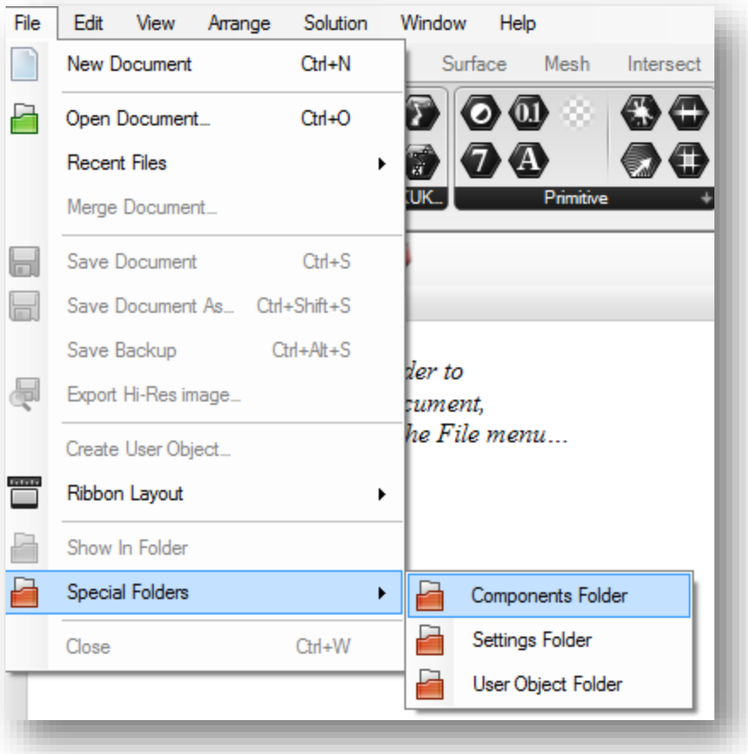
KUKA|prc: Introduction

KUKA|prc
parametric robot control for grasshopper

01 | Installing KUKA|prc

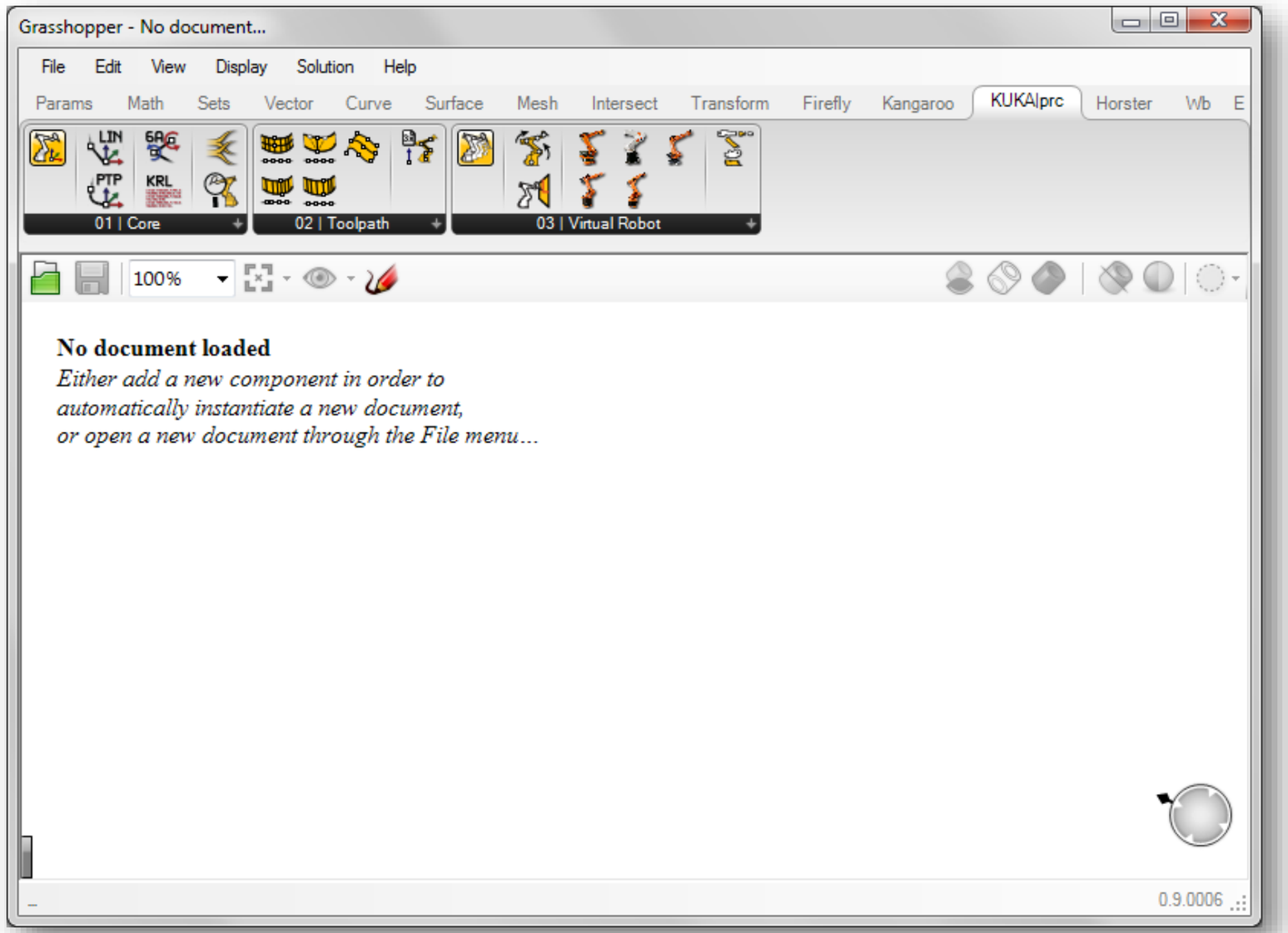
| Download KUKA|prc from www.robotsinarchitecture.org/kuka-prc

| Launch Grasshopper by typing “Grasshopper” in the Rhino command prompt



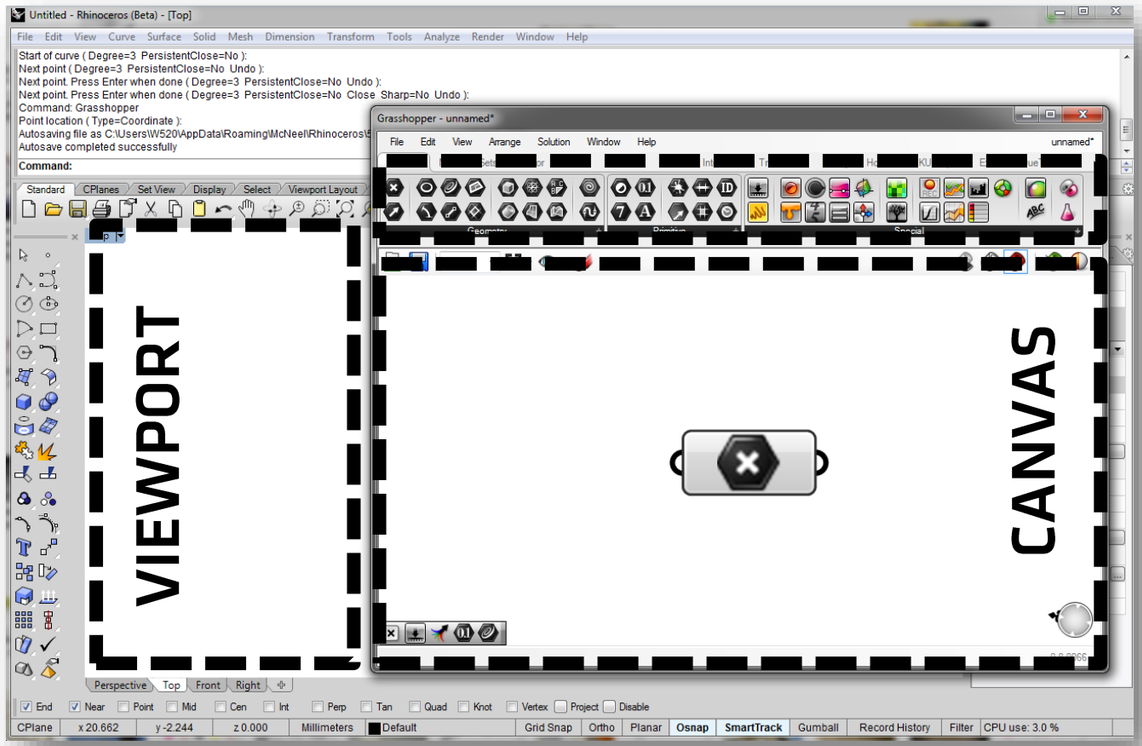
01 | Installing KUKA|prc

| The KUKA|prc tab will now show up in the Grasshopper menu. It should contain the 3 groups of icons as below.



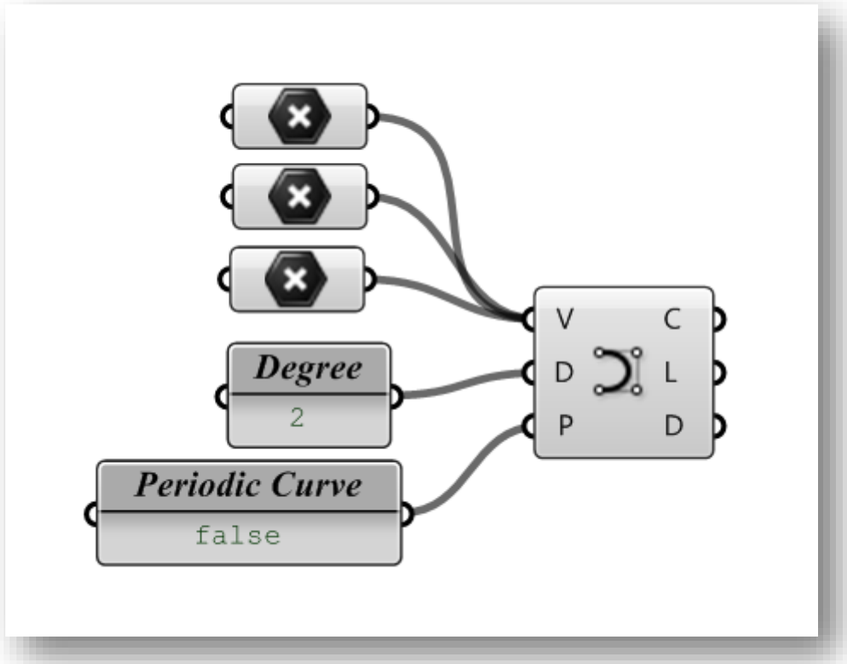
02 | About Grasshopper

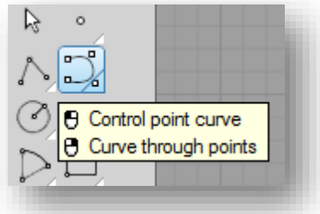
We will refer to the Grasshopper icon groups as *tabs*. They get placed in the Grasshopper *canvas*. Geometry will show up in the Rhinoceros *viewport*

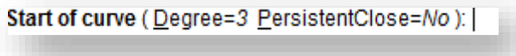


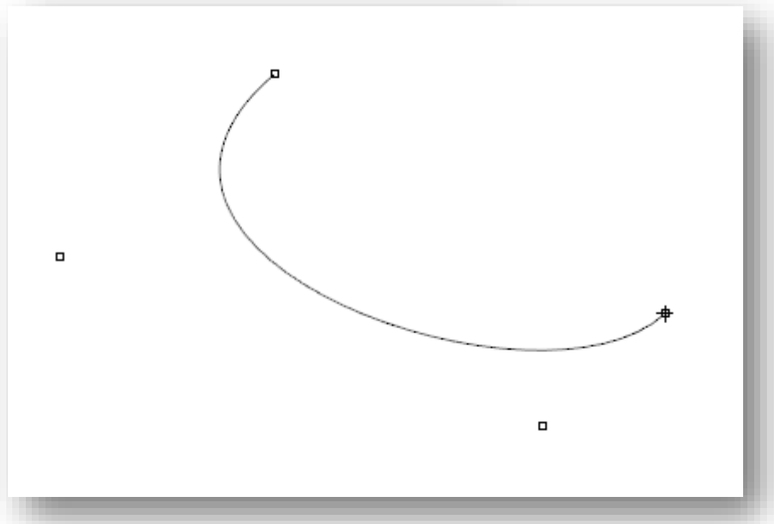
02 | About Grasshopper

| Grasshopper is a *PLUGIN* for Rhinoceros. It uses similar operations/parameters and shares the viewport with Rhino. GH-curve definition (left), Rhino curve creation (right).







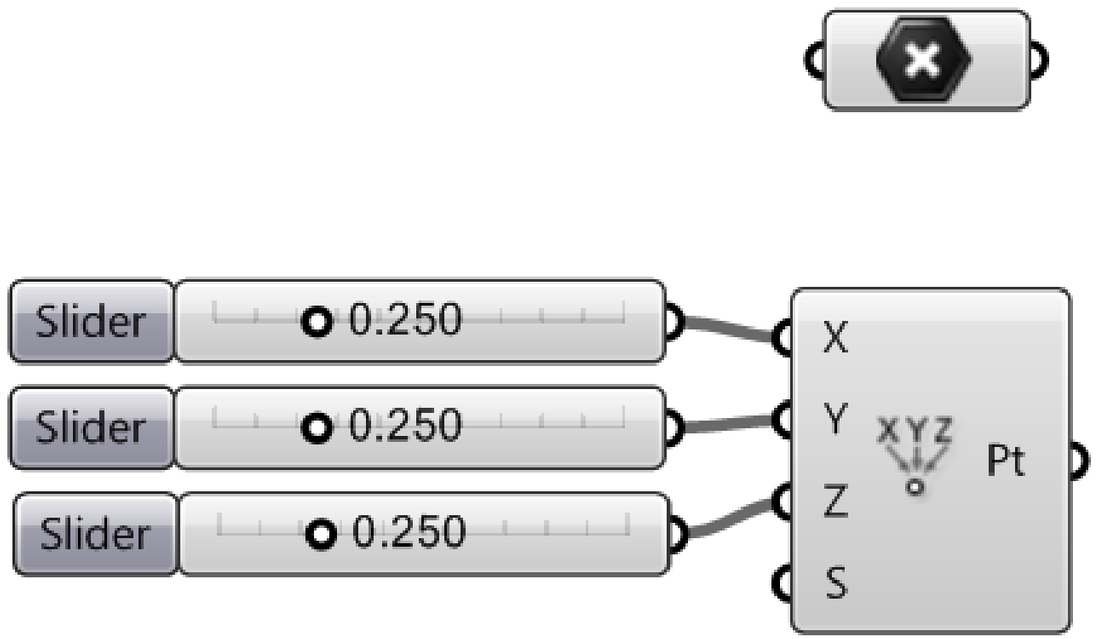


02 | About Grasshopper

| Concept of Grasshopper is that you inform components, and pass information by linking components with each other.

| Information can either come from inside GH, or by linking data directly from Rhino.

| We'll now start by creating a simple point, first using data from Rhino, and then from Grasshopper.



