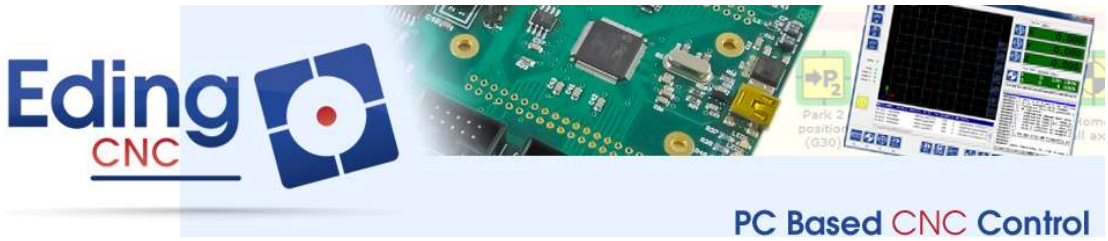


EdingCNC KINEMATICS

Document Release 0.3

Published by: **Bert Eding**
Eindhoven
The Netherlands

Title: Project Document
Author: Bert Eding
Date: Tuesday, 18 June 2019

Document History

Version	Date	Author	Comment
0.1	01-06-2012	B. Eding	Initial version
0.2	29-04-2019	Jan Hummel	Update for output data from kinematics.dll
0.3	18-06-2019	Jan Hummel	Extended standard built-in kinematics

Table of contents

1 Introduction

1.1 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

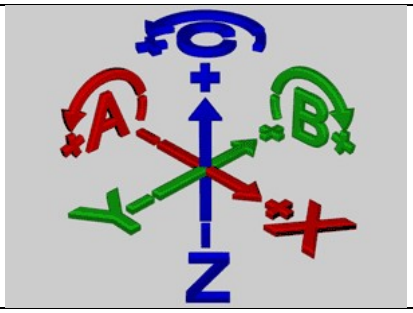
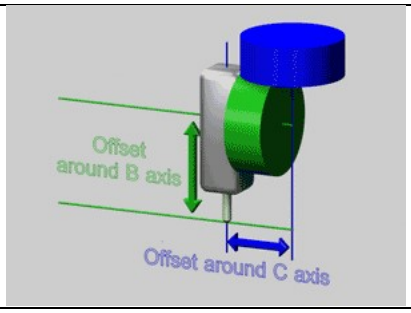
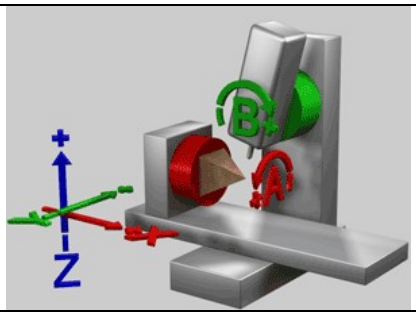
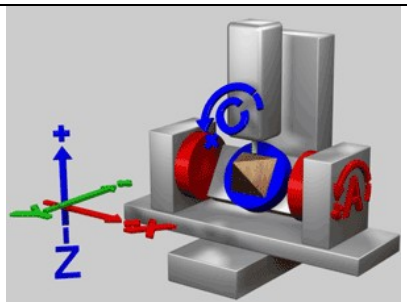
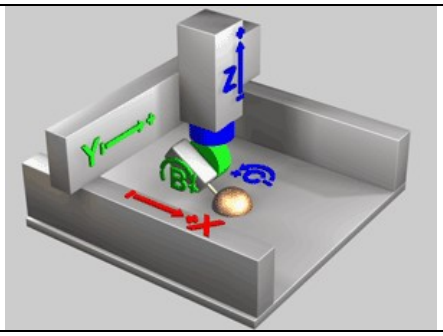
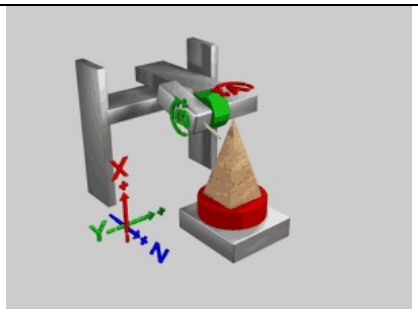
CNC	Computerized Numerical Control
CPU	Central Processor Unit, a PCB board with a Processor on it.
DXF	Drawing Exchange Format) is a CAD data file format developed by Autodesk
FIFO	First In First Out Buffer
HPGL	Hewlet Packard Graphical Language
GUI/UI	Graphical User Interface
INTERPRETER	A software function that is able to read a text file and execute the commands contained therein.
JOBFILE	A *job* is the text file (G code) that will be executed by the interpreter.
GUI	Graphical User Interface.
PWM	Pulse Width Modulation
G-Code	CNC specific language to control the movements and IO of a milling machine.
LAF	Look Ahead Feed, advanced motion algorithm that ensures minimal machining time.

1.2 CONTEXT AND SCOPE

1.2.1 Part coordinates and standard kinematic configurations for CNC machines

The units of measure user interface to a user of a CNC machine is always defined in a Cartesian space. With XY and Z coordinates. The XYZ coordinates define the position of the tool-tip.
Most machines are Cartesian from nature, but this is not always the case.

1.2.2 Common kinematic configurations for CNC machines:

	
	Tool Offsets are the distance between the tool tip and the pivot point of the corresponding rotary axis
	
Configuration 1. 5Axis Mill with A rotary table sits vertically on the mill table The Spindle pivots on the B-Axis which is attached to the Z-Axis	**Confuration 2.** 5Axis Mill The work piece rotates on the C-Axis which rotates on the A-Axis
	
Configuration 3 5Axis Gantry Router The Spindle is attached via the B-Axis to the C-Axis to the Z-Axis	**Configuration 4** 5Axis Vertical Router The work piece sits on a rotating plate.

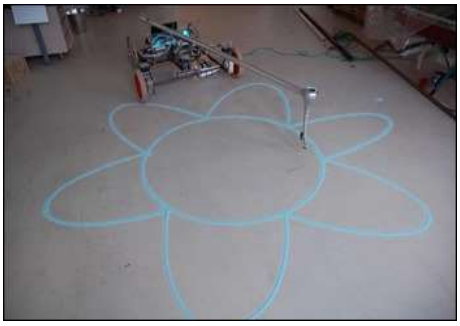
First thing to be noticed is the meaning of ABC in the coordinate system, where XYZ is the position of the tooltip, ABC defines with 3 angles the orientation of the tool tip.

It is clear to see that if the angle of the tooltip is changed that several other motors have to move in order to keep the position of the tool-tip at the same place.

1.2.3 A completely different machine

Besides these common CNC configurations many others are possible, we solve this bottleneck by developing the kinematics as run-time loadable plug in module, which can be developed externally.

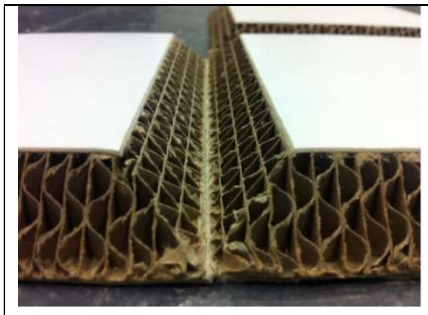
Several wheels with motors can scribe with sand.

