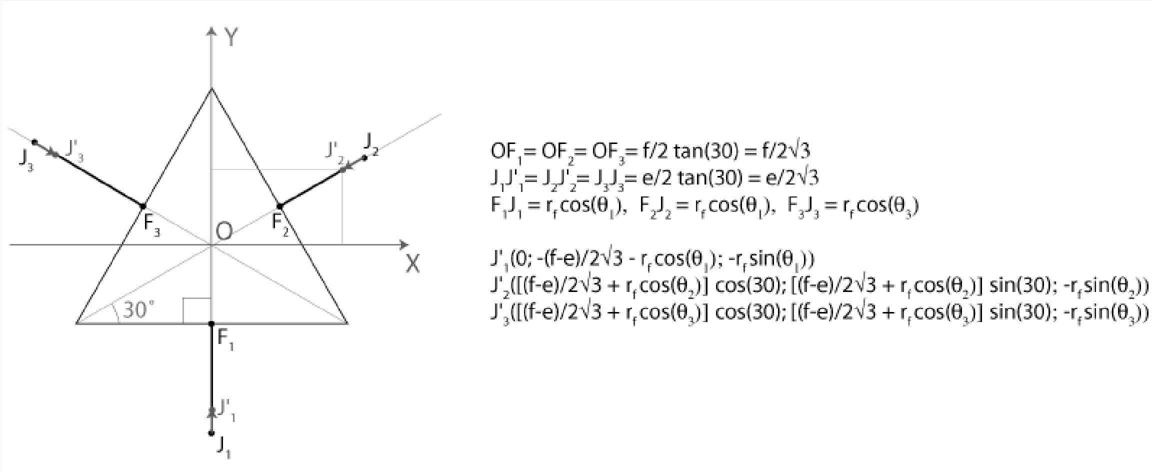


Now let's do the following: move the centers of the spheres from points *J1*, *J2* and *J3* to the points *J'1*, *J'2* and *J'3* using transition vectors *E1E0*, *E2E0* and *E3E0* respectively. After this transition all three spheres will intersect in one point: *E0*, as it is shown in fig. below:



So, to find coordinates (*x0*, *y0*, *z0*) of point *E0*, we need to solve set of three equations like (x-xj)^2+(y-yj)^2+(z-zj)^2 = re^2, where coordinates of sphere centers (*xj*, *yj*, *zj*) and radius *re* are known.

First, let's find coordinates of points *J'1*, *J'2*, *J'3*:



In the following equations I'll designate coordinates of points J1, J2, J3 as (x1, y1, z1), (x2, y2, z2) and (x3, y3, z3). Please note that x0=0. Here are equations of three spheres:

$$\begin{cases} x^2+(y-y_1)^2+(z-z_1)^2=r_e^2 \\ (x-x_2)^2+(y-y_2)^2+(z-z_2)^2=r_e^2 \\ (x-x_3)^2+(y-y_3)^2+(z-z_3)^2=r_e^2 \end{cases} \Rightarrow \begin{cases} x^2+y^2+z^2-2y_1y-2z_1z=r_e^2-y_1^2-z_1^2 & (1) \\ x^2+y^2+z^2-2x_2x-2y_2y-2z_2z=r_e^2-x_2^2-y_2^2-z_2^2 & (2) \\ x^2+y^2+z^2-2x_3x-2y_3y-2z_3z=r_e^2-x_3^2-y_3^2-z_3^2 & (3) \end{cases}$$

$$w_i = x_i^2+y_i^2+z_i^2$$

$$\begin{cases} x_2x+(y_1-y_2)y+(z_1-z_2)z=(w_1-w_2)/2 & (4)=(1)-(2) \\ x_3x+(y_1-y_3)y+(z_1-z_3)z=(w_1-w_3)/2 & (5)=(1)-(3) \\ (x_2-x_3)x+(y_2-y_3)y+(z_2-z_3)z=(w_2-w_3)/2 & (6)=(2)-(3) \end{cases}$$

From (4)-(5):

$$x = a_1z+b_1 \quad (7) \qquad y = a_2z+b_2 \quad (8)$$

$$a_1 = \frac{1}{d}\left[(z_2-z_1)(y_3-y_1)-(z_3-z_1)(y_2-y_1)\right] \qquad a_2 = -\frac{1}{d}\left[(z_2-z_1)x_3-(z_3-z_1)x_2\right]$$

$$b_1 = -\frac{1}{2d}\left[(w_2-w_1)(y_3-y_1)-(w_3-w_1)(y_2-y_1)\right] \qquad b_2 = \frac{1}{2d}\left[(w_2-w_1)x_3-(w_3-w_1)x_2\right]$$

$$d = (y_2-y_1)x_3-(y_3-y_1)x_2$$

Now we can substitute (7) and (8) in (1):

$$(a_1^2+a_2^2+1)z^2+2(a_1+a_2(b_2-y1)-z_1)z+(b_1^2+(b_2-y_1)^2+z_1^2-r_e^2)=0$$

Finally, we need to solve this quadric equation and find *z0* (we should choose the smallest negative equation root), and then calculate *x0* and *y0* from eq. (7) and (8).

