Example Documentation

EASY-ROB™
Kernel V8.3

November 2021

Version 4.1

Subject to change or improve without prior notice
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduction</strong></td>
<td>5</td>
</tr>
<tr>
<td>Layout Diagram of ERK_CAPI</td>
<td>7</td>
</tr>
<tr>
<td>class ERK_CAPI</td>
<td>7</td>
</tr>
<tr>
<td>class ERK_CAPI_ADMIN</td>
<td>7</td>
</tr>
<tr>
<td>class ERK_CAPI_DEVICES</td>
<td>8</td>
</tr>
<tr>
<td>class ERK_CAPI_ROB</td>
<td>8</td>
</tr>
<tr>
<td>class ERK_CAPI_MOP</td>
<td>8</td>
</tr>
<tr>
<td>class ERK_CAPI_TOOLPATH</td>
<td>8</td>
</tr>
<tr>
<td>class ERK_CAPI_SIM</td>
<td>9</td>
</tr>
<tr>
<td>class ERK_CAPI_AUTOPATH</td>
<td>9</td>
</tr>
<tr>
<td>class ERK_CAPI_GEO</td>
<td>9</td>
</tr>
<tr>
<td>class ERK_CAPI_SYS</td>
<td>9</td>
</tr>
<tr>
<td>Output of the example &quot;irb1400H.3&quot;</td>
<td>18</td>
</tr>
<tr>
<td>Getting Started</td>
<td>11</td>
</tr>
<tr>
<td>Important Note for Implementation</td>
<td>13</td>
</tr>
<tr>
<td>Example: Loading Geometries and importing IGP files</td>
<td>14</td>
</tr>
<tr>
<td>Example: Collision Detection</td>
<td>14</td>
</tr>
<tr>
<td>Example: Inverse Kinematics, loading a robot</td>
<td>15</td>
</tr>
<tr>
<td>Example: Trajectory Planner, interpolation PTP, LIN and CIRC</td>
<td>15</td>
</tr>
<tr>
<td>Example: Trajectory Planner with Conveyor Tracking</td>
<td>16</td>
</tr>
<tr>
<td>Example: Trajectory Planner with synchronized Positioner</td>
<td>16</td>
</tr>
<tr>
<td>Example: Trajectory Planner with Robot on a Track</td>
<td>17</td>
</tr>
<tr>
<td>Import of IGP Geometry files</td>
<td>18</td>
</tr>
<tr>
<td>Programming example</td>
<td>19</td>
</tr>
<tr>
<td>Colors and Color-Index</td>
<td>20</td>
</tr>
<tr>
<td>EASY-ROB™ Kernel</td>
<td>22</td>
</tr>
<tr>
<td>Kernel files</td>
<td>22</td>
</tr>
<tr>
<td>Licensing</td>
<td>22</td>
</tr>
<tr>
<td>General Handling</td>
<td>23</td>
</tr>
<tr>
<td>Inverse Kinematics</td>
<td>23</td>
</tr>
<tr>
<td>How does it work?</td>
<td>25</td>
</tr>
<tr>
<td>Trajectory Planner Example</td>
<td>26</td>
</tr>
<tr>
<td>How does it work?</td>
<td>27</td>
</tr>
<tr>
<td>Motion concept with SET_NEXT_TARGET and GET_NEXT_STEP</td>
<td>28</td>
</tr>
<tr>
<td>Other suggestions:</td>
<td>30</td>
</tr>
<tr>
<td>Trajectory Planner Example with synchronized Positioner</td>
<td>31</td>
</tr>
<tr>
<td>How does it work?</td>
<td>32</td>
</tr>
<tr>
<td>Process in detail:</td>
<td>34</td>
</tr>
<tr>
<td>Trajectory Planner Example</td>
<td>37</td>
</tr>
<tr>
<td>Contact</td>
<td>38</td>
</tr>
</tbody>
</table>

Subject to change or improve without prior notice
1. Introduction

Using the Kernel, you have instantly more than 1000 robots for kinematic calculations, and a trajectory planner (motion interpolation) for PTP, SLEW, LIN, and CIRC motions. A powerful collision detection and numerous mathematical functions are also available.

The EASY-ROB™ Kernel is designed as a plug in for integration in technology-based software applications (host application) developed by OEM partners. API-functions and services for the complete robotics functionality are provided. In general, the host application is in charge for the 3D visualization and the administration of geometries and handles for every loaded kinematics.

Features

The EASY-ROB™ Kernel provides mathematics, kinematics and trajectory planner capabilities.

Robot Kinematics

- Load existing EASY-ROB™ robots, devices, tools, etc.
- Huge robot library*) is available
- Access to all robots attributes
- Forward kinematics transformation
- Inverse coordinate transformation
- TCP- and Joint-Jogging
- Access to all robots axis coordinate systems and attributes
  *) ABB / FANUC / KUKA / YASKAWA / STAUBLI / TRICEPT / PKM / Cobots / Universal Robots as well as individual designed robot with new mechanical concepts.

Robot Trajectory planning and -execution

- Motion types: full synchronized PTP, SLEW, LIN and CIRC
- Tool guided movement
- Work piece guided movement, external TCP
- Track Motion
- Conveyor Tracking
- Tracking Windows
- External synchronized positioner
- Wait
- Automatic reduction of the speed, due to singularity
- Cycle time estimation, different velocity profiles
Introduction

Collision detection

- High performed collision detection with three different query types
- COLLIDE detects collision between two Models
- DISTANCE computes the distance between two Models
- TOLERANCE checks if distance between Models is <= tolerance
- Collision Result functions for detailed status information

Mathematics

- Multiplication / inversion / transpose routines for homogeneous transformation matrices (frames)
- Frame conversion into Euler angles (i.e. Yaw Pitch Roll) and quaternion or vice versa
- Routines for building frames (i.e. RotX, RotY, RotZ, TransXYZ and more ...)

Application Programming Interface (API)

The API-Functions and methods respectively services are ajar to the RRS-interface description Realistic Robot Simulation (RRS).

This API is available in two version. The first original version provides Standard ANSI-C compatible functions. The second version wraps these functions and exports the method class ERK_CAPI, which gives a better structure and is easier to use. We recommend Microsoft® Visual Studio C++. Other compilers are possible as well.

The EASY-ROB™ Kernel is available on windows x64 (Linux on request).

To work with the original version, the below two header files required

- erk_capi_types.h
- er_kernel_main.h
- easysimkernel.def

They declare data types and interface functions supplied with this Kernel and for usages in the host application. The exported functions are also available in the module definition file (easysimkernel.def).

To work with the method class ERK_CAPI, the below two header files are required.

- erk_capi_types.h
- erk_capi.h
- erk_capi_definitions.h
Introduction

Below figure shows a layout diagram followed by a brief description of the method classes in ERK_CAPI.

Layout Diagram of ERK_CAPI

Figure: Layout Diagram ERK_CAPI

class ERK_CAPI

A Method class containing all other ERK_CAPI_* classes. The main class ERK_CAPI is sub-divided in following classes.