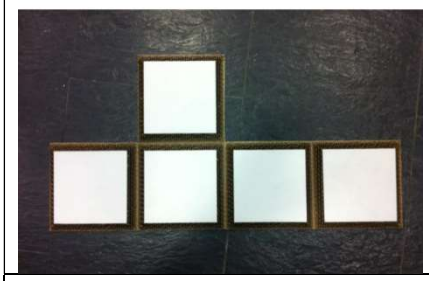


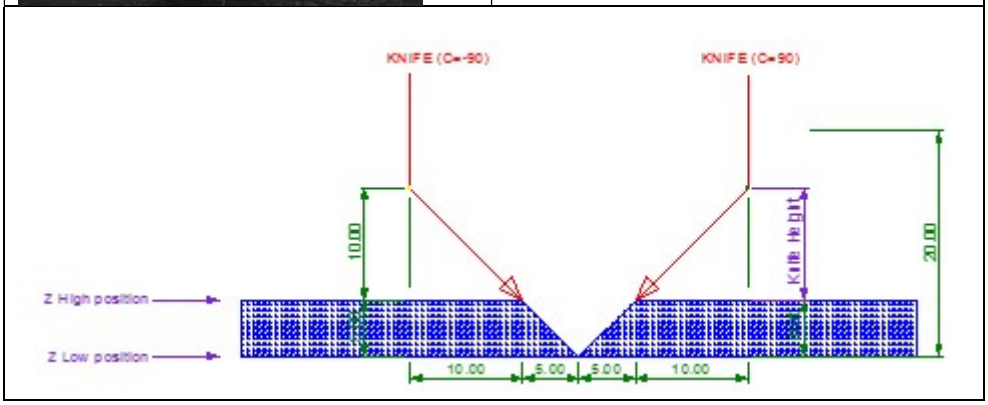
The sand figure is a Cartesian drawing.

1.2.4 Tangential bend knife kinematics

A 45 degree bend tangential knife make certain applications possible:
The software also allows a BEND tangential knife, bend under 45 degrees makes applications like this possible:



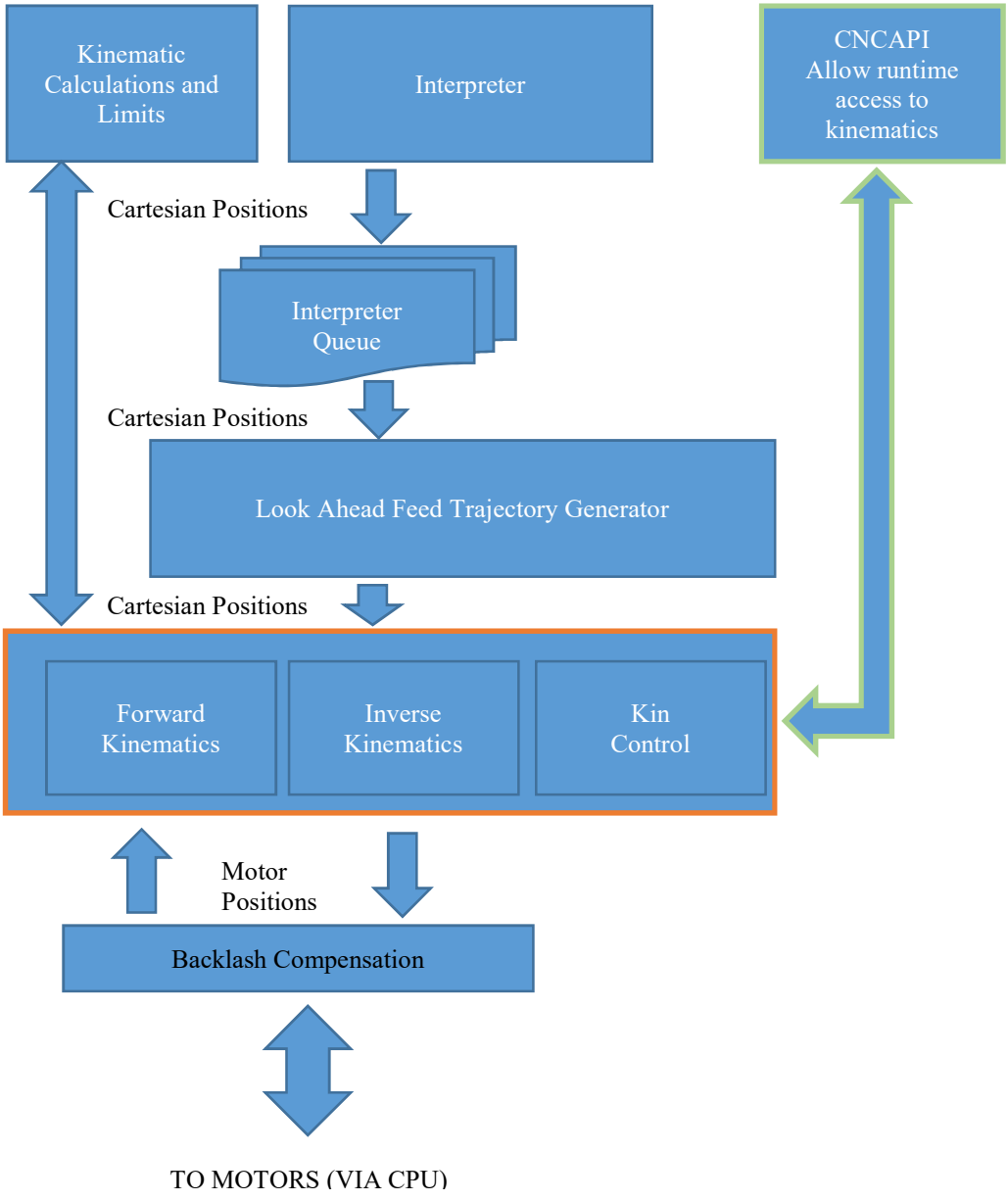




It is clear that during a z-move, the XY axes should move as well with same distance as Z, in the direction such that the sharp side of the knife moves under 45 degrees into the material. The user will program Z moves, the interpreter will rotate the knife (C axis) into the correct XY direction. Then kinematics will add the XY correction depending on the Z position.

1.3 SOFTWARE ARCHITECTURE

Below we see the rough architecture of the kinematic system in the CNC server part. The interpreter interprets the use commands and prepares the motions to be performed by calculating the correct velocities and accelerations depending on the setup values. The bottleneck is how to calculate the correct velocities/accelerations in Cartesian space while the actual motion takes place by the motors. It is possible that a movement in Cartesian space of e.g. the C axis can cause motion on the any of the motors depending on the kinematics transformation functions. The kinematics module is a plugin module, so that a user can develop this himself. Modification of runtime behavior (e.g. a different gripper etc.) is possible via the CNCAPI interface.

