INTERNATIONAL STANDARD

ISO 10303-105

First edition
1996-12-15

Industrial automation systems and integration — Product data representation and exchange —

Part 105:
Integrated application resource: Kinematics

Systèmes d'automatisation industrielle et intégration — Représentation et échange de données de produits —

Partie 105: Ressource d'application intégrée: Cinématique

Reference number
ISO 10303-105:1996(E)

Contents

Page

Tables

Foreword

The International Organization for Standardization (ISO) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

Draft International Standards adopted by technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75% of the member bodies casting a vote.

International Standard ISO 10303–105 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration,* Subcommittee SC4, *Industrial data.*

ISO 10303 consists of the following parts under the general title *Industrial automation systems and integration – Product data representation and exchange*:

– Part 1, Overview and fundamental principles;

– Part 11, Description methods: The *EXPRESS* language reference manual;

– Part 12, Description method: The *EXPRESS-I* language reference manual;

– Part 21, Implementation methods: Clear text encoding of the exchange structure;

– Part 22, Implementation method: Standard data access interface specification;

– Part 23, Implementation method: C++ language binding to the standard data access interface;

– Part 24, Implementation method: C language binding to the standard data access interface;

– Part 26, Implementation method: Interface definition language binding to the standard data access interface;

– Part 31, Conformance testing methodology and framework: General concepts;

– Part 32, Conformance testing methodology and framework: Requirements on testing laboratories and clients;

– Part 33, Conformance testing methodology and framework: Structure and use of abstract test suites;

– Part 34, Conformance testing methodology and framework: Abstract test methods;

– Part 35, Conformance testing methodology and framework: Abstract test methods for SDAI implementations;

– Part 41, Integrated generic resources: Fundamentals of product description and support;

– Part 42, Integrated generic resources: Geometric and topological representation;

– Part 43, Integrated generic resources: Representation structures;

– Part 44, Integrated generic resources: Product structure configuration;

– Part 45, Integrated generic resource: Materials;

– Part 46, Integrated generic resource: Visual presentation;

– Part 47, Integrated generic resource: Shape variation tolerances;

– Part 49, Integrated generic resource: Process structure and properties;

– Part 101, Integrated application resource: Draughting;

– Part 104, Integrated application resource: Finite element analysis;

– Part 105, Integrated application resource: Kinematics;

– Part 106, Integrated application resource: Building construction core model;

– Part 201, Application protocol: Explicit draughting;

– Part 202, Application protocol: Associative draughting;

– Part 203, Application protocol: Configuration controlled design;

– Part 204, Application protocol: Mechanical design using boundary representation;

– Part 205, Application protocol: Mechanical design using surface representation;

– Part 207, Application protocol: Sheet metal die planning and design;

– Part 208, Application protocol: Life cycle management - Change process;

– Part 209, Application protocol: Composite and metallic structural analysis and related design;

– Part 210, Application protocol: Design of layered electronic products;

- Part 211, Application protocol: Electronics test diagnostics and remanufacture;
- Part 212, Application protocol: Electrotechnical design and installation;
- Part 213, Application protocol: Numerical control process plans for machined parts;
- Part 214, Application protocol: Core data for automotive mechanical design;
- Part 215, Application protocol: Ship arrangement;
- Part 216, Application protocol: Ship moulded forms;
- Part 217, Application protocol: Ship piping;
- Part 218, Application protocol: Ship structures;
- Part 220, Application protocol: Process planning, manufacture, and assembly of layered electronic products;
- Part 221, Application protocol: Functional data and their schematic representation for process plant;
- Part 222, Application protocol: Exchange of product data for composite structures;
- Part 223, Application protocol: Exchange of design and manufacturing product information for cast parts;
- Part 224, Application protocol: Mechanical product definition for process plans using mechanical feature;
- Part 225, Application protocol: Building elements using explicit shape representation;
- Part 226, Application protocol: Ship mechanical systems;
- Part 227, Application protocol: Plant spatial configuration;
- Part 228, Application protocol: Building services: Heating, ventilation, and air conditioning;
- Part 229, Application protocol: Exchange of design and manufacturing product information for forged parts;
- Part 230, Application protocol: Building structural frame: Steelwork;
- Part 231, Application protocol: Process engineering data: Process design and process specification of major equipment;

– Part 232, Application protocol: Technical data package;

– Part 301, Abstract test suite: Explicit draughting;

– Part 302, Abstract test suite: Associative draughting;

– Part 303, Abstract test suite: Configuration controlled design;

– Part 304, Abstract test suite: Mechanical design using boundary representation;

– Part 305, Abstract test suite: Mechanical design using surface representation;

– Part 307, Abstract test suite: Sheet metal die planning and design;

– Part 308, Abstract test suite: Life cycle management - Change process;

– Part 309, Abstract test suite: Composite and metallic structural analysis and related design;

– Part 310, Abstract test suite: Design of layered electronic products;

– Part 311, Abstract test suite: Electronics test diagnostics and remanufacture;

– Part 312, Abstract test suite: Electrotechnical design and installation;

– Part 313, Abstract test suite: Numerical control process plans for machined parts;

– Part 314, Abstract test suite: Core data for automotive mechanical design;

– Part 315, Abstract test suite: Ship arrangement;

– Part 316, Abstract test suite: Ship moulded forms;

– Part 317, Abstract test suite: Ship piping;

– Part 318, Abstract test suite: Ship structures;

– Part 320, Abstract test suite: Process planning, manufacture, and assembly of layered electronic products;

– Part 321, Abstract test suite: Functional data and their schematic representation for process plant;

– Part 322, Abstract test suite: Exchange of product data for composite structures;

– Part 323, Abstract test suite: Exchange of design and manufacturing product information for cast parts;

– Part 324, Abstract test suite: Mechanical product definition for process plans using mechanical features;

– Part 325, Abstract test suite: Building elements using explicit shape representation;

– Part 326, Abstract test suite: Ship mechanical systems;

– Part 327, Abstract test suite: Plant spatial configuration;

– Part 328, Abstract test suite: Building services: Heating, ventilation, and air conditioning;

– Part 329, Abstract test suite: Exchange of design and manufacturing product information for forged parts;

– Part 330, Abstract test suite: Building structural frame: Steelwork;

– Part 331, Abstract test suite: Process engineering data: Process design and process specification of major equipment;

– Part 332, Abstract test suite: Technical data package;

– Part 501, Application interpreted construct: Edge-based wireframe;

– Part 502, Application interpreted construct: Shell-based wireframe;

– Part 503, Application interpreted construct: Geometrically bounded 2D wireframe;

– Part 504, Application interpreted construct: Draughting annotation;

– Part 505, Application interpreted construct: Drawing structure and administration;

– Part 506, Application interpreted construct: Draughting elements;

– Part 507, Application interpreted construct: Geometrically bounded surface;

– Part 508, Application interpreted construct: Non-manifold surface;

– Part 509, Application interpreted construct: Manifold surface;

– Part 510, Application interpreted construct: Geometrically bounded wireframe;

– Part 511, Application interpreted construct: Topologically bounded surface;

– Part 512, Application interpreted construct: Faceted boundary representation;

– Part 513, Application interpreted construct: Elementary boundary representation;

- Part 514, Application interpreted construct: Advanced boundary representation;
- Part 515, Application interpreted construct: Constructive solid geometry;
- Part 517, Application interpreted construct: Mechanical design geometric presentation;
- Part 518, Application interpreted construct: Mechanical design shaded representation.

The structure of this International Standard is described in ISO 10303-1. The numbering of the parts of this International Standard reflects its structure:

- Parts 11 to 13 specify the description methods,
- Parts 21 to 26 specify the implementation methods,
- Parts 31 to 35 specify the conformance testing methodology and framework,
- Parts 41 to 49 specify the integrated generic resources,
- Parts 101 to 106 specify the integrated application resources,
- Parts 201 to 232 specify the application protocols,
- Parts 301 to 332 specify the abstract test suites, and
- Parts 501 to 518 specify the application interpreted constructs.

Should further parts be published, they will follow the same numbering pattern.

Annexes A and B are an integral part of this part of ISO 10303. Annexes C, D, E, F, and G are for information only.

Introduction

ISO 10303 is an International Standard for the computer-interpretable representation and exchange of product data. The objective is to provide a neutral mechanism capable of describing product data throughout the life cycle of a product independent from any particular system. The nature of this description makes it suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases and archiving.

This International Standard is organized as a series of parts, each published separately. The parts of ISO 10303 fall into one of the following series: description methods, integrated resources, application interpreted constructs, application protocols, abstract test suites, implementation methods, and conformance testing. The series are described in ISO 10303-1. This part of ISO 10303 is a member of the integrated resources series. Major subdivisions of this International Standard are:

- kinematic structure;
- kinematic motion representation;
- kinematic analysis control and results.

This part of ISO 10303 specifies an information model for the kinematic aspects of a mechanical product as required for the communication between CAD systems and kinematic analysis systems, and among dissimilar kinematic analysis systems. Kinematic information in the context of this part of ISO 10303 may be used in:

- early design stages, where detailed component shape is yet undetermined. The purpose of the kinematic description in these stages is to develop the conceptual model of the mechanical product to understand its motion characteristics;
- detail design stage where detailed shapes of components are determined. The purpose of the kinematic description in that stage is to verify the performance of the kinematic characteristics of a mechanical product using the final shape of its components, e.g., by means of collision checking.

The kinematic structure in this part of ISO 10303 is composed of rigid components related by kinematic pairs along a surface, on a curve, or at a point.

The kinematic structure schema defines the kinematic structure of rigid objects in terms of links, pairs, and joints. A link is the rigid part of a kinematic object. A pair represents the geometric aspects of the kinematic constraints of motion of these objects, and a joint represents the topological aspects of these constraints. The kinematic structures are represented by graphs in which the vertices of the graph correspond to the links, and the edges of the graph correspond to the joints.

The kinematic motion representation schema specifies the motion of a mechanical product by using a parametric path definition.

The kinematic analysis control and result schema specifies configurations of kinematic structures and the interpolation between configurations. This schema describes prescribed paths for a kinematic analysis and paths resulting from such a kinematic analysis.

Figure 1 illustrates these three schemas and the types and entities that are defined in one schema and referenced by another.