- **ERK_CAPI_ADMIN** Method class to administrate the EASY-ROB™ Kernel
- **ERK_CAPI_DEVICES** Method class to create, attach, update devices, for kinematics calculations, tool path definition and for trajectory planning and -execution
- **ERK_CAPI_SIM** Method class for simulation settings and collisions as well
- **ERK_CAPI_AUTOPATH** Method class for collision free path planning
- **ERK_CAPI_GEO** Method class to handle 3D Geometry Data imported by Geometry Manager
- **ERK_CAPI_SYS** Method class for mathematical calculations, simulation status, units

class ERK_CAPI_ADMIN

A Method class to administrate the EASY-ROB™ Kernel

- **ERK_CAPI_CALLBACKS** A Method class for Callback functions
Introduction

class ERK_CAPI_DEVICES
A Method class to create, attach, update devices, for kinematics calculations, tool path definition and for trajectory planning and -execution.

- **ERK_CAPI_ROB** Method class kinematics and transformations
- **ERK_CAPI_MOP** Method class for trajectory planning and -execution
- **ERK_CAPI_TOOLPATH** Method class for tool path definition

class ERK_CAPI_ROB
A Method class kinematics and transformations

- **ERK_CAPI_ROB_ATTRIBUTES** Method class for robot/device attributes, travel ranges, home position, device name, ...
- **ERK_CAPI_ROB_KIN** Method class forward-, Inverse kinematics, desired robot joints, tools, position w.r.t. world and reference system
- **ERK_CAPI_ROB_KIN_API** Method class API for forward- and Inverse kinematics

class ERK_CAPI_MOP
A Method class for trajectory planning and -execution

- **ERK_CAPI_MOP_DATA** Method class for start-, target data, motion time, etc
- **ERK_CAPI_MOP_PATH** Method class for path specifications, motion type (PTP, LIN, CIRC), speeds, acceleration, waiting time, etc.
- **ERK_CAPI_MOP_PREP** Method class for trajectory planning (preparation)
- **ERK_CAPI_MOP_EXEC** Method class for motion execution

class ERK_CAPI_TOOLPATH
A Method class for tool path definition

- **ERK_CAPI_TOOLPATH_CREATE** Method class to create tool paths
- **ERK_CAPI_TOOLPATH_TARGETS** Method class to create, unload and specify target locations
- **ERK_CAPI_TOOLPATH_HEAD** Method class to set and get tool path motion header data for target locations
- **ERK_CAPI_TOOLPATH_INSTRUCTIONS** Method class to set and get Instructions for target locations
- **ERK_CAPI_TOOLPATH_ATTRIBUTES** Method class to set and get tool path motion attributes for target locations
- **ERK_CAPI_TOOLPATH_ATTRIBUTES_AUX** Method class to set and get tool path auxiliary motion attributes for target locations
- **ERK_CAPI_TOOLPATH_MOVE_JOINT** Method class to specify a joint motion for target location
- **ERK_CAPI_TOOLPATH_MOVE_CP** Method class to specify a cp motion for target location
- **ERK_CAPI_TOOLPATH_MOTION_EXEC** Method class to access motion execution data at target
- **ERK_CAPI_TOOLPATH_EXTAX_TRACKMOTION** Method class to specify external axis data for a track/slider-motion for target location
- **ERK_CAPI_TOOLPATH_EXTAX_POSITIONER** Method class to specify external axis data for a positioner for target location
- **ERK_CAPI_TOOLPATH_TOOLBOX** Method class for miscellaneous tool path calculations
- **ERK_CAPI_TOOLPATH_APIPP** Method for post processing, creating a robot program for a tool path
class ERK_CAPI_SIM
A Method class for simulation settings

- **ERK_CAPI_SIM_COLLISION** Method class for collision, tolerances, etc.

class ERK_CAPI_AUTOPATH
A Method class for collision free path planning

class ERK_CAPI_GEO
A Method class to handle 3D Geometry Data

- **ERK_CAPI_GEO_MNGR** Method class to access geometry manager methods.

class ERK_CAPI_SYS
A Method class for mathematical calculations, simulation status, units

- **ERK_CAPI_SYS_UTILITIES** Method class for helping functions, color conversion, etc.
- **ERK_CAPI_SYS_MATHEMATICS** Method class for mathematical calculations, multiplications of homogeneous matrices, conversion Euler angle, triangle calculations, formula parser, etc.
doxygen Documentation

A detailed description of all classes and methods are available as doxygen documentation in sub folder "./Includes/doc_erk_capil/"

Precompiled header file: "EASY-ROB Kernel.chm"

An alternative is to load file "index.html" into your Browser.

Information and Support

Please call us if you have any questions or something is not working as it should. We are happy to offer our services for the integration.

Email: support@easy-rob.com

Web: EASY-ROB™ Kernel

Copyright© February 2021 EASY-ROB™ by EASY-ROB Software GmbH
2. Overview of Project Examples

The Kernel comes with some examples to understand how it works and how to use it. All examples are summarized in one Visual Studio 2017 Project, see folder /ERK-Examples/.

"/er_ERK-UDll-Shell-Examples.sln"

These examples can be compiled for 64-Bit platforms and in release or debug mode. The executables "*.exe" are stored in folder "/EasySimKernel/". This folder is called the destination folder. They are running in a simple Dos-Shell and create a data file "*.dat" and a program files "*.prg". The data file contains relevant information about the communication with the kernel, such as licensing, enabled options, handles and so on. The program file will be used together with the EASY-ROB™ App (see folder /EasyRobExe), to visualize the results. In case of a 64-Bit solution platform these files have a "x64" prefix.

Getting Started

Download link for Kernel Example v8.304

https://easy-rob.com/fileadmin/Userfiles/........../erk-example-suite-v8304.zip

How to proceed?

1. After downloading, extract the zip file somewhere, this creates folder \easy-rob-v83\

2. The license file "license.dat" to access the EASY-ROB™ License Server is already included. You need to create the unique hardware number and send it to us, see below. This allows us to register a license for you to evaluate the kernel.

3. Start the EASY-ROB™ App "easyrobwx64.exe" in folder /EasyRobExe/
   Press Ctrl+T to enable all tool bars and for easier operation.

   Note: VS 2017 x64 redistributable are required
   http://www.easy-rob.com/fileadmin/Userfiles/vcredist/2017/vc_redist.x64.exe

4. Load (drag’n drop) one of below example .cel file from the project /Proj/ folder
   a. Collision detection between two simple geometries:

   ![Collision detection between two simple geometries](image)

<table>
<thead>
<tr>
<th>Input</th>
<th>er_ERK-UDll-Shell-COLLcel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>er_ERK-UDll-Shell-COLL-OBJECT_2.prg, er_ERK-UDll-Shell-COLL-x64.dat</td>
</tr>
<tr>
<td>Program</td>
<td>er_ERK-UDll-Shell-COLLx64.exe</td>
</tr>
</tbody>
</table>
b. Kinematics: Loading a robot, calling inverse kinematics, reading joint data

Input: IRB1400H-SPRING-KIN.cel
Output: er_ERK-UDll-shell-KIN-IRB1400H-SPRINGx64.prg
Program: er_ERK-UDll-Shell-KINx64.exe

c. Trajectory planning: Interpolation with PTP, LIN and CIRC motion

Input: IRB1400H-SPRING-MOP-aesx64.cel
Output: er_ERK-UDll-Shell-MOP-GNS-IRB1400H-SPRING-aesx64.prg
Program: er_ERK-UDll-Shell-MOPx64.exe

d. Trajectory planning: Robot with synchronized position

Input: KR-60-3-HA-DKP-400-ToolPath.cel
Output: er_ERK-UDll-Shell-MOP-Positioner-GNS-KR-60-3-HAx64.prg
Program: er_ERK-UDll-Shell-MOP-Positionenx64.exe
e. Trajectory planning: Robot on a track