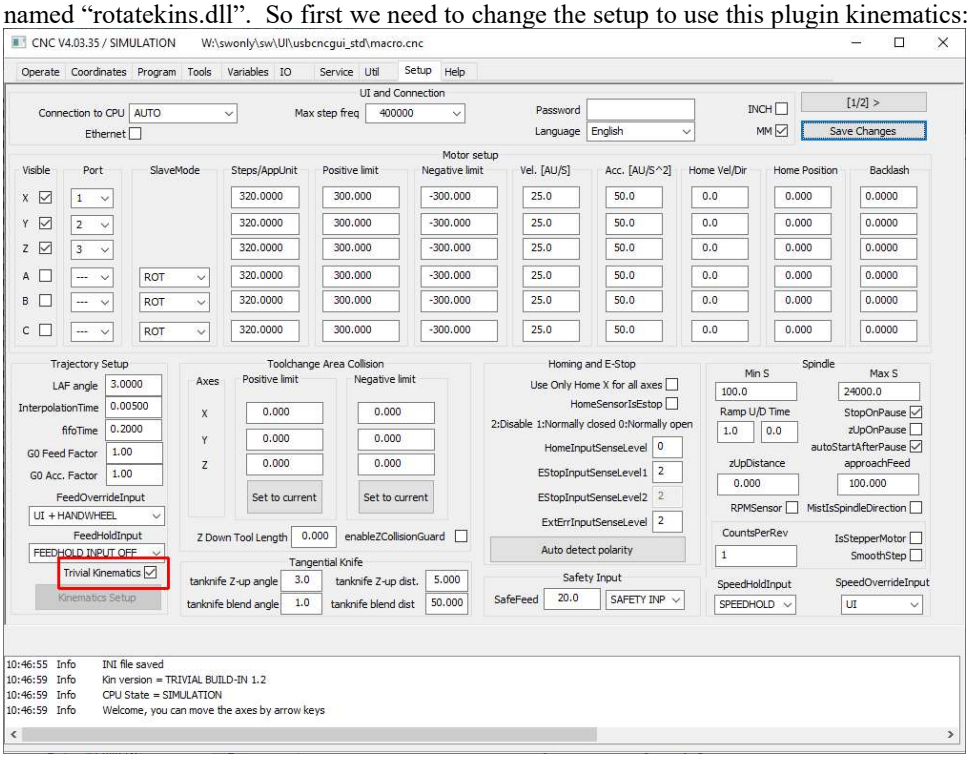


2 The kinematic module

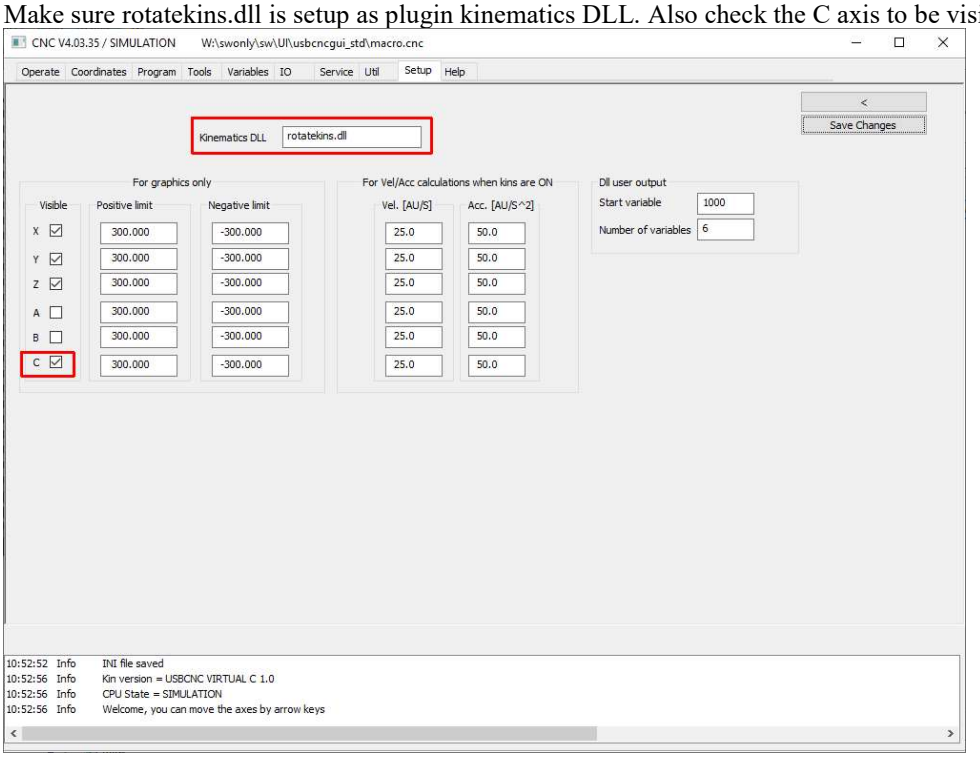
It is created as plug-in DLL.
It contains only 5 functions. These functions will be explained next. For learning purpose, a kinematics DLL project is created that can be used on every Cartesian machine that has an XYZ axis. A 4th virtual axis "C" is added. This C exists only in Cartesian space. The C works in Degree units 0-360 degree. What we will do here is rotate the XY plane with the angle given by C. The rotation center point is the position where the Kinematics is switched on.

2.1 SETUP AND TRY

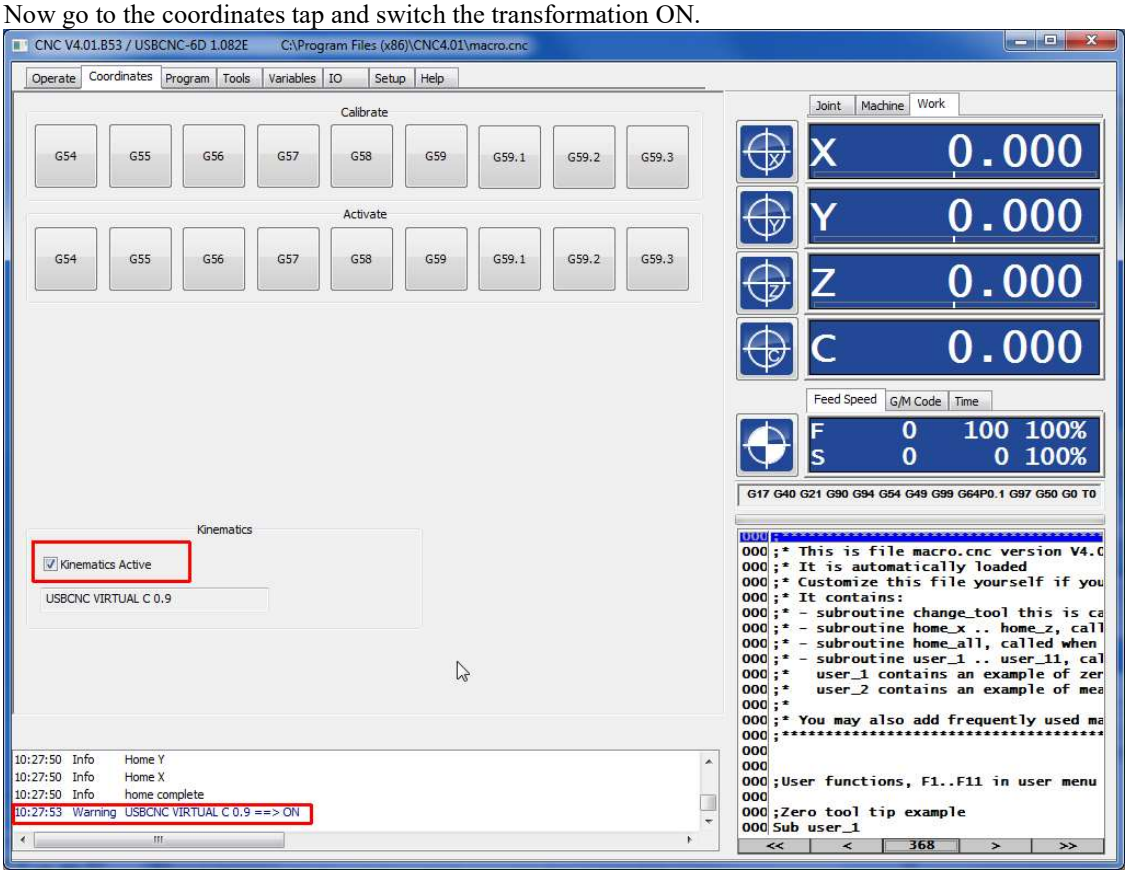
First let us see how it really works, the plugin dll is already compiled and available in the install directory, it is named "rotatekins.dll". So first we need to change the setup to use this plugin kinematics:



Uncheck Trivial Kinematics and press save. Button "Kinematics Setup" becomes available, press it:
Make sure rotatekins.dll is setup as plugin kinematics DLL. Also check the C axis to be visible.

