Practical Exercise 4: Calibration

Objective:

In this practical exercise you should learn how you can calculate the transformations between different coordinate frames by measuring the points of each reference frame. This can be used to calibrate coordinate systems.
For this purpose you implement the upcoming tasks in Python (here we provide a source code framework) or C++ (here, we **do not provide** a source code framework) and evaluate your results by visual inspection using CoppeliaSim.

Task 1:

In the scene *scan_robot.ttt* you see a redundant robot with 7 dof and 3 conference tables with reference frames. You can move the robots tip by activating the object *redundantRob_manip-Sphere*, starting the simulation and using the *Object/item shift button* or the *Object/item rotate button.*

The robot shall measure the coordinates of three points for each reference systems in the order origin, point in x direction and point in y direction.

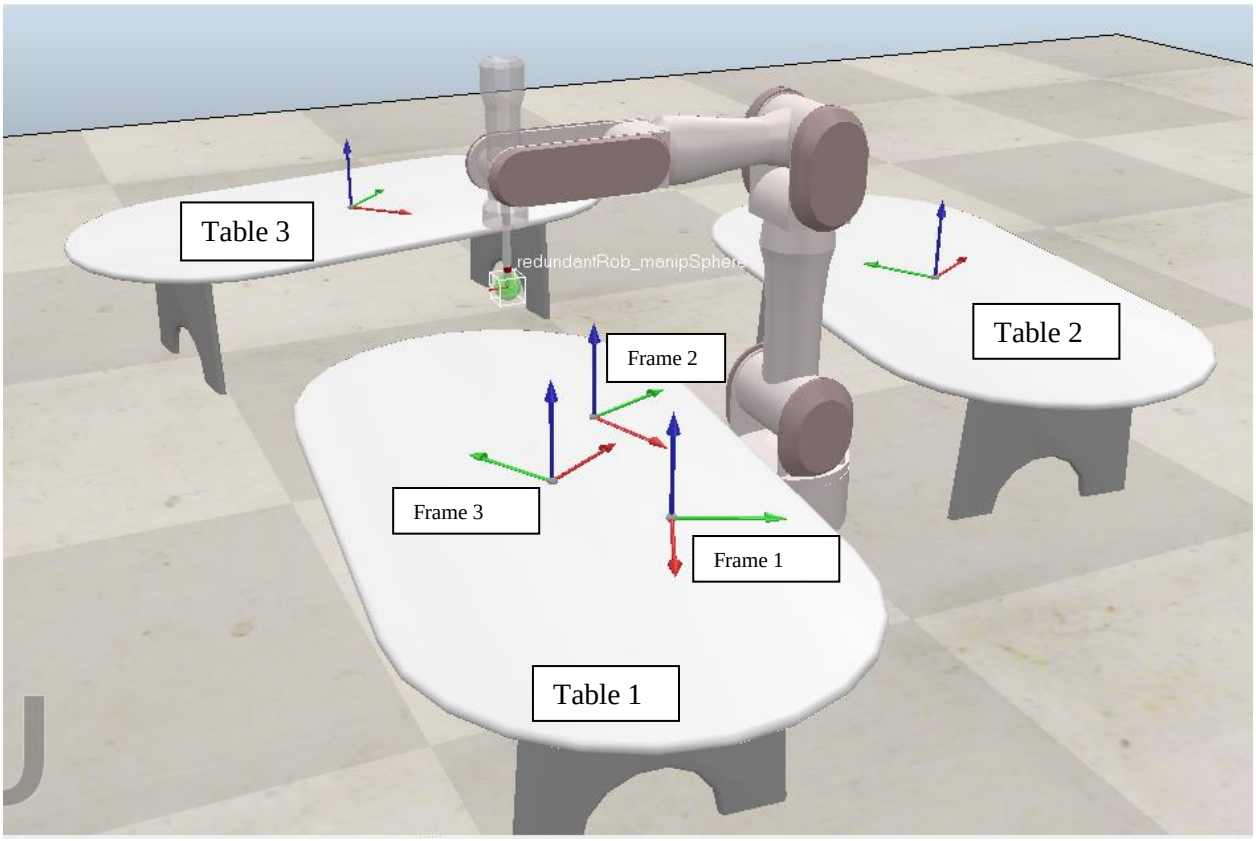

Figure 1: Sample scene with a redundant robot and 3 conference tables with reference frames