Input 
KR-120-R3900-Ultra-K-on-Track_ERK.cel

Output 
er_ERK-UDll-Shell-MOP-TrackMotion-GNS-KR120-R3900-ULTRA-Kx64.prg

Program 
er_ERK-UDll-Shell-MOP-TrackMotionx64.exe

f. Trajectory planning: Robot with conveyor tracking

Input 
er_ERK-UDll-Shell-MOP-Conveyor-IRB1400H-SPRING.cel

Output 
er_ERK-UDll-Shell-MOP-Conveyor-IRB1400H-SPRINGx64.prg

Program 
er_ERK-UDll-Shell-MOP-Conveyorx64.exe

5. Press the RUN Button 📈 to start the simulation. EASY-ROB™ will execute the created program ".prg" files, created by the Kernel examples.

**Important Note for Implementation**

Of course, these are just some simple examples to show the functionality of the Kernel and to explain first steps. We store the interpolated joint data of the robot/devices in each interpolation step in an ascii file, a ".prg" file. Using the ERPL command JUMP_TO_AX allows to jump to each of this interpolated joint data step.

When implementing the Kernel into the “real” host application it is **strongly recommended** to visualize the joint data of all devices in each interpolation step and to verify this status. For example, to check collision as well as travel ranges and other constraints, such as the allowed working space etc.

In each interpolation step the user should have the possibility to stop the motion/interpolation, to change the target position/orientation, speed etc. and to move to that modified target again to verify the results. Only then an error free motion can be realized by the user.

Note: Storing the results in a file or a vector class and visualize them afterwards, will give the user not any chance to influence to robot’s behavior. It is necessary that the user can interact with the simulation all the time.
Overview of Project Examples

Example: Loading Geometries and importing IGP files

Project: er_ERK-Read-Igp-Shell
Purpose: Parses the structure of an IGP files
Target folder: \EasySimKernel-Read-Igp-Shell\er_ERK-Read-Igp-Shell
Executable: er_ERK-Read-Igp-Shellx64.exe
Input: IGP files: irb1400H.3, irb1400H.1a or irb1400.0
Output: out.dat, outx64.dat
Remarks: no Kernel required

Example: Collision Detection

Project: er_ERK-UDll-Shell-COLL
Purpose: Collision between two geometries
Target folder: \ (destination folder)
Executable: er_ERK-UDll-Shell-COLllx64.exe
Input: none, two pyramids with 5 vertices are generated internally
Output: er_ERK-UDll-Shell-COLllx64.dat
Remarks: Kernel license or DEMO Kernel required
Viewer: er_ERK-UDll-Shell-COLL.cel press RUN Button
Overview of Project Examples

Example: Inverse Kinematics, loading a robot

<table>
<thead>
<tr>
<th>Project:</th>
<th>er_ERK-UDll-Shell-KIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Loading a robot, calling forward and inverse kinematics</td>
</tr>
<tr>
<td>Target folder:</td>
<td>.\ (destination folder)</td>
</tr>
<tr>
<td>Executable:</td>
<td>er_ERK-UDll-Shell-KINx64.exe</td>
</tr>
<tr>
<td>Input:</td>
<td>Kinematics, Device or Robot file &quot;IRB1400H-SPRING.rob&quot;</td>
</tr>
<tr>
<td>Output:</td>
<td>er_ERK-UDll-Shell-KIN-IRB1400H-SPRINGx64.dat</td>
</tr>
<tr>
<td></td>
<td>er_ERK-UDll-Shell-KIN-IRB1400H-SPRINGx64.prg</td>
</tr>
<tr>
<td>Remarks:</td>
<td>Kernel license or DEMO Kernel required</td>
</tr>
<tr>
<td>ER-App</td>
<td>IRB1400H-SPRING-KIN.cel press RUN Button</td>
</tr>
</tbody>
</table>

Example: Trajectory Planner, interpolation PTP, LIN and CIRC

<table>
<thead>
<tr>
<th>Project:</th>
<th>er_ERK-UDll-Shell-MOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Loading a robot, PTP; LIN; CIRC motions</td>
</tr>
<tr>
<td>Target folder:</td>
<td>.\ (destination folder)</td>
</tr>
<tr>
<td>Executable:</td>
<td>er_ERK-UDll-Shell-MOPx64.exe</td>
</tr>
<tr>
<td>Input:</td>
<td>Kinematics, Device or Robot file &quot;IRB1400H-SPRING.rob&quot;</td>
</tr>
<tr>
<td>Output:</td>
<td>er_ERK-UDll-Shell-MOP-GNS-IRB1400H-SPRING-aesx64.dat</td>
</tr>
<tr>
<td></td>
<td>er_ERK-UDll-Shell-MOP-GNS-IRB1400H-SPRING-aesx64.prg</td>
</tr>
<tr>
<td>Remarks:</td>
<td>Kernel license or DEMO Kernel required</td>
</tr>
<tr>
<td>ER-App</td>
<td>IRB1400H-SPRING-MOP-aesx64.cel press RUN Button</td>
</tr>
</tbody>
</table>
Example: Trajectory Planner with Conveyor Tracking

Project: er_ERK-UDll-Shell-MOP-Conveyor
Purpose: Loading two devices, PTP; LIN; CIRC motions with conveyor tracking
Target folder: \ (destination folder)
Executable: er_ERK-UDll-Shell-MOP-Conveyorx64.exe
Input: Kinematics, Device or Robot file "IRB1400H-SPRING.rob"
Conveyor device or rob file "Conveyor_01.rob"
Output: er_ERK-UDll-Shell-MOP-Conveyor-IRB1400H-SPRINGx64.dat
er_ERK-UDll-Shell-MOP-Conveyor-IRB1400H-SPRINGx64.prg
Remarks: Kernel license or DEMO Kernel required
ER-App er_ERK-UDll-Shell-MOP-Conveyor-IRB1400H-SPRING.cel
press RUN Button

Example: Trajectory Planner with synchronized Positioner

Project: er_ERK-UDll-Shell-MOP-Positioner
Purpose: Loading two devices, PTP; LIN; CIRC motions with synchronized positioner
Target folder: \ (destination folder)
Executable: er_ERK-UDll-Shell-MOP-Positionerx64.exe
Input: Kinematics, Device or Robot file "KR-60-3-HA.rob"
Positioner device or rob file "DKP-400.rob"
Output: er_ERK-UDll-Shell-MOP-Positioner-GNS-KR-60-3-HAx64.dat
er_ERK-UDll-Shell-MOP-Positioner-GNS-KR-60-3-HAx64.prg
Remarks: Kernel license or DEMO Kernel required
ER-App KR-60-3-HA-DKP-400-ToolPath.cel
press RUN Button
Example: Trajectory Planner with Robot on a Track

Project: er_ERK-UDll-Shell-MOP-TrackMotion
Purpose: Loading two devices, PTP; LIN; CIRC motions with synchronized tracking axis
Target folder: .\ (destination folder)
Executable: er_ERK-UDll-Shell-MOP-TrackMotionx64.exe
Input: Kinematics, Device or Robot file "KR120-R3900-ULTRA-K.rob"
Positioner device or rob file "KUKA-KL-1500-3-3000.rob"
Output: er_ERK-UDll-Shell-MOP-TrackMotion-GNS-KR120-R3900-ULTRA-Kx64.prg.dat
er_ERK-UDll-Shell-MOP-TrackMotion-GNS-KR120-R3900-ULTRA-Kx64.prg.prg
Remarks: Kernel license or DEMO Kernel required
ER-App KR-120-R3900-Ultra-K-on-Track_ERK.cel
press RUN Button
3. Loading Geometries, import IGP files version 11 and 12

All device or robot files (*.rob files) using 3D geometries. EASY-ROB™ can import STL (binary or ASCII, even colored), 3DS and its internal part file format IGP (binary or ASCII, always colored).