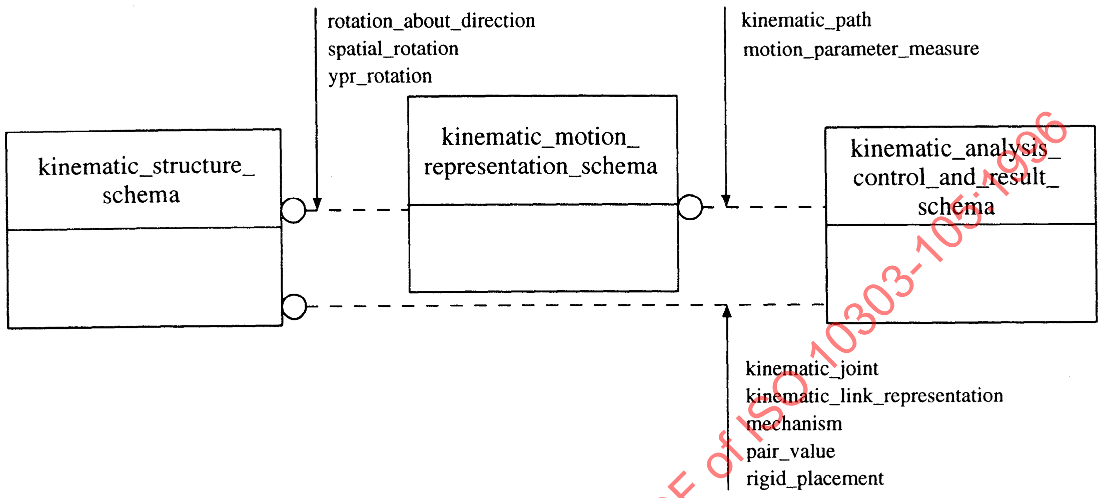


Figure 1 – The relationships between the three schemas of ISO 10303-105: Kinematics

This page intentionally left blank

Industrial automation systems and integration — Product data representation and exchange — Part 105 : Integrated application resource: Kinematics

1 Scope

This part of ISO 10303 specifies the resource constructs for the representation of the kinematic aspects of a mechanical product.

The following are within the scope of this part of ISO 10303:

- definition of kinematic relationships between rigid objects called links;
- representation of the topology for a kinematic structure;
- definition of a kinematic motion as a sequence of discretised positions and orientations;
- representation of the input to and the result of a kinematic analysis.

The following are outside the scope of this part of ISO 10303:

- description of tolerances and clearances in kinematic structures;
- description of motion parameters in terms of a continuous time variable;
- representation of intermittent joints with variable constraints on motion;
- representation of dynamic mechanical assemblies;
- representation of force, mass, and friction.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO 10303. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO 10303 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 8824-1:1995, *Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation.*

ISO 8855:1991, *Road vehicles - Vehicle dynamics and road-holding ability - Vocabulary.*

ISO 10303-1:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 1: Overview and fundamental principles.*

ISO 10303-11:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 11: Description methods: The EXPRESS language reference manual.*

ISO 10303-21:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 21: Implementation methods: Clear text encoding of the exchange structure.*

ISO 10303-41:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 41: Integrated generic resources: Fundamentals of product description and support.*

ISO 10303-42:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 42: Integrated generic resources: Geometric and topological representation.*

ISO 10303-43:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 43: Integrated generic resources: Representation structures.*

3 Definitions

3.1 Terms defined in ISO 10303-1

This part of ISO 10303 makes use of the following terms defined in ISO 10303-1:

- application resource;
- data;
- generic resource;
- information;
- integrated resource;
- product.

3.2 Terms defined in ISO 10303-42

This part of ISO 10303 makes use of the following terms defined in ISO 10303-42:

- geometric coordinate system;

– geometrically founded;

– graph.

3.3 Terms defined in ISO 8855

This part of ISO 10303 makes use of the following terms defined in ISO 8855:

– pitch angle;

– roll angle;

– yaw angle.

3.4 Other definitions

For the purposes of this part of ISO 10303, the following definitions apply.

3.4.1 base: a link which is fixed to the ground or whose motion is prescribed along a defined path.

3.4.2 frame: a coordinate system used in kinematic representations.

3.4.3 ground: the immobile part of a mechanical product in the world coordinate system.

3.4.4 joint: the topological aspect of an ordered connection or motion constraint between two and only two links.

3.4.5 kinematics: the description of movable mechanical structures consisting of joints and links including the number, location, and orientation of the joints.

3.4.6 kinematic chain: a sequence of kinematic links connected by joints.

3.4.7 link: a rigid kinematic object whose motion is constrained by one or more joints.

3.4.8 link frame: the local coordinate system of a link, implicitly defined as the geometric context of the link, with respect to which all geometric definitions of the link are defined.

3.4.9 loop: a part of a mechanical structure whose joints and links together form a closed kinematic chain.

3.4.10 mechanism: a mechanical product whose motion is constrained by joints.

3.4.11 motion: change of position and orientation of a rigid body.

3.4.12 pair: the geometric aspect of either an ordered connection or motion constraint between two and only two links.

3.4.13 pair actuation: the assignment of a pair value to a kinematic pair by an application.

3.4.14 placement: the location and orientation of a frame.

3.4.15 transform: a representation of a placement that is specified with respect to an independent reference frame.

3.4.16 SU-parameters: the parameters that specify the placement between a pair of frames.

3.4.17 world coordinate system: the initial coordinate system from which all other frames are defined.

4 Symbols and abbreviations

For the purposes of this part of ISO 10303, the following symbols and abbreviations apply.

4.1 Mathematical symbols

The mathematical symbol convention used in this part of ISO 10303 is given in table 1.

4.2 Graphic convention for matrix representation

In mathematical formulas, a transform representation (three position values and three rotation values) is written as ***BOLD*** capital letters.

4.3 Abbreviations

SU	Sheth-Uicker
ypr	rotation angles called yaw, pitch, and roll.

Table 1 – Mathematical symbology

Symbol	Definition
a	Scalar quantity a
$\overline{a}$	Vector quantity $\overline{a}$
x, y, z	Coordinate axes
$\overline{x}$, $\overline{y}$, $\overline{z}$	Unit vectors along coordinate axes
$\boldsymbol{F}$	Matrix $\boldsymbol{F}$
$\boldsymbol{F}^{-1}$	The inverse of the matrix $\boldsymbol{F}$
$\times$	Vector (cross) product
$\cdot$	Scalar product
$f(u)$	Parametric curve f
$f(u)\Big\vert_{u=u_p} = f(u_p)$	Value of $f(u)$ at the point denoted by p
$\frac{df(u)}{du}$	Derivative of $f(u)$ with respect to u
$f(u,v)$	Parametric surface f
$f(u,v)\Big\vert_{u=u_p,v=v_p}$	Value of $f(u,v)$ at the point denoted by p
$\frac{\partial f(u,v)}{\partial u}$	Partial differential of f with respect to u
$x \in]a,b]$	x is element of the range between a (excluded) and b (included)
$sign(x)$	Function returns the sign of the argument x
$\vert\vert$	Absolute value, or magnitude, or determinant
R^m	m-dimensional real space

5 Kinematic structure

The following *EXPRESS* declaration begins the **kinematic_structure_schema** and identifies the necessary references.

EXPRESS specification:

```
*)
SCHEMA kinematic_structure_schema;

REFERENCE FROM geometry_schema
    (axis2_placement_3d,
     cartesian_transformation_operator_3d,
     curve,
     direction,
     geometric_representation_context,
     normalise,
     point,
     point_on_curve,
     point_on_surface,
     surface,
     rectangular_trimmed_surface,
     trimmed_curve);

REFERENCE FROM measure_schema
    (conversion_based_unit,
     global_unit_assigned_context,
     length_measure,
     plane_angle_measure,
     si_prefix,
     si_unit,
     si_unit_name,
     unit);

REFERENCE FROM product_property_definition_schema
    (characterized_definition,
     property_definition);

REFERENCE FROM product_property_representation_schema
    (property_definition_representation);

REFERENCE FROM representation_schema
    (functionally_defined_transformation,
     item_defined_transformation,
     representation,
     representation_context,
     representation_item,
     representation_relationship,
     representation_relationship_with_transformation);

REFERENCE FROM support_resource_schema
    (bag_to_set,
    label);
(*
```

NOTES

1 – The schemas referenced above can be found in the following parts of ISO 10303:

geometry_schema	ISO 10303-42
measure_schema	ISO 10303-41
product_property_definition_schema	ISO 10303-41
product_property_representation_schema	ISO 10303-41
representation_schema	ISO 10303-43
support_resource_schema	ISO 10303-41

2 – See annex D for a graphical representation of this schema using the *EXPRESS-G* notation.

5.1 Introduction

The subject of the **kinematic_structure_schema** is the description of the kinematic information of a mechanical product defining its motion capabilities.

Motion in this part of ISO 10303 comprises both free, unconstrained motion of objects in a Cartesian coordinate system and motion constrained by kinematic joints and pairs.

This part of ISO 10303 covers the following types of pairs.

1) revolute pair;
2) prismatic pair;
3) screw pair;
4) cylindrical pair;
5) spherical pair;
6) universal pair;
7) planar pair;
8) gear pair;
9) rack and pinion pair;
10) unconstrained pair;
11) fully constrained pair;
12) point on surface pair;

13) sliding surface pair;

14) rolling surface pair;

15) point on planar curve pair;

16) sliding curve pair;

17) rolling curve pair.

Pair types 1 through 11 are called low order pairs and the remaining pair types are called high order pairs.

NOTE – Pair types 1 through 11 do not require explicit geometric information to describe their kinematic behaviour. Pair types 12 through 17 require a reference to a surface or to a curve in order to be defined kinematically.

For surface pairs and curve pairs, two specialisations are distinguished: the sliding and rolling subtypes. A rolling pair is constrained in the relative motion of the two surfaces or curves such that they cannot slide on each other.

EXAMPLE 1 – High levels of friction between two surfaces or curves may be the cause for relative motion to be constrained to rolling.

For sliding pairs such a constraint does not exist.

Low order pairs can be represented as sliding high order pairs. If a pair can be represented as a low order pair, it should be so represented and not as a high order pair.

5.2 Fundamental concepts and assumptions

5.2.1 Kinematic model structure

Mechanical products with kinematic representations may have arbitrary topological structures. The topological structures are not constrained to simple chains (i.e., open kinematic chains without branches) or to tree-type structures; they may include loops or a network structure.

NOTE – Every kinematic structure can be described in terms of its joints and links. Joints constrain the motion between two rigid objects called links. Joints and links are sufficient to describe the topology of kinematic structures, but for computation it is meaningful to introduce additional levels of structure based on graph theory. Information related to the graph representation of a kinematic structure is given in 5.4.70.

EXAMPLES

2 – Simple chains are suitable kinematic structures for many industrial robots or cranes. Tree-type structures are used to represent multi-arm manipulators.

3 – Network structures occur in industrial products. One example is the mechanism used for windscreen wipers.

In this part of ISO 10303, kinematic structures are represented by graphs where the links represent the vertices of the graph, and the joints represent the edges. The rigid objects with kinematic representations are defined kinematically in terms of links and geometrically in terms of their associated shape representation. All coordinate systems related to a kinematic link are founded in the link frame, which is the geometric context of the related link representation. These coordinate systems are called frames in this part of ISO 10303. For the purpose of representing the kinematic aspect of a mechanism, the shape of the link is represented by the relative location and orientation of all its pair frames with respect to its link frame.

