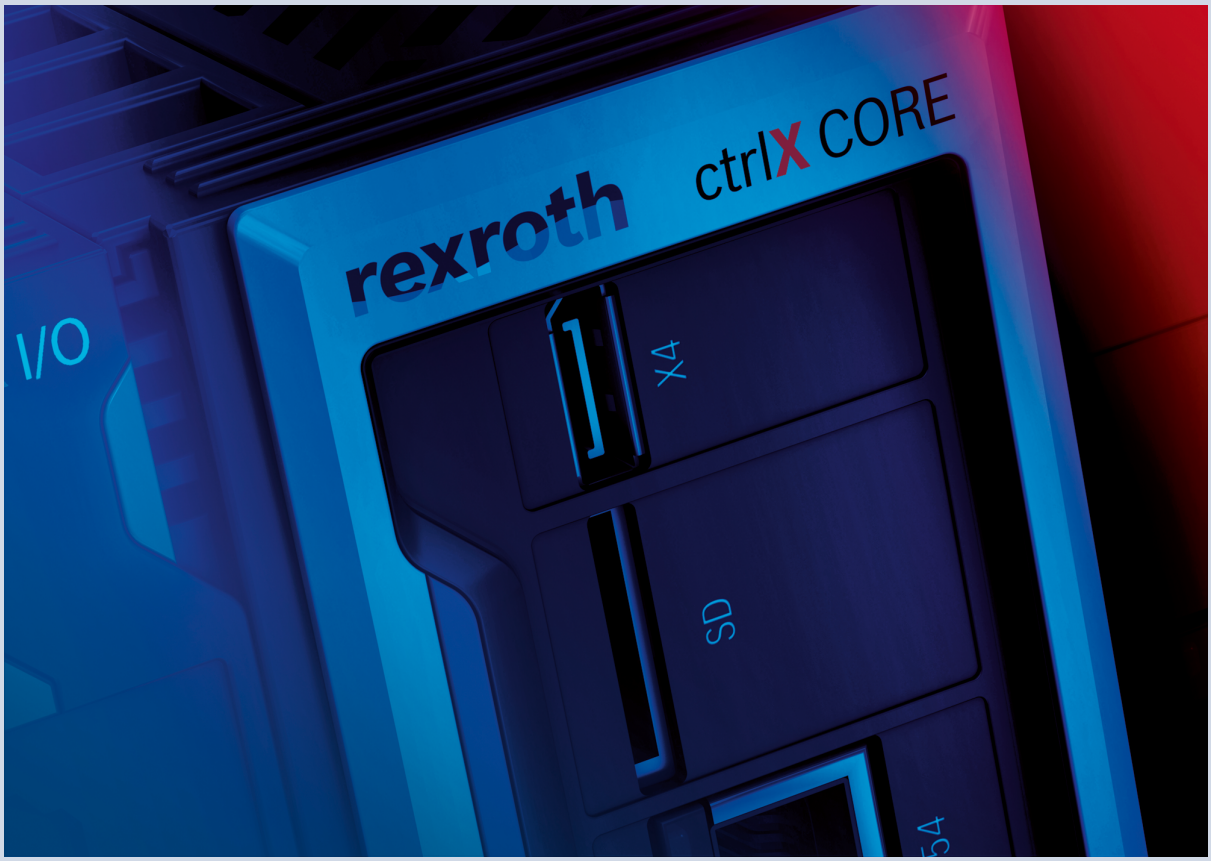
rexroth
A Bosch Company

ctrlX CORE: CODESYS SoftMotion

Insights - ctrlX AUTOMATION: Engineering

January 2022



1 Introduction

The CODESYS SoftMotion suite includes functionality to support simple point-to-point motion, camming, electronic gearing, robotic functionality and CNC.

In this note we look at the robotic functionality, focusing on how users may support their own kinematic geometries.

Our example will be a simple parallel kinematic, known from IndraMotion MLC as kinematic type 12:

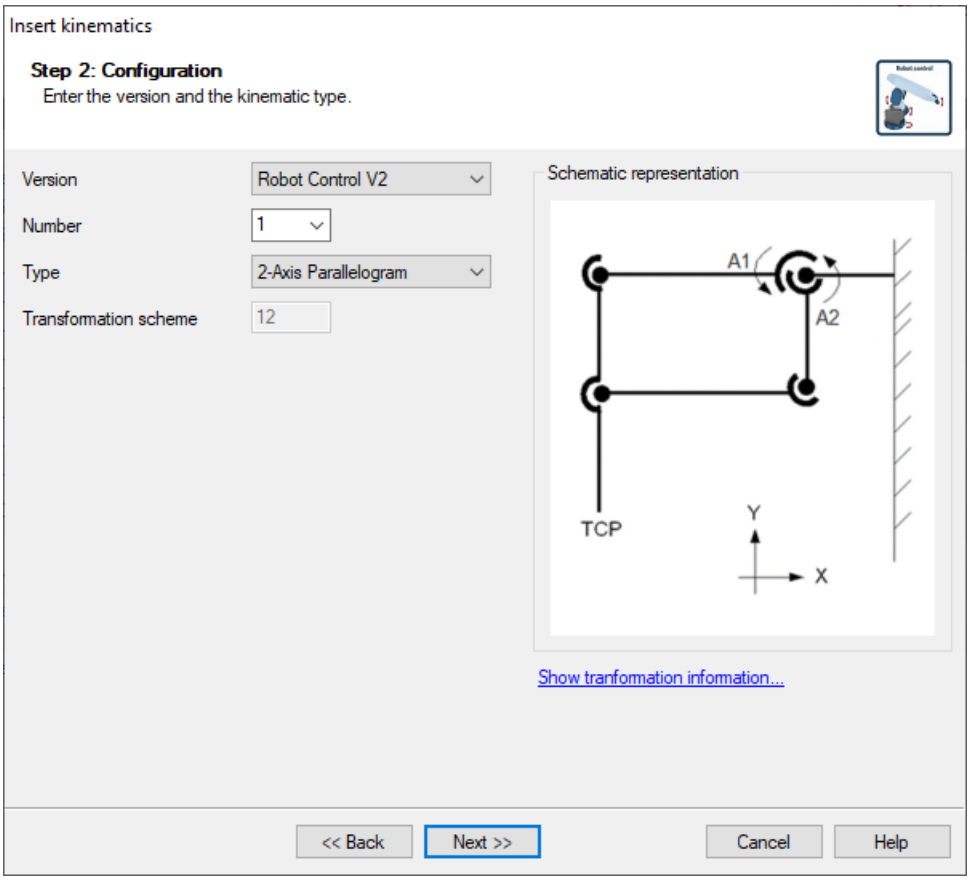


Figure 1: CODESYS SoftMotion allows users to support their own kinematic geometries. Here we take as an example a simple parallelogram kinematic from the IndraMotion MLC library.