These geometry data are not included within the rob file. Per default they are stored in a sub folder ".\igp". In the rob file you will find only the file names of the geometries belonging to the device.

The following chapter explains the structure of an IGP file. This is necessary to import and visualize the file format in the 3D scene of the host application.

Another possibility to visualize the 3D geometries is the usage of the “new” Geometry Manager. When loading a robot or device file (*.rob) the Geometry Manager imports all belonging geometries. Via API methods “ERK_CAPI_GEO_MNGR” the Host Application can access all the geometries. Hereby the access does not depend on the internal file format. It does not matter if the geometry exists as STL, IGP, binary/ascii or another file format. Geometries that have been read by the Geometry Manager can, for example, be extracted, entered your own data structure, and then deleted.

For the sake of completeness, we would like to explain for IGP data format (ASCII), allowing you to decide importing this format by yourself.

Import of IGP Geometry files

Using the callback-function LoadGeometry(), the name of the geometry-file fileName will be transferred. The type will be concluded on the basis of the file extension.

File extensions

<table>
<thead>
<tr>
<th>File</th>
<th>extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>STL</td>
<td>// STL file, binary or ASCII (even colored), neutral format</td>
</tr>
<tr>
<td>3DS</td>
<td>// 3D-Studio FILE, neutral format</td>
</tr>
<tr>
<td>IGP</td>
<td>// IGP Part file format (binary or ASCII), always colored.</td>
</tr>
</tbody>
</table>

The structure of an IGP-file version 11, version 12 is slightly modified (see also the programming example)

An IGP-file consists of a header with several objects. The header describes the version number, the number of coordinate systems, number of b-spline curves and –surfaces as well as the number of objects. Every object consists of a color-index and points, as well as indices for lines and polygons.

IGP part file: "irb1400.0", see folder \Proj\igp

<table>
<thead>
<tr>
<th>Version</th>
<th>Version number</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_coorsys</td>
<td>Number of coordinate systems</td>
</tr>
<tr>
<td>normal</td>
<td>orientation</td>
</tr>
<tr>
<td>approach</td>
<td>position</td>
</tr>
<tr>
<td>n_curve</td>
<td>Number of b-spline curves</td>
</tr>
<tr>
<td>n_surface</td>
<td>Number of b-spline surfaces</td>
</tr>
<tr>
<td>n_obj</td>
<td>Number of Objects</td>
</tr>
<tr>
<td>n_color_idx</td>
<td>Color-Index of the 1. Object</td>
</tr>
<tr>
<td>n_point</td>
<td>Number of points in [mm]</td>
</tr>
<tr>
<td>x y z</td>
<td>2. Pkt</td>
</tr>
</tbody>
</table>

Subject to change or improve without prior notice
### Programming example

Visual Studio 2017 Project „er_ERK-Read-Igp-Shell“

In C-example `int read_igp(VRMLBODY *gb)` the structure `VRMLBODY` is passed, which contains the file-name `gb->fln` of the IGP-file and the desired scaling `gb->fscal[0..2]` vector. The file is parsed and all objects will be read. Furthermore, all normal vectors for every polygon and the bounding boxes for every object are calculated.

The following 3 IGP-files serve as an example:

- "irb1400.0" // version 11, 6 objects all objects with User-Color, color idx = -2
- "irb1400H.1a" // version 12, 2 objects with lines only
- "irb1400H.3" // version 12, 10 objects, last object with Color-Idx = 2144517376 --> RGB 255 165 122

The whole geometry is stored in `struct VRMLBODY`. The memory will not be released.
Loading Geometries, import IGP files version 11 and 12

**Output of the example “irb1400H.3”**

```plaintext
number of object data 10
Results reading igp file 'irb1400H.3'
n_obj 10
n_coorsys_total 2
n_point_total 340
n_line_total 0
n_poly_total 239
fscal 1.000 1.000 1.000
size 0.383956 0.253000 0.288000 max size 0.383956
1. object
color index 6 --> RGB 76 76 76
n_coorsys 2
n_point 24
n_line 0
n_poly 13
2. object
color index -2 --> RGB 0 255 255
n_coorsys 0
n_point 32
n_line 0
n_poly 16
10. object
color index 2144517376 --> RGB 255 165 122
n_coorsys 0
n_point 44
n_line 0
n_poly 35
done...
```

**Colors and Color-Index**

Each object has a color, which is described by the color-index.

```
Color-Index =
-2  User-Color, the color RGBColor is used for the object, which has been transferred with LOAD_GEOMETRY_DATA after LoadGeometry().
0-127 The object color is „Standard“ and can be seen in the color_table col_tab[] see also t_color_den2rgb(int den_color, float *color_rgb).
>127 The color of the object is stored in Bit 8 till Bit 31. This is (reading from right)
Bit 32: 0 not used
Bit 31-24: Blue share
Bit 23-16: Green share
Bit 15-8: Red share
Bit 7-1: Default-Colors
```
Example:

Color-Index \[ 6553727 \text{ decimal} = 0 \times 00 \ 64 \ 00 \ 7F \text{ hex} \]

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\[ \text{bin} \]

is the RGB value 0 200 0

with green = 200 decimal = 0xC8 = \[ 1100 \ 1000 \] \[ \text{bin} \]

The programming example should serve as presentation. Ultimately polygons, lines and attributes, e.g. colors have to be added to the geometry-structure of the host application. Furthermore, a triangulation of polygons with more than three indices is necessary respectively recommended.
4. Kinematics Example

**EASY-ROB™ Kernel**

The EASY-ROB™ Kernel is a development version for integration into custom applications. This kernel does all the calculations, such as forward and inverse coordinate-transformation, trajectory-planning and -execution (PTP, SLEW, LIN, CIRC) for all available types of robots. To control it, only C/C++ API-functions/services for robot functionality are given. The custom application does its own 3D-visualization, as well as the management of all geometries and handling of every loaded kinematics. The EASY-ROB™ Kernel returns a handle for every loaded kinematics.

The following kinematics example should support you with your first steps.

Choose an appropriate PC with your development environment and unzip the Zip-file on your hard disk, to create the directory „./ERK_Examples“.