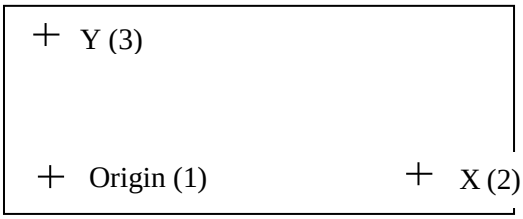

Figure 2: Measuring three points of a reference frame in that order

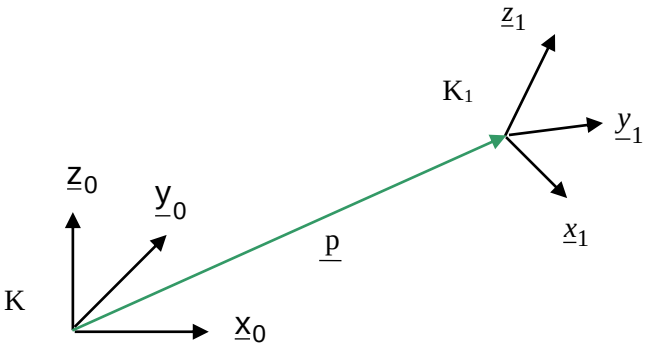

Figure 3: Transformation from reference system K to reference system K_1.

The file P4_Task1.py uses scene file *scan_robot.ttt* and can be used to save the measured points. For each reference frame you save three points. The program automatically saves a set of 3 points in one file called *framesn.txt*, where n is the number of the frame starting from 1. There is no programming necessary for this task.

You steer the robot's tip to each point and press the s key for saving the point (see source code). With the q key you close the program. It makes sense to use the orthographic scene view from the top.

So each file has 3 rows:
x_1, y_1, z_1: O: Origin of the coordinate system
x_2, y_2, z_2: X: is a point on the x-axis (red) – you can choose the tip of the red arrow
x_3, y_3, z_3: Y: is a point on the y-axis (green) – you can choose the tip of the green arrow

Task 2

Write a separate python file to calculate the following transformations of table 1:

- T_{12}: frame1 to frame2
- T_{23}: frame2 to frame3
- T_{31}: frame3 to frame1
- and the product $T = T_{12} * T_{23} * T_{31}$

Proof of Correctness:

Compare your results, with information given in the scene *scan_robot.ttt* by looking at the parent frame transformations of the reference frame 2 (this describes the transformation T_{12}) and reference frame 3 (this describes the transformation T_{23})

Hints:

- The x and y directions of your measuring are not necessarily orthogonal due to inaccuracies in the measurement
- The distance of the points X and Y to the origin O do not necessarily have unit length. They need to be normalized (Length = 1).
- The x and y directions of your measuring are not necessarily are orthogonal due to inaccuracies in the measurement, they need to orthogonalized.
- Use the following orthonormalisation method to orthogonalize the first two direction vectors x and y.

Orthonormalisation method from the lecture:

$$x = \frac{X - Origin}{|X - Origin|}, b_2 = Y - Origin - \frac{(Y - Origin)^T (X - Origin)}{(X - Origin)^T (X - Origin)}(X - Origin), y = \frac{b_2}{|b_2|}$$

$$z = x \times y$$

- Use the cross product to calculate the direction vector in z-direction
- Use appropriate numpy method for your coding

How can you prove your results? What is the expected result for T?