2 SoftMotion packages

To use CODESYS SoftMotion on the ctrlX AUTOMATION platform, install the associated package using the Package Manager available in ctrlX PLC Engineering. Installation is straightforward. For details see document *Softmotion on ctrlX CORE_HowTo.pdf*.

Note that a free adaptor package provided by Bosch Rexroth, *ctrlX CODESYS SoftMotion Adaption*, is also required.

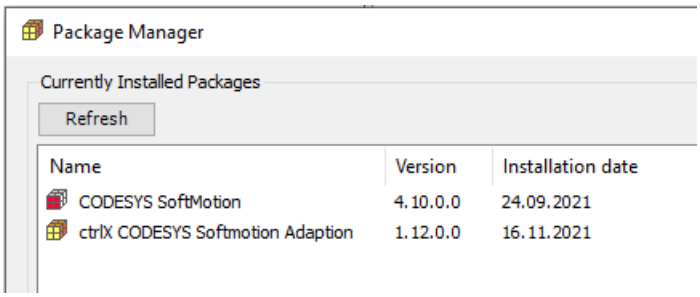


Figure 2: Use of CODESYS SoftMotion on the ctrlX AUTOMATION platform requires the two packages shown. Additional licensing requirements are described in the next section.

3 CODESYS SoftMotion

The CODESYS SoftMotion suite is divided into libraries for basic single axis motion (SM3_Basic), CNC (SM3_CNC) and robotics (SM3_Robotics). Note that SM3_Basic, in addition to simple point-to-point motion, supports camming and electronic gearing.

A fourth library, SM3_Transformation, includes interfaces that allow users to provide support for their own kinematic mechanisms. Standard mechanisms, like SCARA or two- or three-axis delta, are also supported by this library.

The CODESYS SoftMotion Basic functionality is licensed separate from the CNC and Kinematic functionalities[1]. Licensing details, as well as some additional software requirements are listed at the end of this document.

When unlicensed, packages will run in demo mode for roughly 120 minutes. Demo mode is fully featured.

[1] Note that in the use-case described here, both licenses (i.e Basic + CNC/Kinematic) are required.

3.1 SM3_Robotics

Library SM3_Robotics provides functionality for robot operation. Users will employ SMC_GroupPower to apply power to all axes included in the kinematic group, MC_GroupEnable to enable coordinated (”grouped”) motion and MC_MoveLinearAbsolute or MC_MoveLinearRelative to move the robot’s tool center point (TCP) along a linear path. Helper blocks for resetting all group axes (MC_GroupReset) and reading the active robot status (MC_GroupReadActual-Position, MC_GroupReadActualVelocity, MC_GroupReadActualAcceleration) are also provided.

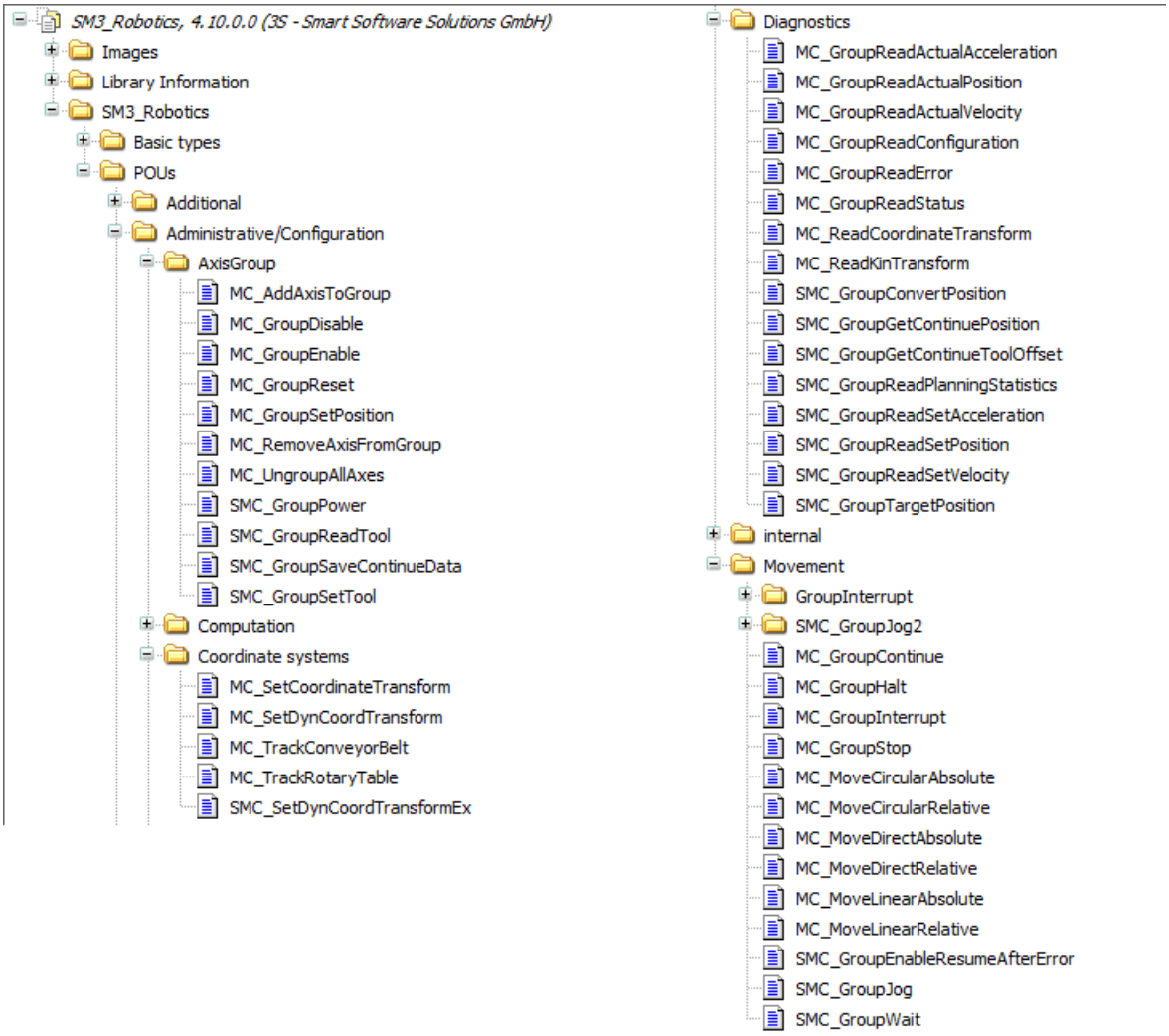


Figure 3: Library SM3_Robotics provides functionality for basic robot operation.