Attachment: << erk-example-suite-v8304.zip >>

**Kernel files:**

The following files belong to the EASY-ROB™ Kernel, see folder \Includes \Libs for header and library files and folder \ERK-Examples\EasySimKernel\ for the Dlls

- erk_version.txt // current Kernel version
- erk_capi_types.h // Header file, declaration of data types and constants
- erk_capi.h // Header file, class ERK_CAPI
- erk_capi_definitions.h // Header file, prototype definitions, etc.
- er_Kernel_main.h // Header file, prototype definitions, etc.
- EasySimKernel.def // Module definition file
- EASY-ROB Kernel.chm // doxygen documentation as compiled html file

- 64-Bit version
- EasySimKernelx64.dll // Windows DLL containing Kernel functionality.
- EasySimKernelx64.lib // Kernel Library for linking
- EasySimKernel_GeoMngrx64.dll // Geometry Manager
- EasySimKernel_tboxx64.dll // Toolbox for individual user customization
- EasySimKernel_apipxx64.dll // User API for user defined post processing
- EasySimKernel_apikinx64.dll // User API for user defined kinematics
- er_wibukeyx64.dll // for WibuKey USB Dongle licensing
- er.codemeterx64.dll // for CodeMeter USB Dongle licensing
- er_matrixlockx64.dll // for MatrixLock USB Dongle licensing

**Licensing:**

For the temporary hardware connected licensing please download the program

Link: https://easy-rob.com/fileadmin/Userfiles/er_hwnr.zip

and execute the application "er_hwnr.exe" with admin rights. Send us the created file „HardwarNumber_xxx.dat and HardwarNumber_xxx.enc“ by E-Mail to sales@easy-rob.com. We will generate a temporary license for your selected computer.
General Handling

When loading rob-files the host application will be informed about the geometries which belong to the kinematics. For this reason, it is necessary to define the following three Callback-functions (see Basic-example er_ERK_UTIL.cpp, er_ERK-UDll-Shell-KIN.cpp)

```c
// erSetCallBack_LoadGeometryProc
erSetCallBack_LoadGeometryProc(&LoadGeometry);

// erSetCallBack_UpdateGeometryProc
erSetCallBack_UpdateGeometryProc(&UpdateGeometry);

// erSetCallBack_FreeGeometryProc
erSetCallBack_FreeGeometryProc(&FreeGeometry);
```

Calling function `erLoadKin(c_er_hnd, fln_rob)` results in a call of the Callback-function.

```
LoadGeometry (ER_HND ErHandle, LOAD_GEOMETRY_DATA *p_load_geometry_data)
```

for each geometry in the rob-file

The actual kinematics-handle ErHandle and the structure LOAD_GEOMETRY_DATA will be transferred. `LoadGeometry(...)` creates a new geometry-handle and saves the transferred attributes like FileName GeoName RGBColor Scaling GeoMat and ax_idx. The geometry-handle is managed by the host application and returned to the kernel.

A call of `erUpdateKin(c_er_hnd)` causes an update of the kinematics respectively a calculation of coordinate systems of each axis as well as the Tip and TCP. To get the associated position of the geometries, `erUpdateGeo(c_er_hnd)` must be called. This causes a single call of the callback-function `UpdateGeometry(…)` for every to the kinematics related geometry.

```
UpdateGeometry(ER_HND ErHandle, TErGeoHandle GeoHandle, DFRAME *)(...)
```

The kinematics-handle ErHandle, the geometry-handle GeoHandle and the transformation KinMat with respect to the robot base will be transferred. The host application must place the appropriate geometry using the geometry-handle. The resulting position of the geometry with respect to the robot base results of the multiplication of KinMat *GeoMat.

When unloading the kinematics `erUnloadKin(c_er_hnd)`, the callback-function `FreeGeometry()` will be called for each geometry. The host application must unload the appropriate geometries and release the geometry-handles.

Inverse Kinematics

This EASY-ROB™ Kernel example "er_ERK-UDll-Shell.exe" will open a Dos-Shell and read-in the Robot-file "IRB1400H-SPRING.rob". The files "er_ERK-UDll-Shell-IRB1400H-SPRING.prg" and "er_ERK-UDll-Shell-IRB1400H-SPRING.dat" will be created.
Kinematics Example

Running this example (Visual Studio 2017 Project "er_ERK-UDll-Shell-KIN") changes TCP position with respect to the Robot base several times and calls the inverse kinematics to calculate the all joint value solutions for each possible configuration. This process will go on until either the position is unreachable, or the travel ranges are exceeded.

The result can be visualized in the EASY-ROB™ App, which can be found on the in folder "EasyRobExe/".

After starting the EASY-ROB™ App, load the cell-file "IRB1400H-SPRING-KIN.cel" first and press RUN Button. This will "run" the output file "er_ERK-UDll-Shell-KIN-IRB1400H-SPRINGx64.prg".
Kinematics Example

How does it work?

Project "er_ERK-UDll-Shell-KIN" has been developed using VS 2017 and creates the executable "er_ERK-UDll-Shellx64.exe". The coding can be found in "er_ERK-UDll-Shell.cpp" and begins with the line main() respectively pn_ERK->do_erk(PROJECT_PATH,"IRB1400H-SPRING.rob");

Procedure:

a) First the kernel has to be initialized. This happens in
ErkInit();// Initialize the Kernel: define Callback Fcts, verify License, Reset robot and geo handles --> m_erk_init=0 is success
For the first step we recommend to define the Callback Fct. LogProc with erSetCallBack_LogProc(). Only then you receive back messages from the kernel which are quite helpful.
By calling
CheckLogproc(m_LogProc=1);// enable LogMessages after Kernel Init, per default m_LogProc=1 when _DEBUG else =0 when NDEBUG
the message logging will be activated.

b) Generate one or more handle kinematics
This happens in
ErkInitKin();// Create some empty robot handles for later usage

c) Now load the .rob-file with the previously created handle (here only one handle is used)
ErkLoadKin(robfile);// Load a robot using robfile, --> c_er_hnd is the current robot handle

d) Now changes happen, starting at line 809
- read the number of Dofs
- read the JointType (if trans or Rot, which is only used for the Output mm or deg)
- read the current Tool, which is here positioned at "0", which is the Robot-Tip
- set the TCP to z=100mm and update the kinematics in the kernel using erUpdateKin(c_er_hnd), because the TCP in relation to the Base has been changed.
- Change the TCP location as far as an "error" or "warning" occurs, see also CErERK::Erk_Do_Kinematics(), where the calculated Joint values can be written into the prg-file with the command Jump_to_ax.

e) Quit the kernel
-ErkUnloadKin();// unload all kinematics
-ErkInit();// Unload Kernel (ErkInit can also be quit if m_erk_init = false, otherwise it restarts again)
- close the .dat and .prg-file

This example shows only the essentials. In addition we used some common kernel methods needed quickly.

The type of management of the kinematics and geo-handles in the host application is up to the developer. Here we are using a simple class CErERK.

An example EASY-ROB™ robot library is available by download.

Link: https://easy-rob.com/fileadmin/Userfiles/robotlib.zip

Note: The robot geometries should be per default always in sub folder "./igp".
5. Trajectory Planner Example

The following Trajectory Planner-example should support you with your first steps.

Visual Studio Project: "er_ERK-UDll-Shell-MOP"

The result can be visualized in the EASY-ROB™ App, which can be found on the in folder "./EasyRobExe"/

After starting the EASY-ROB™ App load the cell-file "IRB1400H-SPRING-MOP-aesx64.cel" first and press RUN Button . This will “run” the output file "er_ERK-UDll-Shell-MOP-GNS-IRB1400H-SPRING-aesx64.prg".