Sample programs

The following code is written in C, all variable names correspond to designations I've used above. Angles *theta1*, *theta2* and *theta3* are in degrees. You can freely use this code in your applications.

Code:

```
// robot geometry
// (look at pics above for explanation)
const float e = 115.0;     // end effector
const float f = 457.3;     // base
const float re = 232.0;
const float rf = 112.0;

// trigonometric constants
const float sqrt3 = sqrt(3.0);
const float pi = 3.141592653;    // PI
const float sin120 = sqrt3/2.0;
const float cos120 = -0.5;
const float tan60 = sqrt3;
const float sin30 = 0.5;
const float tan30 = 1/sqrt3;

// forward kinematics: (theta1, theta2, theta3) -> (x0, y0, z0)
// returned status: 0=OK, -1=non-existing position
int delta_calcForward(float theta1, float theta2, float theta3, float &x0, float &y0, float &z0) {
    float t = (f-e)*tan30/2;
```

```
    float dtr = pi/(float)180.0;

    theta1 *= dtr;
    theta2 *= dtr;
    theta3 *= dtr;

    float y1 = -(t + rf*cos(theta1));
    float z1 = -rf*sin(theta1);

    float y2 = (t + rf*cos(theta2))*sin30;
```

Acknowledgements

All core ideas about delta robot kinematics are taken from the article of Prof. Paul Zsombor-Murray Descriptive Geometric Kinematic Analysis of Clavel's "Delta" Robot. It's a great publication, but it requires a very strong mathematical background for understanding.

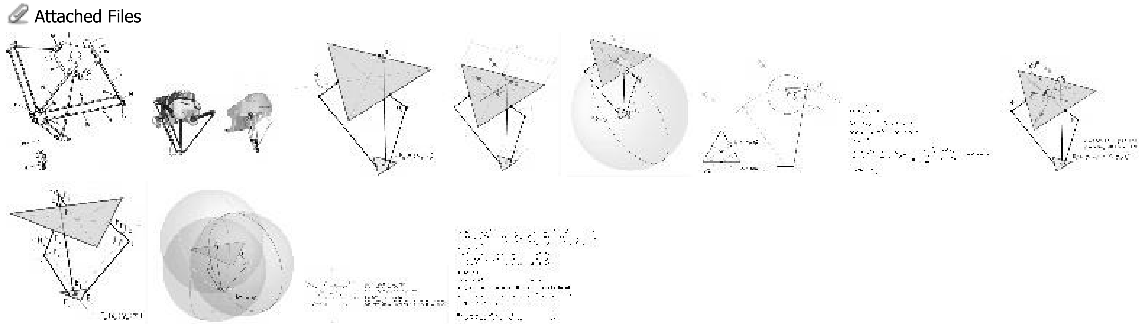
Sample build

I've used the programs listed above in my model of delta robot made of standard Lego parts. It uses Lego Mindstorms NXT as "brain", and a program written in RobotC. Below the robot is shown in action, so you can be sure that this programs realy work

[youtube]V_FoJWm2lOI[/youtube]

Now you know enough about delta robot kinematics to build your own. Enjoy!

Attached Files



Tags: delta, robot, kinematics, forward, reverse, programming, program

« Previous Tutorial | Next Tutorial »

Replies to Tutorial: Delta robot kinematics

Page 1 of 3 **1** 2 3 Last

03-14-2010

djl5053
Abacus

Join Date: Feb 2010
Posts: 0

Re: Delta robot kinematics

mzavatsky,

This is a fantastic tutorial. Thanks for breaking down the research paper you mentioned. My buddy Andy and I are working on a similar delta robot. We follow you up to the point where you translate you inverse kinematics code into C code. We found a solution process on line for the intersection of two circles, but it does look like you implement that method.

http://local.wasp.uwa.edu.au/~pbourke/geometry/2circle/

I also played around with the algebra. Using the last two equations of from you Inv Kine. I solved the 2nd to last equation for yj1 and zj1 and plugged into the very last equation. I wound up with zj1 as a function of yj1. I guess that gives me a line my solution lies on. Then I think I need to use on of the equations again to find where that line intersects one of the circles.