3.2 User-defined kinematics

If a robot geometry is not supported by the standard kinematics available in SM3_Transformation, users may create a so-called user-defined kinematic. To do this, one defines a function block that includes, at a minimum, methods defining the forward and inverse transformations of the geometry, as well as some additional methods required for system integration.

The function block we describe here implements the interfaces MC_KIN_REF_SM3 and ISMKinematicWithInfo2, both described by this library (SM3_Transformation).

Interface MC_KIN_REF_SM3 requires methods AxesToCartesian and CartesianToAxes as well as property NumAxes. Interface ISMKinematicWithInfo2 requires methods GetAxisProperties, GetKinematicsName and IsSin-

```
{attribute 'sm_kin_axes' := 'X,Y'}
{attribute 'sm_kin_libdoc' := ''}
FUNCTION_BLOCK MC_KIN_TYPE_12_XY IMPLEMENTS TRAFO.MC_KIN_REF_SM3,
                                            TRAFO.ISMKinematicWithInfo2
VAR_INPUT
    {attribute 'sm_kin_param_unit' := 'u'}
    {attribute 'sm_kin_param_range' := ']0..+inf['}
    L1: LREAL := 200;
    {attribute 'sm_kin_param_unit' := 'u'}
    {attribute 'sm_kin_param_range' := ']0..+inf['}
    L2: LREAL := 150;
    {attribute 'sm_kin_param_unit' := 'u'}
    {attribute 'sm_kin_param_range' := ']0..+inf['}
    L3: LREAL := 100;
    {attribute 'sm_kin_param_unit' := 'u'}
    {attribute 'sm_kin_param_range' := ']0..+inf['}
    L4: LREAL := 200;
    {attribute 'sm_kin_param_unit' := 'u'}
    {attribute 'sm_kin_param_range' := ']0..+inf['}
    L5: LREAL := 100;
END_VAR
VAR CONSTANT
    PI: LREAL := 3.1415926535897931;
END_VAR
```

Figure 4: To create a user-defined kinematic, begin by defining a function block that implements the required interfaces defined by library SM3_Transformation. For additional interface options, see CODESYS online help. Note that the attribute pragmas shown provide information to ctrlX PLC Engineering required for integration into the kinematic selection wizard. This topic will be discussed later.

gularity. Next we describe methods AxesToCartesian and CartesianToAxes in detail. The requirements of the additional methods are minimal and are well-described by the CODESYS online help.

3.2.1 AxesToCartesian

AxesToCartesian defines the forward kinematic transformation. The joint, or axis, positions are provided by variable a (type AXISPOS_REF). From this we calculate the TCP frame, f (type SMC_Frame). Note that we return an error in case the given joint angles do not yield a (real) solution.

The code listing is shown below. Method CircleCircle_Intersection, which is used internally to calculate the intersection points of circles, is described in the appendix.

```
/// Computes position/orientation of the TCP from axis positions.
/// Returns whether the computation was successful.
METHOD AxesToCartesian : TRAFO.SMC_Error
VAR_IN_OUT
    /// Out: The position and orientation of the TCP
    f: SMC_Frame;
    /// Out: The serialized configuration data
    cd: TRAFO.CONFIGDATA;
END_VAR
VAR_IN_OUT CONSTANT
    /// In: The position of all axes
    a: TRAFO.AXISPOS_REF;
END_VAR
VAR
    T: ML_CARTESIAN_POINT_TYPE;
    E: ML_CARTESIAN_POINT_TYPE;
    W: ML_CARTESIAN_POINT_TYPE;
END_VAR
```

```
// Calculate TCP world coordinates from given joint coordinates.

// Endpoint of linkage L1 (axis 1)
T.X := L1 * COS((a.a0 + 180.0) * PI / 180.0);
T.Y := L1 * SIN((a.a0 + 180.0) * PI / 180.0);

// Endpoint of linkage L5 (axis 2)
W.X := L5 * COS((a.a1 + 270.0) * PI / 180.0);
W.Y := L5 * SIN((a.a1 + 270.0) * PI / 180.0);

// Point E is intersection of linkages L3, L4
IF NOT CircleCircle_Intersection(T.X, T.Y, L3, W.X, W.Y, L4, -1, E.X, E.Y) THEN
    AxesToCartesian := SMC_Error.SMC_IPR_DIVISION_BY_ZERO;
    RETURN;
END_IF;

f.vT.dX := T.X + (E.X - T.X) * L2 / L3;
f.vT.dY := T.Y + (E.Y - T.Y) * L2 / L3;
f.vT.dZ := 0;
SMC_Matrix3_RotateZ(f.mR, 0);

AxesToCartesian := SMC_ERROR.SMC_NO_ERROR;
```

Figure 5: Code listing for AxesToCartesian. Values L1, L2, L3, L4 and L5 are function block inputs assigned when configuring the AxisGroup object in the kinematic selection wizard.

A geometric representation of the calculation is given in the figure below.