EASY-ROB™ App with robot and created program
Trajectory Planner Example

How does it work?

The project "er_ERK-UDll-Shell-MOP" has been developed with VS 2017 and creates the executable "er_ERK-UDll-Shell-MOP.exe". The coding can be found in "er_ERK-UDll-Shell-MOP.cpp" and begins with the main() respectively pn_ERK->do_erk(PROJECT_PATH, "IRB1400H-SPRING-aes.rob");

Procedure:

a) First the kernel has to be initialized. This happens in
ErkInit (); // Initialize the Kernel: define Callback Fcts, verify License, Reset robot and geo handles --> m_erk_init=0 is success
For the first step we recommend to define the Callback Fct. LogProc with erSetCallBack_LogProc(). Only then you receive back messages from the kernel which are quite helpful.
By calling CheckLogproc(m_LogProc=1);// enable LogMessages after Kernel Init, per default m_LogProc=1 when _DEBUG else =0 when NDEBUG the message logging will be activated.

b) Generate one or more handle kinematics
This happens in
ErkInitKin(); // Create some empty robot handles for later usage

c) Now load now the rob-file with the previously created handle (here only one handle is used)
ErkLoadKin(robfile); // Load a robot using robfile, --> c_er_hnd is the current robot handle

d) No changes happen, see line 1327, „num_dofs=erGet_num_dofs(c_er_hnd);“
- read the number of Dofs
- read the JointType (if translational or rotational, only used for output mm or deg)
- read the current Tool, which is here positioned at "0", which is the Robot-Tip
- set the TCP to z=100mm (comment out!) and update the kinematics in the kernel using erUpdateKin(c_er_hnd), because the TCP in relation to the Base has been changed.

e) Trajectory Planner, some given target positions for interpolation, see also
„do_pause("init + start trajectory planner");“
- ErkMopResetInit(); // Reading respectively setting of robot axe position
- ErkMopSetInitPos(); // Initializing of the Trajectory Planners
- SetNextTarget_01(); // Setting the first target
- ErkMopGetNxtStep (); // Interpolation and setting of other targets

The calculated axis values will be written into the .prg-file using the command JUMP_TO_AX
The .dat-file consists return messages from the kernel.

For more information read „Motion concept with SET_NEXT_TARGET and GET_NEXT_STEP“ on the following pages.
Trajectory Planner Example

f) Quit the kernel
   - ErkUnloadKin(); // unload all kinematics
   - ErkInit();       // Unload Kernel (ErkInit can also be quit if
                       m_erk_init = false, otherwise it restarts again)
   - close the .dat and .prg-file

Motion concept with SET_NEXT_TARGET and GET_NEXT_STEP

The EASY-ROB™ Kernel has been focused while the development on the RRS I Specification (Realistic Robot Simulation). So two API-functions respectively RRS-Services SET_NEXT_TARGET and GET_NEXT_STEP build the engine of the kernel.

INITIALIZE

SET_INITIAL_POSITION

SET_NEXT_TARGET

GET_NEXT_STEP

TERMINATE

Principle RRS Services
Fraunhofer-Institute for Production Systems and Design Technology (IPK), 1994

After initializing the Trajectory Planner the first target will be set using SET_NEXT_TARGET. After that GET_NEXT_STEP will be called as long as „need more data“ or „final step, target reached“ will be returned. In case of an error, a negative value will be returned, see also „er_Kernel_main.h“ or “erk_capih.h”. While interpolating, usually the value 0 for “OK, next step is calculated successful” will be returned. After each GET_NEXT_STEP the robot axis values will be read and visualized in the host application respectively saved as in this example.

Process in detail:

Using SetNextTarget 01(); the first target will be set. The target will be specified, e.g. Motion type, interpolation mode, velocities, accelerations, target ID, etc. The target coordinates will be written into the structure NEXT_TARGET_DATA , before erSET_NEXT_TARGET(c_er_hnd,&next_target_data); is called. If the trace has been successfully prepared, 0 will be returned otherwise the error will be evaluated a.
Trajectory Planner Example

In the example the robot should move to all targets, which are programmed in `SetNextTarget_01()`, with `DFRAME dT[] = {...}`. We set `m_autoplay = 1` and call `ErkMopGetNxtStep();` respectively. `GetNextStep_01();`.

The step size will be set with `m_InterpolationTime` respectively with the service `erSET_INTERPOLATION_TIME(c_er_hnd,m_InterpolationTime);`.

A loop shall run as long as the motion ends (end=1 or negative) or `GET_NEXT_STEP „target reached”` is returned and the last target has been reached.

Code snippet from `CErERK::GetNextStep_01()`

```cpp
float gtime_offset = gtime;
for (t=dt;m_activ_thread_01 && (!end || t<t_min);t+=dt)
{
    // erGET_NEXT_STEP
    ret=erGET_NEXT_STEP(c_er_hnd,output_format,&next_step_data,gtime);
    // erGET_CURRENT_TARGETID
    target_id=next_step_data.TargetID;
    gtime = gtime_offset + (float)t;
    etime = (float)next_step_data.ElapsedTime;
    ...
}
```

GET_NEXT_STEP writes the results into the structure `NEXT_STEP_DATA`. The return value has to be evaluated.

```cpp
if (ret==0)   // next step is calculated successful
{    end=0;
}
else if (ret==2) // target reached
{    // if autoplay and last target not reached yet
        // go to next target
    SetNextTarget_01(m_motion_type,m_target_type);
        end=0;
    // otherwise quit loop
    end=1;
}
else if (ret==1) // need more data
{    // The movement will reach the target in the next step
        // if autoplay and last target not reached yet
        // go to next target
    SetNextTarget_01(m_motion_type,m_target_type);
        end=0;
    // otherwise quit loop
    end=1;
}
else if (ret<0) // error and warning Codes
{    end= -1;   // stop for loop
}
```
Trajectory Planner Example

Representation of the results when no error, \( ret>0 \)

\[
\text{if } (\text{end}>=0) \ // \text{Visualization in host application} \\
\{ \\
\text{ret=erUpdateGeo}(c_{\text{er}_\text{hnd}}); // \text{erUpdateGeo} \\
// \text{current axis values of the cRobot can be found in JointPos} \\
\text{VisuER}(\text{num_dofs}, \text{JointPos}, v, a); // \text{output} \\
\} \\
// \text{go for next step}
\]

Other suggestions:

If you want to drive through the targets with constant speed you have to call \text{erSELECT_FLYBY_MODE}(c_{\text{er}_\text{hnd}}, \text{flyby}_\text{on}) with \text{flyby}_\text{on} = 1, before the trace is planned with the command \text{SET_NEXT_TARGET}.

In the example this happens always if \text{m_autoplay} > 0 is set, see also \text{SetNextTarget}_01();

The next target will be planned (one stroke before) at each „need more data“.

\textbf{Important:} EASY-ROB™ Simulation Kernel does currently not support Rounding (Flyby).

The global Simulations time „gtime“ is managed by the host application itself. The variable „etime“ indicates how much time the current trace will take.

This example shows only the essentials. In addition we used some common kernel methods needed quickly.