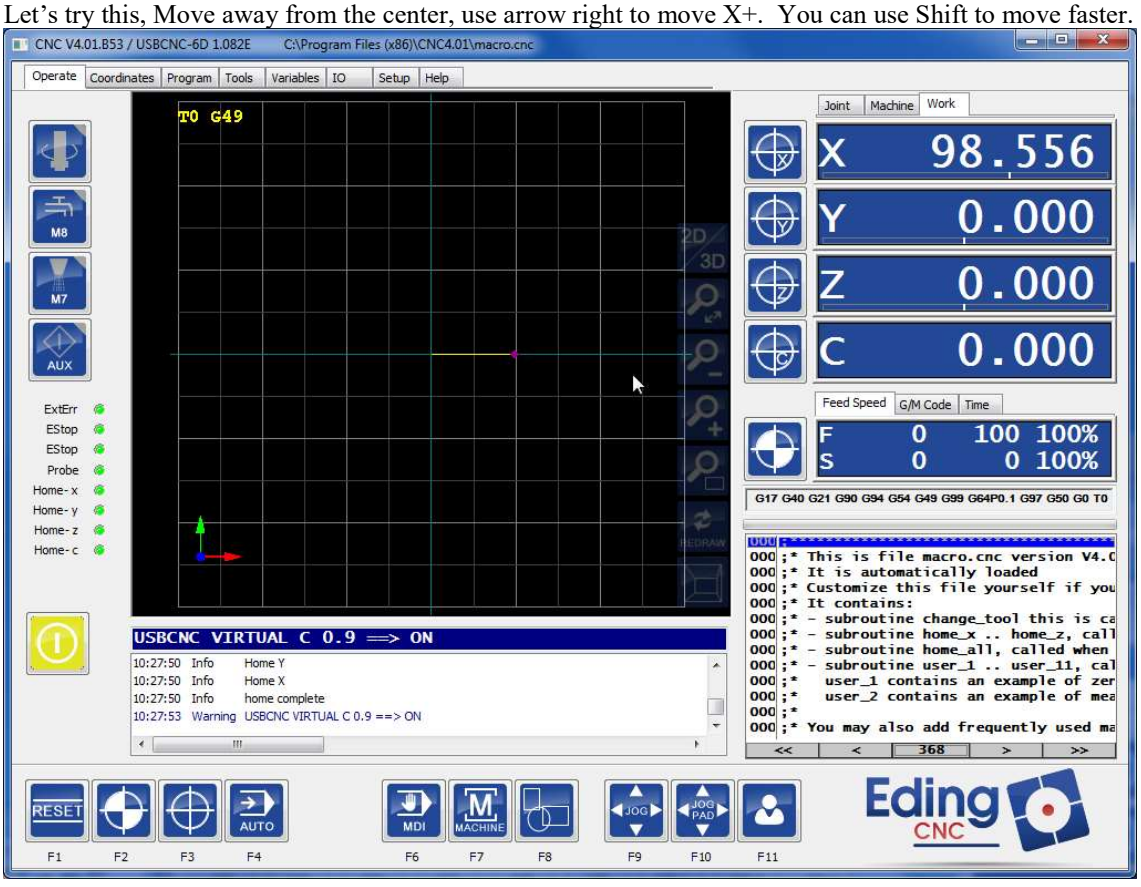


We are ready to use it now.
See that our current position is somewhere in the middle of the machine area.
Now go to the coordinates tap and switch the transformation ON.

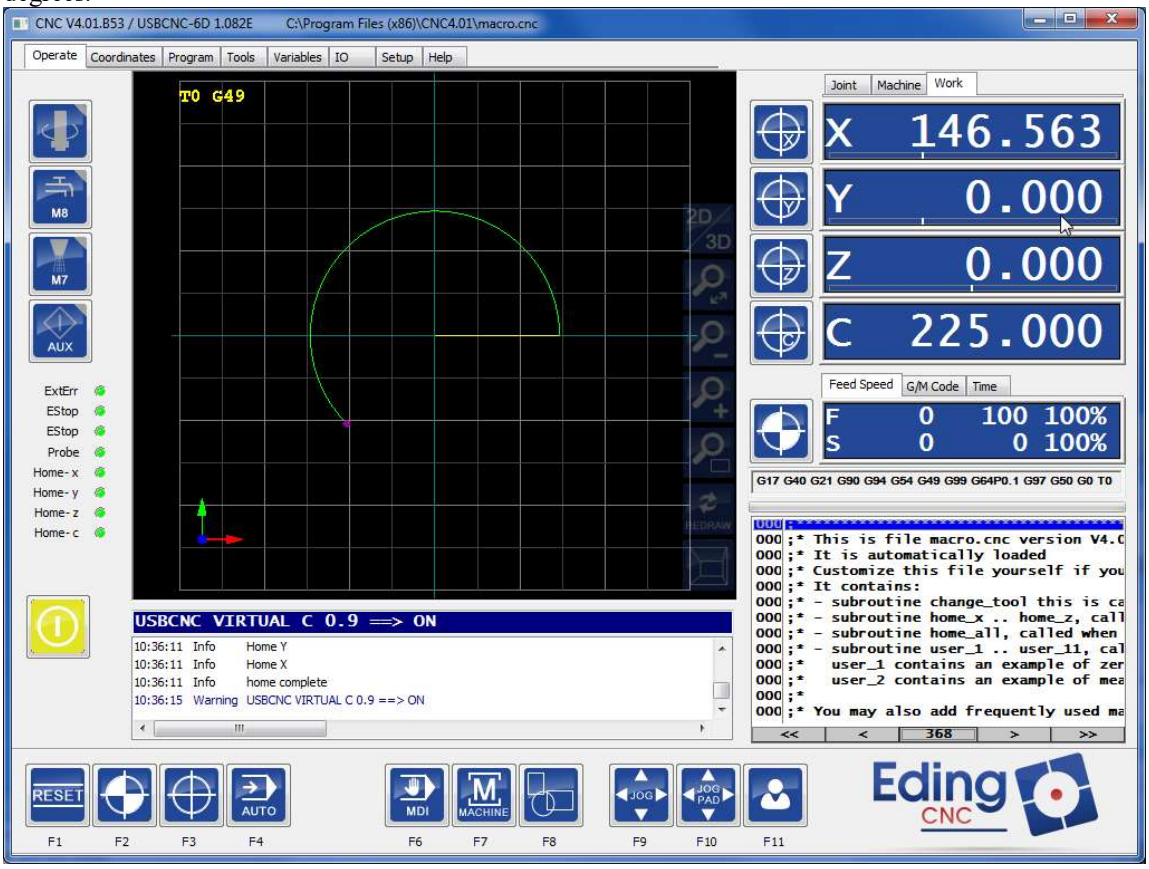


You will see that USBCNC VITUAL C 0.9 is switched on. The kinematics are active now and the center of rotation is the current positin.

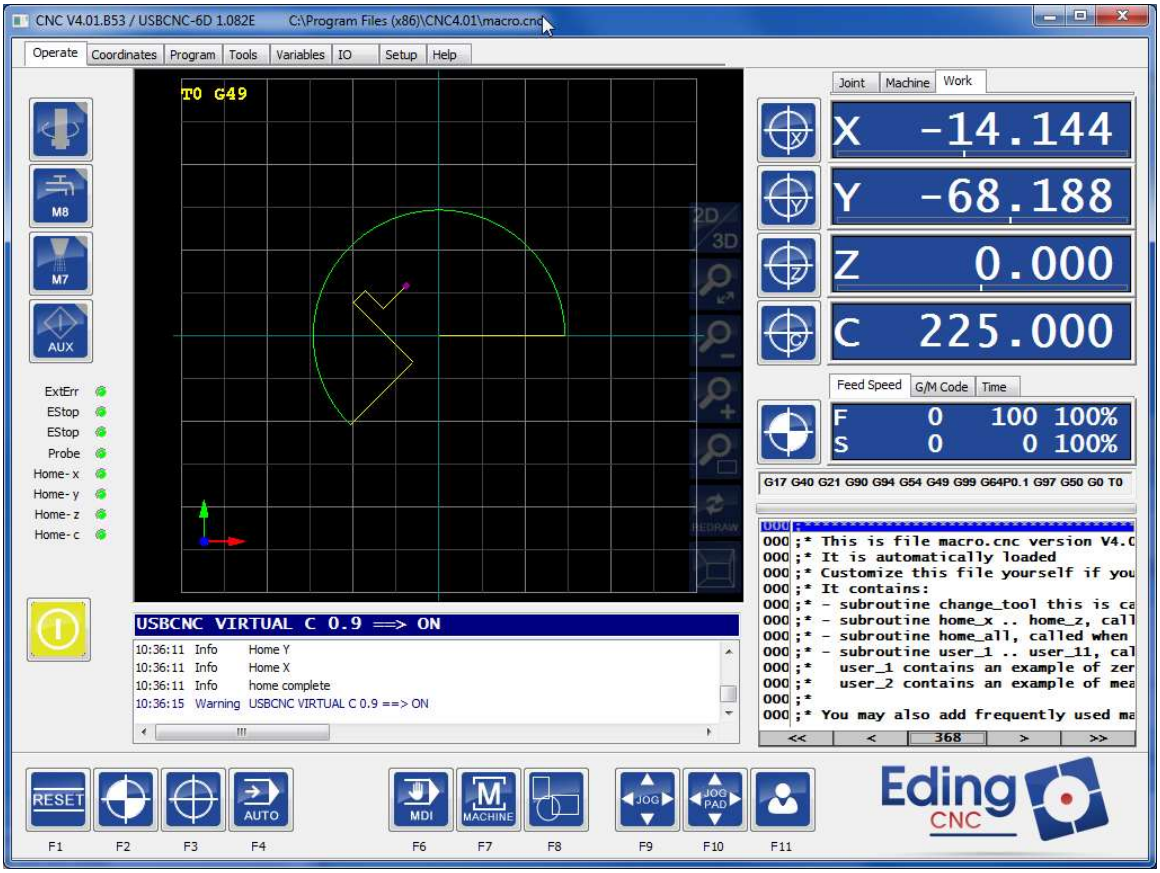
Let's try this, Move away from the center, use arrow right to move X+. You can use Shift to move faster.