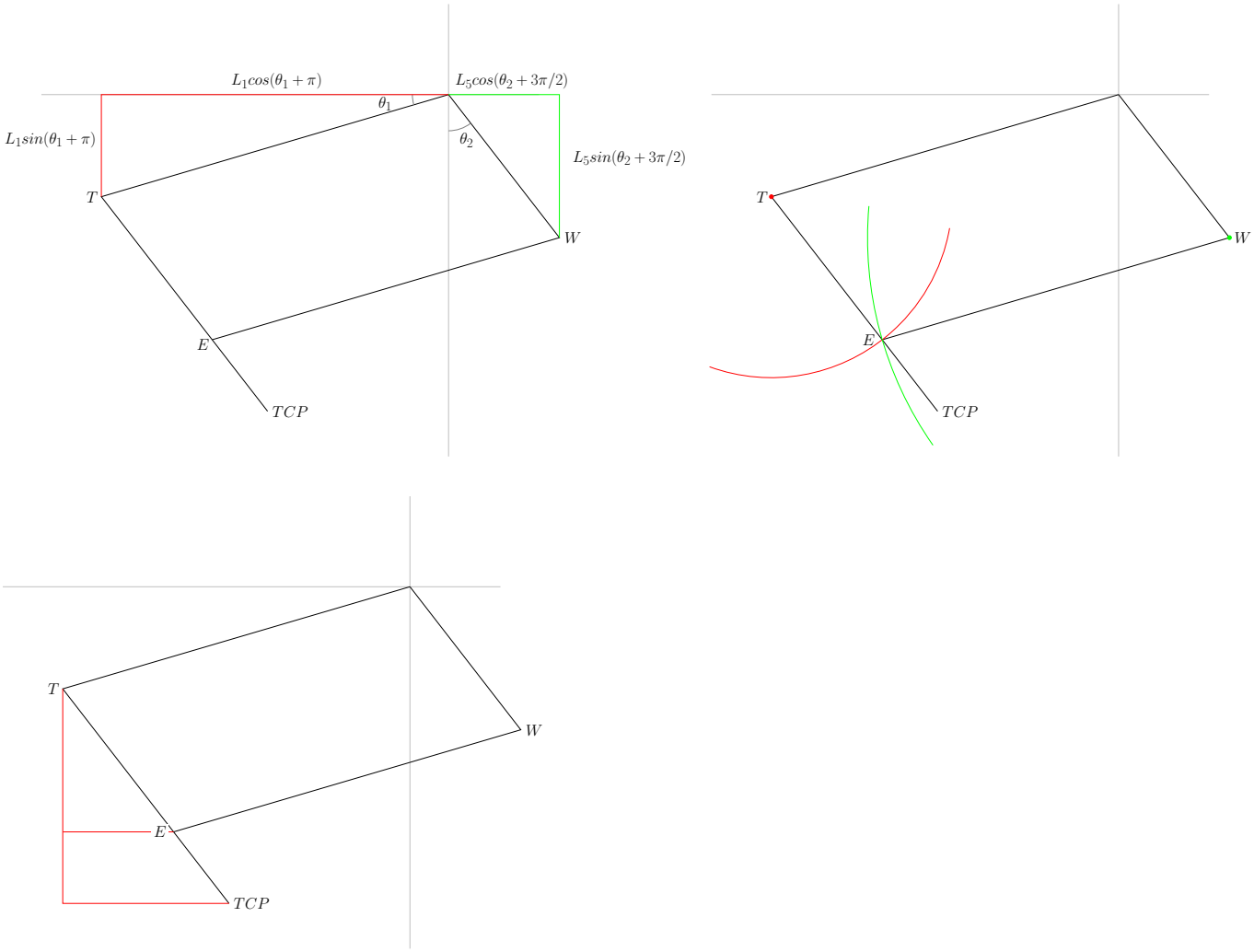


Figure 6: To calculate the TCP position from the given joint co-ordinates (θ_1, θ_2), first calculate the positions T, W using standard sine, cosine functions. Next calculate position E by finding the intersection point of two circles,centered at T and W, with radii corresponding to the given linkage lengths. Next calculate TCP by considering the similar triangles shown. Note that in case the robot origin together with points T, E, W form a parallelogram, position E may be determined simply by vector addition. We consider only the general case here.

An important point is that SoftMotion will call AxesToCartesian as required *internally*. Users are never required to call it in the application explicitly. This is also true for the other methods described by interfaces MC_KIN_REF_SM3 and ISMKinematicWithInfo2.

3.2.2 CartesianToAxes

CartesianToAxes defines the reverse or inverse kinematic transformation. In this case the TCP frame is provided by variable f. From this we calculate the corresponding joint angles. As before, note that we return an error in case the given TCP does not yield a (real) solution.

```
/// Computes the axis positions from position/orientation of the TCP.
/// Returns whether the computation was successful.
METHOD CartesianToAxes : TRAFO.SMC_Error
VAR_IN_OUT
    /// Out: The position of all axes
    a: TRAFO.AXISPOS_REF;
END_VAR
VAR_IN_OUT CONSTANT
    /// In: The position and orientation of the TCP
    f: SMC_Frame;
    aRef: TRAFO.AXISPOS_REF;
    cd: TRAFO.CONFIGDATA;
END_VAR
VAR
```

```
// Calculate TCP world coordinates from given joint coordinates.

IF NOT CircleCircle_Intersection(0, 0, L1, f.vT.dX, f.vT.dY, L2, -1, T.X, T.Y) THEN
    CartesianToAxes := SMC_Error.SMC_IPR_DIVISION_BY_ZERO;
    RETURN;
END_IF

a.a0 := ATAN_2(T.X, T.Y) - 180.0;
a.a0 := Modulo(0, 360.0, a.a0, 0); // Axis 1 command position range: [-180, 180]

E.X := T.X + (f.vT.dX - T.X) * L3 / L2;     // Calculate E, the intersection of linkages L3, L4
E.Y := T.Y + (f.vT.dY - T.Y) * L3 / L2;

// Calculate W, the endpoint of linkage L5 (axis 2)
IF NOT CircleCircle_Intersection(0, 0, L5, E.X, E.Y, L4, 1, W.X, W.Y) THEN
    CartesianToAxes := SMC_Error.SMC_IPR_DIVISION_BY_ZERO;
    RETURN;
END_IF

a.a1 := ATAN_2(W.X, W.Y) - 270.0;
a.a1 := Modulo(0, 360.0, a.a1, 0.0); // Axis 2 command position range: [-180, 180]

CartesianToAxes := SMC_ERROR.SMC_NO_ERROR;
```

Figure 7: Code listing for CartesianToAxes. Values L1, L2, L3, L4 and L5 are function block inputs assigned when configuring the AxisGroup object in the kinematic selection wizard.

A geometric representation of the calculation is given in the figure below.