The type of management of the kinematics and geo-handles in the host application is up to the developer.

Here we are using a simple class \text{CErERK}.

An example EASY-ROB™ robot library is available by download.

\textbf{Link:} https://easy-rob.com/fileadmin/Userfiles/robotlib.zip

Note: The robot geometries should be per default always in sub folder „./igp“.
6. Trajectory Planner Example with synchronized Positioner

Visual Studio Project:  "er_ERK-UDll-Shell-MOP-Positioner"

The result can be visualized in the EASY-ROB™ App, which can be found on the in folder „/EasyRobExe/”.

After starting the EASY-ROB™ App, load the cell-file "KR-60-3-HA-DKP-400-ToolPath.cel" first and press RUN Button ➤. This will "run" the output file "er_ERK-UDll-Shell-MOP-Positioner-GNS-KR-60-3-HAx64.prg".

The EASY-ROB™ Kernel sample "er_ERK-UDll-Shell-MOP-Positioner.exe" will open a Dos-Shell and read-in the Robot-file "KR-60-3-HA.rob" and the Positioner "DKP-400.rob". The files "er_ERK-UDll-Shell-MOP-Positioner-GNS-KR-60-3-HAx64.prg" and "er_ERK-UDll-Shell-MOP-Positioner-GNS-KR-60-3-HAx64.dat" will be created.
How does it work?

The project "er_ERK-UDll-Shell-MOP-Positioner" has been developed with VS 2017 and creates the executable "er_ERK-UDll-Shell-MOP-Positionerx64.exe". The coding can be found in "er_ERK-UDll-Shell-MOP-Positioner.cpp" and begins with the line main() respectively pn_ERK->do_erk_example(PROJECT_PATH,"KR-60-3-HA.rob","DKP-400.rob");.

Structure of the work cell:
In this example robot and positioner are placed with respect to the world. The robot has its position at z=500mm with the tool data z=100mm. The positioner has its position at x=1000mm and is rotated around the z-axis by 180 degrees.

The targets are set to the tip respectively the tip of the positioner. It makes sense to set the tool data of the positioner to "0" = identity.
As you can see on the screenshot in addition to the TCP-Trace, the direction of the tool axis is displayed to show the change of orientation in a much better way.

Procedure:

a) First the kernel has to be initialized. This happens in
ErkInit (); // Initialize the Kernel: define Callback Fcts, verify License, Reset robot and geo handles --> m_erk_init=0 is success
For the first step we recommend to define the Callback Fct. LogProc with erSetCallBack_LogProc(). Only then you receive back messages from the kernel which are quite helpful.
By calling
CheckLogproc(m_LogProc=1); // enable LogMessages after Kernel Init, per default m_LogProc=1 when _DEBUG else =0 when NDEBUG
the message logging will be activated.

b) Then at least two kinematics will be created
This happens in
int number_of_kin = 2; // robot and positioner
ErkInitKin(number_of_kin); // Create some empty robot handles for later usage

c) Now load the rob-file and positioner-file with the previously created handle
ErkLoadKin(robfile,0); // Load a robot using robfile, --> c_erb_hnd is the current robot handle
ErkLoadKin(posfile,1); // Load a positioner using posfile, --> c_erb_hnd is the current robot handle
Trajectory Planner Example with synchronized Positioner

d) Place robot and positioner with respect to the world
   // 1. Robot
   c_er_hnd = Get_c_ER_HND(1); // get handle for 1st robot (idx=1)

   // Tool data of the robot
   erGetTool(c_er_hnd,&tTw); // read tool
   vec_to_frame(0,0,100*mm2m, 0,0,0,&tTw); // set tool to 100 mm in z direction
   erSetTool(c_er_hnd,&tTw); // set new tool

   // RobotBase of the robot
   erGetRobotBase(c_er_hnd,&iTb); // read robot base position
   vec_to_frame(0,0,500*mm2m, 0,0,0,&iTb); // set robot base position w.r.t World 'i' to 500 mm in z direction
   erSetRobotBase(c_er_hnd,&iTb); // set robot base position

   // 2. Positioner
   c_er_hnd = Get_c_ER_HND(2); // get handle for 2nd robot (idx=2)

   // Tool data of the positioner
   rob_kin_frame_ident(Convert_DFRAME_frame(&tTw)); // positioner without tool!
   erSetTool(c_er_hnd,&tTw); // set new positioner tool

   // RobotBase of the positioner
   vec_to_frame(1000*mm2m,0,0, 0,0,180*RAD,&iTb); // x=1000mm Rz(180°)
   erSetRobotBase(c_er_hnd,&iTb); // set positionerbase position

   // Save the handle of the positioner for later access
   er_hnd_positioner = c_er_hnd; // store this positioner handle

e) Connect robot with the positioner.
   Use the previously saved handle of the positioner
   // Connect Robot with Positioner
   erConnectPositioner(c_er_hnd,er_hnd_positioner);

   The Trajectory-Planner of the robot is now able to move the two axes of the positioner either in a synchronized or not-synchronized way.
   The targets for the robot include the axes of the positioner as external axis. This includes also the axis-speeds of the positioner, see also ER_EXTAX_KIN_DATA

f) Trajectory Planner, Initializing and Start
   "do_pause("init + start trajectory planner");"
   -ErKmopResetInit(); // Read and Set robot joints
   -ErKmopSetInitPos(); // Initializes the Trajectory Planner

   Moving to some targets. The calculated joint values for the robot and positioner are written into the prg-file with the command JUMP_TO_AX.
   The dat-file includes more callback-messages from the kernel
   -ErKrunTargets(); // move along targets

   More information about „Motion concept with SET_NEXT_TARGET and GET_NEXT_STEP“ on the following pages
Trajectory Planner Example with synchronized Positioner

**g) Quit the kernel**

- ErkUnloadKin();  // unload all kinematics
- ErkInit();  // Unload Kernel (ErkInit can also be quit if m_erk_init = false, otherwise it will start again)

- close the .dat- and the .prg-file.

**Process in detail:**

Unlike the first trajectory planner example these targets are outsourced to method GetNextTarget(). Method GetNextTarget() returns 0 if there is a target, otherwise -1 will be returned. Each target has to describe the motion step completely and conclusively.

```cpp
typedef struct {
    long motype;  // ER_JOINT, ER_LIN, ER_SLEW, ER_CIRC
    long flyby_on;  // 0-OFF, 1-ON only when LIN or CIRC
    double speed_percent, speed_cp;  // [%], [m/s]
    double JointPos[DOF6];  // Wait time before and after move
    double CartPosVec[DOF6];  // Target Joint Location for ER_SLEW move
                           // Cart. Target Location for ER_JOINT, ER_LIN and
                           // ER_CIRC move, Pxyz Rxyz
    long sync_type;  // ER_SYNC_OFF or ER_SYNC_ON
    ER_EXTAX_KIN_DATA eakd_1;  // 1st external axis
    ER_EXTAX_KIN_DATA eakd_2;  // 2nd external axis
} TARGET_LOCATION;
```

**Example for the first target**

```cpp
static TARGET_LOCATION targets[] = {
    ER_JOINT, 0, 0, 0.5, 0, 0, // Motion Type, FlyBy
    0, 0, 0, 0, 0, 0, // JointPos
    0.0*mm2m, 0.0*mm2m, 400.0*mm2m, 180*RAD, 0.0*RAD, 0.0*RAD, // CartPosVec Pxyz Rxyz
    ER_SYNC_ON, // Synchronization
    er_hnd_positioner, 0, 0*RAD, 40*RAD, 0*RAD, 30*RAD, 0*RAD, // 1st ER_HND, axis_idx, axis_value, axis_speed
    er_hnd_positioner, 1, 90*RAD, 30*RAD, // 2nd ER_HND, axis_idx, axis_value, axis_speed
};
```

Using PTP motion, a movement happens to the coordinate CartPosVec. The axis-speed is set to 20% of the maximum joint speeds. Flyby is set to 0, so there is a stop at the target position. The command Lag_Time results in a stop and wait of 0.5s at the target position.