Task 2.1

The measurement space of the robot is restricted by its work space, which depends on the kinematic structure of the robot. It can be enlarged by “leap frogging”: measuring reference 1 from table 1 and the reference frame of table 2 from one robot base position. Then changing the position of the robot so, that the robot can measure the reference frame of table 2 and the reference frame of table 3. In a third position of the robot base measures reference frame of table 3 and reference frame 1 of table 1. Make the same calculations as in task 2. Again, what is the expected result for T?

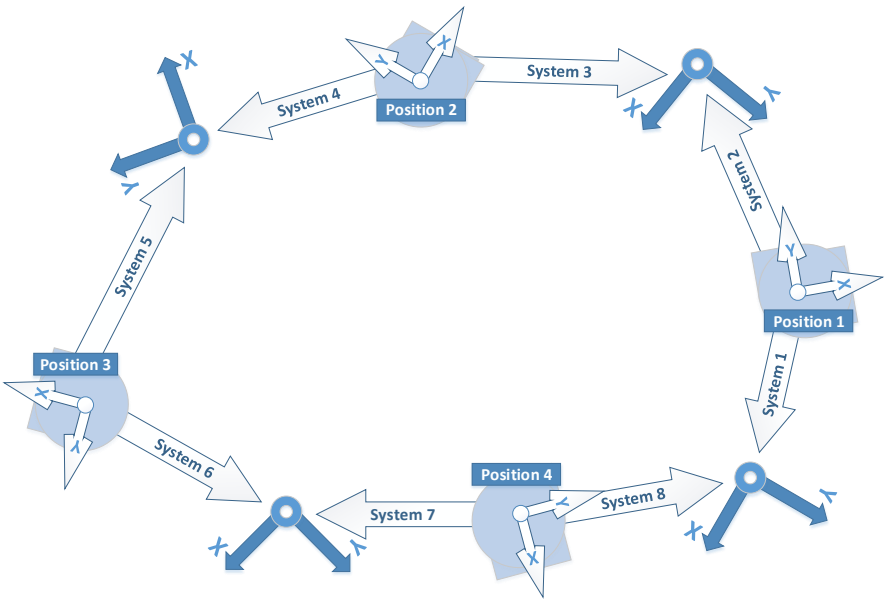

Figure 4: “Leap frogging” between reference systems by measuring from 4 different positions

Task 3 (optional)

A frequent task during the commissioning of a robot is the measurement of the tool and the resulting calculation of the TCP reference system. For this purpose, the user defines a fixed point P_t in the tool coordinate system and a reference point P_{pivot} in the world coordinate system. The robot is moved so, that the tip of the robot is changed in its orientation but not in its position (see figure 5).