02 | About Grasshopper

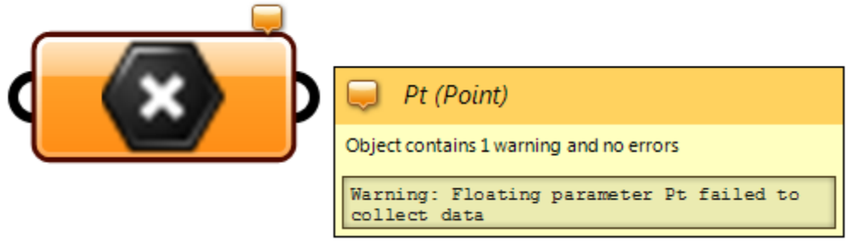
| In Grasshopper, open the Params tab and then click on the “Point” icon in the upper left corner. Drag it onto the Grasshopper canvas.

| The icon will show up in yellow. The following color-

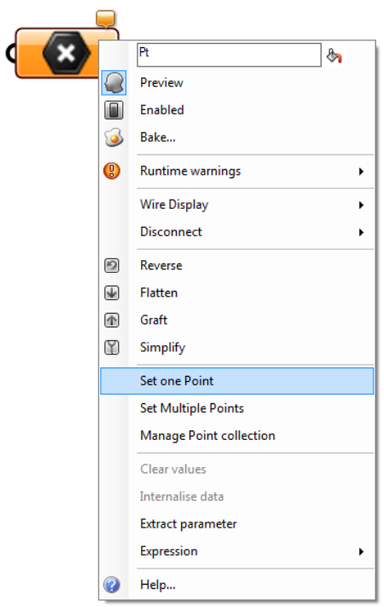
Grey – the component is working as intended
Yellow - the component is empty or that there are minor problems
Green – the component is selected
Red – there are serious problems with the component
Dark Grey – the component is hidden
Pale Grey – the component is disabled

02 | About Grasshopper

| When you put the cursor over the balloon of colored components, GH will list the problem. Our point says:

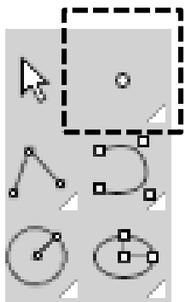


| The problem is that the component is empty. To fill it with data from Rhinoceros, right-click it and choose “Set One Point”



02 | About Grasshopper

| You can now place a point in the Rhino viewport. It is also possible to reference existing geometry. To do so, switch to Rhino and use the Point command to place a few points.



| Back in Grasshopper, right-click again on the Point component and choose again “Set one Point”. However, in Rhino go to the command bar and click on <Coordinate>. This will show the following choice. Click on Point.

Grasshopper Point type <Coordinate> (Coordinate Point Curve): |

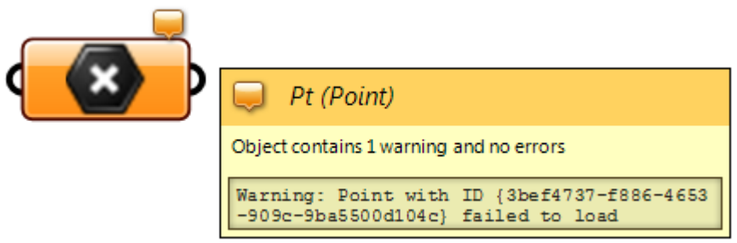
02 | About Grasshopper

| The point is now referenced in Rhino and a red cross ✖ will be overlaid over the regular Rhino-point object.

| When a Grasshopper component is selected, both the component and the contained geometry will turn green:  ✖

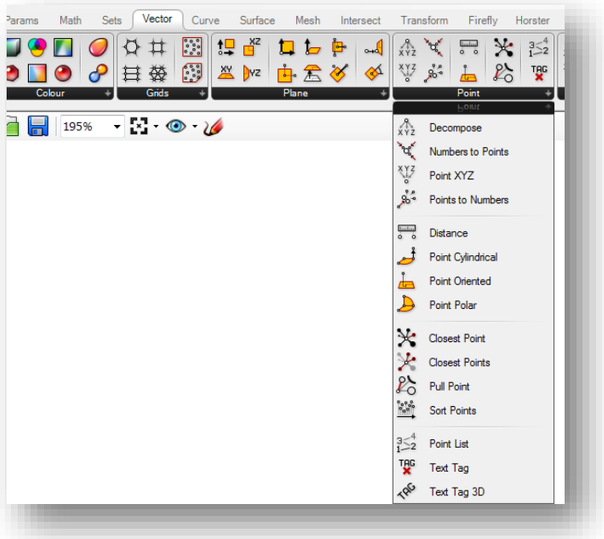
| Whenever you move a referenced object, the linked Grasshopper object will update as well. Move a point!

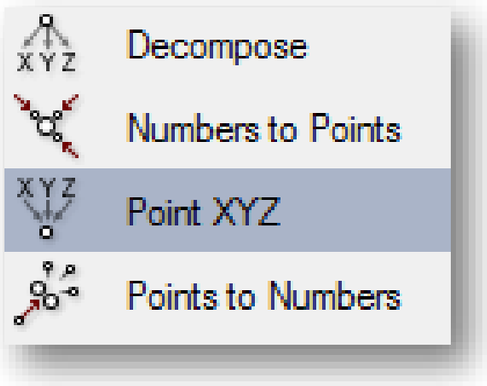
| If you delete an object that is referenced in Grasshopper, the GH component will turn yellow to show the error:



02 | About Grasshopper

| We will now create a point inside Grasshopper. In GH, go to the Vector tab and then to the Point subsection. Click the small arrow on the lower right so that all components are shown.

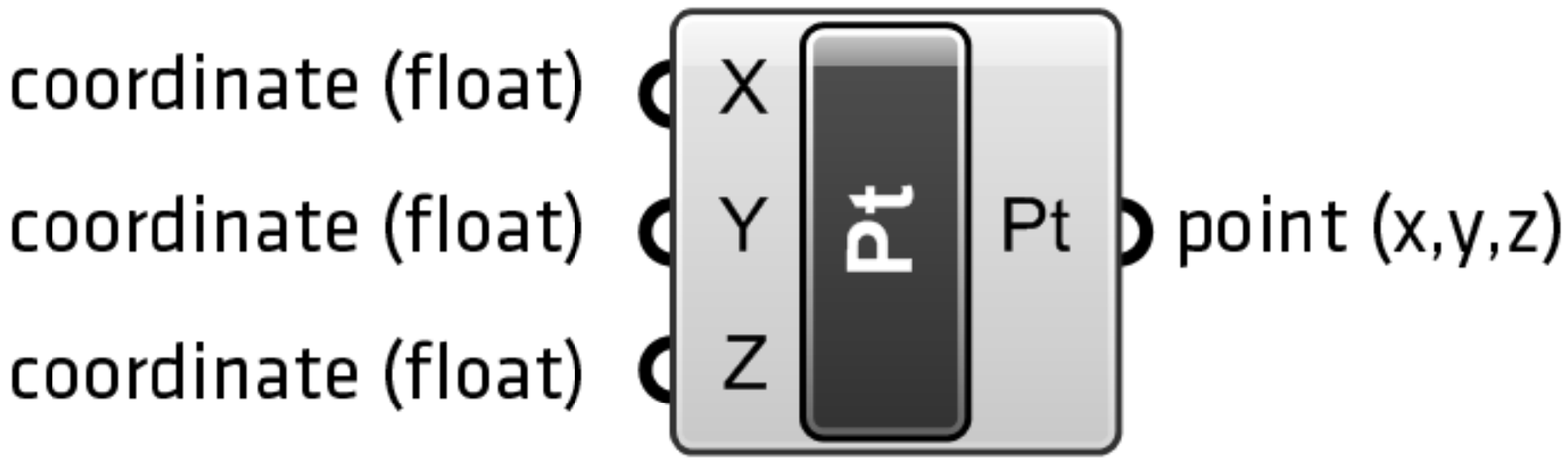




| Choose “Point XYZ” and drag the component onto the canvas.

02 | About Grasshopper

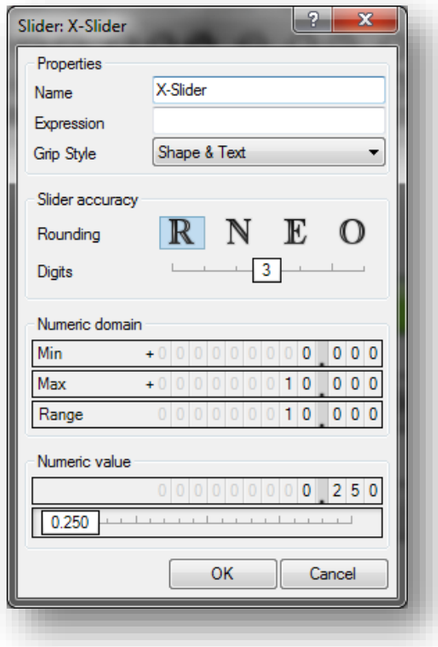
| Take a look at the component and mouse over the inputs. You will see that the Point XYZ component requires three numbers (XYZ) and outputs a point.



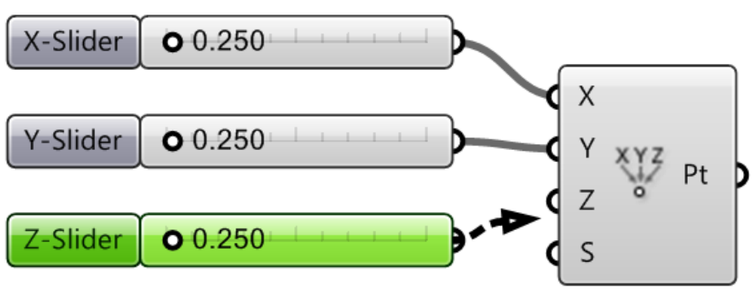
| Let's create some slider to create the XYZ values. You can find the Number Slider component in the Params tab, in the upper left corner of the Special subsection. Drag three slider onto the canvas.

02 | About Grasshopper

| You can set the properties of the slider by double-clicking its name. Call the first slider “X-Slider” and set its range to 0.0-10.0. Repeat it for the remaining to sliders, Y-Slider and Z-Slider.



| Now connect the slider output of the slider into the corresponding input of the Point component. Do this for all sliders.

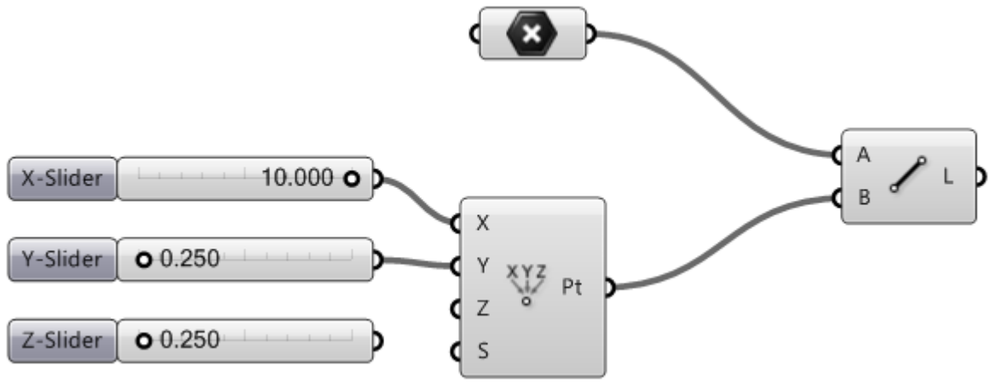


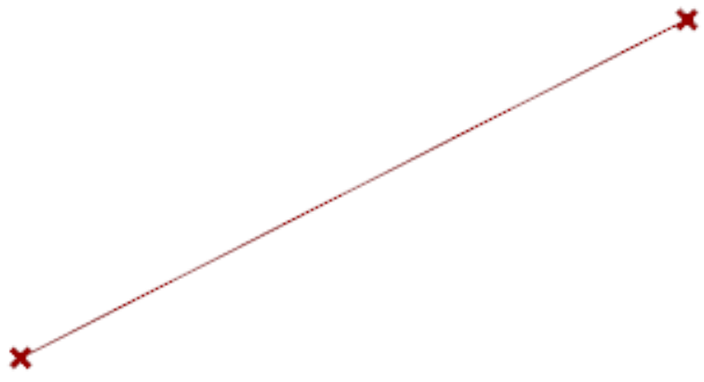
02 | About Grasshopper

| What you get, is a parametric point whose XYZ position is defined by a slider value. Note that you cannot select this point in Rhino, as it only exists in Grasshopper.

| Next, we'll create a parametric line. Get the Line component from the Curve tab, subsection Primitive and drag it onto the canvas.

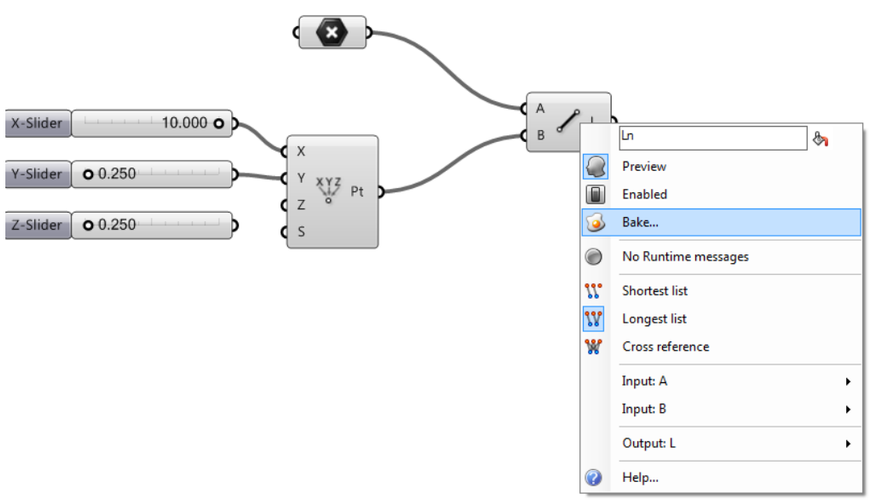
| Connect the referenced point to input A and the slider-defined point to input B. You will receive a parametric line that moves with the points.

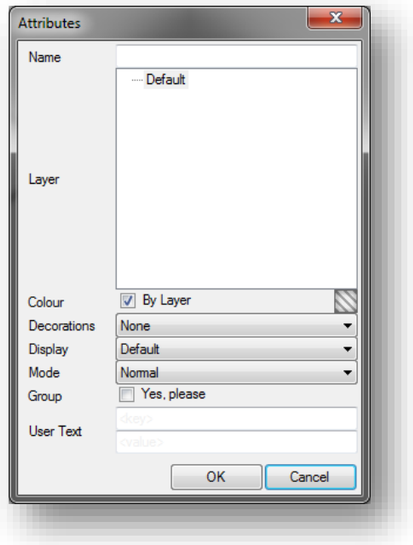




02 | About Grasshopper

| To get the line into Rhinoceros, e.g. to print it or to export it, you have to “Bake” it. Right-click on the line component and click on Bake. Choose a layer and confirm with OK.

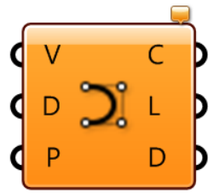




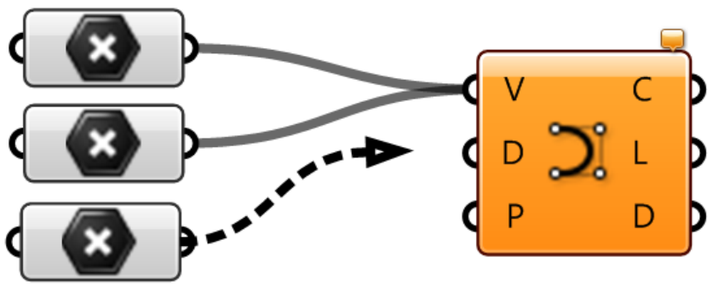
| The line will now show up in Rhinoceros. However, it is decoupled from Grasshopper, i.e. if you change the line in Grasshopper, the Rhino-line will stay the same.

02 | About Grasshopper

| Let's go back to the very first example, the control point curve. Create three points in Rhino and three point component in GH that reference one point each. Put a Control Curve component from the Curve tab, Spline subsection next to it.