The motion is synchronized with the positioner, which is realized by ER_SYNC_ON. That means, the target position is interpreted with respect to the positioner. At the target position both axes of the positioner should interpolate to 0°.

**Example for another target**

```cpp
ER_SLEW, 0, 20, 100*mm2m, 0.5, // Motion Type, FlyBy
0 *RAD, 0 *RAD, 0 *RAD, 90 *RAD, 0 *RAD, // JointPos
0.0*mm2m, 0.0*mm2m, 0.0*mm2m, 0.0*RAD, 0.0*RAD, // CartPosVec Pxyz Rxyz
ER_SYNC_OFF, // Synchronization
er_hnd_positioner, 0, 0*RAD, 40*RAD, // 1st ER_HND, axis_idx, axis_value, axis_speed
er_hnd_positioner, 1, 90*RAD, 30*RAD, 0*RAD, // 2nd ER_HND, axis_idx, axis_value, axis_speed
```

The motion type ER_SLEW is interpreted as PTP motion. The axes-values JointPos of the robot are given (e.g. ABB Rapid MoveAbsJ). The axis-speed is set to 20% of the max. joint speeds. Flyby is set to 0, so there is a stop at the target position. The robot waits 0.5s before he moves again, which can be set with Lead_Time.
Trajectory Planner Example with synchronized Positioner

The robot will wait 1.0s at the target, which can be set with Lag_Time. The motion is not synchronized with the positioner, which is set with ER_SYNC_OFF. At the target position the axis of the positioner should interpolate to 0° and 90°. In this case the external axes are synchronized only in time with the movement of the robot.

These two examples show that each target depends on the previous respectively the following target. A target must be conclusive in itself. It makes no sense to set a Lag_Time, if at the target Flyby has been set to = 1.

ErkRunTargets() has the following sequence. The parameter

\texttt{int run\_target\_status}

shows the interpolation-status of the robot.

\begin{verbatim}
const int RUN_TARGET_STATUS_UNDEF = 0;
const int RUN_TARGET_STATUS_GET_NEXT_TARGET = 1;
const int RUN_TARGET_STATUS_SET_NEXT_TARGET = 2;
const int RUN_TARGET_STATUS_GET_NEXT_STEP = 3;
\end{verbatim}

1. If run\_target\_status==RUN_TARGET_STATUS_GET_NEXT_TARGET, read the next target with GetNextTarget()
2. Check result, set run\_target\_status
3. If run\_target\_status==RUN_TARGET_STATUS_SET_NEXT_TARGET, set the target with SetNextTarget(), the trace will be prepared.
4. Check result, set run\_target\_status
5. If run\_target\_status== RUN_TARGET_STATUS_GET_NEXT_STEP, call erGET\_NEXT\_STEP(), until “need more data” or “target reached” is returned.
6. Check result, set run\_target\_status

A CIRC motion consists of two following CIRC sets. The first CIRC Set provides SetNextTarget()=1 “need more data”, so that the next target, another CIRC set, can be read only before the trace has been planned and the interpolation has started with erGET\_NEXT\_STEP().

A visualization and storage of results happens only if erGET\_NEXT\_STEP() returns a value of >=0.

The step size can be set with m\_InterpolationTime respectively the service erSET\_INTERPOLATION\_TIME(c\_er\_hnd,m\_InterpolationTime);.
Trajectory Planner Example with synchronized Positioner

The sequence of the example shows the following output-screen:

Use robfile IRB1400H-SPRING.rob
Use posfile positioner_01.rob
Write dat file er_ERK-UDll-Shell-MOP-Positioner-IRB1400H-SPRING.dat
Write prg file er_ERK-UDll-Shell-MOP-Positioner-IRB1400H-SPRING.prg
press any key to Connect Robot with Positioner ....
press any key to init + start trajectory planner ....
press any key to set next target ....

GetNextTarget() gt 0.000   TargetIdx 1 of 8
SetNextTarget() gt 0.000   MotionType ER_JOINT
   erGET_NEXT_STEP() 2 - target reached gt 3.400   TrajTime 3.381s

GetNextTarget() gt 3.400   TargetIdx 2 of 8
SetNextTarget() gt 3.400   MotionType ER_LIN
   erGET_NEXT_STEP() 2 - target reached gt 6.150   TrajTime 2.732s

GetNextTarget() gt 6.150   TargetIdx 3 of 8
SetNextTarget() gt 6.150   MotionType ER_LIN
   erGET_NEXT_STEP() 1 - need more data gt 8.625   TrajTime 2.500s

GetNextTarget() gt 8.625   TargetIdx 4 of 8
SetNextTarget() gt 8.625   MotionType ER_CIRC ... need more data

GetNextTarget() gt 8.625   TargetIdx 5 of 8
SetNextTarget() gt 8.625   MotionType ER_CIRC
   erGET_NEXT_STEP() 2 - target reached gt 13.600   TrajTime 4.943s

GetNextTarget() gt 13.600   TargetIdx 6 of 8
SetNextTarget() gt 13.600   MotionType ER_LIN
   erGET_NEXT_STEP() 2 - target reached gt 15.625   TrajTime 2.000s

GetNextTarget() gt 15.625   TargetIdx 7 of 8
SetNextTarget() gt 15.625   MotionType ER_SLEW
   erGET_NEXT_STEP() 2 - target reached gt 21.125   TrajTime 5.480s

GetNextTarget() gt 21.125   TargetIdx 8 of 8
SetNextTarget() gt 21.125   MotionType ER_JOINT
   erGET_NEXT_STEP() 2 - target reached gt 25.475   TrajTime 4.333s
press any key to unload ....
press any key to Done ....

This example shows only the essentials. In addition we used some common kernel methods needed quickly.

The type of management of the kinematics and geo-handles in the host application is up to the developer. Here we are using a simple class CErERK.

An example EASY-ROB™ robot library is available by download.

   Link: https://easy-rob.com/fileadmin/Userfiles/robotlib.zip

Note: The robot geometries should be per default always in sub folder "./igp".
7. EASY-ROB Contact

EASY-ROB Software GmbH

Address: Hauptstr. 42
65719 Hofheim am Taunus
Germany

Contact: Mr. Stefan Anton

Phone: +49 6192 921 70 77
FAX: +49 6192 921 70 66

Email: contact@easy-rob.com
sales@easy-rob.com

Url: www.easy-rob.com

EASY-ROB customer area

Content: Program updates and robot libraries


Log in data:
User name: customer
Password: **********
8. Space for you notes