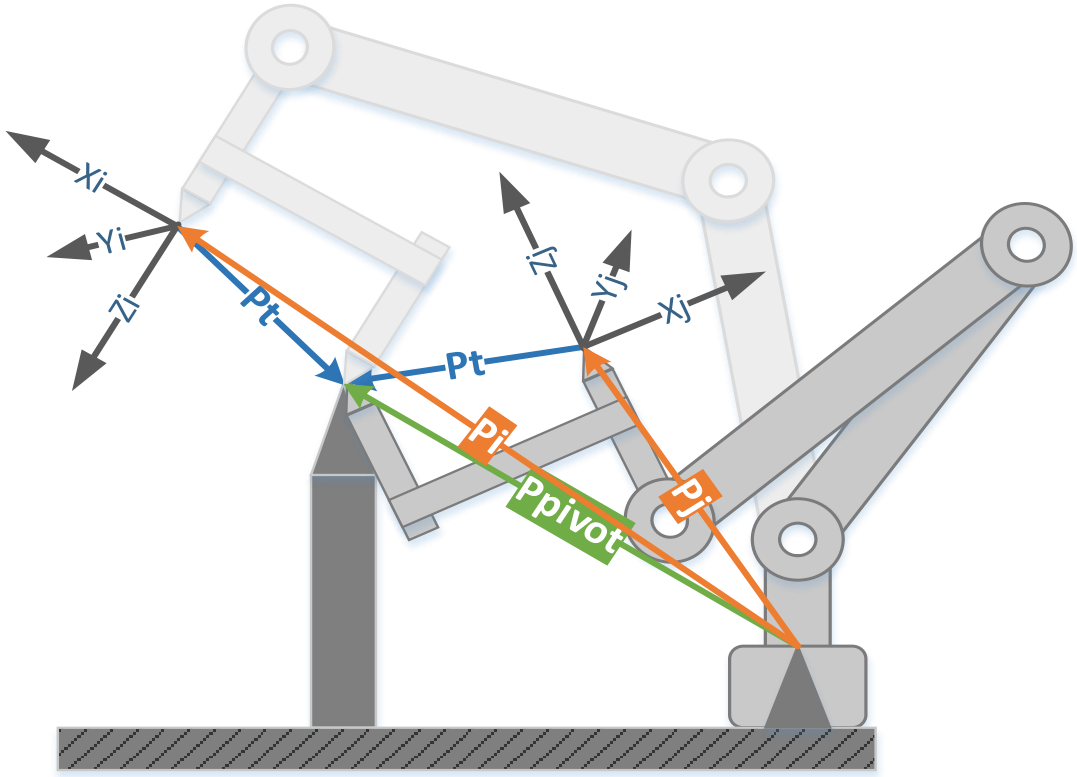

Figure 5: Measuring the robot’s tip from different poses

Since P_{pivot} does not change during the entire measurement and P_t is also invariant with respect to the coordinate systems R_i and R_j, the following relationships can be established:

$$P_i + R_i P_t - P_{pivot} \cong 0, i = 1..n$$

$$\rightarrow \underbrace{\begin{pmatrix} R_1 & -I \\ \vdots & \vdots \\ R_n & -I \end{pmatrix}}_{\substack{coefficient\ matrix \\ A}} \underbrace{\begin{pmatrix} P_t \\ P_{pivot} \end{pmatrix}}_{\substack{Parameter \\ vector \\ x}} \cong \underbrace{\begin{pmatrix} -P_1 \\ \vdots \\ -P_n \end{pmatrix}}_{\substack{observations \\ b}}$$

In general this will be an overdetermined system with less variables than observations. This system can be formulated as a least squares problem to minimize the errors in the observations

$$Ax = b + e$$

In a least squares problem the parameter vector x is determined in such, that the sum of all errors squared is minimized, e.g. $e^T e \rightarrow min$.

$$x = (A^T A)^{-1} A^T b$$

In your task you will simulate a tool calibration with CoppeliaSim and you can therefore prove your results with transformations displayed in this simulation system.

Take 6 different poses with the same orientation of the robot's tip. These poses are saved in a file. For this purpose you can use the same python file (change the task variable to 2).

Write a separate python script to calculate P_{pivot} and P_t and compare the results with CoppeliaSim (they should be roughly the same).

Proof of correctness: If you use the provided file systems.txt, then you should get the following solution:
Translation: [1.23650818e-04 -2.12293968e-05 -5.64324476e-02]
Pivot Point: [0.32348329 -0.24998911 0.31394935]

Preparation:

- Understand the orthonormalisation method
- Make yourself familiar with the given code example
- Make yourself familiar with the numpy functions to use
- Have a scaffold of your program ready

Report:

The report should contain

- the task description itself (you can reuse this document, at the top your name(s) must be mentioned)
- a comprehensive description of the implemented algorithm
- Sketches and explaining figures
- source code with result output
- you deliver one zipped file. It contains the report as PDF, the .ttt scene files and the solution python files including all additional project files necessary, to run the program. Your program must start out of the box.
- You can provide one solution for one group consisting of max. 2 students.

Naming Conventions: as in exercise 2