Now move the C, click in the C position readout and specify 270 degrees. Press OK, the C will move with maximum velocity to 225 degree and watch what happens, the XY plane is rotated resulting in a move of 225 degrees.



Now try to move X and Y, to see clearly that the XY plane is rotated, press the Arrow Keys (with Shift) to see it:



Cool isn't it ? You can use this on your normal Cartesian milling machine.

2.2 DETAILED EXPLANATION OF THE ROTATE KINEMATICS PLUGIN DLL

To understand this part, you'll need some programming "C" knowledge and a little bit of math.
All the functions are in one file, "rotatekins.cpp"

These are the prototype definitions of the functions that must be made. This content is found in the "rotatekins.h" header file. It is the same for all kinematic types.

There are only 5 functions the needs to be implemented, they are explained in next chapter.

2.2.1 Function proto types and explanation

The following part is the prototype definition with explanation as in the "C" Header file rotatekins.h.

```
/*
 * Name: Kinematics Inverse, converts Cartesian coordinates to Motor coordinates
 *
 * Input:  pos,
 *         pos.x, pos.y, pos.z, are the tool-tip position in millimeter.
 *         pos.a, pos.b, pos.c are the tool orientation angles in degrees.
 *         iflags, not used
 *
 * Output: joints,
 *         joints.jx, joints.jy. joints.jz joints.ja, joints.jb, joints.jc
 *         are the motor positions.
 *         Motor positions usually also in mm or degrees, depending on the motor setup.
 *         fflags, not used.
 *
 * Remark: This function is called by after the trajectory generation inside the cncserver.
 *         Typically calling frequency is 200 Hz (5 Milli second).
 *
 *
 */
int KINEMATICS_API _stdcall KinematicsInverse(CNC_CART_DOUBLE   pos,
                                      CNC_JOINT_DOUBLE *joints,
                                      const CNC_KINEMATICS_INVERSE_FLAGS * iflags,
                                      CNC_KINEMATICS_FORWARD_FLAGS * fflags,
                                      double  toolLength);
/*
 * Name: Kinematics Forward, converts Motor coordinates to Cartesian coordinates
 *
 * Input: joints,
 *        joints.jx, joints.jy. joints.jz joints.ja, joints.jb, joints.jc
 *        are the motor positions.
 *        Motor positions usually also in mm or degrees, depending on the motor setup.
 *        fflags, not used.
 *
 * Output: pos,
 *         pos.x, pos.y, pos.z, are the tool-tip position in millimeter.
 *         pos.a, pos.b, pos.c are the tool orientation angles in degrees.
 *         iflags, not used
 *
 * Remark: This function is called after homing and after switching on kinematics, to obtain
 *         the actual Cartesian position given the actual motor positions.
 *
 */
int KINEMATICS_API _stdcall KinematicsForward(CNC_JOINT_DOUBLE joints,
                                      CNC_CART_DOUBLE  *pos,
                                      const CNC_KINEMATICS_FORWARD_FLAGS * fflags,
                                      CNC_KINEMATICS_INVERSE_FLAGS * iflags,
                                       double toolLength);

/*
 * Name:   KinematicsControl, general purpose communication with the kinematics module.
 *         Provide parameters and switching ON/OFF of the kinematics.
 *         This function can be used to change the kinematics behavior run-time.
 *
 * Input:  ControlID,  there is a list of predefined Control ID's and free user control ID's,
 *         see cnc_kin_types.h
 *
 *         controlData, a buffer of data, double integer or char for communication to and from
```

```
 *         this plug-in DLL.
 *
 * Remark: This function can be called indirectly from a user application or from the GUI by
 *         CNCAPI function "CncKinControl" with this, the user can have full control
 *         over the behavior  of the
 *         kinematics.
 *
 */
int KINEMATICS_API __stdcall KinematicsControl(KIN_CONTROL_ID controlID,
                                               KIN_CONTROLDATA *controlData);



/*
 *  Name: KinematicsType, this function returns the kinematics type, See cnc_kin_types.h
 */
CNC_KINEMATICS_TYPE KINEMATICS_API _stdcall KinematicsType();

/*
 *  Name: KinematicsType, this function returns the kinematics version string
 */
char KINEMATICS_API * _stdcall KinematicsVersion(void);
```