5.2.2 Positioning and placement

Whenever constraints are imposed on the relative motion of two kinematic links, this fact is topologically represented by a joint. The geometric aspect of this motion constraint is represented by the corresponding pair.

A placement describes a relative position and orientation of a frame with respect to another frame. There are different ways of representing a placement. Eight different representations of placement are used in this part of ISO 10303:

- rigid homogenous matrix;
- **axis2_placement_3d**;
- **transform** (see 6.4.2);
- **su_parameters**;
- **point_on_curve**;
- **point_on_surface**;
- a pair type with its **initial_state**;
- **cartesian_transformation_operator_3d**.

The rigid homogeneous matrix is used only as a notation in mathematical formulas. It defines the relative position and orientation of one frame with respect to another. The terms of this matrix are as follows:

$$^{k}\boldsymbol{P}_i = \begin{bmatrix} & 3\times 3 & & \delta x \\ & rotational & & \delta y \\ & matrix & & \delta z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where the notation used for a frame **P** shall be $^{k}\boldsymbol{P}_i$ with:

k = identification of the reference frame in which the position of frame i is described;
i = identification of the frame whose position is described with respect to frame k;
δx, δy, δz = the components of the translation vector.

${}^0\boldsymbol{P}_i$ represents the placement relative to the world coordinate system.

The transform includes three rotational and three translational components. The transform is the only form of placement representation which is suitable for interpolation or approximation via weighting functions.

The remaining representations are described in the schema.

5.2.3 Positive sense of rotations

For any rotation defined in the **kinematic_structure_schema**, a positive value of the rotation angle indicates a counter-clockwise rotation when looking in the negative direction with respect to the axis of that rotation.

5.2.4 Shape representations

Kinematic structure representations are frequently associated with shape representations for single parts of the structure at various levels of detail. In this part of ISO 10303, three kinds of shape representations are distinguished:

– shape representations of parts that belong to the immobile environment of a kinematic structure, referred to as the *ground shape*;

– shape representations describing the shape of a link of a kinematic structure, referred to as the *link shape*;

– shape representations describing specific aspects of the shape of a kinematic pair, referred to as the *pair shape*.

The **geometric_representation_items** which contribute to the ground shape representation may be in the **items** set either of the **kinematic_ground_representation** of the kinematic structure which, in this case, should be simultaneously a **shape_representation**, or of a **shape_representation** which is related to the **kinematic_ground_representation** of the kinematic structure by any means. This part of ISO 10303 does not specify any specific constraints on the representation of ground shape.

The **geometric_representation_items** contributing to a link shape representation shall not be in the **items** set of the corresponding **kinematic_link_representation**; rather they shall be in the **items** set of a **shape_representation** which is related to this **kinematic_link_representation** by means of a **kinematic_link_representation_association**. Any number of **shape_representations** may be related in this way to a specific **kinematic_link_representation**.

For the purpose of a pair shape representation, only **geometric_representation_items** of type **kinematic_frame_background**, i.e., **point**, **curve**, or **surface** are allowed. They shall be in the **items** set of a **kinematic_frame_background_representation** which, in turn, is related to one of the **kinematic_link_representations** of the pair by means of a **kinematic_frame_background_representation_association**. The **items** set of a **kinematic_frame_background_representation** may contain more than one item, and in general any number of

kinematic_frame_background_representations may be related to a **kinematic_link_representation** and to any frame in its **items** set. In the case of the higher pair types, however, which need explicit geometric information to describe their kinematic behaviour, at least one **kinematic_frame_background_representation** is required for each **curve** or **surface** that is referenced in its specification.

5.2.5 Consistent use of units

It is required that units are used consistently throughout a kinematic structure, although such a kinematic structure will include a variety of **representations** and related **representation_-contexts**. To ensure this consistency, it is further required that all units which are to be used in a kinematic structure are assigned globally to the items of the respective **representations** by means of the **global_unit_assigned_context** subtype of the **representation_context** (see ISO 10303-41). Units for **measure_values** of the same kind shall be identical throughout all the **representations** in a kinematic structure.

EXAMPLE 4 – The pair placements of a **kinematic_pair** are founded in two different **kinematic_link_representations**. Consistent use of units requires that these two **kinematic_link_representations** share the same units via their **global_unit_assigned_contexts**.

5.3 kinematic_structure_schema type definitions

5.3.1 rigid_placement

The **rigid_placement** type is a means for referencing either an **axis2_placement_3d** or **su_parameters**.

<u>*EXPRESS* specification:</u>

```
*)
TYPE rigid_placement = SELECT
  (axis2_placement_3d,
   su_parameters);
END_TYPE;
(*
```

5.3.2 rotational_range_measure

The **rotational_range_measure** type is a means for referencing either a **plane_angle_measure** or an **unlimited_range**.

<u>*EXPRESS* specification:</u>

```
*)
TYPE rotational_range_measure = SELECT
  (plane_angle_measure,
   unlimited_range);
END_TYPE;
(*
```

5.3.3 translational_range_measure

The **translational_range_measure** type is a means for referencing either a **length_measure** or an **unlimited_range**.

EXPRESS specification:

```
*)
TYPE translational_range_measure = SELECT
  (length_measure,
   unlimited_range);
END_TYPE;
(*
```

5.3.4 unlimited_range

The **unlimited_range** type is a means by which an unlimited value can be assigned to a range bound of a pair.

EXPRESS specification:

```
*)
TYPE unlimited_range = ENUMERATION OF
  (unlimited);
END_TYPE;
(*
```

Enumerated item definitions:

unlimited: an indication that the range bound is unlimited.

5.3.5 spatial_rotation

The **spatial_rotation** type enables an arbitrary rotation in 3D space ($\boldsymbol{R}^3$) to be specified either as a **ypr_rotation** or as a **rotation_about_direction**.

EXPRESS specification:

```
*)
TYPE spatial_rotation = SELECT
  (ypr_rotation,
   rotation_about_direction);
END_TYPE;
(*
```

5.3.6 ypr_enumeration

The **ypr_enumeration** is a means by which the angles of rotation about the origin in Cartesian coordinate space $\boldsymbol{R}^3$ are distinguished. **Yaw** represents the angle by which the original frame (x_0, y_0, z_0) is rotated first about the z-axis to yield an intermediate (x', y', z') frame. **Pitch** represents the angle by which the intermediate frame (x', y', z') then is rotated about the y$'$-axis to yield another intermediate (x'', y'', z'') frame. **Roll** represents the angle by which the (x'', y'', z'') frame is rotated about the x$''$ -axis to yield the desired final (x, y, z) frame.

NOTE – For more detailed information on yaw, pitch, and roll convention see ISO 8855:1991 (clause 2.2).

EXPRESS specification:

```
*)
TYPE ypr_enumeration = ENUMERATION OF
  (yaw,
   pitch,
   roll);
END_TYPE;
(*
```

Enumerated item definitions:

yaw: the angle of rotation about the z-axis.

pitch: the angle of rotation about the y′-axis.

roll: the angle of rotation about the x″-axis.

5.3.7 ypr_rotation

The **ypr_rotation** type is a means for specifying an ordered sequence of the angles of rotation. The **yaw** value is the first, the **pitch** value the second, and the **roll** value the last element in the array. The yaw rotation shall be performed first, followed by the pitch rotation. The roll rotation shall be performed as the last action of a **ypr_rotation**.

NOTE – An *EXPRESS* function which returns the rotation matrix corresponding to an input **ypr_rotation** is given in annex E. This function has been provided for applications that use this part of ISO 10303, but need the rotational transformation to be expressed in terms of a rotation matrix rather than in terms of a sequence of rotation angles.

EXPRESS specification:

```
*)
TYPE ypr_rotation = ARRAY [ypr_index(yaw) : ypr_index(roll)]
                     OF plane_angle_measure;
END_TYPE;
(*
```

Informal propositions:

angle bounds: The values of the yaw, pitch, and roll angle shall be in the ranges equivalent to the following ranges, where the angles are measured in radians:

$$\begin{aligned} &\textit{yaw angle} \in \,]-\pi,\pi] \\ &\textit{pitch angle} \in [-\pi/2,\pi/2] \\ &\textit{roll angle} \in \,]-\pi,\pi] \end{aligned}$$

rectangular pitch angle: If the absolute value of the pitch angle is equal to the equivalent value of $\pi/2$, the roll angle shall be zero.

5.3.8 kinematic_frame_background

The **kinematic_frame_background** type is a means for referencing either a **point**, a **curve**, or a **surface**.

EXPRESS specification:

```
*)
TYPE kinematic_frame_background = SELECT
  (point,
   curve,
   surface);
END_TYPE;
(*
```

5.4 kinematic_structure_schema entity definitions

5.4.1 rotation_about_direction

A **rotation_about_direction** specifies a rotation in Cartesian coordinate space R^3. The axis of rotation passes the origin of the related coordinate system; its direction is given explicitely as **direction_of_axis**. Furthermore, the amount of rotation is specified by **rotation_angle** which may have any finite value; i.e., its absolute value in radians is not restricted to be less than π. A positive value of **rotation_angle** indicates a counter-clockwise rotation when looking in the negative direction with respect to the **direction_of_axis**.

EXPRESS specification:

```
*)
ENTITY rotation_about_direction;
  direction_of_axis : direction;
  rotation_angle    : plane_angle_measure;
WHERE
  WR1: SIZEOF (direction_of_axis.direction_ratios) = 3;
END_ENTITY;
(*
```

Attribute definitions:

direction_of_axis: the three-dimensional direction of the axis of rotation.

rotation_angle: the angle of rotation about the axis of rotation.

Formal propositions:

WR1: The **direction_of_axis** shall be a three-dimensional direction.

5.4.2 kinematic_property_definition

The **kinematic_property_definition** specifies the kinematic property of a product.

EXPRESS specification:

```
*)
ENTITY kinematic_property_definition
  SUBTYPE OF (property_definition);
  ground_definition : characterized_definition;
END_ENTITY;
(*
```

Attribute definitions:

ground_definition: the **characterized_object**, **characterized_product_definition**, or **shape_definition** related to the **kinematic_ground_representation** which is associated with the **kinematic_property_definition**.

NOTE – The **ground_definition** may be a **characterized_object** if it does not need to be associated with a product definition.

5.4.3 kinematic_property_representation_relation

A **kinematic_property_representation_relation** is a **property_definition_representation** that relates a **kinematic_ground_representation** to a **kinematic_property_definition**.

EXPRESS specification:

```
*)
ENTITY kinematic_property_representation_relation
  SUBTYPE OF (property_definition_representation);
UNIQUE
  UR1: SELF\property_definition_representation.definition;
WHERE
  WR1: 'KINEMATIC_STRUCTURE_SCHEMA.KINEMATIC_PROPERTY_DEFINITION'
         IN TYPEOF (SELF\property_definition_representation.definition);
  WR2: 'KINEMATIC_STRUCTURE_SCHEMA.KINEMATIC_GROUND_REPRESENTATION'
         IN TYPEOF (SELF\property_definition_representation.used_representation);
END_ENTITY;
(*
```

Formal propositions:

UR1: A **kinematic_property_definition** shall be related to a **kinematic_ground_representation** by at most one **kinematic_property_representation_relation**.