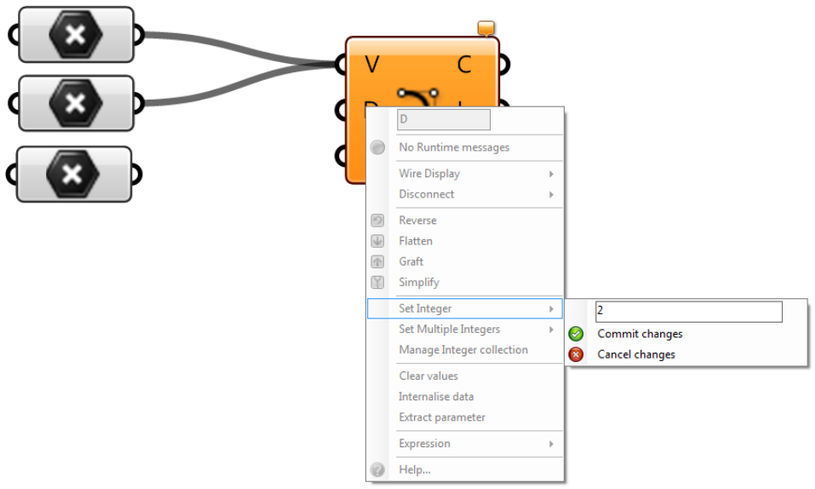


| You can connect multiple components to one input. Keep the Shift key pressed, while dragging the wire from the point component to the V[ertex] input of the curve. The order is important!



02 | About Grasshopper

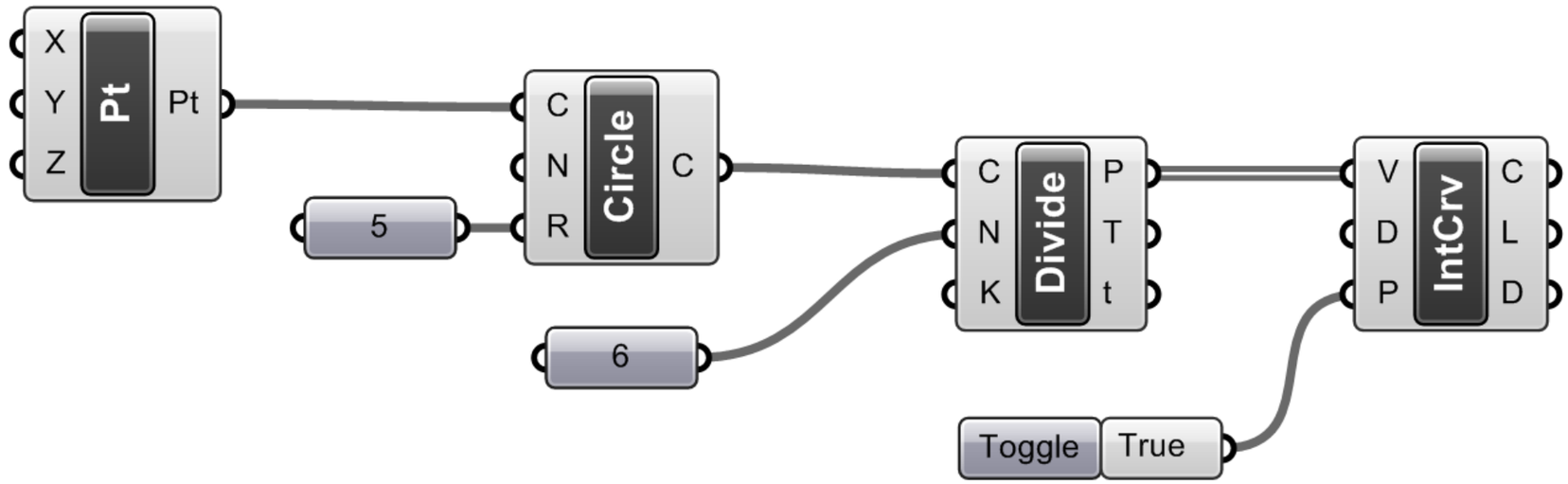
| Even after you connected the points, the curve component is still yellow. When you mouse over, it will tell you that the degree must be less than the number of points (remember Barch geometry!). The D input sets the degree of the curve. We can either connect a number component (even a slider!) to the input, or we can set it inside the component. To do so, right-click the D and go to Set Integer. Enter “2” and confirm by clicking the green check-icon.



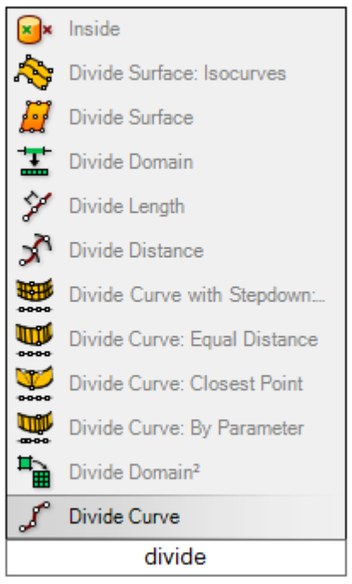
| Try to find out what the P input does and how you can change it!

02 | About Grasshopper

| Can you guess what the following definition creates? If not, try it out.

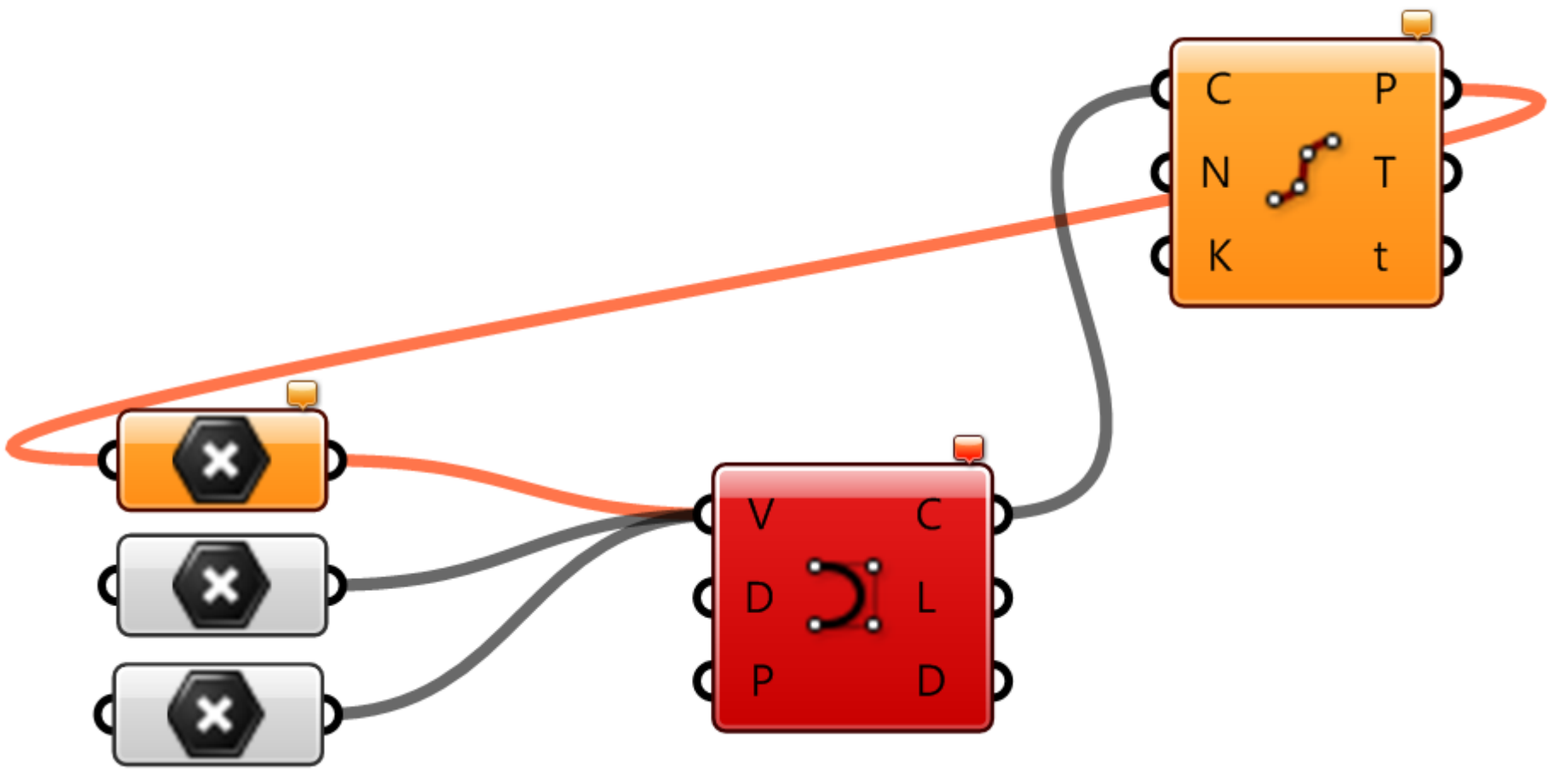


| If you cannot find some of the components, try searching for them. Double-click in empty canvas space and enter the first few letters of the component.



02 | About Grasshopper

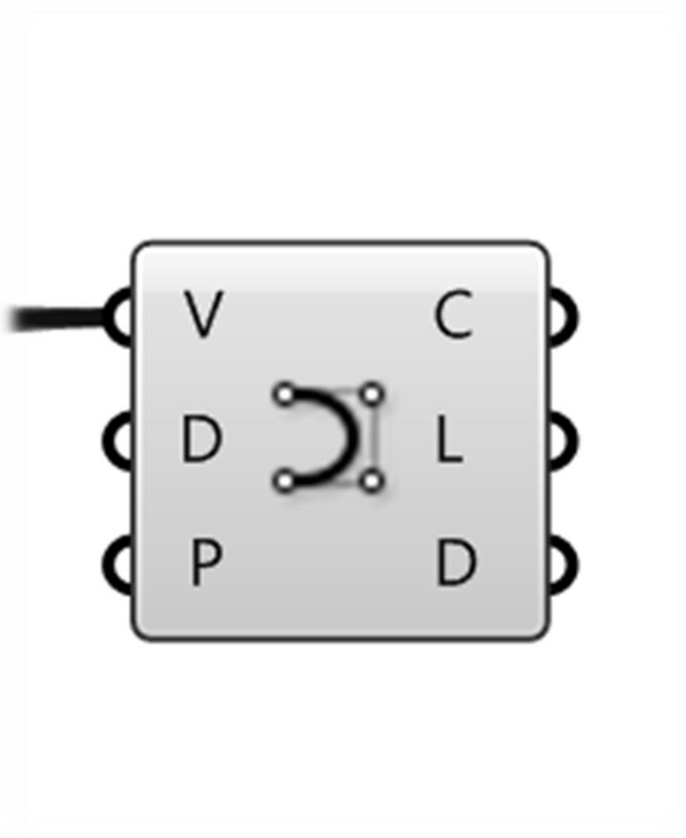
| Two more important aspects: Definitions must never loop!



02 | About Grasshopper

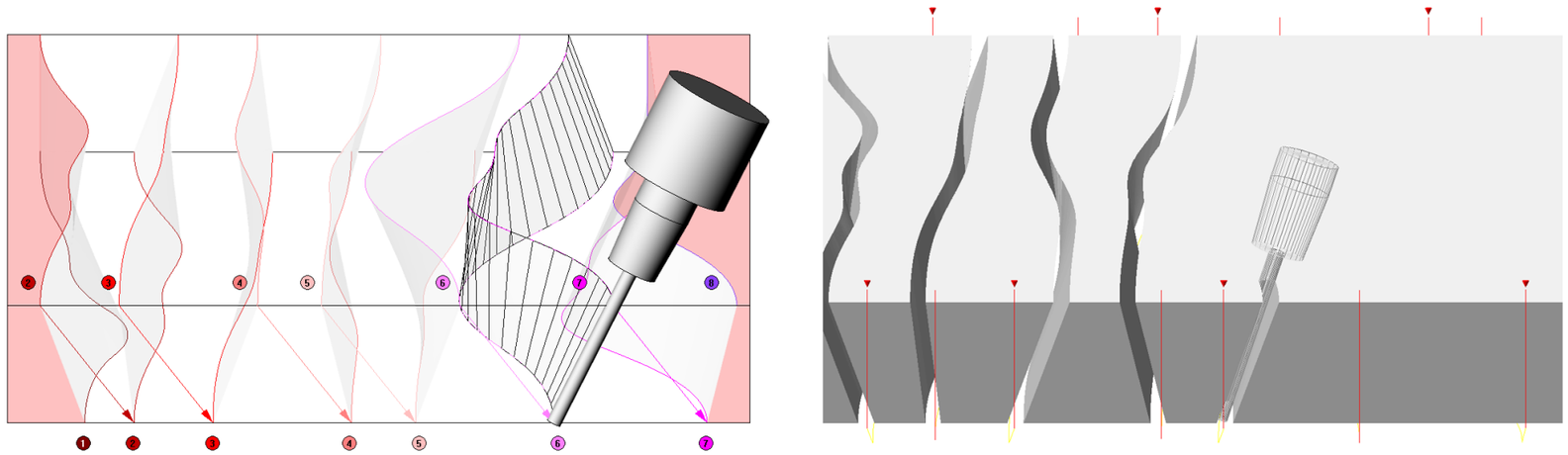
| Every GH component contains code, but the user cannot access it, i.e. it's a black box. Because of this, the user can focus on the logic, instead of having to write code himself.

```
protected override void SolveInstance(IGH_DataAccess DA)
{
  int destination = 0;
  bool flag = false;
  if (DA.GetData<int>(1, ref destination) && DA.GetData<bool>(2, ref flag))
  {
    if (destination < 1)
    {
      this.AddRuntimeMessage(GH_RuntimeMessageLevel.Error, "Degree must be higher than or equal to 1.");
      destination = 1;
    }
    if (destination > 11)
    {
      this.AddRuntimeMessage(GH_RuntimeMessageLevel.Error, "Degree must be smaller than or equal to 11.");
      destination = 11;
    }
    List<GH_Point> list = new List<GH_Point>();
    if (DA.GetDataList<GH_Point>(0, list))
    {
      List<Point3d> points = new List<Point3d>();
      foreach (GH_Point point in list)
      {
        if ((point != null) && point.IsValid)
        {
          points.Add(point.Value);
        }
      }
      if (points.Count < 2)
      {
        this.AddRuntimeMessage(GH_RuntimeMessageLevel.Error, "Insufficient vertices for a curve");
      }
      else
      {
        if (destination >= points.Count)
        {
          this.AddRuntimeMessage(GH_RuntimeMessageLevel.Blank | GH_RuntimeMessageLevel.Warning, "The degree must be less than the number of control-points");
          destination = points.Count - 1;
        }
        NurbsCurve data = NurbsCurve.Create(flag, destination, points);
        if (data != null)
        {
          DA.SetData(0, data);
          double length = data.GetLength();
          if (RhinoMath.IsValidDouble(length))
          {
            DA.SetData(1, length);
          }
          else
          {
            DA.SetData(1, 0.0);
          }
          DA.SetData(2, data.Domain);
        }
      }
    }
  }
}
```



03 | About Robots

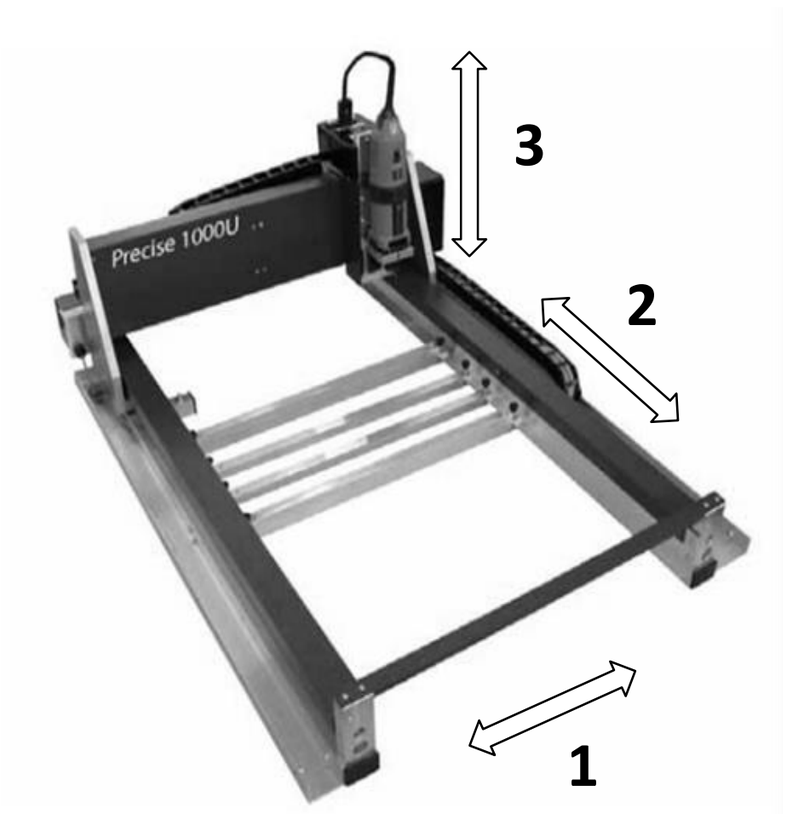
| Robots basically work like CNC machines: The first step is generating toolpaths, and the second is translating that data into a format that the machine can understand.