2.2.2 The implementation in “C”

In this section the implementation of the function sis described, this is the contents of the rotatekins.cpp source code file.

```
// rotatekins.cpp : Defines the exported functions for the DLL application.
//

#include <math.h>
#include <stdio.h>
#include "rotatekins.h"

static char KIN_VERSION[] = "USBCNC VIRTUAL C 1.0";

static int    kinEnabled     = 0; /* 0 if disabled, 1 if enabled        */
static double rotationPointX = 0; /* rotation point X                   */
static double rotationPointY = 0; /* rotation point Y                   */
static double rotationAngle  = 0; /* Degrees for easy debugging purpose */

int __stdcall KinematicsControl(int controlID, KIN_CONTROLDATA *controlData)
{
    switch (controlID)
    {
    case CNC_KIN_CONTROL_ID_OPEN:
        /* nothing to do for this kins implementation */
        printf("Kins opened\n");
        break;

    case CNC_KIN_CONTROL_ID_CLOSE:
        /* nothing to do for this kins implementation */
        printf("Kins closed\n");
        break;

    case CNC_KIN_CONTROL_ID_OFF:
        /* disable kins */
        kinEnabled = 0;
        printf("Kins disabled\n");
        break;

    case CNC_KIN_CONTROL_ID_ON:
        /* enable kins */
        printf("Kins enabled\n");
        kinEnabled = 1;
        break;

    case KIN_CONTROL_ID_USER1:
        /* Set rotation point */
        rotationPointX = controlData->dData[0];
        rotationPointY = controlData->dData[1];
        /* printf("Kins rotation point set to x=%f, y=%f\n", rotationPointX, rotationPointY); */
        break;

    case KIN_CONTROL_ID_USER2:
        /* Set rotation angle */
        rotationAngle = controlData->dData[0];
        /* printf("Kins rotation angle set to ang=%f\n", rotationAngle); */
        break;

    case KIN_CONTROL_ID_USER3:
        /* Get rotation point */
        controlData->dData[0] = rotationPointX;
        controlData->dData[1] = rotationPointY;
        break;


    default:
        /* return error */
        return (-1);
        break;

    }

    /* return OK */
    return(0);
}


/*
 * This function is a local function, that rotates the XY plane.
```

```
 * Reverse is used to rotate in opposite direction or normal direction
 * rotationPointX, rotationPointY is the center point for the rotation.
 * x and y are input and rotated output
 */
static void rotate_scale(bool reverse, double rotationPointX, double rotationPointY, double &x, double &y)
{
    double xr, yr;
    double t = CNC_D2R(rotationAngle);
    //Note conversion Degree to Radian is needed because in
    //"C" the Goniometric functions Sin, Cos etc always work with radians.

    double sinR = sin(reverse ? -t : t);
    double cosR = cos(reverse ? -t : t);


    xr = rotationPointX + (double)(((x - rotationPointX) * cosR) - ((y - rotationPointY) * sinR));
    yr = rotationPointY + (double)(((x - rotationPointX) * sinR) + ((y - rotationPointY) * cosR));

    x = xr;
    y = yr;
}


//Convert motor coordinates to Cartesian coordinates.
int __stdcall KinematicsForward(CNC_JOINT_DOUBLE joints,
                                CNC_CART_DOUBLE  *pos,
                                const CNC_KINEMATICS_FORWARD_FLAGS * fflags,
                                CNC_KINEMATICS_INVERSE_FLAGS * iflags, double toolLength)
{
    CNC_UNUSED(toolLength);
    CNC_UNUSED(fflags);
    CNC_UNUSED(iflags);

    if (kinEnabled)
    {
        double motorX = joints.jx;
        double motorY = joints.jy;

        //Rotate, reverse
        rotate_scale(true, rotationPointX, rotationPointY, motorX, motorY);

        //Output calculated Cartesian position in mm and degrees.
        //Because motor C is not physically present, we use last rotation angle from inverse
        //kinematics.
        pos->x = motorX;
        pos->y = motorY;
        pos->z = joints.jz;
        pos->a = joints.ja;
        pos->b = joints.jb;
        pos->c = rotationAngle;
    }
    else
    {
        pos->x = joints.jx;
        pos->y = joints.jy;
        pos->z = joints.jz;
        pos->a = joints.ja;
        pos->b = joints.jb;
        pos->c = rotationAngle;
    }


    return 0;
}
```

```
int __stdcall KinematicsInverse(CNC_CART_DOUBLE pos,
                                CNC_JOINT_DOUBLE *joints,
                                const CNC_KINEMATICS_INVERSE_FLAGS * iflags,
                                CNC_KINEMATICS_FORWARD_FLAGS * fflags, double toolLength)
{
    CNC_UNUSED(toolLength);
    CNC_UNUSED(fflags);
    CNC_UNUSED(iflags);


    if (kinEnabled)
    {
        // C axis gives rotation for X,Y axes
        // We keep the C value stored in rotation angle because we use it in
        // the forward kinematics.
        rotationAngle = pos.c;

        joints->jx = pos.x;
        joints->jy = pos.y;

        //Rotate the coordinate system normally, not reversed.
        rotate_scale(false, rotationPointX, rotationPointY, joints->jx, joints->jy);

        joints->jz = pos.z;
        joints->ja = pos.a;
        joints->jb = pos.b;
        joints->jc = 0; /* C axis is virtual, do not output to joint */

    }
    else
    {
        joints->jx = pos.x;
        joints->jy = pos.y;
        joints->jz = pos.z;
        joints->ja = pos.a;
        joints->jb = pos.b;
        joints->jc = 0;
    }

    return 0;
}

CNC_KINEMATICS_TYPE __stdcall KinematicsType(void)
{
    //Return the kinematics type.
    //This type is a predefined type.
    //Change to one of the CUSTOM types if you make your own kinematics!
    return (CNC_KINEMATICS_TYPE_VIRTUAL_C);
}

char * __stdcall KinematicsVersion(void)
{
    //Return the version string that is defined above in this file.
    return(KIN_VERSION);
}
```

2.3 FULL CONTROL

During the motion, every interpolation cycle the Kinematics Inverse function is called. The function calculates from the Cartesian input and local parameters the output to the motors. So this function determines how the motion is done.

During the motion, the user can communicate with the kinematics module by means of the CNCAPI CncKinControl function. And so if he wishes adapt the motion of the system at run-time.

2.4 STANDARD BUILD IN KINEMATICS

A few standard Kinematic types are already in place:

- This rotation kinematics, which serves as example, but may also have functional uses.
- Tangential bend knife kinematics which takes care that the Z move is nit straight down, but under 45 degree into the material. This can be controlled by the “tanknife” command, see main software manual.

- A-axis mapping Y to A movements which allows to mill on a cylinder as if it is the XY plane. A program with XYZ coordinates is transformed to XAZ movements, the software calculates all that is needed, the user has to calibrate the A axis rotation point and radius. See main software manual.
- Parallel processing of 2 work pieces is supported by linking Z2ZC (or if 2 motors are used for the Y motion, Y2YAZ2ZC

Paramter for 'KIN' command	Behavior
X2A	Map X-axis movement to A-axis
X2XA	Move A-axis also when X-axis moves
Y2A	Map Y-axis movement to A-axis
Y2YA	Move A-axis also when Y-axis moves
Z2C	Map Z-axis movement to C-axis
Z2ZC	Move C-axis also when Z-axis moves
Y2YAZ2ZC	Move A-axis when Y-axis moves and move C-axis also when Z-axis moves

2.5 INTERPRETER COMMANDS

There are a few commands that allow the Kinematics to be controlled from the interpreter.

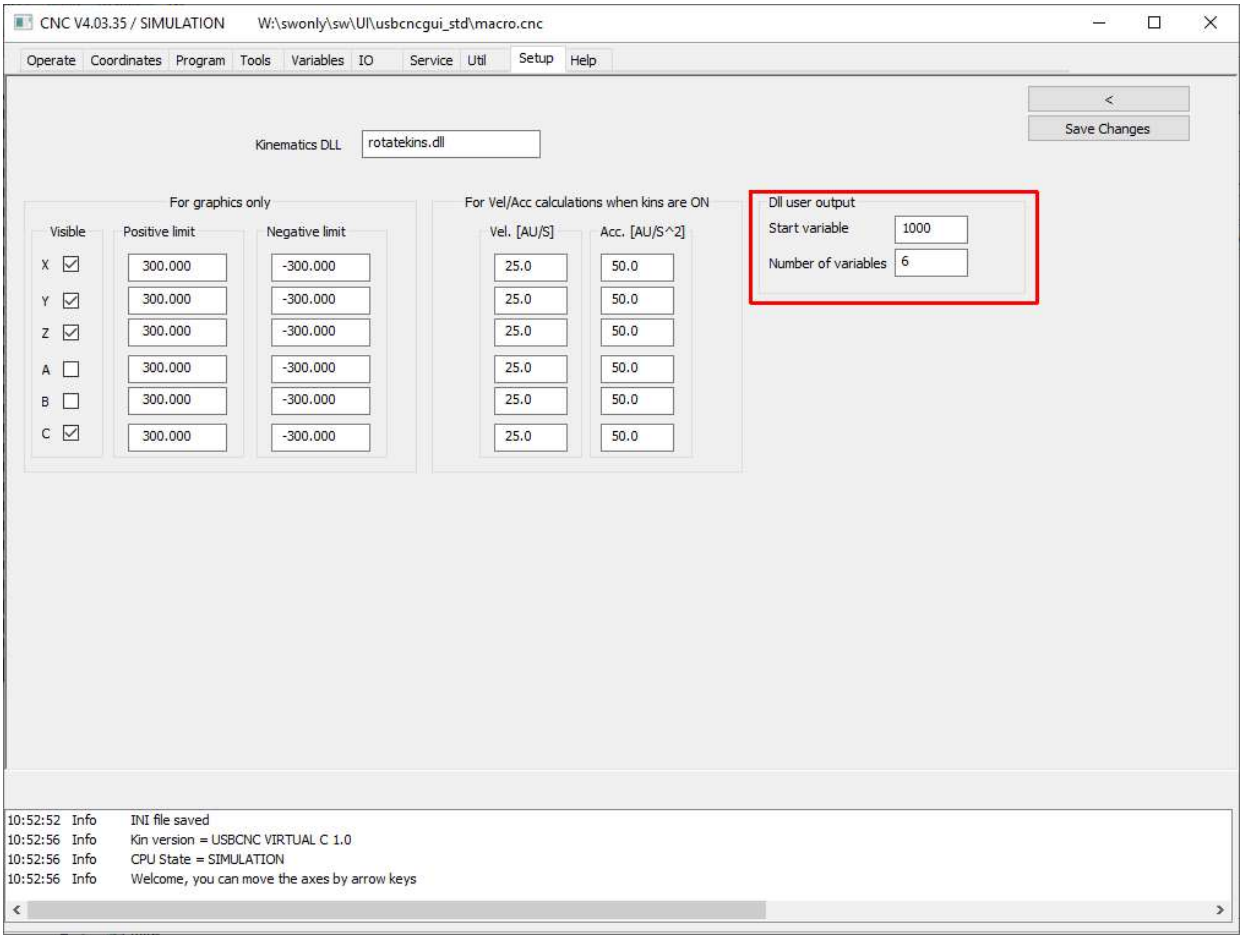
KIN ON
KIN OFF
KIN CTRL <CTRLID> <CTRL PAR1> …. <CTRL PAR6>

These are self-explaining. CTRL PAR1 to CTRL PAR6 are passed to the KinControl function via the buffer.