WR1: The **definition** of a **kinematic_property_representation_relation** shall be a **kinematic_property_definition**.

WR2: The **used_representation** of a **kinematic_property_representation_relation** shall be a **kinematic_ground_representation**.

5.4.4 kinematic_ground_representation

A **kinematic_ground_representation** specifies the world coordinate system.

NOTE – A **shape_representation** may be given together with a **kinematic_ground_representation** to specify the shape of those parts of a **kinematic_property_definition** that are immobile.

EXPRESS specification:

```
*)
ENTITY kinematic_ground_representation
  SUBTYPE OF (representation);
INVERSE
  property : kinematic_property_representation_relation FOR used_representation;
WHERE
  WR1: 'GEOMETRY_SCHEMA.GEOMETRIC_REPRESENTATION_CONTEXT' IN
       TYPEOF (SELF\representation.context_of_items);
END_ENTITY;
(*
```

Attribute definitions:

SELF\representation.items: a set of **representation_items** that are related in the **context_of_items**.

ground_context: the identification of a world coordinate system against which all placements in the **kinematic_property_definition** are measured, either directly or indirectly.

property: the **kinematic_property_representation_relation** that relates the **kinematic_ground_representation** to the **kinematic_property_definition** to which it belongs.

Formal propositions:

WR1: The **context_of_items** of a **kinematic_ground_representation** shall always be a **geometric_representation_context**.

Informal propositions:

ground definition: The **kinematic_ground_representation** shall be associated to **property\property_definition_representation.definition.ground_definition**.

5.4.5 mechanism

A **mechanism** specifies the capabilities of relative motion of its links under the constraints imposed by its joints. It is a kinematic structure with a single link fixed to the ground or to some other **mechanism**. The entire **mechanism** may be subject to the rigid body motion imposed to its **base**.

NOTE – A mechanism may have different bases at different times. In space applications, worm-like mechanisms have been designed which grab to a space-structure with one end, move on with the other end, fix themselves with an end-effector establishing a new base, and then loosen the original grip.

EXPRESS specification:

```
*)
ENTITY mechanism;
  structure_definition : kinematic_structure;
  base                 : kinematic_link;
  containing_property  : kinematic_property_definition;
WHERE
  WR1: SIZEOF (QUERY (joint <* structure_definition.joints |
               (base :=: joint.first_link) OR
               (base :=: joint.second_link) )) > 0;
END_ENTITY;
(*
```

Attribute definitions:

structure_definition: the topological structure of the **mechanism**.

base: the initial link of the **mechanism**.

containing_property: the **kinematic_property_definition** to which the **mechanism** belongs.

Formal propositions:

WR1: The **base** shall be either the first link or the second link in at least one joint in the set **structure_definition.joints**. This ensures that the **base** is within the kinematic structure.

5.4.6 mechanism_base_placement

A **mechanism_base_placement** is a means to define the placement of the link frame of the **base** of a mechanism either relative to the world coordinate system or relative to the local coordinate system of a link that belongs to another **mechanism**.

NOTE 1 – If the placement is defined relative to the link of another **mechanism**, the **mechanism** whose **base** is being placed is said to be mounted on the other **mechanism**.

If the placement is defined relative to the world coordinate system, **mechanism_base_placement** relates the link frame of the **base** in the **context_of_items** of a **kinematic_ground_representation**. If the placement is defined relative to the local coordinate system of another link, **mechanism_base_placement** relates the link frame of the **base** in the **link_frame** of the **kinematic_link_representation** which is associated with this other link.

EXPRESS specification:

```
*)
ENTITY mechanism_base_placement
  SUBTYPE OF (representation_relationship_with_transformation);
  base_of_mechanism          : mechanism;
  SELF\representation_relationship_with_transformation.
    transformation_operator : cartesian_transformation_operator_3d;
DERIVE
  SELF\representation_relationship.rep_2
                             : kinematic_link_representation
                             := representation_of_link (base_of_mechanism.base);
UNIQUE
  UR1: base_of_mechanism;
WHERE
  WR1: ('KINEMATIC_STRUCTURE_SCHEMA.KINEMATIC_GROUND_REPRESENTATION' IN
        TYPEOF (SELF\representation_relationship.rep_1))
```

```
       OR
       ('KINEMATIC_STRUCTURE_SCHEMA.KINEMATIC_LINK_REPRESENTATION' IN
         TYPEOF (SELF\representation_relationship.rep_1));
  WR2: suitably_based_mechanism (SELF, base_of_mechanism);
  WR3: SELF\representation_relationship_with_transformation.
         transformation_operator\representation_item IN
       SELF\representation_relationship.rep_1.items;
END_ENTITY;
(*
```

Attribute definitions:

base_of_mechanism: the **mechanism** whose **base** is being placed.

SELF\representation_relationship_with_transformation.transformation_operator: the transformation between the context of **rep_1** and the context of **rep_2**.

NOTE 2 – Legal values for the attributes of the **cartesian_transformation_operator_3d** that is used as the **transformation_operator**, are specified in application protocols which use this entity.

SELF\representation_relationship.rep_2: the **kinematic_link_representation** that is related to the **base** of **base_of_mechanism**. This is derived from **base_of_mechanism.base** by the function **representation_of_link**.

Formal propositions:

UR1: A **mechanism** shall be the **base_of_mechanism** for at most one **mechanism_base_-placement**.

WR1: The **rep_1** of a **mechanism_base_placement** shall be a **kinematic_ground_representation** or a **kinematic_link_representation**.

WR2: The **mechanism_base_placement** shall place the **base** of **base_of_mechanism** either with respect to the **kinematic_ground_representation** which is associated with its **containing_property**, or with respect to a **kinematic_link_representation** that belongs to another **mechanism**. This other **mechanism** shall also be in the **mechanisms** set of **containing_-property**. In this way, the **base** of **base_of_mechanism** shall be placed with respect to the **kinematic_ground_representation**, directly or indirectly. If the placement is done indirectly, a **mechanism** shall not participate in the placement of its own **base**. These conditions are checked by the function **suitably_based_mechanism**.

WR3: The **cartesian_transformation_operator_3d** that is used as the **transformation_-operator** in a **mechanism_base_placement** shall be in the **items** set of **rep_1**.

5.4.7 initial_state

The **initial_state** specifies initial values of the pair parameters of a **mechanism**.

NOTE – Some types of kinematic structures, such as four bar linkages, can only be assembled uniquely if configuration information is provided in addition to shape parameters, such as link lengths. In those cases the **initial_state** provides enough pair parameter information to make the configuration unique.

EXPRESS specification:

```
*)
ENTITY initial_state;
  applies_to_mechanism : mechanism;
  pair_values          : SET [ 1 : ?] OF pair_value;
WHERE
  WR1: SIZEOF (QUERY (joint <* applies_to_mechanism.structure_definition.joints |
                SIZEOF (QUERY (init_val <* pair_values |
                        init_val.applies_to_pair.joint :=: joint)) <> 1)) = 0;
END_ENTITY;
(*
```

Attribute definitions:

applies_to_mechanism: the **mechanism** to which the **initial_state** applies.

pair_values: the pair parameter values of each pair in the **mechanism**.

Formal propositions:

WR1: An **initial_state** shall provide in its set of **pair_values** exactly one **pair_value** for each **kinematic_pair** in the **mechanism** to which it applies.

5.4.8 kinematic_structure

The **kinematic_structure** specifies the topological aspects of a kinematic representation. The topology of the **kinematic_structure** is represented by means of **kinematic_joints** which, in turn, establish a relationship between pairs of **kinematic_links**.

NOTE – Every kinematic structure can be described in terms of its joints and links. Joints constrain the motion between two rigid objects called links. Joints and links are sufficient to describe the topology of kinematic structures, but for computation it is meaningful to introduce additional levels of structure based on graph theory. Information related to the graph representation of a kinematic structure is given in 5.4.70.

EXPRESS specification:

```
*)
ENTITY kinematic_structure;
  joints : SET [1 : ?] OF kinematic_joint;
END_ENTITY;
(*
```

Attribute definitions:

joints: the set of **kinematic_joints** that compose the **kinematic_structure** of a **mechanism**.

5.4.9 kinematic_joint

The **kinematic_joint** specifies the topological aspect of an ordered connection between two and only two links.

A topological orientation is always implicitly associated with a joint. This orientation is defined as being from the first link to the second link.

For a **kinematic_pair** referencing a **kinematic_joint**, the **pair_placement_in_first_link_context** shall be an item of the **kinematic_link_representation** of the **first_link** of the **kinematic_joint**.

Similarly, the **pair_placement_in_second_context** shall be an item of the **kinematic_link_representation** of the **second_link** of the **kinematic_joint**.

NOTES

1 – The type of constraint on the relative motion is defined by the **kinematic_pair** that has the joint as its **joint** attribute. In a graph representation of the topology, the **kinematic_joint** represents an oriented edge connecting two vertices in the graph. The orientation of the **kinematic_joint** reflects an implicit orientation which is arbitrary, except for the rack-and-pinion pair, the point-on-surface pair, and the point-on-planar-curve pair.

2 – If the sequence of **first_link** and **second_link** is interchanged, both, pair values and pair range values, have to be used with the opposite signs, and in the subtypes of **simple_pair_range** additionally the attributes giving a lower limit are to be interchanged with those giving the corresponding upper limit.

EXPRESS specification:

```
*)
ENTITY kinematic_joint ;
  first_link  : kinematic_link;
  second_link : kinematic_link;
INVERSE
  structure   : kinematic_structure FOR joints;
WHERE
  WR1: first_link :<>: second_link;
END_ENTITY;
(*
```

Attribute definitions:

first_link: the link through which the joint is entered.

second_link: the link through which the joint is left.

structure: the **kinematic_structure** containing the **kinematic_joint** in its set of **joints**.

Formal propositions:

WR1: **first_link** and **second_link** shall not be identical.

5.4.10 kinematic_link

A **kinematic_link** represents the topological aspects associated with a rigid part of a **mechanism**.

EXPRESS specification:

```
*)
ENTITY kinematic_link;
WHERE
  WR1: SIZEOF (USEDIN (SELF,
                 'KINEMATIC_STRUCTURE_SCHEMA.KINEMATIC_JOINT.FIRST_LINK') +
               USEDIN (SELF,
                 'KINEMATIC_STRUCTURE_SCHEMA.KINEMATIC_JOINT.SECOND_LINK')) > 0;
  WR2: unique_link_usage (SELF);
END_ENTITY;
(*
```

Formal propositions:

WR1: A **kinematic_link** shall be the **first_link** or the **second_link** of at least one **kinematic_joint**.