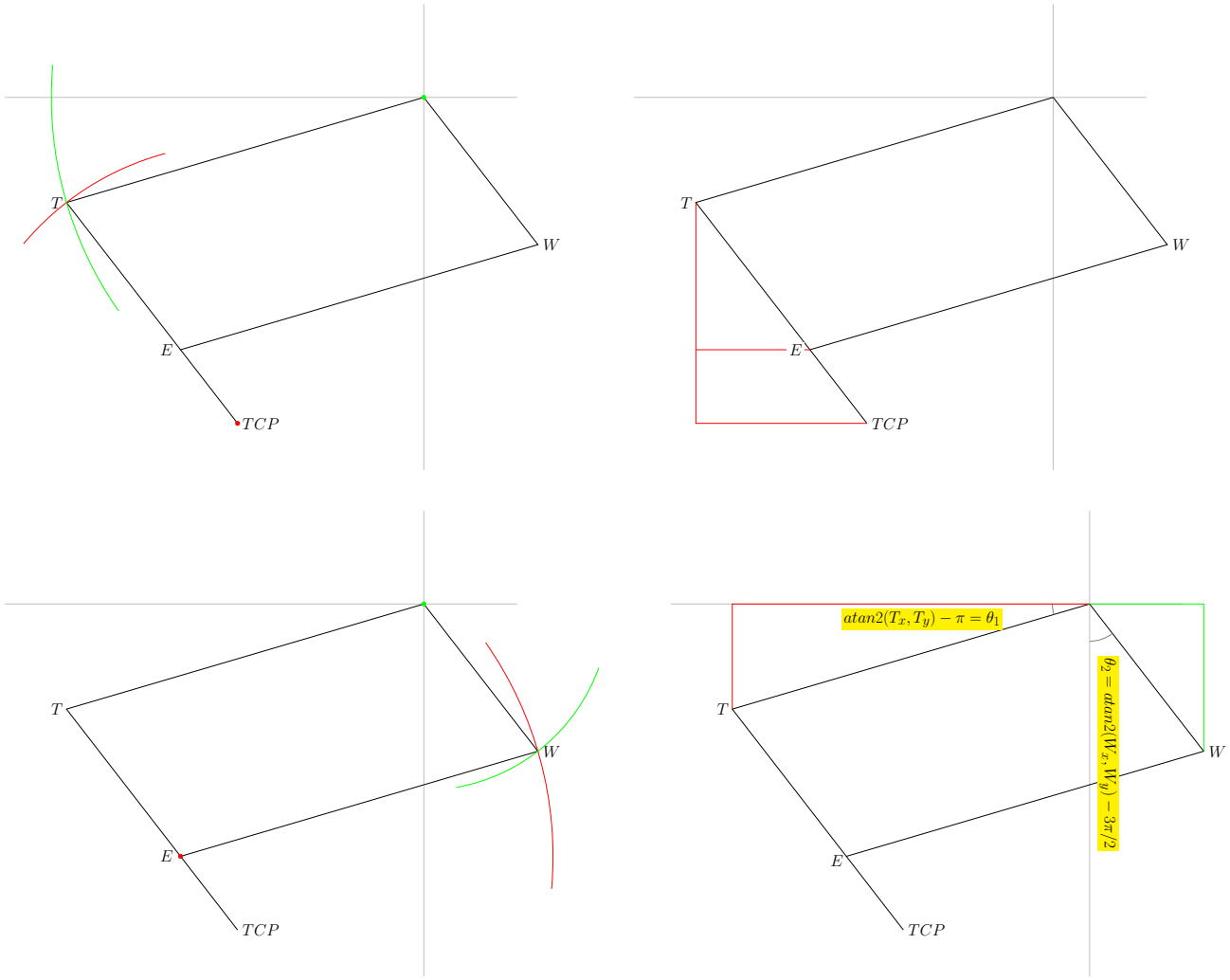


Figure 8: To calculate the joint positions from the given TCP frame (X, Y), first calculate position T by finding the intersection point of two circles, centered at TCP and the robot origin, with radii corresponding to the given linkage lengths. Calculate position E by considering the similar triangles shown. Next calculate position W by finding the intersection point of circles centered at E and the robot origin as shown. Once positions T and W are known, the joint positions may be calculated using the arctangent in the obvious way. Note that in case the robot origin together with points T, E, W form a parallelogram, position W may be determined simply by vector addition. We consider only the general case here.

3.3 reStructuredText

reStructuredText is a simple markup syntax that may be used to format the documentation displayed by ctrlX PLC Engineering's Library Manager as well as the kinematic selection wizard.

For example, with the following markup we produce the document shown below.

```
(*
==================================
Short description
==================================

*MC_KIN_TYPE_12_XY* implements the forward and reverse transformations for the
parallel mechanism shown below.  The mechanism corresponds to IndraMotion
kinematic TYPE 12.

==================================
Functional description
==================================

In the posture shown, links L1, L5 are in their reference (null) position:
JC = (0, 0).

.. image:: @(kin12_xy.png)
:width: 300

IndraMotion kinematic TYPE 12
*)
```

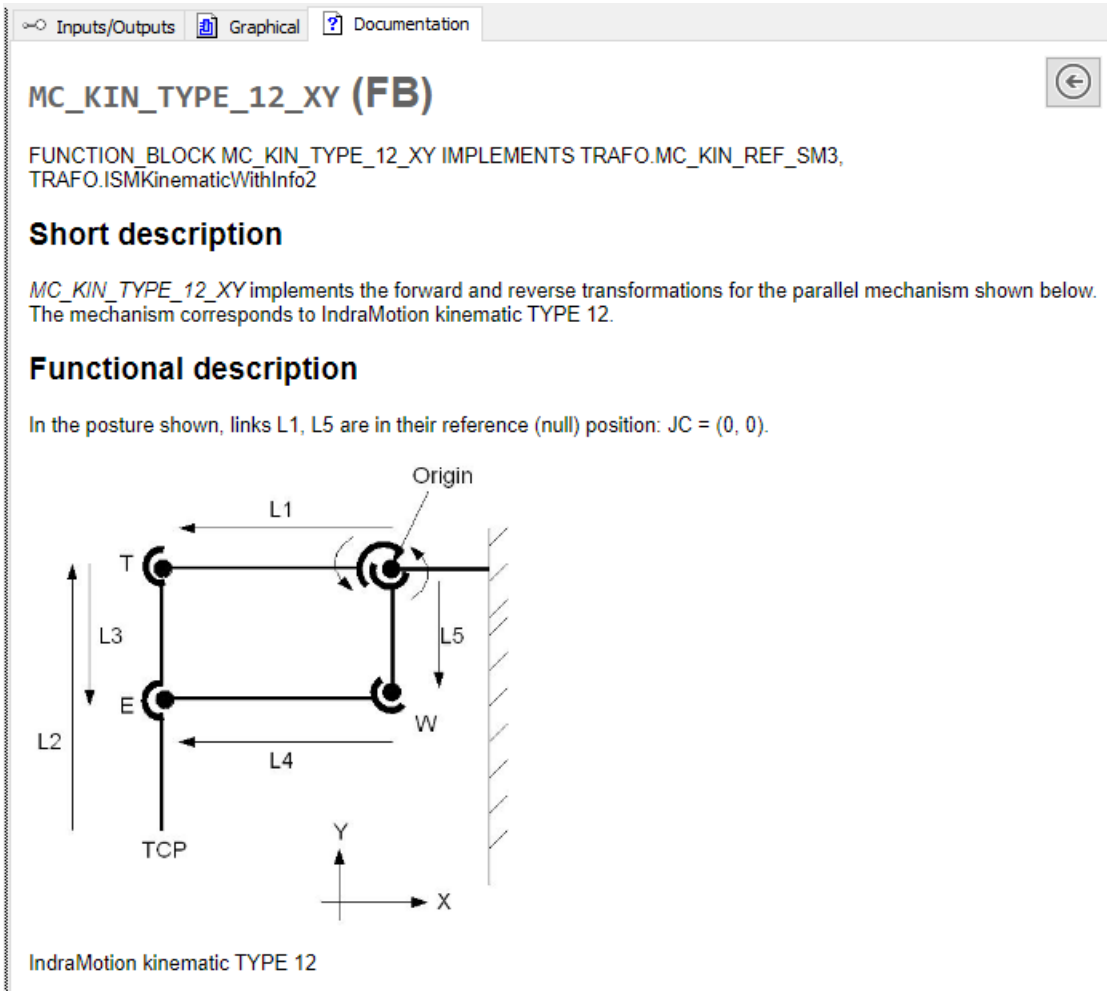


Figure 9: Use reStructuredText to create the internal library documentation. Note that extension wkhtmltox.dll is required to build the documentation and must be configured manually in the ctrlX PLC Engineering installation directory. See CODESYS online help for more information.