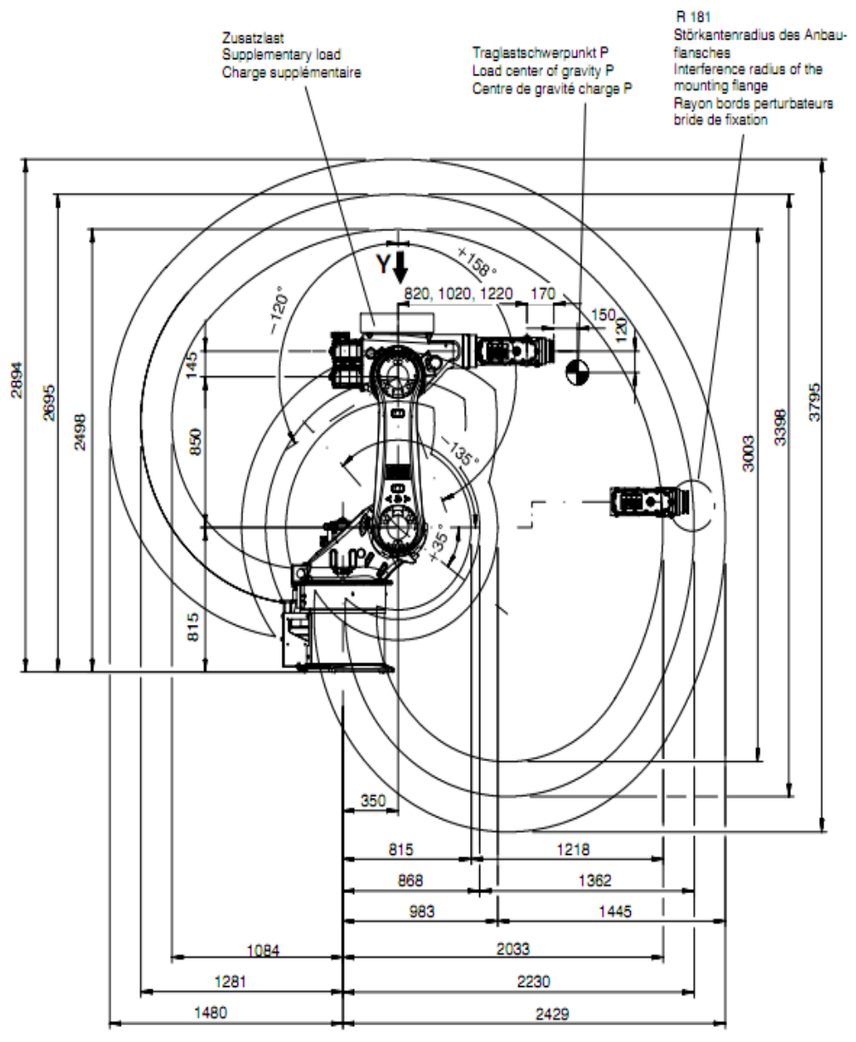


```
&ACCESS RVP
&REL 1
&PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
&PARAM EDITMASK = *
DEFDAT kukaprc_project
;FOLD EXTERNAL DECLARATIONS;%{PE}%MKUKATPBASIS,%CEXT,%VCOMMON,%P
;FOLD BASISTECH EXT;%{PE}%MKUKATPBASIS,%CEXT,%VEXT,%P
EXT  BAS (BAS_COMMAND  :IN,REAL  :IN )
DECL INT SUCCESS
;ENDFOLD (BASISTECH EXT)
;FOLD USER EXT;%{E}%MKUKATPUSER,%CEXT,%VEXT,%P
;Make here your modifications
;ENDFOLD (USER EXT)
;ENDFOLD (EXTERNAL DECLERATIONS)
INT CR_GENNUMBER = 525475
INT CR_ANSWER
DECL PDAT PPDATP2={VEL 50,ACC 100,APO_DIST 100}
DECL FDAT FP2={TOOL_NO 0,BASE_NO 0,IPO_FRAME #BASE,POINT2[] " "}
DECL E6AXIS XP2 = {A1 5,A2 -90,A3 100,A4 5,A5 -10,A6 -5,E1 0,E2 0,E3 0,E4 0,E5 0,E6 0}
DECL PDAT PPDATP4={VEL 50,ACC 100,APO_DIST 100}
DECL FDAT FP4={TOOL_NO 0,BASE_NO 0,IPO_FRAME #BASE,POINT2[] " "}
DECL FRAME XP4={X 408.930, Y 408.930, Z 118.896, A 45.000, B 16.576, C 179.321}
DECL PDAT PPDATP5={VEL 50,ACC 100,APO_DIST 100}
DECL FDAT FP5={TOOL_NO 0,BASE_NO 0,IPO_FRAME #BASE,POINT2[] " "}
DECL FRAME XP5={X 532.239, Y 554.277, Z 755.706, A 41.949, B 16.576, C 177.653}
DECL PDAT PPDATP3={VEL 50,ACC 100,APO_DIST 100}
DECL FDAT FP3={TOOL_NO 0,BASE_NO 0,IPO_FRAME #BASE,POINT2[] " "}
DECL E6AXIS XP3 = {A1 5,A2 -90,A3 100,A4 5,A5 -10,A6 -5,E1 0,E2 0,E3 0,E4 0,E5 0,E6 0}
ENDDAT
```

03 | About Robots

| We'll start with the second step. Try to figure out what control data a three axis milling machine (left) requires, as opposed to a six-axis industrial robot.

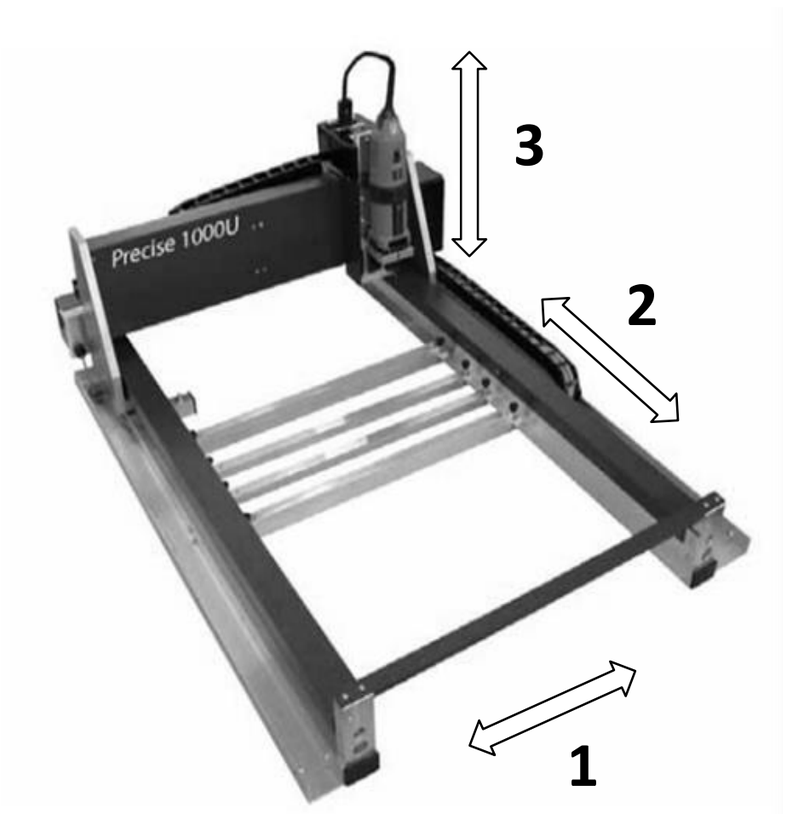


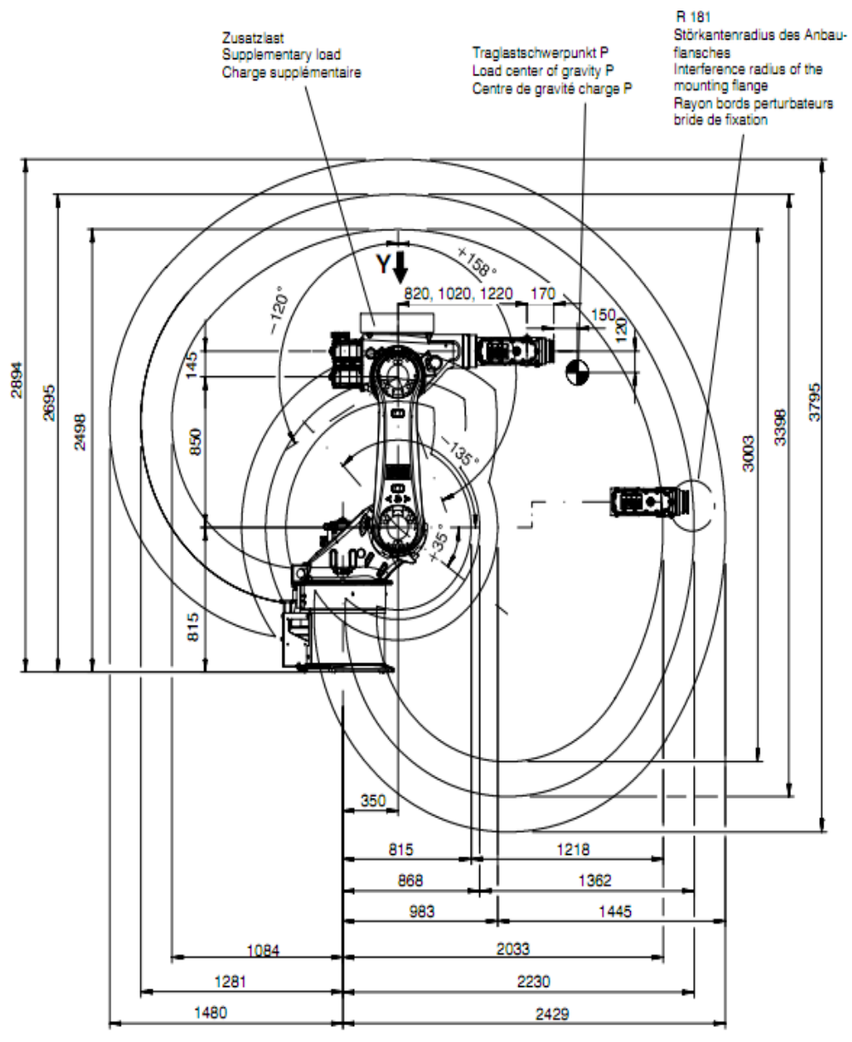


03 | About Robots

| We'll start with the second step. Try to figure out what control data a three axis milling machine (left) requires, as opposed to a six-axis industrial robot.

XYZ

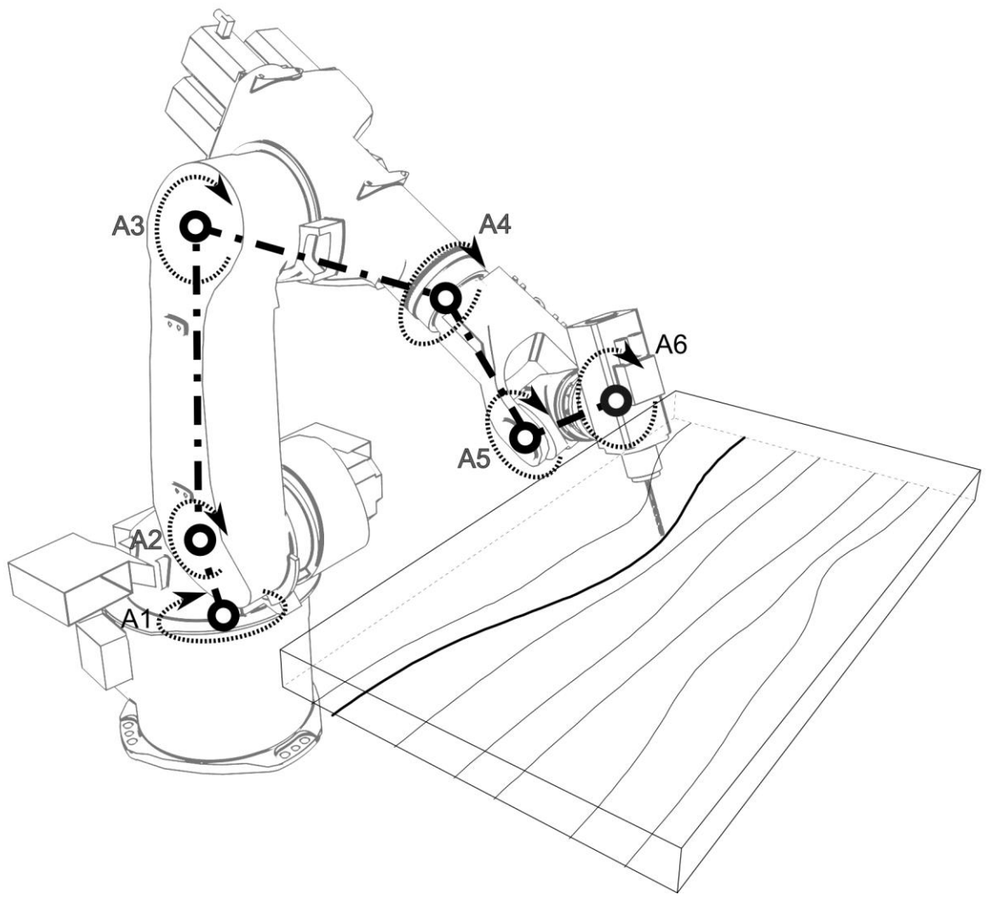




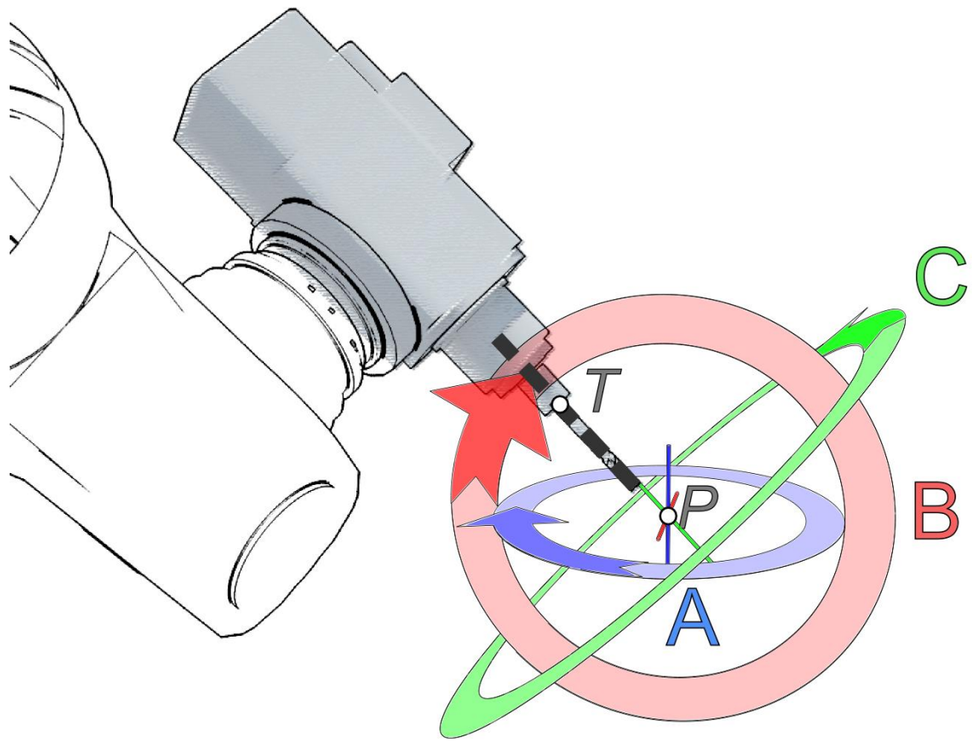
03 | About Robots

| A six axis robot either requires one rotation value for each axis, or a defined toolframe via XYZ ABC. This creates a plane, defining the position of the end-effector in 3D space.