WR2: A **kinematic_link** shall be a constituent of one mechanism only and shall occur only once in the kinematic structure of that mechanism.

5.4.11 kinematic_link_representation_relation

A **kinematic_link_representation_relation** relates the geometric aspects of a **kinematic_link** with its topological aspects.

EXPRESS specification:

```
*)
ENTITY kinematic_link_representation_relation;
  topological_aspects : kinematic_link;
  geometric_aspects   : kinematic_link_representation;
UNIQUE
  UR1: topological_aspects;
END_ENTITY;
(*
```

Attribute definitions:

topological_aspects: the **kinematic_link** whose geometric aspects are represented by the **kinematic_link_representation**.

geometric_aspects: the representation of the context of the rigid part of a mechanical product and of the frames related to it for a **kinematic_link**.

Formal propositions:

UR1: A **kinematic_link** shall be related to a **kinematic_link_representation** by at most one **kinematic_link_representation_relation**.

5.4.12 kinematic_link_representation

A **kinematic_link_representation** specifies the geometric aspects of a **kinematic_link**. The context of the representation is established by the **link_frame** attribute.

EXPRESS specification:

```
*)
ENTITY kinematic_link_representation
  SUBTYPE OF (representation);
  SELF\representation.context_of_items : geometric_representation_context;
DERIVE
  link_frame                     : geometric_representation_context
                                 := SELF\representation.context_of_items;
INVERSE
  link_representation_relation : kinematic_link_representation_relation FOR
                                  geometric_aspects;
WHERE
  WR1: SIZEOF (QUERY (item <* SELF\representation.items |
             NOT (('KINEMATIC_STRUCTURE_SCHEMA.RIGID_PLACEMENT' IN
                    TYPEOF (item))
                  OR
                  ('GEOMETRY_SCHEMA.CARTESIAN_TRANSFORMATION_OPERATOR_3D' IN
                    TYPEOF (item))) )) = 0;
END_ENTITY;
(*
```

Attribute definitions:

SELF\representation.items: a set of **representation_items** that are related in the **link_frame**. It contains at least one pair frame that is used by a **kinematic_pair** as **pair_placement_in_first_link_context** or as **pair_placement_in_second_link_context**. It may contain any number of **cartesian_transformation_operator_3d**s which serve as the **transformation_operator** of a **mechanism_base_placement**. Furthermore, it may contain additional frames that are attached to the link.

NOTE – The **items** set of a **kinematic_link_representation** does not contain **geometric_representation_items** that define the shape of the link.

SELF\representation.context_of_items: the **geometric_representation_context** providing the context for the **items**.

link_frame: the link frame of the related **kinematic_link**.

link_representation_relation: the **kinematic_link_representation_relation** that relates a **kinematic_link** to the **kinematic_link_representation**.

Formal propositions:

WR1: All elements in the **items** set of a **kinematic_link_representation** shall be of type **rigid_placement** or of type **cartesian_transformation_operator_3d**.

5.4.13 kinematic_link_representation_association

A **kinematic_link_representation_association** is a **representation_relationship** that associates a **representation** with a **kinematic_link_representation**.

NOTE – A **kinematic_link_representation_association** is used to define the shape of a link by associating a **shape_representation** to the corresponding **kinematic_link_representation**.

EXPRESS specification:

```
*)
ENTITY kinematic_link_representation_association
  SUBTYPE OF (representation_relationship);
  SELF\representation_relationship.rep_1 : kinematic_link_representation;
WHERE
  WR1: SELF\representation_relationship.rep_2.context_of_items :=:
       SELF\representation_relationship.rep_1\representation.context_of_items;
  WR2: SIZEOF (['KINEMATIC_STRUCTURE_SCHEMA.KINEMATIC_GROUND_REPRESENTATION',
                'KINEMATIC_STRUCTURE_SCHEMA.KINEMATIC_LINK_REPRESENTATION'] *
               TYPEOF (SELF\representation_relationship.rep_2)) = 0;
END_ENTITY;
(*
```

Attribute definitions:

SELF\representation_relationship.rep_1: the **kinematic_link_representation** with which **rep_2** is associated.

Formal propositions:

WR1: The **context_of_items** of **rep_2** shall be identical to the **link_frame** of the **kinematic_link_representation** with which **rep_2** is associated.

WR2: Neither a **kinematic_ground_representation** nor a **kinematic_link_representation** shall be associated with a **kinematic_link_representation** by this entity.

5.4.14 kinematic_frame_background_representation

A kinematic_frame_background_representation specifies the shape aspects which are associated with a **kinematic_pair**.

EXPRESS specification:

```
*)
ENTITY kinematic_frame_background_representation
  SUBTYPE OF (representation);
  SELF\representation.items                  : SET [1 : ?] OF
                                               kinematic_frame_background;
  SELF\representation.context_of_items : geometric_representation_context;
WHERE
  WR1: SELF\representation.context_of_items\
         geometric_representation_context.coordinate_space_dimension = 3;
END_ENTITY;
(*
```

Attribute definitions:

SELF\representation.items: a set of instances of **kinematic_frame_background** which specify shapes that are to be associated with a **kinematic_pair**.

SELF\representation.context_of_items: the context in which the **items** are founded.

Formal propositions:

WR1: The **context_of_items** of a **kinematic_frame_background_representation** shall be three-dimensional.

5.4.15 kinematic_frame_based_transformation

The **kinematic_frame_based_transformation** is a **functionally_defined_transformation** which uses a **rigid_placement** as the transformation function.

EXPRESS specification:

```
*)
ENTITY kinematic_frame_based_transformation
  SUBTYPE OF (functionally_defined_transformation);
  transformer_frame : rigid_placement;
END_ENTITY;
(*
```

Attribute definitions:

transformer_frame: the **rigid_placement** which is used as the transformation function for the **kinematic_frame_based_transformation**.

5.4.16 kinematic_frame_background_representation_association

A **kinematic_frame_background_representation_association** is a **representation_relationship_with_transformation** which uses a **kinematic_frame_based_transformation** as the **transformation_operator**. The **transformer_frame** of this operator is founded in the **kinematic_link_representation** used as **rep_1**; the representation being transformed is a **kinematic_frame_background_representation** which is referenced by **rep_2**.

EXPRESS specification:

```
*)
ENTITY kinematic_frame_background_representation_association
  SUBTYPE OF (representation_relationship_with_transformation);
  SELF\representation_relationship_with_transformation.
    transformation_operator : kinematic_frame_based_transformation;
WHERE
  WR1: 'KINEMATIC_STRUCTURE_SCHEMA.KINEMATIC_LINK_REPRESENTATION' IN
         TYPEOF (SELF\representation_relationship.rep_1);
  WR2: 'KINEMATIC_STRUCTURE_SCHEMA.KINEMATIC_FRAME_BACKGROUND_REPRESENTATION'
         IN TYPEOF (SELF\representation_relationship.rep_2);
  WR3: SELF\representation_relationship_with_transformation.
           transformation_operator\kinematic_frame_based_transformation.
           transformer_frame\representation_item IN
         SELF\representation_relationship.rep_1.items;
END_ENTITY;
(*
```

Attribute definitions:

SELF\representation_relationship_with_transformation.transformation_operator: the transformation operator for the **kinematic_frame_background_representation_association**.

Formal propositions:

WR1: The first **representation** related in a **kinematic_frame_background_representation_association** shall be a **kinematic_link_representation**.

WR2: The second **representation** related in a **kinematic_frame_background_representation_association** shall be a **kinematic_frame_background_representation**.

WR3: The **rigid_placement** which serves as the **transformer_frame** in the **transformation_operator** of a **kinematic_frame_background_representation_association** shall be in the **items** set of the **kinematic_link_representation** used as **rep_1**.

5.4.17 su_parameters

su_parameters are an alternative method to **axis2_placement_3d** for specifying the placement of pairs on a link.

To describe the kinematic behaviour, the full shape representation of a link is not needed; rather it is sufficient to characterize the link by the transition from the link frame to all the pair frames. This shape information is captured by a set of characteristic parameters called Sheth-Uicker-Parameters, henceforth abbreviated "SU-parameters".

NOTE 1 – See figure 2. For a **kinematic_link** with two pairs and for the link frame coinciding with the frame of one of the pairs, this may be interpreted as the transition from the frame at the beginning of the link to the frame at the following end.

The following notation is used for frame definitions:

x_k^r, y_k^r, z_k^r	coordinate axes of a link frame;
$\bar{o}_k^r$	origin of that link frame;
x_k, y_k, z_k	axes of the frame of a kinematic pair on that link;
$\bar{o}_k$	origin of that pair frame.

The following auxiliary vectors are introduced:

$\bar{t}_k$	unit vector along the common perpendicular between the two axes z_k^r and z_k, directed from z_k^r towards z_k. The common perpendicular itself is denoted by t_k;
$\bar{s}_k^r$	intersection point of the common perpendicular t_k with z_k^r;
$\bar{s}_k$	intersection point of the common perpendicular t_k with z_k.

Then the following notation applies to SU-parameters:

a_k the (positive) distance from z_k^r to z_k, given by

$$a_k = |\bar{s}_k - \bar{s}_k^r|$$

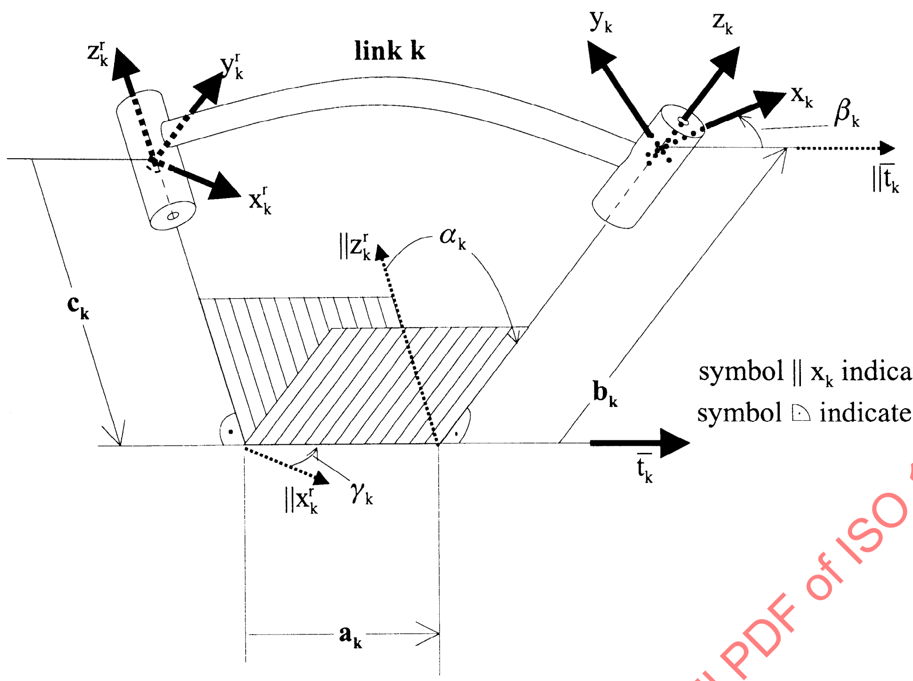


Figure 2 – Definition of the su_parameters

α_k the angle from positive z^r_k to positive z_k, measured in the mathematically positive sense about $\bar{t}_k$ and determined by