4 Visualization

In the next sections we briefly describe software used to create a browser-based visualization tool for our robot. This is provided for information only. The topics described in this section are not required when using SoftMotion.

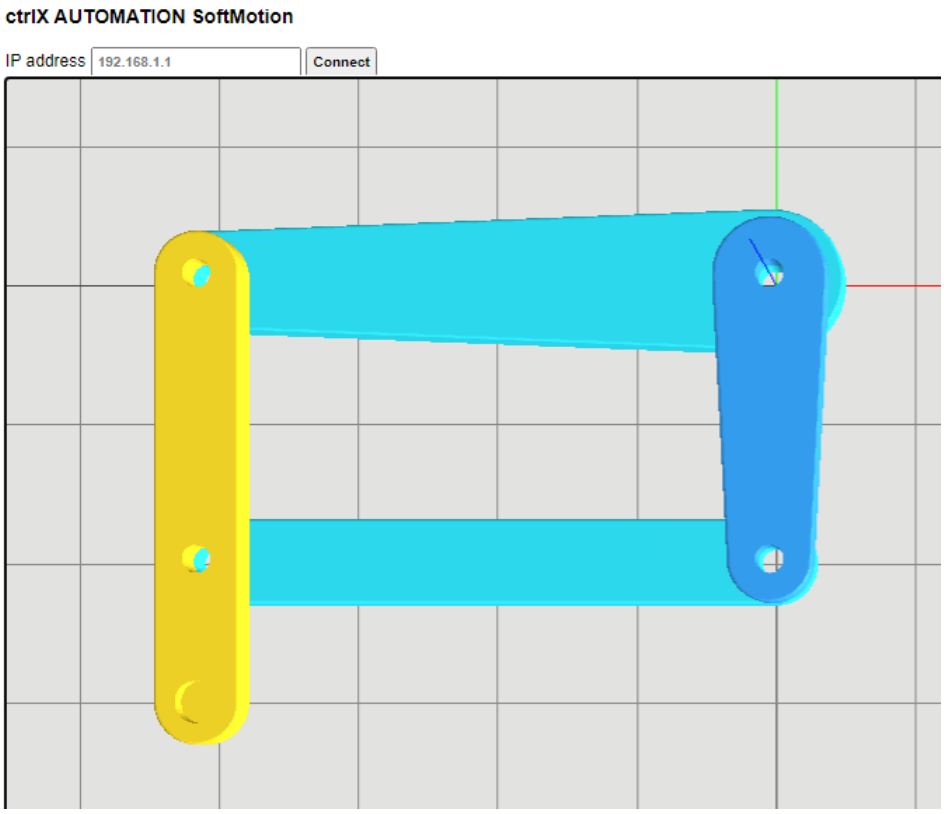


Figure 10: Real-time visualization tool for our user-defined kinematic created using Blender and animated for the browser with JavaScript library three.js.

4.1 Blender

Blender is open-source software used to create and render 3D computer models. It claims to support the ”entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation[2].”

Blender’s use here is very limited: We create models of each of the linkages in our kinematic assembly and export this assembly as a file of type .gltf.

[2]See https://www.blender.org/about/

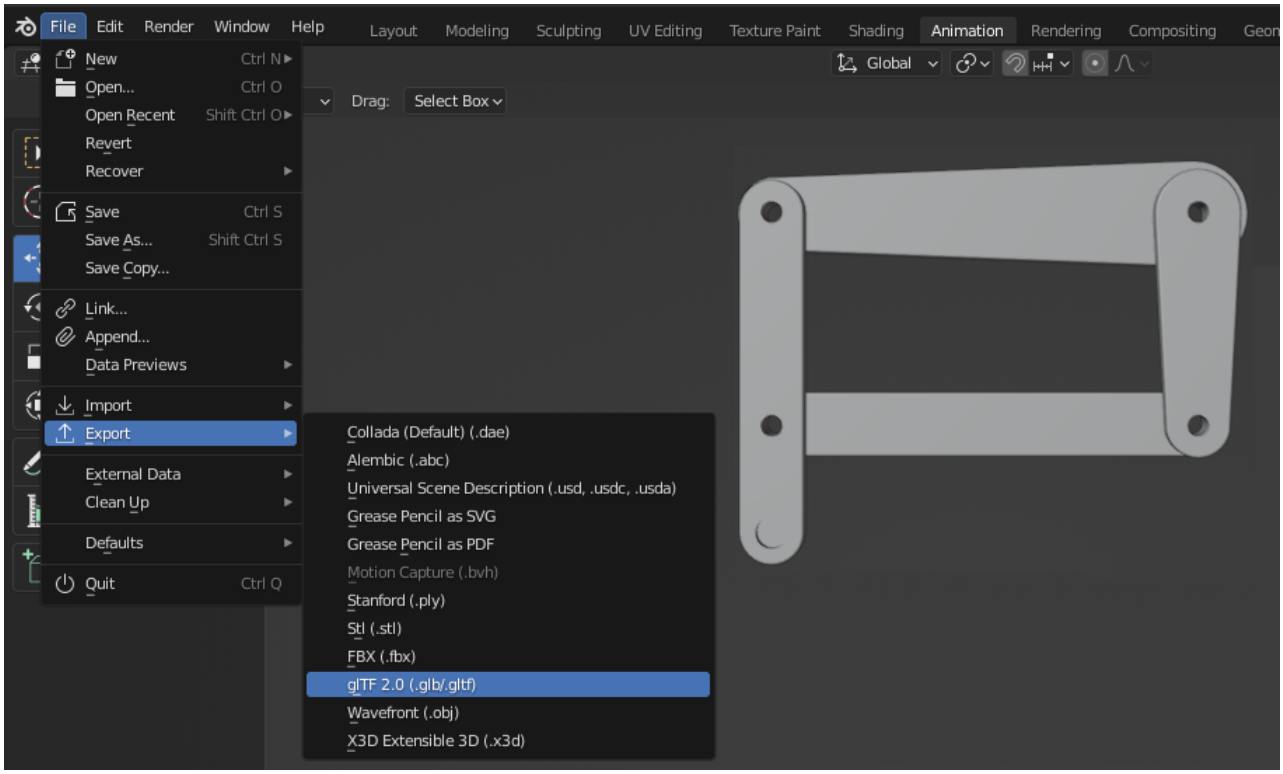


Figure 11: Blender allows users to create 3D models and export these as a set of individual linkages in the gltf format. Note that each linkage has a canonical pivot point and we move each linkage's canonical "origin" to this point.