2.5.1 Output data from kinematics dll

Up to a maximum of 12 doubles can be exported from the kinematics dll to user variables. Configuration for this behavior is done in the kinematics setup page as shown in paragraph 2.1.

Configuration consists of a start variable number and the number of bytes that will be written to the user variables.



Maximum number of variables is 12 and start value must be below 3980. To disable the functionality, set start variable to 0.

3 EdingCNC software setup

3.1 LIMITS

There are in fact two places where limits are specified

- The standard setup for motor X .. motor C, where position limits, velocity limits and acceleration limits are specified.
- The kinematic setup becomes available when a non-trivial kinematics type is setup. Here there are also position limits and acceleration limits specified.

The second one is needed because the software cannot always limit the Cartesian move correctly with as only knowledge the motor limits. A single Cartesian move can easily cause moves for multiple motors and also violate the max velocities and accelerations of the motors.
For these cases where the software cannot determine the limits from the motor parameters, the kinematical Cartesian limits will be used.

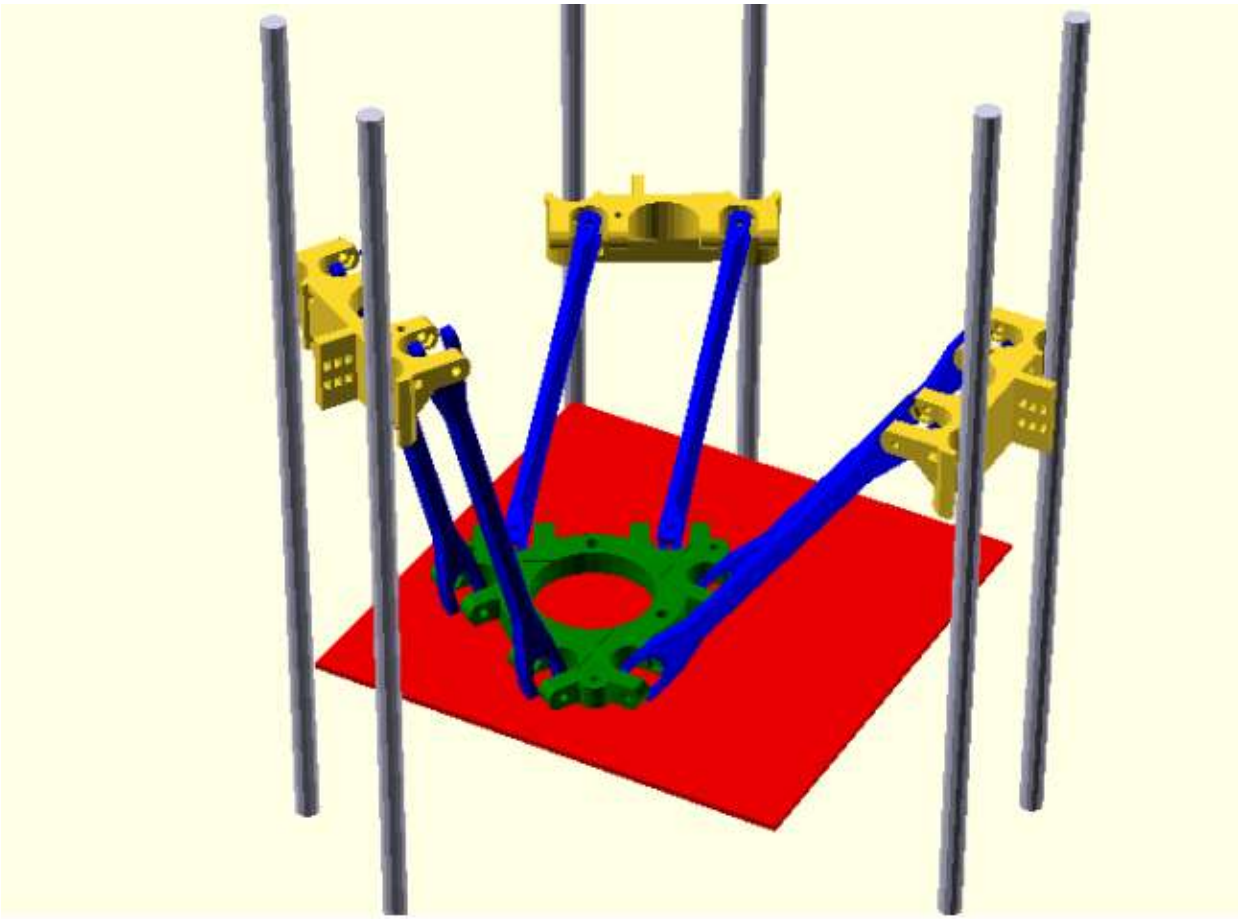
3.2 HOMING

Homing always takes place at motor level and kinematics are switches off wile homing.
In the home sequence (macro.cnc) at the end of the homing sequence a “KIN ON” command may be placed to switch on the kinematics immediately after homing.