Did you happen to follow a process like this?
http://mathworld.wolfram.com/Circle-...ersection.html

Reply With Quote

08-16-2011

aggrav8d

Join Date: Sep 2010
Posts: 9

Relay

Re: Delta robot kinematics

Using the tutorial above I have solved for both forward and inverse kinematics, and build them into a 3D OpenGL simulation. There is also Arduino code fir use with hobby servos.

http://visual-delta3.sourceforge.net/

and the project home page with videos,

http://www.marginallyclever.com/delta3/

Reply With Quote

09-30-2011

The mask
Abacus

Join Date: Sep 2011
Posts: 0

Re: Delta robot kinematics

It would be great if you could post the same code in NXC or an extended version of the robotC code. The problem is that I can't figure out how to use robotC functions.

Thanks in advanced.

Reply With Quote

10-26-2011

ushnish
Abacus

Join Date: Oct 2011
Posts: 0

Re: Delta robot kinematics

it was a good tutorial....i hope all of you know, for the standard anthropomorphic serial robotic arm, the problem of forward kinematics is simpler with the D-H convention, but the Inverse Kinematics is not straight forward,and on the contrary, this is quite the opposite for parallel manipulators,where IK is easier than FK. So , I was thinking, if we could find out a methodology, in which, every serial robot will have its parallel dual and viceversa. In that case , we can easily calculate the FK from the serial form, and IK from the parallel form.As far as my intuition goes, this duality will be nothing but a mapping from a serial manipulator joint space to a parallel manipulator joint space and between the workspaces also...think over guys!!

Reply With Quote

01-08-2012

Doc
Abacus

Join Date: Jan 2012
Posts: 0

Re: Delta robot kinematics

Thanks for the great tutorial.
Any suggestions on how I would alter the program to work on a 4 arm Delta robot?

Reply With Quote

01-26-2012

Shani
Abacus

Join Date: Jan 2012
Posts: 1

Re: Delta robot kinematics

Hi,

I implement the given code in the MATLAB to find the forward and Inverse Kinematics.
When I did the cross check then I get the wrong results.
First I put the dummy angle in the forward kinematics and get the position, after that I put that position in the Inverse Kinematics but at that time I get the different angles as I put in the forward kinematics before.
Can anyone explain this deviation in the cross check ? ?
Thanks in Advance
Zeeshan

Reply With Quote

01-31-2012

aggrav8d
Relay

Join Date: Sep 2010
Posts: 9

Re: Delta robot kinematics

Shani, I confirm your results. I took the code above, put it into javascript, and I see that calcForward(calcInverse(A)) != A, which is wrong. Can anyone spot the bug?

Reply With Quote

02-01-2012

aggrav8d
Relay

Join Date: Sep 2010
Posts: 9

Re: Delta robot kinematics

No, it seems I spoke too soon. Everything checks out just fine.

Reply With Quote

02-05-2012

Shani
Abacus

Join Date: Jan 2012
Posts: 1

Re: Delta robot kinematics

Dear aggrav8d,

Thank you very much for your confirmation. Might be there is some mistake in my programming. I have one more question. The link that you mentioned in your comment
http://www.marginallyclever.com/samples/fk-ik-test.html

It have a term "Steps Per Turn", what does it mean ? Can you explain me this term ?
How can I implement this in my programming ?

Thanks in Advance,
Shani

Reply With Quote

02-09-2012

Shani
Abacus

Join Date: Jan 2012
Posts: 1

Re: Delta robot kinematics

Hi All,
I have solved the inverse Kinematics. But when I implement it then on some position the moving plate is not parallel to the base plate.
Is there any special condition to maintain the moving plate parallel to the base plate ?
Can I measure the slope of the moving plate plate if it is not parallel to the base plate.
Regards,
Zeeshan

Reply With Quote

Reply To Tutorial

Page 1 of 3 **1** 2 3 Last

Quick Navigation Introduction Top

Quick Reply

Rich Text Editor
Toolbar Source Remove Format Paste as plain text Bold Italic UnderlineFontFontSizeSize Text Color Smiley Link Unlink Image Insert Video Wrap [QUOTE] tags around selected text

☐ Subscribe to This Tutorial
☐ Add to Favorites

Post Quick Reply Go Advanced Cancel

Contact Us Trossen Robotics Community Top

All times are GMT -5. The time now is 08:09 AM.