$$\sin\alpha_k = (\bar{z}^r_k \times \bar{z}_k)\cdot\bar{t}_k; \qquad \cos\alpha_k = \bar{z}^r_k\cdot\bar{z}_k$$

b_k the deviation of $\bar{o}_k$ from $\bar{s}_k$, measured along positive z_k and given by

$$b_k = (\bar{o}_k - \bar{s}_k)\cdot\bar{z}_k$$

β_k the angle from positive $\bar{t}_k$ to positive x_k, measured in the mathematically positive sense about positive z_k and determined by

$$\sin\beta_k = (\bar{t}_k \times \bar{x}_k)\cdot\bar{z}_k; \qquad \cos\beta_k = \bar{t}_k\cdot\bar{x}_k$$

c_k the deviation of $\bar{s}^r_k$ from $\bar{o}^r_k$, measured along positive z^r_k and given by

$$c_k = (\bar{s}^r_k - \bar{o}^r_k)\cdot\bar{z}^r_k$$

γ_k the angle from positive x^r_k to positive $\bar{t}_k$, measured in the mathematically positive sense about positive z^r_k and determined by

$$\sin\gamma_k = (\bar{x}^r_k \times \bar{t}_k)\cdot\bar{z}^r_k; \qquad \cos\gamma_k = \bar{x}^r_k\cdot\bar{t}_k$$

NOTES

2 – For a detailed description of the Sheth-Uicker parameters see [1].

3 – In industrial robot technology the Denavit-Hartenberg parameters (DH-parameters) are widely

used. See [2]. For kinematic structures with loops it is important to represent the information about the relative placement of the link axes separately from the present state of the pair. In contrast to DH-parameters, SU-parameters provide for this separation.

4 – See annex F for information on the replacement of DH-parameters by SU-parameters.

5 – An instance of **su_parameters** with SU-parameters as defined above is equivalent to an instance of **axis2_placement_3d** that has the following attribute values:

SELF\placement.location: **cartesian_point** $([(a_k \cos\gamma_k + b_k \sin\gamma_k \sin\alpha_k), (a_k \sin\gamma_k - b_k \cos\gamma_k \sin\alpha_k), (c_k + b_k \cos\alpha_k)]);$

axis: **direction** $([(\sin\gamma_k \sin\alpha_k), (-\cos\gamma_k \sin\alpha_k), \cos\alpha_k]);$

ref_direction: **direction** $([(\cos\gamma_k \cos\beta_k - \sin\gamma_k \cos\alpha_k \sin\beta_k), (\sin\gamma_k \cos\beta_k + \cos\gamma_k \cos\alpha_k \sin\beta_k), (\sin\alpha_k \sin\beta_k)]).$

EXPRESS specification:

```
*)
ENTITY su_parameters
  SUBTYPE OF (representation_item);
  a     : length_measure;
  alpha : plane_angle_measure;
  b     : length_measure;
  beta  : plane_angle_measure;
  c     : length_measure;
  gamma : plane_angle_measure;
END_ENTITY;
(*
```

Attribute definitions:

a: SU-parameter a_k.

alpha: SU-parameter α_k.

b: SU-parameter b_k.

beta: SU-parameter β_k.

c: SU-parameter c_k.

gamma: SU-parameter γ_k.

5.4.18 kinematic_pair

A **kinematic_pair** defines the kinematic constraints between two adjacent links coinciding at a joint.

EXPRESS specification:

```
*)
ENTITY kinematic_pair
  SUBTYPE OF (item_defined_transformation);
  joint : kinematic_joint;
DERIVE
  pair_placement_in_first_link_context
        : rigid_placement
        := SELF\item_defined_transformation.transform_item_1;
  pair_placement_in_second_link_context
        : rigid_placement
        := SELF\item_defined_transformation.transform_item_2;
UNIQUE
  UR1: joint;
WHERE
  WR1: coordinated_pair_link_representation
         (joint.first_link, pair_placement_in_first_link_context);
  WR2: coordinated_pair_link_representation
         (joint.second_link, pair_placement_in_second_link_context);
END_ENTITY;
(*
```

Attribute definitions:

joint: the joint whose motion is constrained by the **kinematic_pair**.

pair_placement_in_first_link_context: the placement of the pair frame on the first link in the context of the first link, i.e., its link frame.

pair_placement_in_second_link_context: the placement of the pair frame on the second link in the context of the second link, i.e., its link frame.

Formal propositions:

UR1: Each **kinematic_pair** shall apply to a different **joint**.

WR1: There shall be a **kinematic_link_representation** related to **joint.first_link**, and it shall contain **pair_placement_in_first_link_context** in its **items** set. This condition is checked by the function **coordinated_pair_link_representation**.

WR2: There shall be a **kinematic_link_representation** related to **joint.second_link**, and it shall contain **pair_placement_in_second_link_context** in its **items** set. This condition is checked by the function **coordinated_pair_link_representation**.

NOTES

1 – The relative position of the second link with respect to the first link is defined on the basis of three frames:

– one frame, rigidly connected to the first link and placed in the context (i.e., with respect to the link frame) of the first link;

– another frame, rigidly connected to the second link and placed in the context (i.e., with respect to the link frame) of the second link;

– a contact frame which specifies the point and plane of contact between the two links. This contact frame may or may not be physically connected with one of the links.

The type of the pair, together with the pair parameters, defines the placement of the contact frame relative to the first pair frame as well as the placement of the contact frame relative to the second pair frame. Thus, indirectly the placement of the two pair frames relative to each other is defined. See figure 3.

2 – The goal of a kinematic analysis of a mechanism is to determine the link frames for all the links of that mechanism such that they are consistent with the actual pair parameters. The consistency condition is as follows:

Consider one pair which connects two links and let
j indicate the first link of that pair;
k indicate the second link of that pair;
p indicate that pair as seen from the first link;
q indicate that pair as seen from the second link;
c indicate the common contact point (which may or may not be physically connected with both links);
${}^0\boldsymbol{LF}_j$ and ${}^0\boldsymbol{LF}_k$ be the rigid homogeneous matrix representation of the link frames of those links (both relative to the world coordinate system);
${}^j\boldsymbol{PF}_p$ and ${}^k\boldsymbol{PF}_q$ be the pair placements of pair p with respect to the first and second link of that pair, respectively,

then
${}^0\boldsymbol{PF}_p = {}^0\boldsymbol{LF}_j\ {}^j\boldsymbol{PF}_p$ is the rigid homogeneous matrix placing the first pair frame relative to the world coordinate system while

${}^p\boldsymbol{PF}_c$ is the placement for the contact frame relative to the first pair frame. This placement is determined by the type of the pair and its actual pair parameters.

${}^q\boldsymbol{PF}_c$ is the placement for the contact frame relative to the second pair frame. This placement is determined by the type of the pair and its actual pair parameters.

Consistency now requires that
$${}^0\boldsymbol{LF}_j\ {}^j\boldsymbol{PF}_p\ {}^p\boldsymbol{PF}_c = {}^0\boldsymbol{LF}_k\ {}^k\boldsymbol{PF}_q\ {}^q\boldsymbol{PF}_c$$
Or else, for the second link placement relative to the world coordinate system is obtained
$${}^0\boldsymbol{LF}_k = {}^0\boldsymbol{LF}_j\ {}^j\boldsymbol{PF}_p\ {}^p\boldsymbol{PF}_c\ {}^q\boldsymbol{PF}_c^{-1}\ {}^k\boldsymbol{PF}_q^{-1}$$

3 – For lower pairs the contact frame and the second pair frame shall be treated as identical. Hence, for lower pairs the rigid homogeneous matrix representation of ${}^q\boldsymbol{PF}_c$ is a unit matrix.

5.4.19 pair_actuator

The **pair_actuator** identifies a **kinematic_pair** that is actuated. All degrees of freedom of a **kinematic_pair** that is identified by a **pair_actuator** are treated as being actuated.

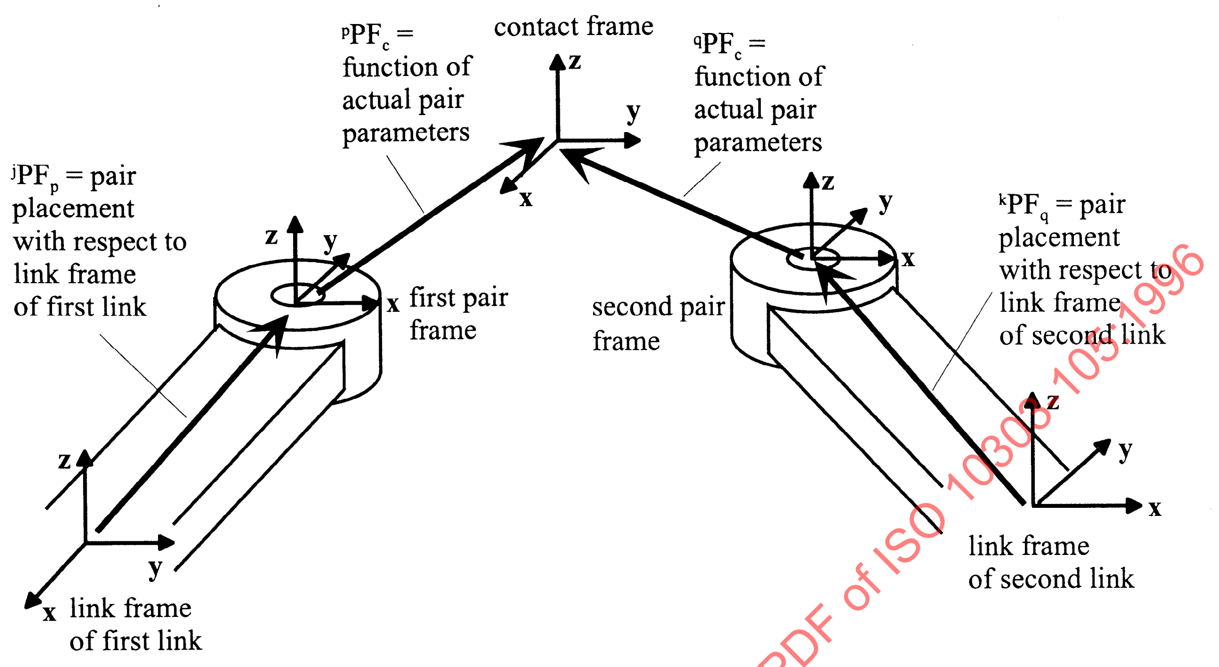

^pPF_c = function of actual pair parameters
contact frame
z
y
x
^qPF_c = function of actual pair parameters
^jPF_p = pair placement with respect to link frame of first link
z
y
x
first pair frame
second pair frame
z
y
x
^kPF_q = pair placement with respect to link frame of second link
z
y
x
link frame of second link
z
y
x
link frame of first link

Figure 3 – Placement relationships of the pair frames relative to the mating links

NOTES

1 – For a backward kinematic analysis the actuated pairs are used to achieve a prescribed path or a prescribed position. For a forward kinematic analysis pair values may be prescribed only for actuated pairs.