A01 A02 A03 A04 A05 A06

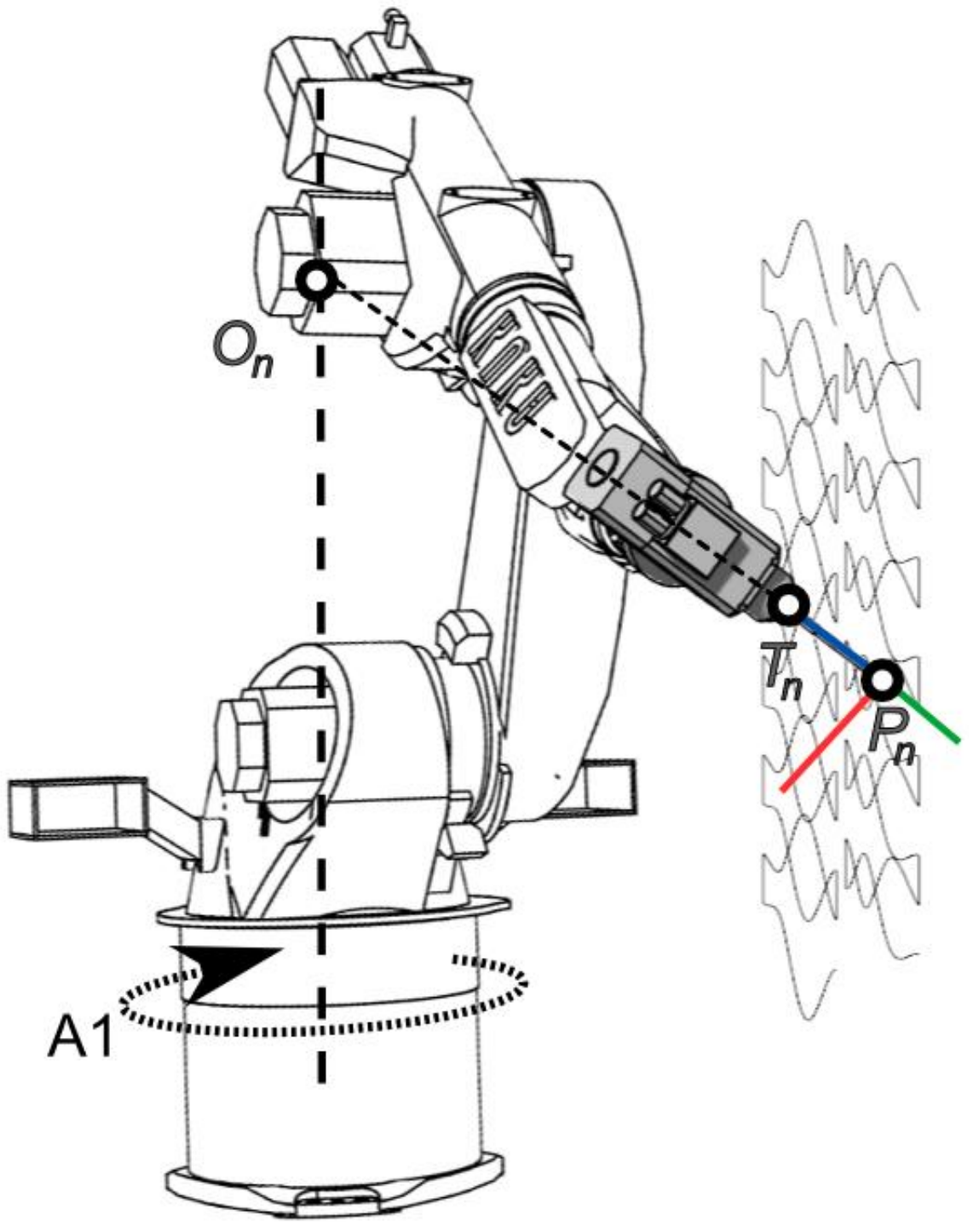


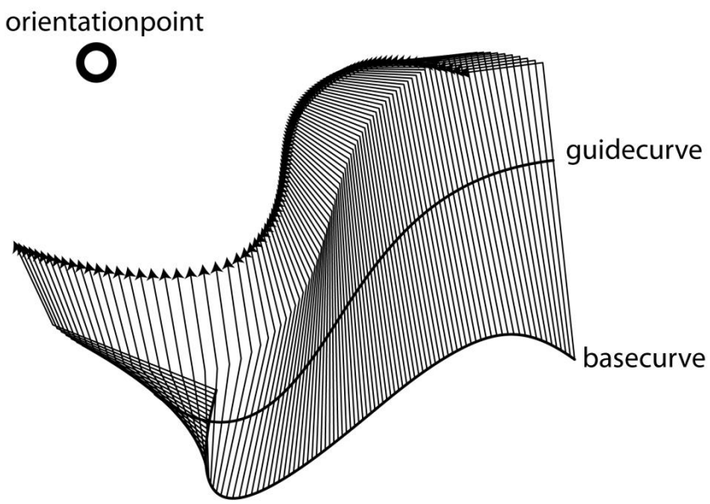
XYZ ABC

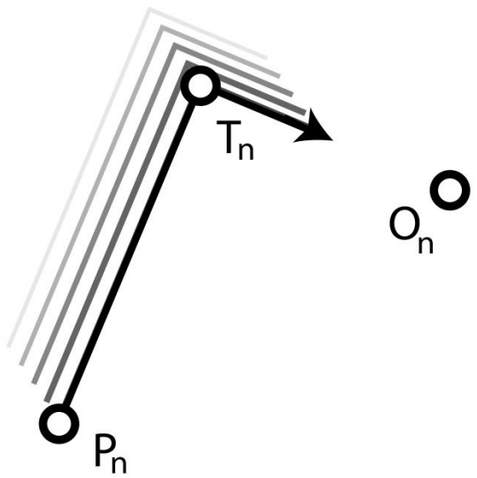


03 | About Robots

| Our task is therefore to create toolpaths that contain enough information to control a robot, i.e. we need three points at every toolpath position to define the toolplane

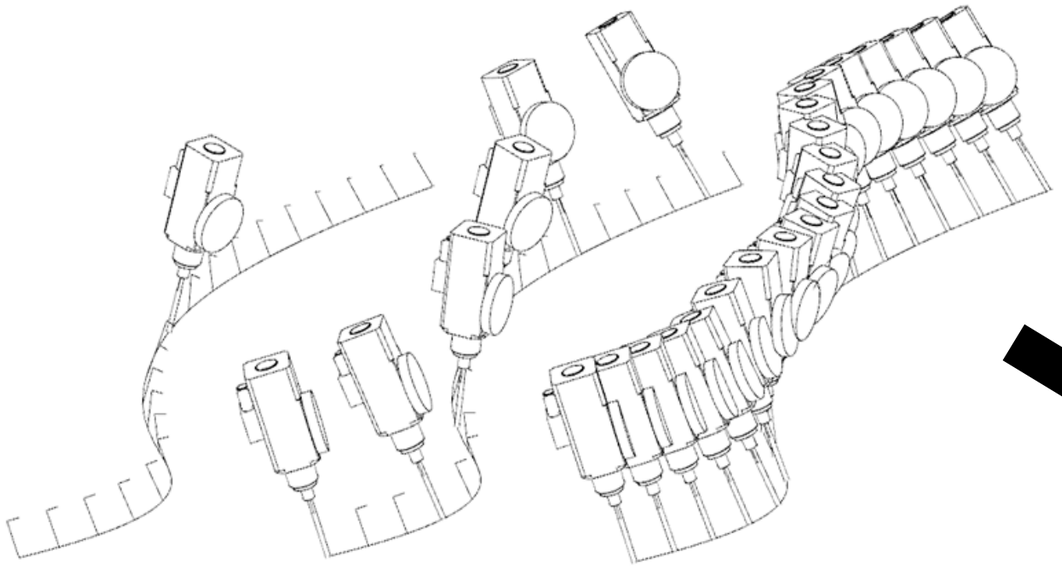






03 | About Robots

| Once we have the toolpaths, KUKA|prc will take care of the formatting and code.



```
&ACCESS RVP
&REL 1
&PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
&PARAM EDITMASK = *
DEFDAT kukaprc_project
;FOLD EXTERNAL DECLARATIONS;%{PE}%MKUKATPBASIS,%CEXT,%VCOMMON,%P
;FOLD BASISTECH EXT;%{PE}%MKUKATPBASIS,%CEXT,%VEXT,%P
EXT  BAS (BAS_COMMAND  :IN,REAL  :IN )
DECL INT SUCCESS
;ENDFOLD (BASISTECH EXT)
;FOLD USER EXT;%{E}%MKUKATPUSER,%CEXT,%VEXT,%P
;Make here your modifications
;ENDFOLD (USER EXT)
;ENDFOLD (EXTERNAL DECLERATIONS)
INT CR_GENNUMBER = 525475
INT CR_ANSWER
DECL PDAT PPDATP2={VEL 50,ACC 100,APO_DIST 100}
DECL FDAT FP2={TOOL_NO 0,BASE_NO 0,IPO_FRAME #BASE,POINT2[] " "}
DECL E6AXIS XP2 = {A1 5,A2 -90,A3 100,A4 5,A5 -10,A6 -5,E1 0,E2 0,E3 0,E4 0,E5 0,E6 0}
DECL PDAT PPDATP4={VEL 50,ACC 100,APO_DIST 100}
DECL FDAT FP4={TOOL_NO 0,BASE_NO 0,IPO_FRAME #BASE,POINT2[] " "}
DECL FRAME XP4={X 408.930, Y 408.930, Z 118.896, A 45.000, B 16.576, C 179.321}
DECL PDAT PPDATP5={VEL 50,ACC 100,APO_DIST 100}
DECL FDAT FP5={TOOL_NO 0,BASE_NO 0,IPO_FRAME #BASE,POINT2[] " "}
DECL FRAME XP5={X 532.239, Y 554.277, Z 755.706, A 41.949, B 16.576, C 177.653}
DECL PDAT PPDATP3={VEL 50,ACC 100,APO_DIST 100}
DECL FDAT FP3={TOOL_NO 0,BASE_NO 0,IPO_FRAME #BASE,POINT2[] " "}
DECL E6AXIS XP3 = {A1 5,A2 -90,A3 100,A4 5,A5 -10,A6 -5,E1 0,E2 0,E3 0,E4 0,E5 0,E6 0}
ENDDAT
```

Tilted Forms: Introduction to KUKA|prc

KUKA|prc

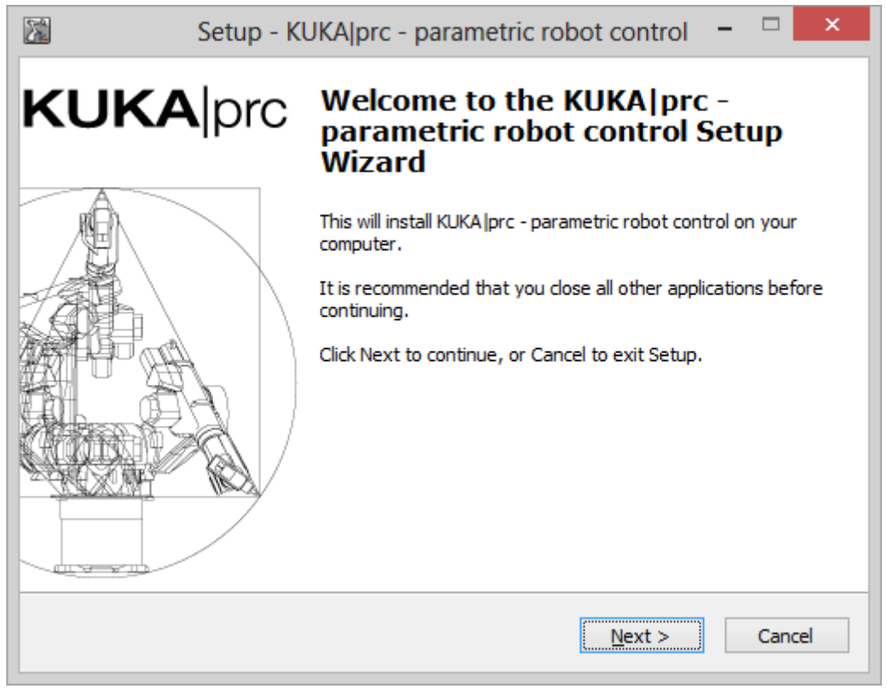
parametric robot control for grasshopper

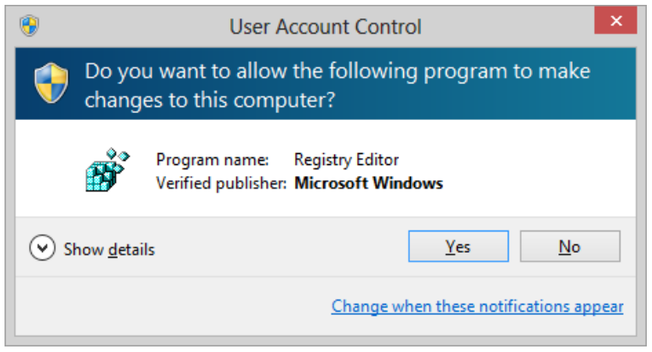
04 | Robot Programming using KUKA|prc

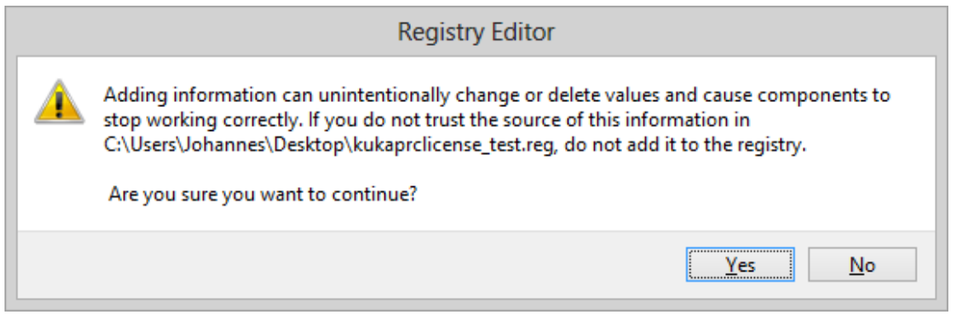
| Close all Rhinoceros windows.