4.2 Three.js

Three.js is a JavaScript-based WebGL[3] framework, allowing users to create high-quality 3D graphical animations in a web browser.

Key to our implementation here is the ability to import a graphics model of type .gltf using a loader supplied by the three.js library:

```
var loader = new THREE.GLTFLoader();

loader.load('kin12_V0_3.gltf', function (gltf) {
L_5 = gltf.scene.children[3];
scene.add(L_5);

L_2 = gltf.scene.children[2];
scene.add(L_2);

L_1 = gltf.scene.children[1];
scene.add(L_1);

L_4 = gltf.scene.children[0];
scene.add(L_4);
});
```

[3]WebGL, or Web Graphics Library, is an open-source JavaScript API for rendering 2D and 3D graphics within a web browser.

Additionally, we note that we may position each of the linkages (objects L_1, L_2, L_3, L_4 above, imported as separate ”children”) within our scene directly using their position and rotation fields. For example:

```
L_4.position.x = 0.030;   //base position in meters
L_4.position.z = 0.020;  //base position in meters
L_4.rotation.y = 0.1; //angular position about base in radians
```

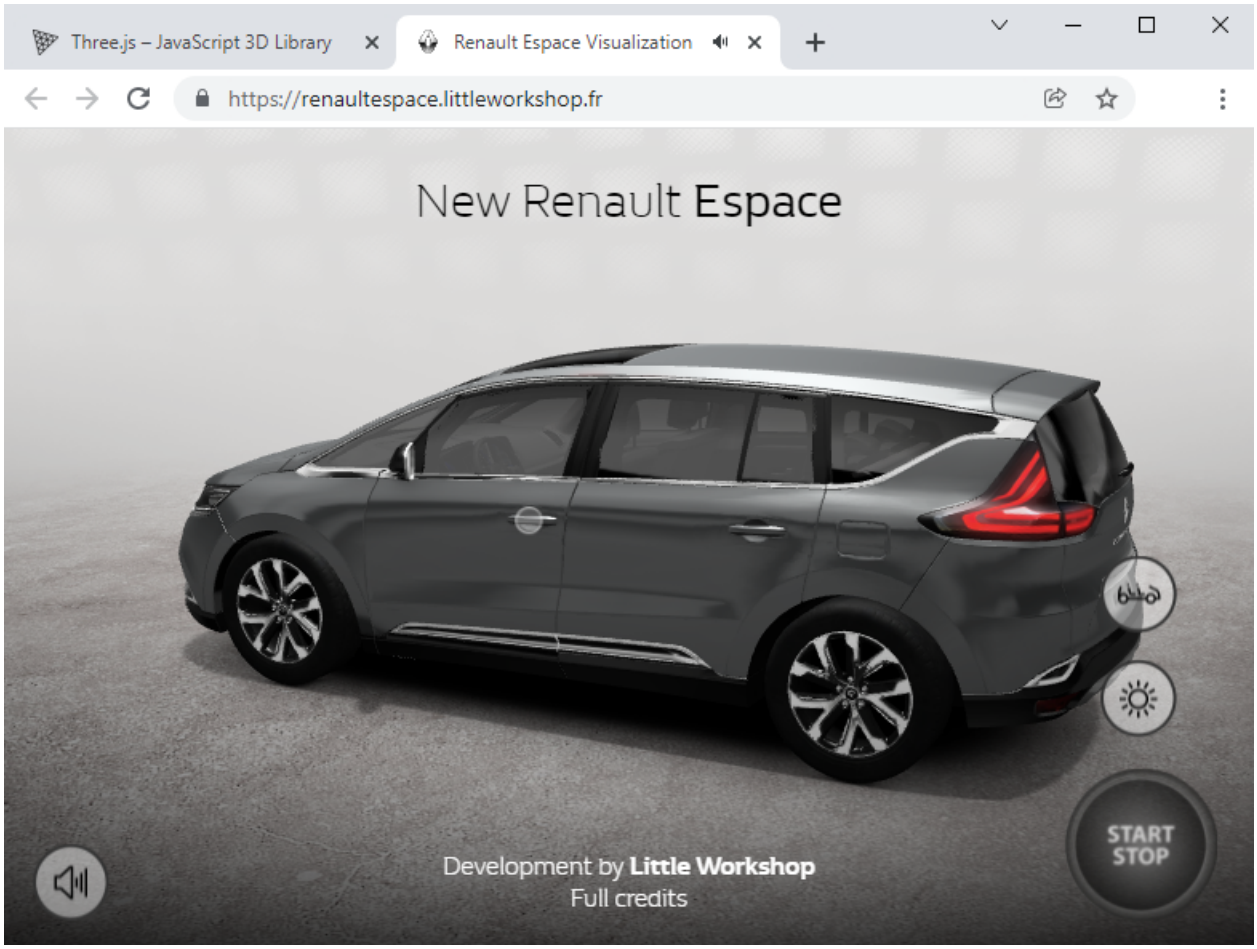


Figure 12: Three.js provides tools for the rendering and animation of sophisticated 3D models. See https://threejs.org/ for more examples.

4.3 Python

Our visualization tool requires that the real-time robot position be supplied using a WebSocket. The main Python routine is shown below. The complete source code, including the configuration files required to build for ctrlX CORE, is available at https://github.com/bostroemc/webconnector-lite.

```
def main():
import sys
import logging
import asyncio
import websockets
import requests
import json
import aiohttp
import async_timeout

async def getToken(session, url, user):
  with async_timeout.timeout(10):
    async with session.post(url, json=user, ssl=False) as response:
      temp = json.loads(await response.text())
      return str(temp["access_token"])


async def fetchData(session, url, headers):
  await asyncio.sleep(0.05)
  with async_timeout.timeout(10):
    async with session.get(url, headers=headers, ssl=False) as response:
      return await response.text()


async def main(websocket, path):
  async with aiohttp.ClientSession() as session:
    payload = {"name": "boschrexroth", "password": "boschrexroth"}
    token = await getToken(session,\
    "https://127.0.0.1/identity-manager/api/v1.0/auth/token",\
    payload)
    headers = {'Authorization': 'Bearer ' + token}

    while True:
      value = await fetchData(session,\
      "https://127.0.0.1/automation/api/v1/plc/app/Application/sym/PLC_PRG/dTest",\
      headers)
      await websocket.send(value)


start_server = websockets.serve(main, "0.0.0.0", 8765)

asyncio.get_event_loop().run_until_complete(start_server)
asyncio.get_event_loop().run_forever()


if __name__ == '__main__':
main()
```