2 – This part of ISO 10303 does not support actuation of individual degrees of freedom of a kinematic pair. Modelling of kinematic systems needs to be performed such that a kinematic pair is either fully actuated or not at all. Mixed situations such as, for example, a cylindrical pair can be avoided by using a revolute pair and a prismatic pair separately.

EXPRESS specification:

```
*)
ENTITY pair_actuator;
  actuated_pair : kinematic_pair;
  name          : label;
UNIQUE
  UR1: actuated_pair;
END_ENTITY;
(*
```

Attribute definitions:

actuated_pair: the kinematic pair which is actuated.

name: the word or group of words by which the **pair_actuator** is referred to.

Formal propositions:

UR1: Each **pair_actuator** shall apply to a different **actuated_pair**.

5.4.20 pair_value

The **pair_value** specifies the configuration of the two links that join a **kinematic_pair**.

EXPRESS specification:

```
*)
ENTITY pair_value;
  applies_to_pair: kinematic_pair;
END_ENTITY;
(*
```

Attribute definitions:

applies_to_pair: the **kinematic_pair** to which the **pair_value** applies.

5.4.21 simple_pair_range

The **simple_pair_range** specifies the allowable range of configurations of the two links that join a **kinematic_pair**.

NOTE – The range specification allows for only simple range specifications of each individual pair parameter value in the form "*low bound – high bound*".

EXPRESS specification:

```
*)
ENTITY simple_pair_range;
  applies_to_pair: kinematic_pair;
END_ENTITY;
(*
```

Attribute definitions:

applies_to_pair: the pair to which the range applies.

5.4.22 revolute_pair

The **revolute_pair** constrains the motion between two adjacent links to a rotation about a common axis. To measure the angle of rotation a frame is defined on each of the links such that local origins and the z-axes coincide and their positive directions agree. The motion of the second link with respect to the first link is defined as the angle required to rotate the x-axis of the first pair frame in positive direction around the common z-axis until it matches the x-axis of the second pair frame.

NOTE – See figure 4.

EXPRESS specification:

```
*)
ENTITY revolute_pair
```

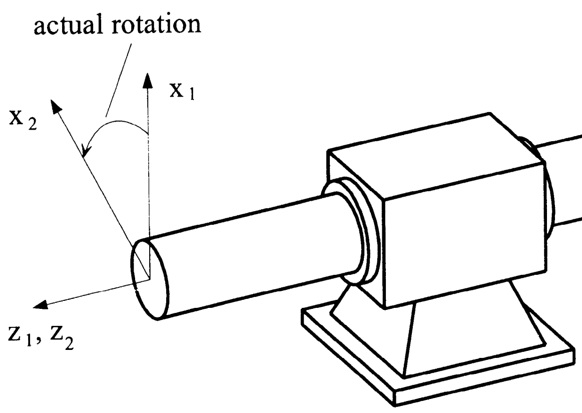


Figure 4 – Example of a joint representing a revolute pair

```
  SUBTYPE OF (kinematic_pair);
END_ENTITY;
(*
```

5.4.23 revolute_pair_value

The **revolute_pair_value** specifies the value of the pair parameter for the **revolute_pair**.

EXPRESS specification:

```
*)
ENTITY revolute_pair_value
  SUBTYPE OF (pair_value);
  SELF\pair_value.applies_to_pair : revolute_pair;
  actual_rotation                 : plane_angle_measure ;
END_ENTITY;
(*
```

Attribute definitions:

SELF\pair_value.applies_to_pair: the **revolute_pair** to which the **revolute_pair_value** applies.

actual_rotation: the value of the pair parameter.

5.4.24 revolute_pair_range

The **revolute_pair_range** specifies the lower bound and the upper bound of the parameter range for the **revolute_pair**.

EXPRESS specification:

```
*)
ENTITY revolute_pair_range
  SUBTYPE OF (simple_pair_range);
  SELF\simple_pair_range.applies_to_pair : revolute_pair;
  lower_limit_actual_rotation           : rotational_range_measure;
  upper_limit_actual_rotation           : rotational_range_measure;
WHERE
  WR1: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (lower_limit_actual_rotation))
        OR
        ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (upper_limit_actual_rotation)))
       XOR
       (lower_limit_actual_rotation < upper_limit_actual_rotation);
END_ENTITY;
(*
```

Attribute definitions:

SELF\simple_pair_range.applies_to_pair: the **revolute_pair** to which the **revolute_pair_range** applies.

lower_limit_actual_rotation: the minimum value of the pair parameter.

upper_limit_actual_rotation: the maximum value of the pair parameter.

Formal propositions:

WR1: The range of the **revolute_pair** shall be positive when both bounds are not given as unlimited.

5.4.25 prismatic_pair

The **prismatic_pair** constrains the motion between two adjacent links to a translation along a common axis. To measure the distance of translation a frame is defined on each of the links such that their corresponding coordinate axes coincide and their positive directions agree. The motion of the second link with respect to the first link is defined as the distance required to translate the xy-plane of the first pair frame in positive direction of the common z-axis until it coincides with the xy-plane of the second pair frame.

NOTE – See figure 5.

EXPRESS specification:

```
*)
ENTITY prismatic_pair
  SUBTYPE OF ( kinematic_pair);
END_ENTITY;
(*
```

5.4.26 prismatic_pair_value

The **prismatic_pair_value** is the value of the pair parameter for the **prismatic_pair**.

EXPRESS specification:

```
*)
ENTITY prismatic_pair_value
```

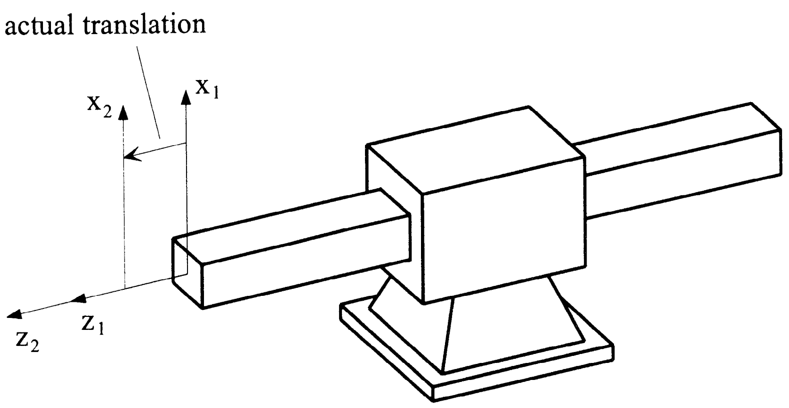


Figure 5 – Example of a joint representing a prismatic pair

```
  SUBTYPE OF (pair_value);
  SELF\pair_value.applies_to_pair : prismatic_pair;
  actual_translation              : length_measure;
END_ENTITY;
(*
```

Attribute definitions:

SELF\pair_value.applies_to_pair: the **prismatic_pair** to which the **prismatic_pair_value** applies.

actual_translation: the value of the pair parameter.

5.4.27 prismatic_pair_range

The **prismatic_pair_range** specifies the lower bound and the upper bound of the parameter range for the **prismatic_pair**.

EXPRESS specification:

```
*)
ENTITY prismatic_pair_range
  SUBTYPE OF (simple_pair_range);
  SELF\simple_pair_range.applies_to_pair : prismatic_pair;
  lower_limit_actual_translation         : translational_range_measure;
  upper_limit_actual_translation         : translational_range_measure;
WHERE
  WR1: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (lower_limit_actual_translation))
        OR
        ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (upper_limit_actual_translation)))
       XOR
       (lower_limit_actual_translation < upper_limit_actual_translation);
END_ENTITY;
(*
```

Attribute definitions:

SELF\simple_pair_range.applies_to_pair: the **prismatic_pair** to which the **prismatic_-pair_range** applies.

lower_limit_actual_translation: the minimum value of the pair parameter.

upper_limit_actual_translation: the maximum value of the pair parameter.

Formal propositions:

WR1: The range of the **prismatic_pair** shall be positive when both bounds are not given as unlimited.

5.4.28 screw_pair

A **screw_pair** constrains the motion between two adjacent links to a rotation about, and translation along, a common axis, where the translation is proportional to the rotation. The factor of proportionality is given by **pitch** which defines the translational displacement for one full rotation. The frames are defined on each of the links such that the z-axes coincide and the positive directions agree.

NOTE 1 – See figure 6.

The motion of the second link with respect to the first link is specified by the angle required to rotate the x-axis of the first pair frame in positive direction around the common z-axis until its direction coincides with the direction of the x-axis of the second pair frame. The translation of the second link along the z-axis shall depend on the rotation according to

$$translation = \frac{pitch \cdot actual\_rotation}{2\pi}$$

where *actual_rotation* is given in radians.

Hence a full rotation ($actual\_rotation = 2\pi$) displaces the xy-plane of the second pair frame by the value *pitch* in the positive z-direction relative to the first pair frame.

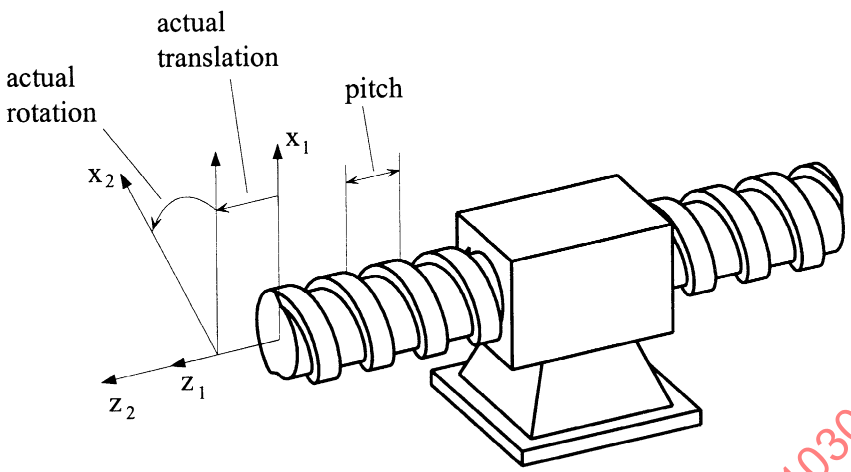
NOTE 2 – The **screw_pair** does not distinguish which one of its links is driving or driven.