| Double-click the kukaprcinstaller.exe to install KUKA|prc into the Grasshopper component library.

| Then double-click the license file (*.reg) and confirm all security prompts with YES



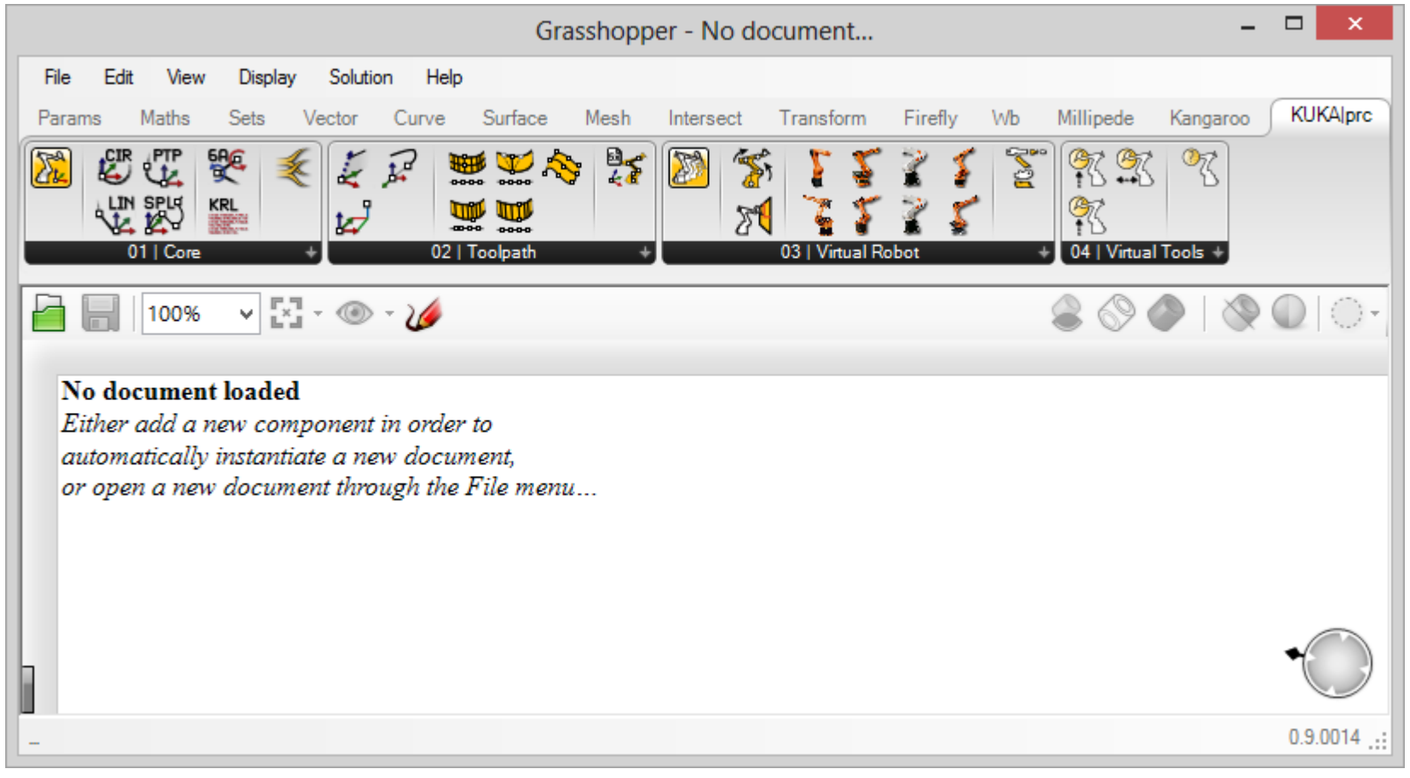




04 | Robot Programming using KUKA|prc

| Restart Rhino and create a new Rhinoceros file. Use millimetres as units, as nearly all CNC machines work with mm-units.

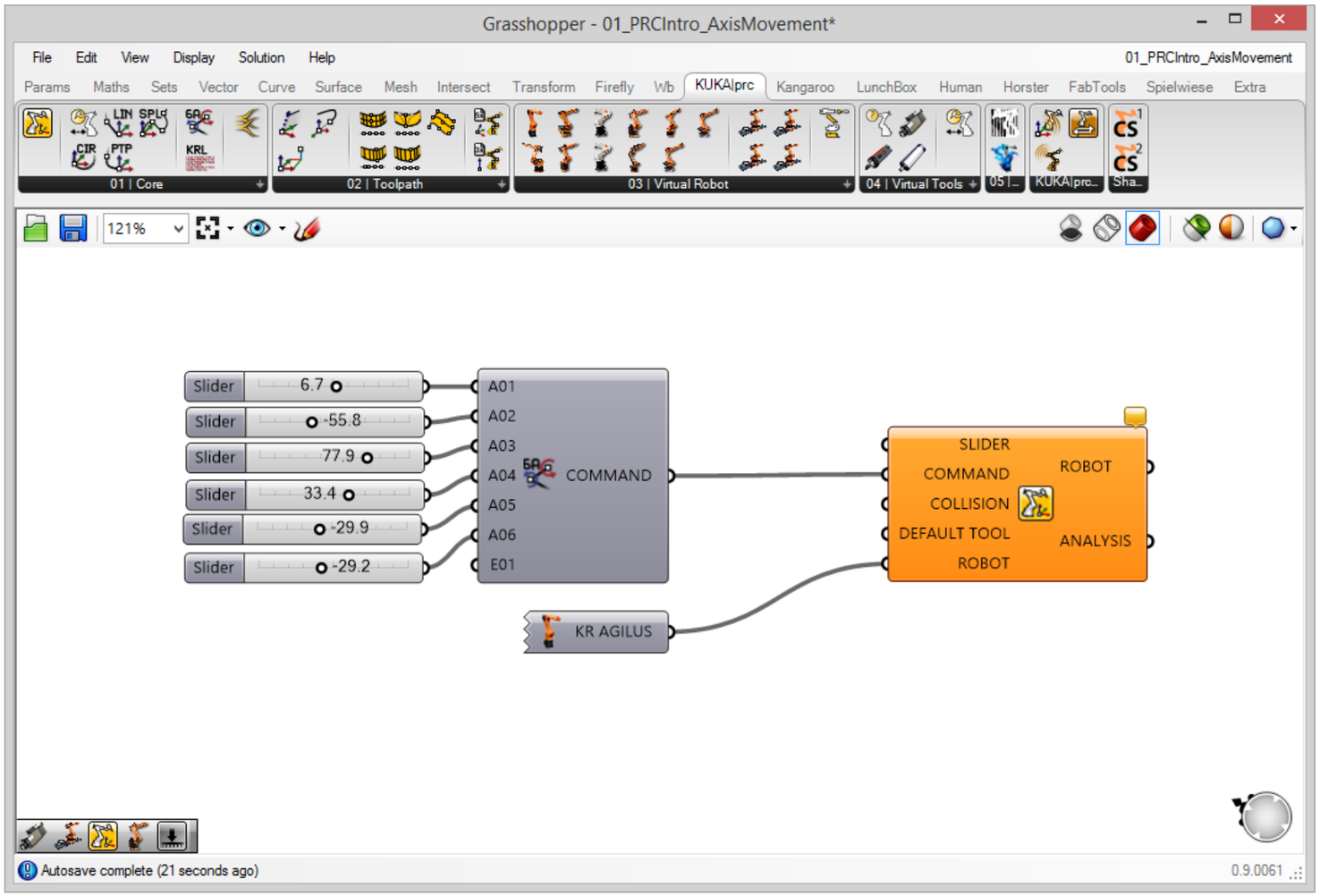
| Launch Grasshopper. If the installation was successful, you will see a new tab called “KUKA|prc”

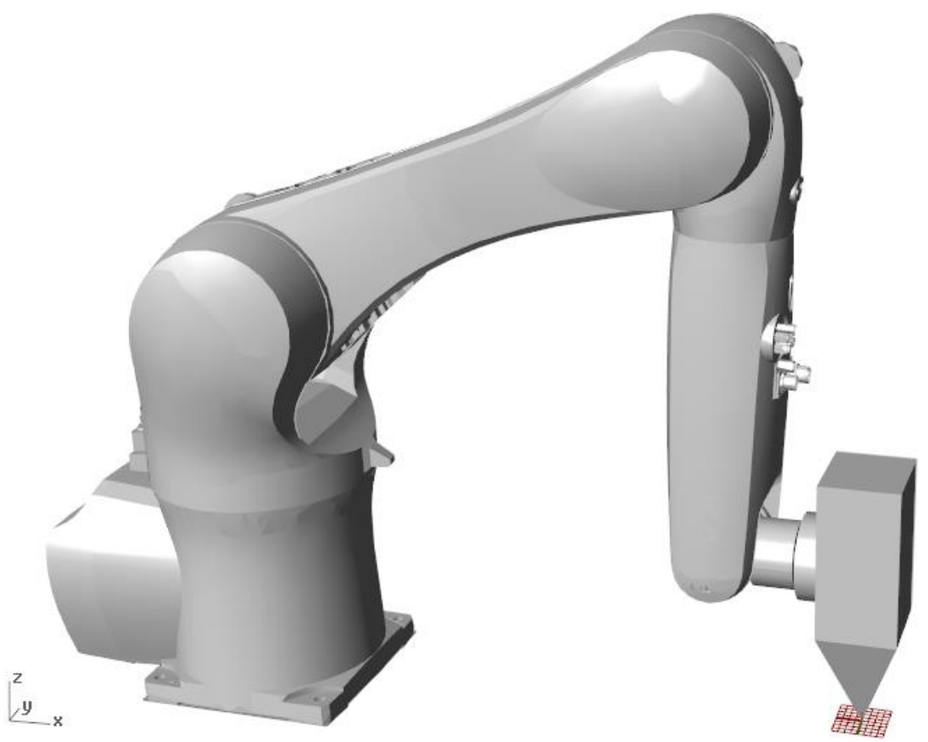


04 | Robot Programming using KUKA|prc

| Open the 01_PRCIntro_AxisMovement Grasshopper file.

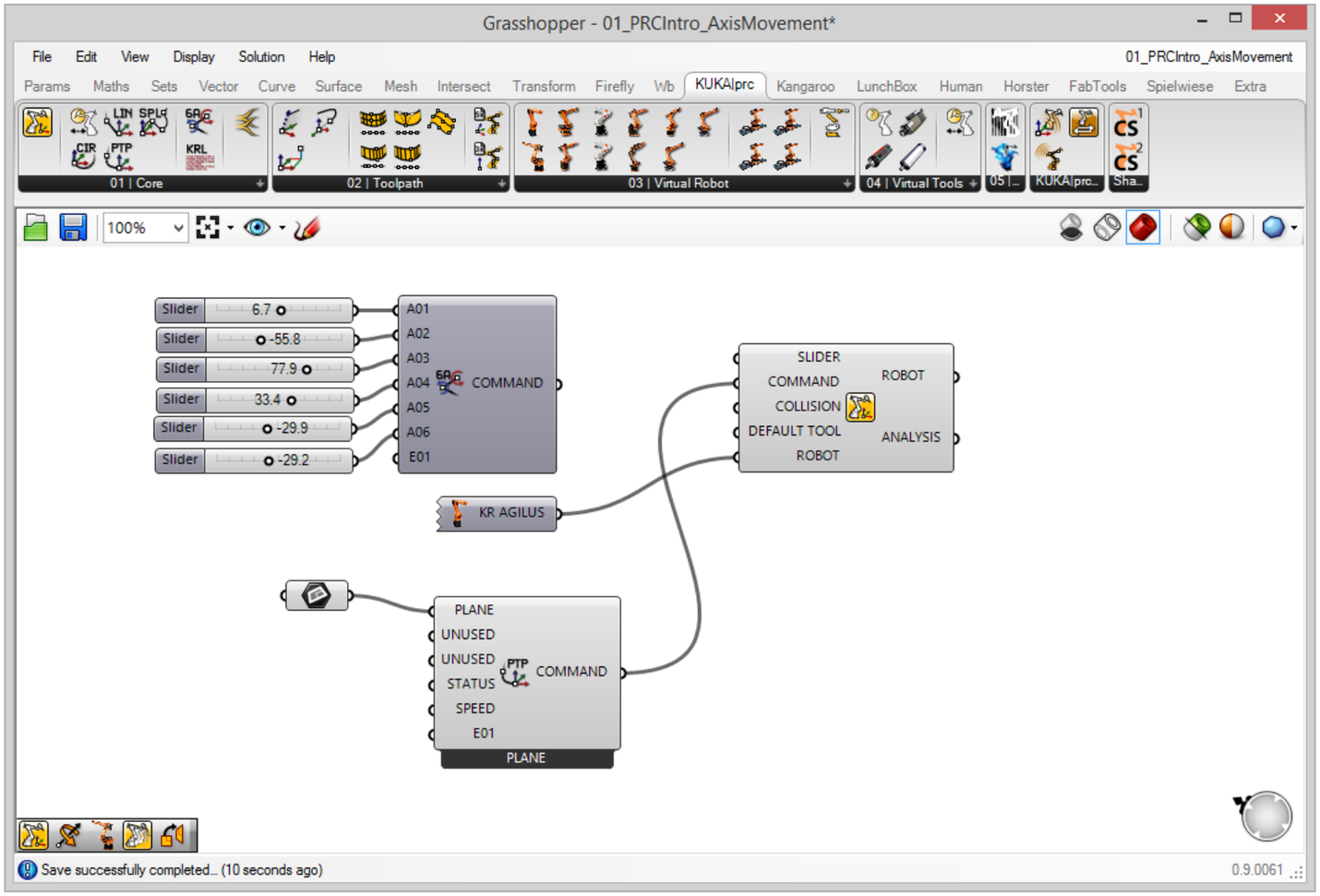
| Move the robot's tooltip to the plane position as quickly and accurately as possible.