5 Appendix

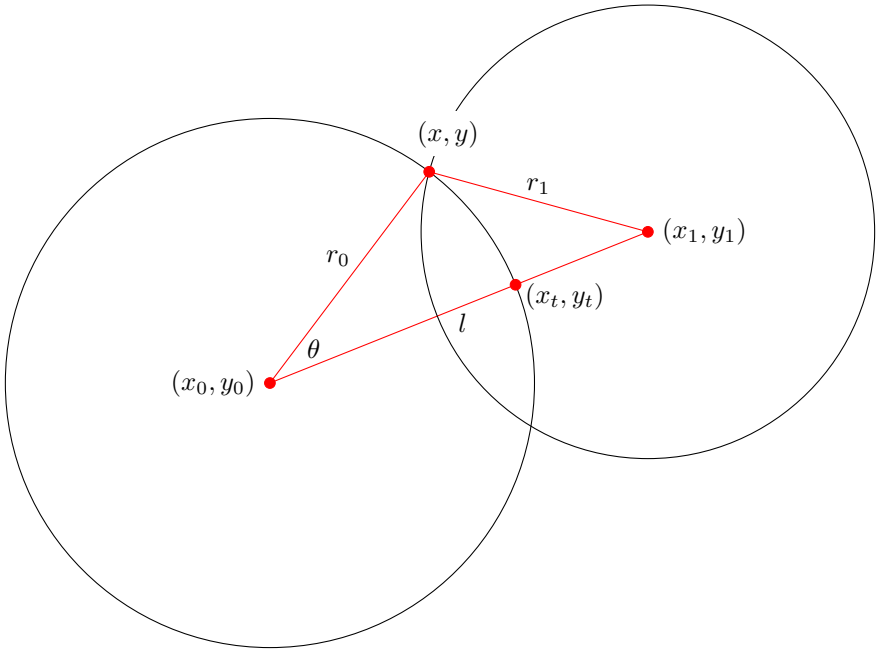
5.1 Intersection of circles

The intersection points of two circles are given by the solution to the following system of equations:

$$\begin{cases} (x - x_0)^2 + (y - y_0)^2 = r_0^2 \\ (x - x_1)^2 + (y - y_1)^2 = r_1^2 \end{cases} \tag{1}$$

After expanding the two equations and simplifying, it is a simple matter to show the the system may be reduced to a single quadratic equation and solved using the quadratic formula. This approach is straightforward, but additional calculation is required to distinguish between the ”left-” and ”right-hand” solutions.

Instead, we use a more geometric approach and leverage the law of cosines. The main point here is that we produce the left- or right-hand solution, whichever is required for the given robot posture, in a natural, ”baked-in” way.



First we calculate the distance between the circle centers, l.

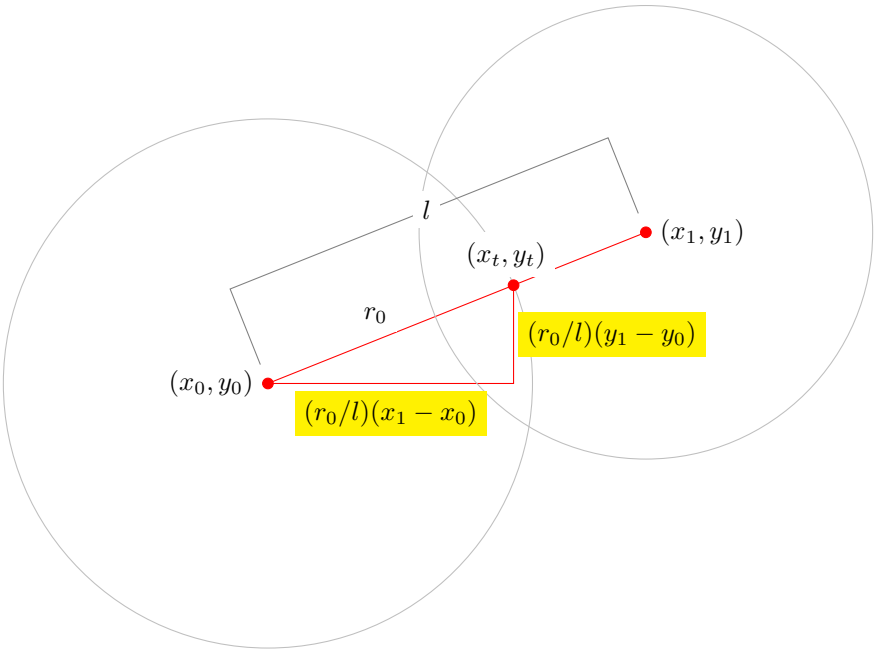
$$l = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \tag{2}$$

We assume the the circles intersect and draw the triangle shown in the figure above. By the law of cosines, $cos(\theta)$ may be given in terms of l and the two radii.

$$cos(\theta) = \frac{l^2 + r_0^2 - r_1^2}{2lr_0} \tag{3}$$

Next we calculate the co-ordinates of the point at the intersection of the first circle with the line segment connecting the two centers, (x_t, y_t). This follows directly from considering the two similar right triangles with hypotenuses r_0 and l respectively.

$$(x_t - x_0, y_t - x_0) = (\frac{r_0}{l}(x_1 - x_0), \frac{r_0}{l}(y_1 - y_0)) \tag{4}$$



Next we rotate (x_t, y_t) about center of the first circle onto the left-hand solution using the rotation matrix $\begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$.

$$\begin{bmatrix} (x - x_0) \\ (y - x_0) \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} (x_t - x_0) \\ (y_t - x_0) \end{bmatrix} \tag{5}$$

If the right-hand solution is desired, we instead rotate by $-\theta$: $\begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix}$.

An important point here is that we calculate $sin(\theta)$ using the Pythagorean identity; we do not need to calculate the value of the angle itself.

$$sin(\theta) = \sqrt{1 - cos^2(\theta)} \tag{6}$$

Finally, we note without proof that the two circles intersect in exactly two points if and only if the following relation holds:

$$0 < \frac{l^2 + r_0^2 - r_1^2}{2lr_0} < 1 \tag{7}$$

5.2 Hardware

Control	ctrlX CORE, COREX-C-X3-11-ANNN-21.01-01RS-NN-NN, RM21.07

5.3 Software

Commissioning	ctrlX WORKS 1.10.6 ctrlX PLC 1.10.6 (CODESYS 3.5.17.2)
CODESYS packages	CODESYS SoftMotion, 4.10.0.0 ctrlX CODESYS SoftMotion Adaption, 1.10.0.3 (pre-release)
ctrlX CORE apps	PLC, 1.10.1 EtherCAT Master, 1.10.0
Required licenses	CODESYS SoftMotion R911311672 SWL-W-XC*-COSY*SMOT******-NNNN CODESYS SoftMotion CNC R911311674 SWL-W-XC*-COSY*SMOT*CNC**-NNNN