4 Specific kinematics

4.1 DELTA ROBOT

Principle drawing

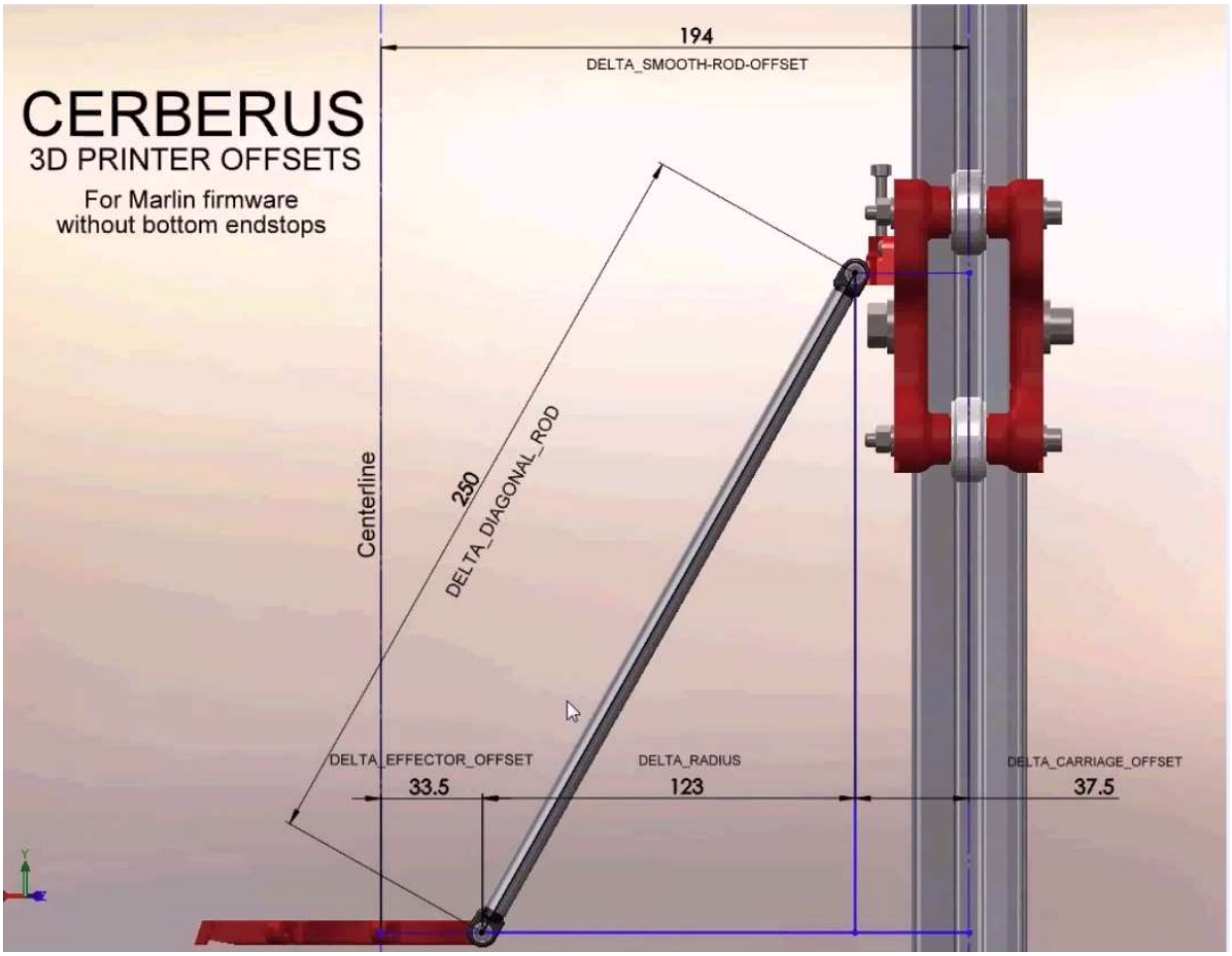


Eding CNC uses 2 parameters in the CNC.INI file under **[KINEMATICS]**

The parameters are:

linDeltaRadius = 125.000000
linDeltaArmLength = 250.000000

The can be calculated from drawing below as:
linDeltaArmLength = DELTA_DIAGONAL_ROD
linDeltaRadius = (DELTA_SMOOTH_ROD_OFFSET –
DELTA_EFFECTOR_OFFSET –
DELTA_CARRIAGE_OFFSET)



The positions:

The home position is defined as follows:

The lowest position of the end-effector is Z = 0 in cartesian coordinates.
The XY position is 0,0 when the endeffector is in the mid sitiation, all joint poisitions at the same level.

The joint positions are defined from the same level, so when the end-effector is at Z = 0, then the joints level cabn ce calculated from Pythargoras:

jointZ = sqrt (DELTA_DIAGONAL_ROD * DELTA_DIAGONAL_ROD – DELTA_RADIUS * DELTA_RADIUS) + delta effector Z.

You must use this to define the home position for the joints in the Eding CNC setup.
The positions of the home sensors are defined in the joint space.

The easiest way is to give an example:

Supose you have an arm length of 250mm and a radius of 125mm and you system can move 200mm up.
So the Z of the end-effector can be 200mm above zero.
The homeswitch position would be at:

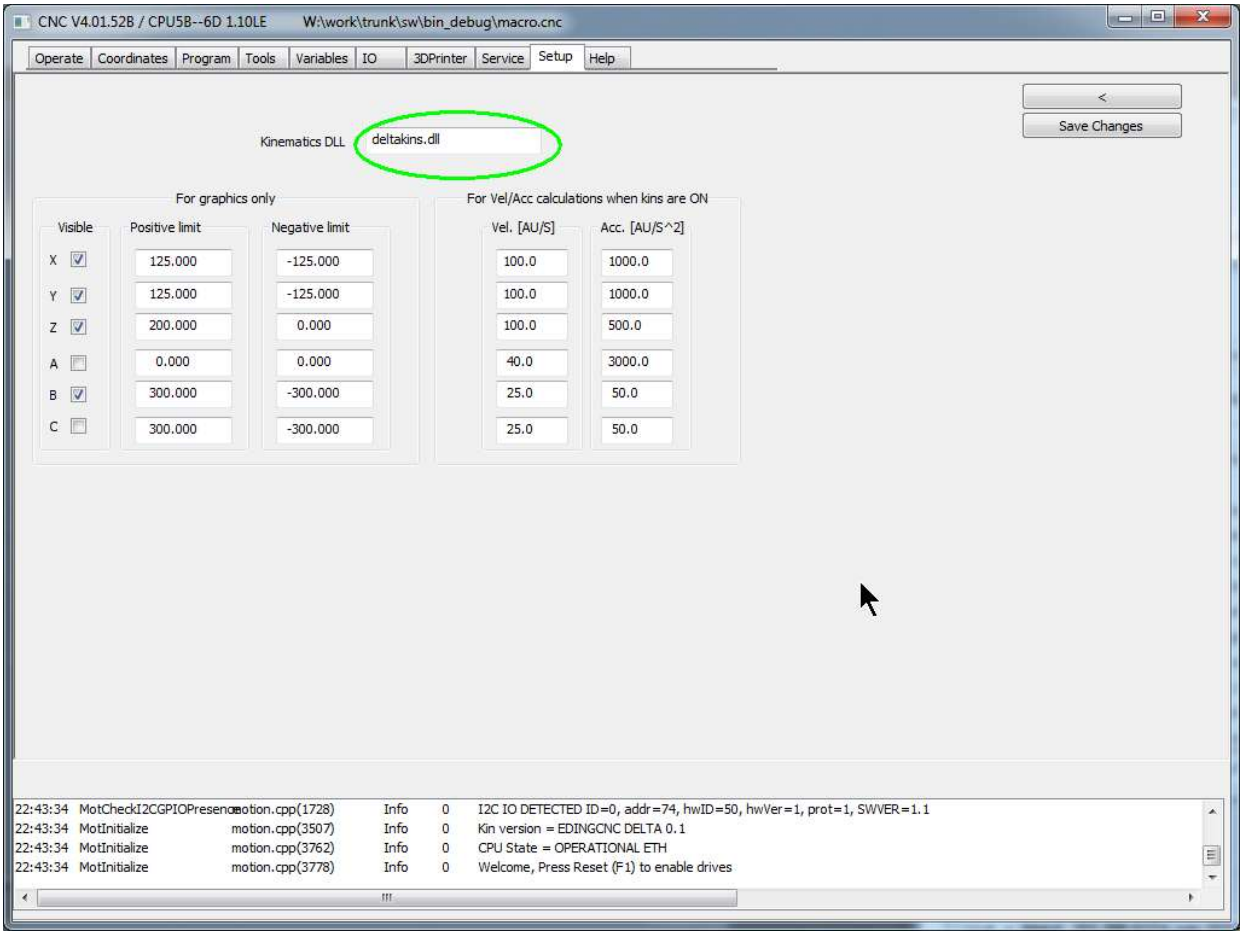
Sqrt (250*250 – 125*125) + 200 = 416.5063509 mm
This is the value for the home positions.

A new function is defined to prepare the delta robot for homing, this function will move alle 3 joints until all 3 home sensors are activated.

The function is called `PrepareLinDeltaHome`
It can be called wit a velocity : `PrepareLinDeltaHome 10` this will do the move with 10 mm/second.
Without velocity, the home velocity of the X motor is taken.
When this is done, the joints can be homed individually by Home X, Home Y, Home Z
The sequence is in macro.cnc and PrepareLimDeltaHome can be added there too.

When this is done, the EdingCNC software is still in Yrivial kinematisc mode, but can be switch to Linear Delta Kinematics if the setup is correct.

The Setup should look like this:
First in the standard setup, trivial kinematics is switched off and save parameters is pressed. Now the Kinemtics setup screen can be accessed:



The deltakins.dll is specified here and the graphics limits and velocities are setup.
Note that the position linits here are only for the graphics, the actual limits are determined by the software using the inverse kinematics and the joint limits.

After this is configured and the home sequence is performed, the kinematics kan be switched on, by command “kin on” or in the GUI at the coordnates page.

For people who want to work only with kinematics, the can modyfy home all in the macro.cnc like this:

```
Sub Home_all
  PrepareLineDeltaHome
  Home X
  Home Y
  Home Z
  Kin ON
endSub
```

Now the system will move with kinematics, try to move X and Y and Z and see what happens.