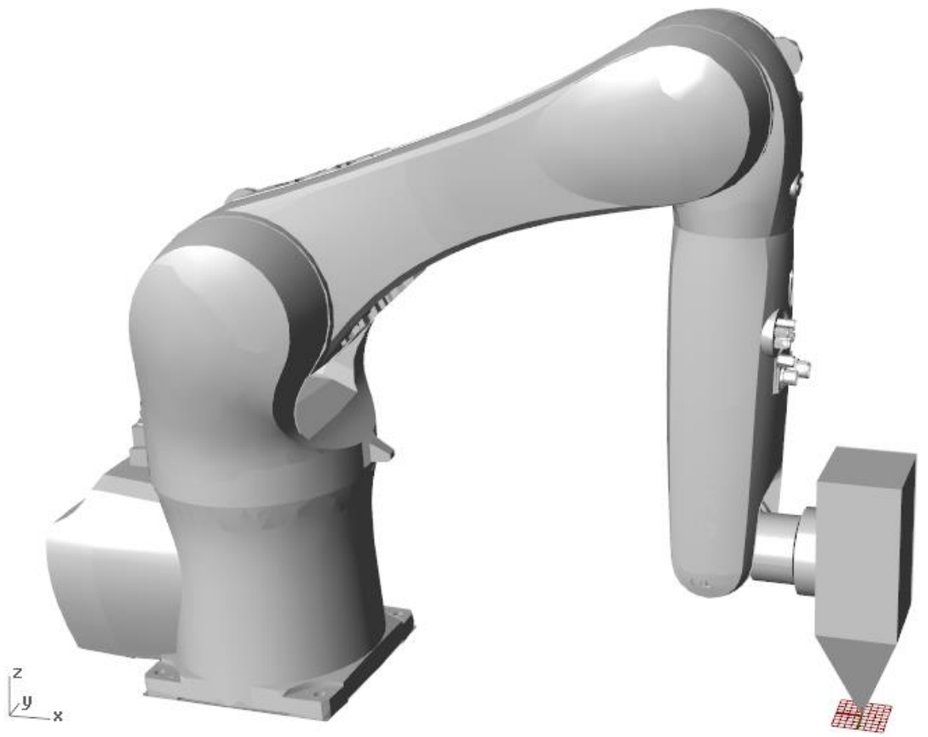




04 | Robot Programming using KUKA|prc

| Its much quicker to let mathematic algorithms solve that problem automatically: Inverse Kinematics.





04 | Robot Programming using KUKA|prc

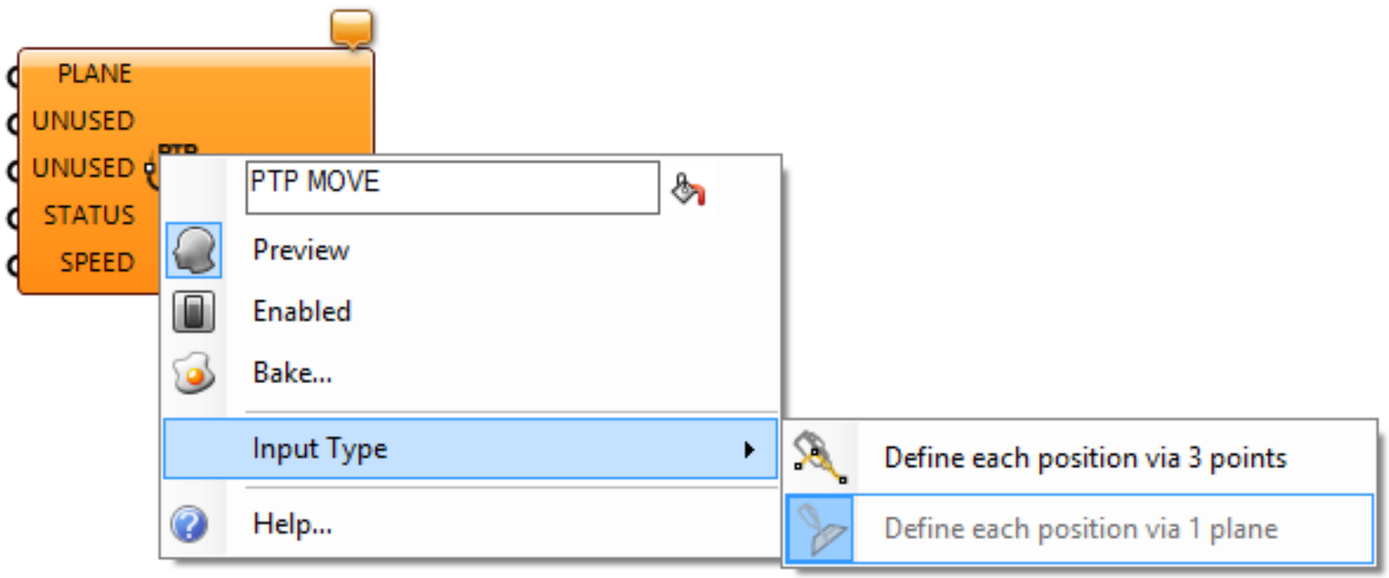
| KUKA|prc offers four categories:

The Core category containing the core component and essential movement commands.
The Toolpath category with components that e.g. facilitate toolpath operations.
The Virtual Robot category, with the virtual robot models and linear axes.
The Virtual Tools category with a selection of robotic tools.

04 | Robot Programming using KUKA|prc

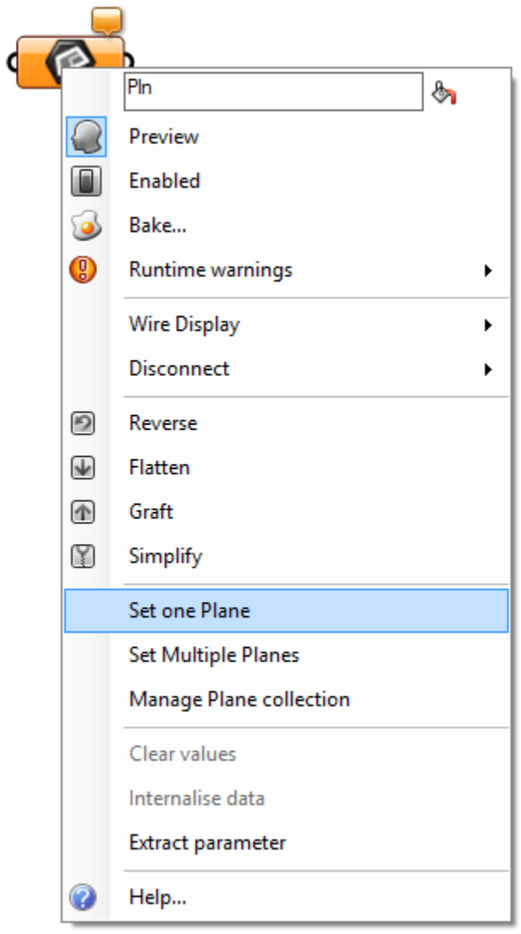
| Before, we programmed robotic movements by recording positions and the replaying them. At a basic level, KUKA|prc works exactly like that.

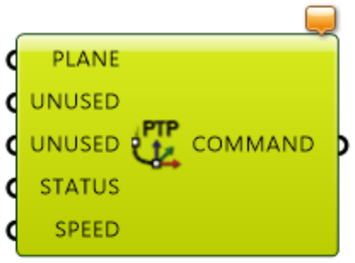
| Get a “PTP Movement” component. By default, each movement requires the definition of three points, but we can also switch the input to planes. To do so, right-click the center of the component and choose the plane option as below:

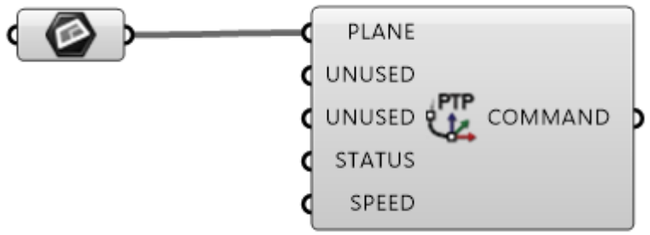


04 | Robot Programming using KUKA|prc

| Now get a Plane container from the Params tab of Grasshopper. Right-click it and choose “Set One Plane”. Define a plane in Rhino and connect the plane to the PLANE input of the PTP component.

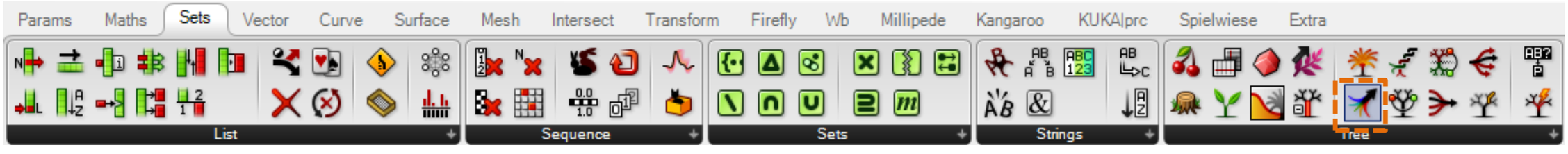




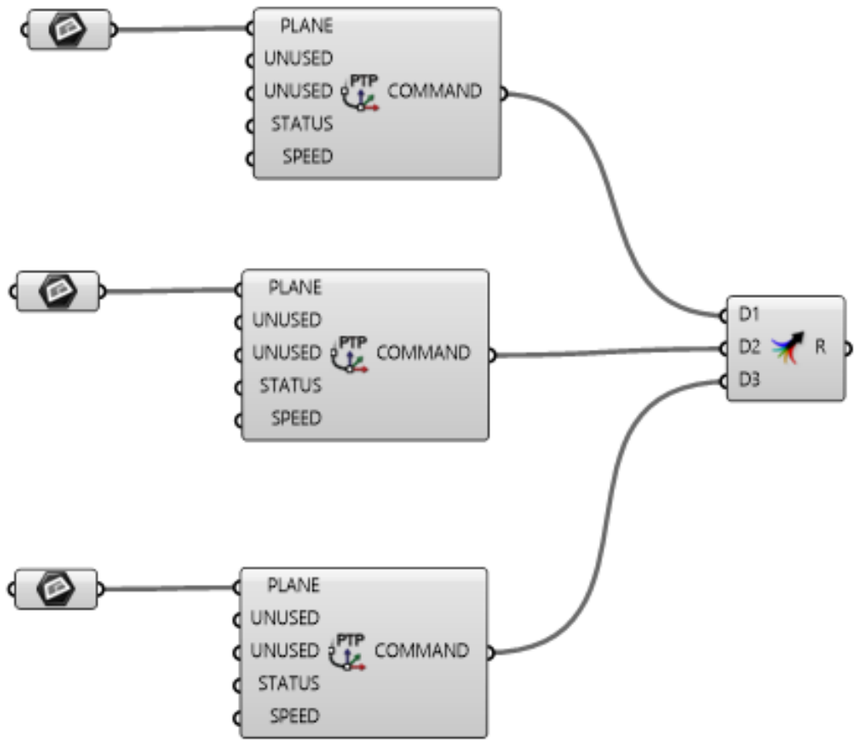


04 | Robot Programming using KUKA|prc

| Now get a Merge component from Sets/Tree and connect the movements in order.

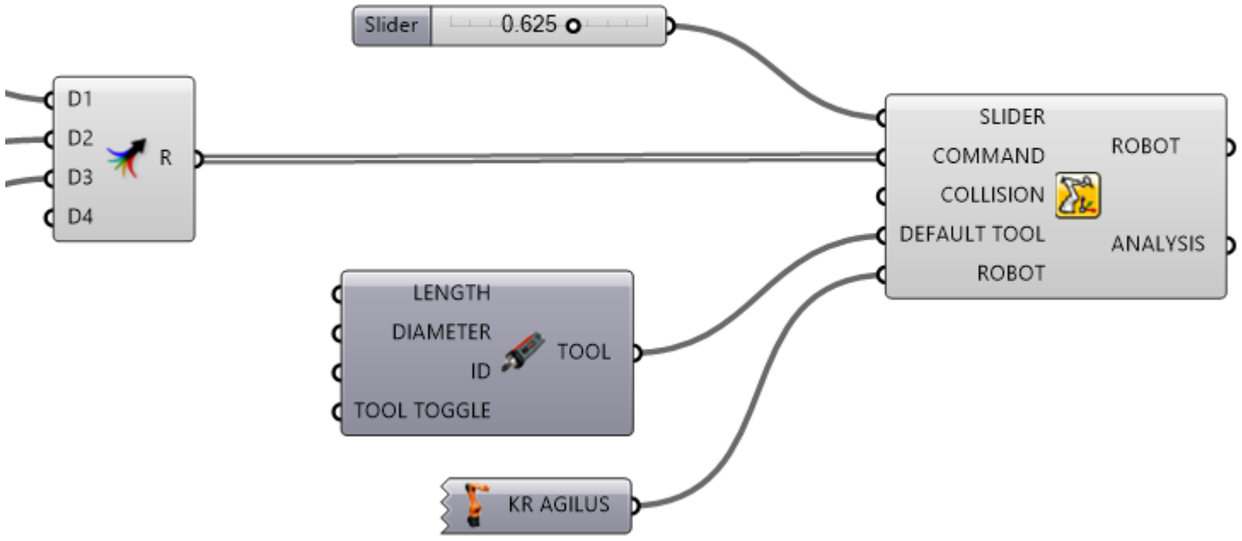


| It should look as below. The order in the Merge component is reflected in the order the commands are processed.



04 | Robot Programming using KUKA|prc

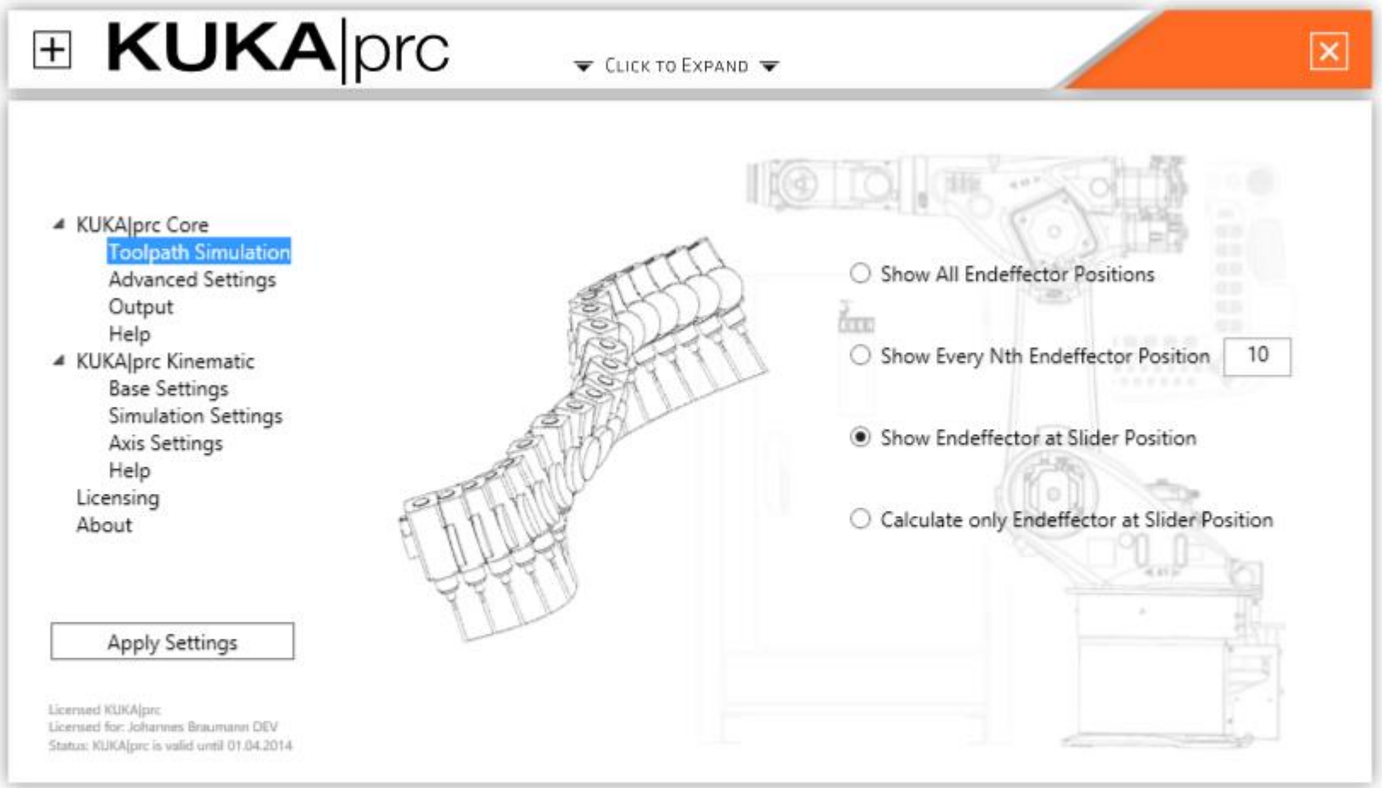
| These commands will be processed by the KUKA|prc Core component from the KUKA|prc Core tab. It has five inputs: Connect a number slider to the SLIDER input. The generated PTP commands should be plugged into the COMMAND input. KUKA|prc contains several predefined tools. Choose the Kress 1050 spindle from the Virtual Tools tab and connect it to DEFAULT TOOL. As the ROBOT, use the KUKA Agilus component.



04 | Robot Programming using KUKA|prc

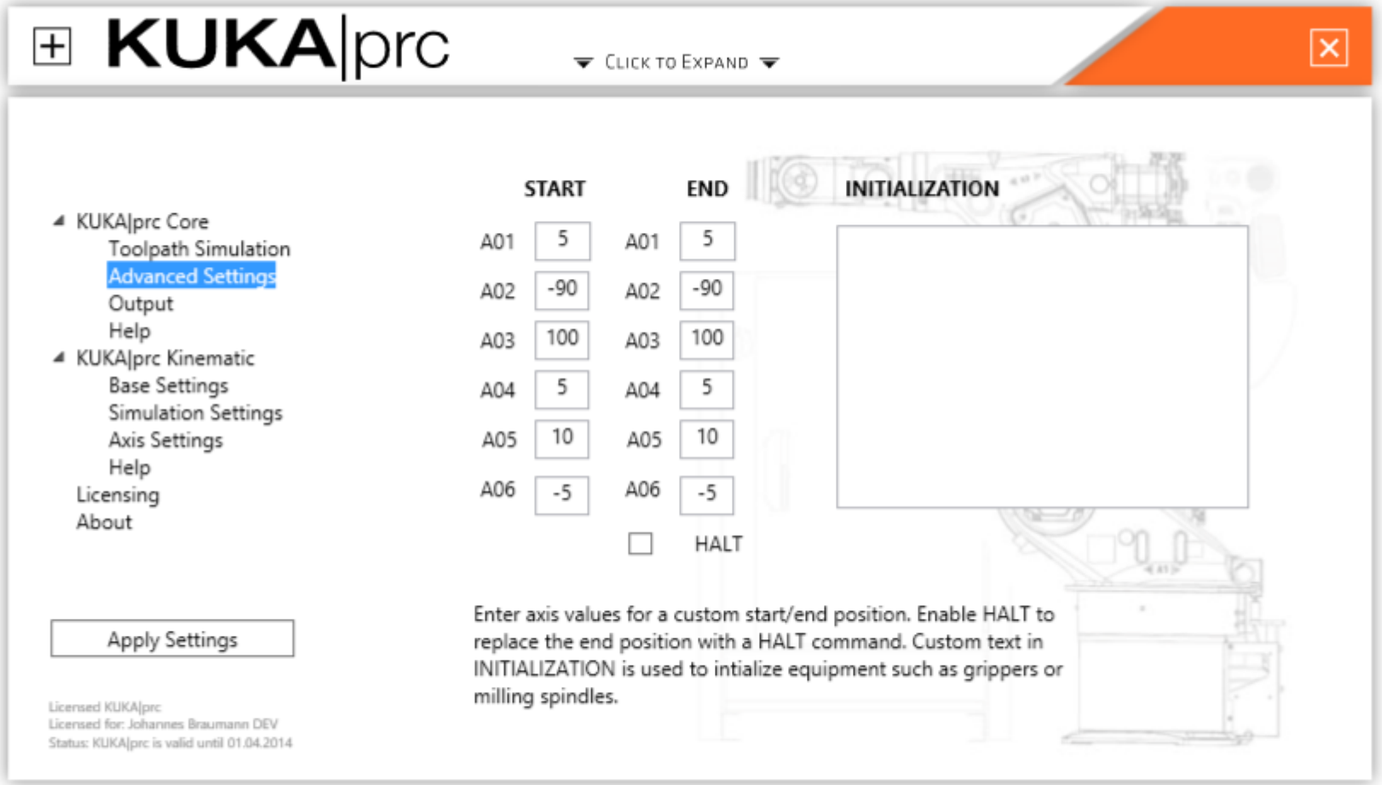
| The KUKA|prc has to be set up before it can be used. To do so, double-click it, and a new window will pop up.

| By default, the endeffector will be shown at the position defined by the slider.



04 | Robot Programming using KUKA|prc

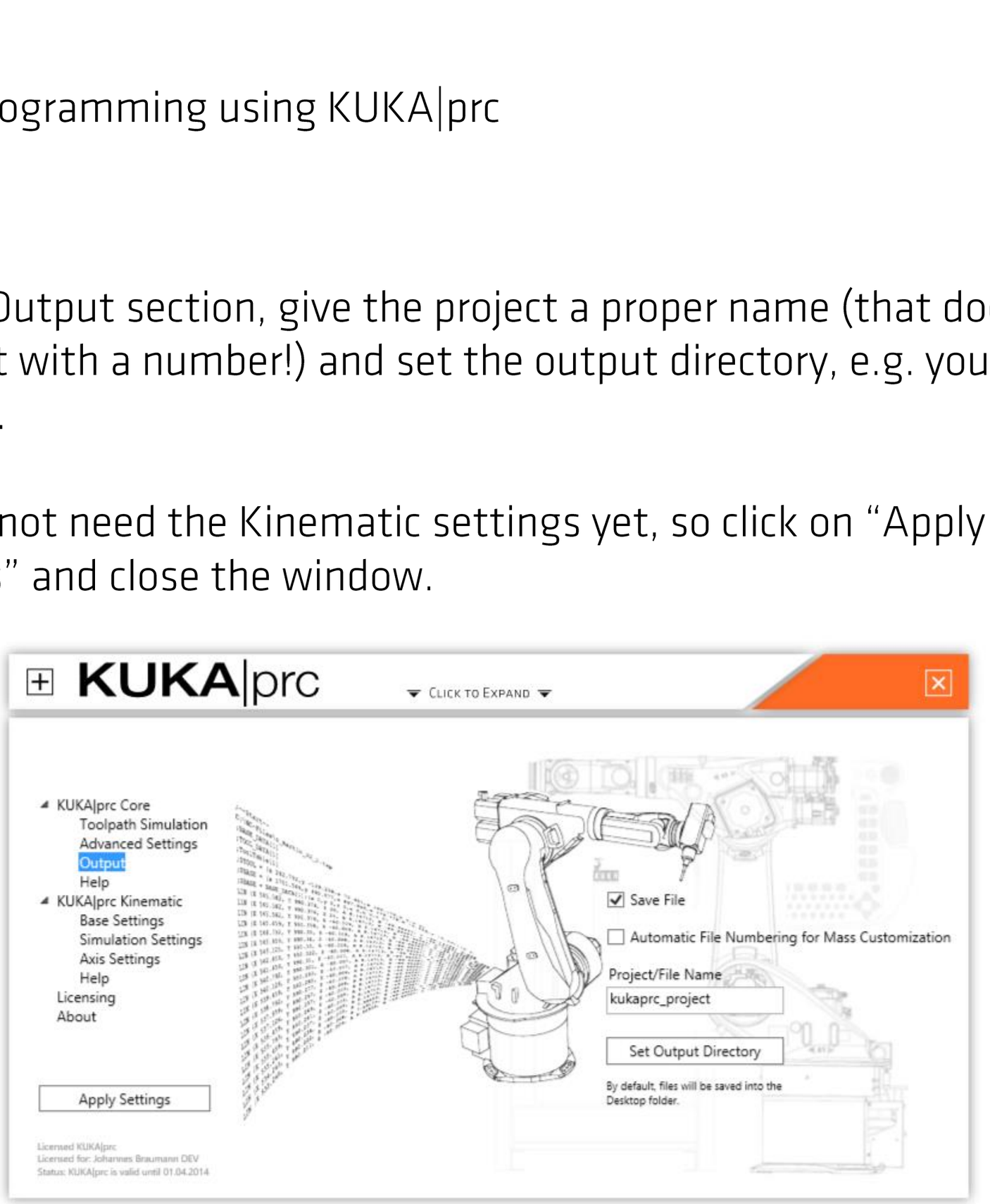
| In Advanced Settings, you can set the start and end position of the robot via direct axis values.



04 | Robot Programming using KUKA|prc

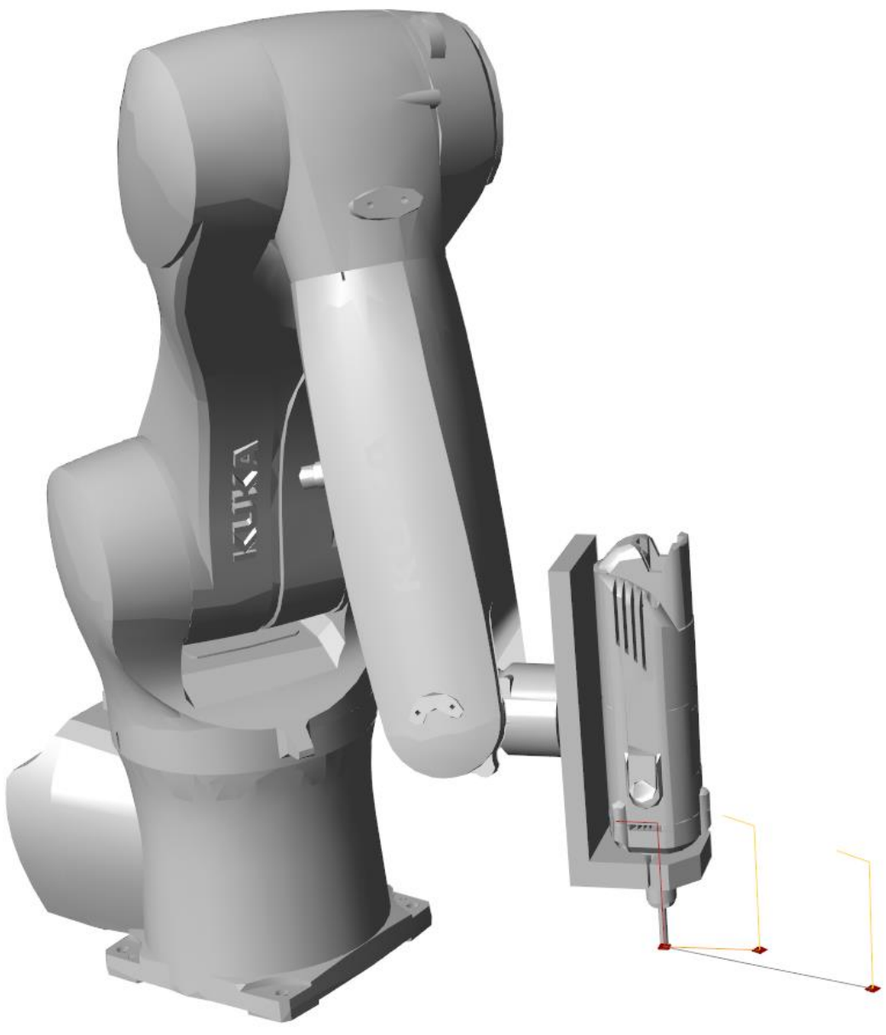
| In the Output section, give the project a proper name (that does not start with a number!) and set the output directory, e.g. your desktop.

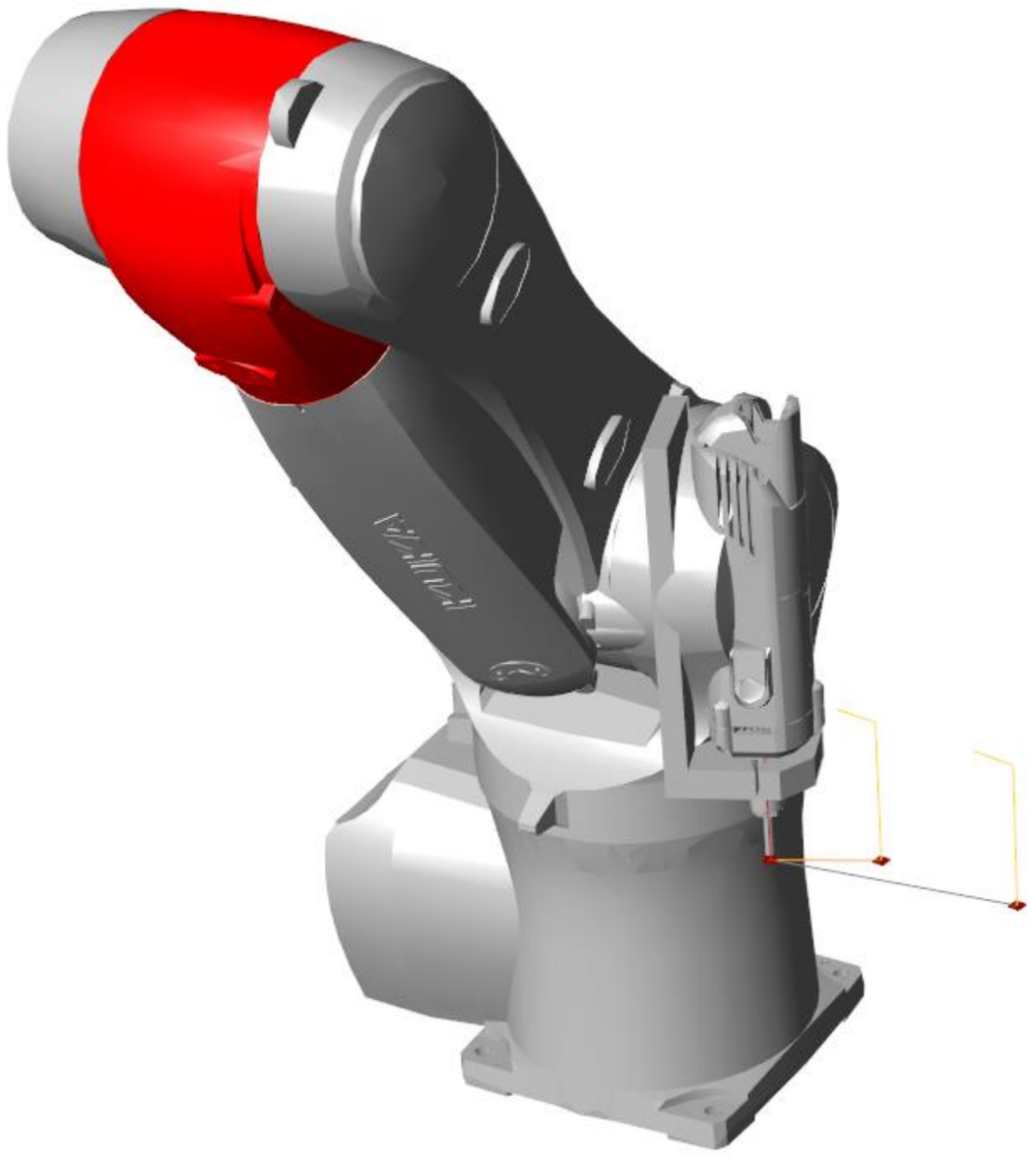
| We do not need the Kinematic settings yet, so click on “Apply Settings” and close the window.



04 | Robot Programming using KUKA|prc

| You now see for every position how the robot is going to be positioned. If any part of the robot is RED, it is outside its axis limits.





04 | Robot Programming using KUKA|prc

| For a finer simulation, you can enable “Smooth Robot Simulation” in the KUKA|prc settings.