EXPRESS specification:

```
*)
ENTITY screw_pair
  SUBTYPE OF (kinematic_pair);
  pitch : length_measure;
END_ENTITY;
(*
```

Attribute definitions:

pitch: the pitch of the screw.



NOTE – The relationship between *actual rotation* and *actual translation* as drawn in the figure does not actually correspond to *pitch*. *actual translation* as drawn would result from an *actual rotation* covering more than a full revolution.

Figure 6 – Example of a joint representing a screw pair

5.4.29 screw_pair_value

A **screw_pair_value** specifies the value of the pair parameter of the **screw_pair**.

EXPRESS specification:

```
*)
ENTITY screw_pair_value
  SUBTYPE OF (pair_value);
  SELF\pair_value.applies_to_pair : screw_pair;
  actual_rotation                 : plane_angle_measure;
DERIVE
  actual_translation              : length_measure
                                  := SELF\pair_value.applies_to_pair\
                                     screw_pair.pitch *
                                     plane_angle_for_pair_in_radian
                                      (SELF\pair_value.applies_to_pair,
                                       actual_rotation) / (2 * PI);
END_ENTITY;
(*
```

Attribute definitions:

SELF\pair_value.applies_to_pair: the **screw_pair** to which the **screw_pair_value** applies.

actual_rotation: the rotation angle of the **screw_pair**.

actual_translation: the resulting translation.

5.4.30 screw_pair_range

The **screw_pair_range** specifies the lower bound and the upper bound of the parameter range for the **screw_pair**.

EXPRESS specification:

```
*)
ENTITY screw_pair_range
  SUBTYPE OF (simple_pair_range);
  SELF\simple_pair_range.applies_to_pair : screw_pair;
  lower_limit_actual_rotation            : rotational_range_measure;
  upper_limit_actual_rotation            : rotational_range_measure;
WHERE
  WR1: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (lower_limit_actual_rotation))
        OR
        ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (upper_limit_actual_rotation)))
       XOR
       (lower_limit_actual_rotation < upper_limit_actual_rotation);
END_ENTITY;
(*
```

Attribute definitions:

SELF\simple_pair_range.applies_to_pair: the **screw_pair** to which the **screw_pair_range** applies.

lower_limit_actual_rotation: the minimum value of the pair parameter.

upper_limit_actual_rotation: the maximum value of the pair parameter.

Formal propositions:

WR1: The range of the **screw_pair** shall be positive when both bounds are not given as unlimited.

5.4.31 cylindrical_pair

The **cylindrical_pair** constrains the motion between two adjacent links to a translation along a common axis and a rotation about it.

NOTE – See figure 7.

To measure the displacement and the angle of rotation, a frame is defined on each of the links such that the z-axes coincide and the positive directions agree. The distance of translation is defined as the distance required to translate the xy-plane of the first pair frame in the positive direction of the common z-axis until it coincides with the xy-plane of the second pair frame. The angle of rotation is defined as the angle required to rotate the x-axis of the first pair frame in the positive direction around the common z-axis until its direction coincides with the direction of the x-axis of the second pair frame.

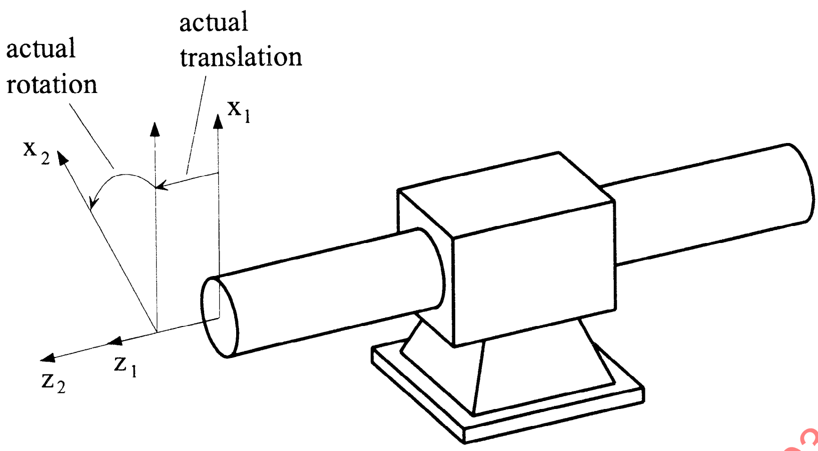


Figure 7 – Example of a joint representing a cylindrical pair

EXPRESS specification:

```
*)
ENTITY cylindrical_pair
  SUBTYPE OF (kinematic_pair);
END_ENTITY;
(*
```

5.4.32 cylindrical_pair_value

The **cylindrical_pair_value** specifies the set of pair parameters for a **cylindrical_pair**.

EXPRESS specification:

```
*)
ENTITY cylindrical_pair_value
  SUBTYPE OF (pair_value);
  SELF\pair_value.applies_to_pair : cylindrical_pair;
  actual_translation              : length_measure;
  actual_rotation                 : plane_angle_measure;
END_ENTITY;
(*
```

Attribute definitions:

SELF\pair_value.applies_to_pair: the **cylindrical_pair** to which the **cylindrical_pair_-value** applies.

actual_translation: the translation value.

actual_rotation: the angle of rotation.

5.4.33 cylindrical_pair_range

The **cylindrical_pair_range** specifies the lower bound and the upper bound of the parameter range for the **cylindrical_pair**.

EXPRESS specification:

```
*)
ENTITY cylindrical_pair_range
  SUBTYPE OF (simple_pair_range);
  SELF\simple_pair_range.applies_to_pair : cylindrical_pair;
  lower_limit_actual_translation         : translational_range_measure;
  upper_limit_actual_translation         : translational_range_measure;
  lower_limit_actual_rotation            : rotational_range_measure;
  upper_limit_actual_rotation            : rotational_range_measure;
WHERE
  WR1: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (lower_limit_actual_translation))
        OR
        ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (upper_limit_actual_translation)))
       XOR
       (lower_limit_actual_translation < upper_limit_actual_translation);
  WR2: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (lower_limit_actual_rotation))
        OR
        ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (upper_limit_actual_rotation)))
       XOR
       (lower_limit_actual_rotation < upper_limit_actual_rotation);
END_ENTITY;
(*
```

Attribute definitions:

SELF\simple_pair_range.applies_to_pair: the **cylindrical_pair** to which the **cylindrical_pair_range** applies.

lower_limit_actual_translation: the minimum value of translation for the **cylindrical_pair**.

upper_limit_actual_translation: the maximum value of translation for the **cylindrical_pair**.

lower_limit_actual_rotation: the minimum value of rotation for the **cylindrical_pair**.

upper_limit_actual_rotation: the maximum value of rotation for the **cylindrical_pair**.

Formal propositions:

WR1: The range of translation of the **cylindrical_pair** shall be positive when both bounds are not given as unlimited.

WR2: The range of rotation of the **cylindrical_pair** shall be positive when both bounds are not given as unlimited.

5.4.34 spherical_pair

The **spherical_pair** constrains the motion between two adjacent links to the rotation about three independent axes that intersect in a common point. To measure the three angles of rotation, a frame system is defined on each of the links such that the origins of the frames coincide. The three angles of rotation are defined as the yaw, pitch, and roll angles. See 5.3.6. They are required to rotate the x-, y-, and z-axes of the first pair frame until the axis directions of this frame coincide with the axis directions of the second pair frame.

NOTE – See figure 8.

EXPRESS specification:

```
*)
ENTITY spherical_pair
  SUBTYPE OF (kinematic_pair);
END_ENTITY;
(*
```

5.4.35 spherical_pair_value

The **spherical_pair_value** specifies the set of pair parameters for a **spherical_pair**.

EXPRESS specification:

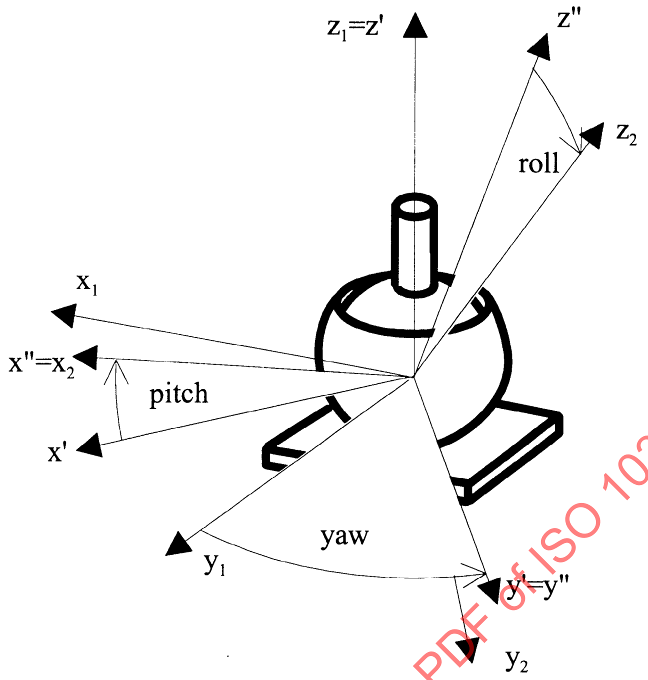
```
*)
ENTITY spherical_pair_value
  SUBTYPE OF (pair_value);
  SELF\pair_value.applies_to_pair : spherical_pair;
  input_orientation               : spatial_rotation;
DERIVE
  actual_orientation              : ypr_rotation
                                  := convert_spatial_to_ypr_rotation
                                     (SELF\pair_value.applies_to_pair,
                                      input_orientation);
END_ENTITY;
(*
```

Attribute definitions:

SELF\pair_value.applies_to_pair: the **spherical_pair** to which the **spherical_pair_value** applies.

input_orientation: the input specification of the pair parameter values from which the **actual_orientation** is derived. **input_orientation** is either a **ypr_rotation** or a **rotation_about_direction**.

actual_orientation: the set of pair parameter values for a **spherical_pair** according to the ypr notation. This is derived from **input_orientation** by the function **convert_spatial_to_ypr_rotation**.



NOTE – (x_1, y_1, z_1) is the first pair frame, and (x_2, y_2, z_2) is the second pair frame. (x', y', z') is an intermediate frame resulting from the yaw rotation of (x_1, y_1, z_1) about z_1. (x'', y'', z'') results from the pitch rotation of (x', y', z') about y'. The roll rotation of (x'', y'', z'') about x'' produces (x_2, y_2, z_2).

Figure 8 – Example of a joint representing a spherical pair

5.4.36 spherical_pair_range

The **spherical_pair_range** specifies the lower bound and the upper bound of the parameter range for a **spherical_pair**.

EXPRESS specification:

```
*)
ENTITY spherical_pair_range
  SUBTYPE OF (simple_pair_range);
  SELF\simple_pair_range.applies_to_pair : spherical_pair;
  lower_limit_yaw                        : rotational_range_measure;
  upper_limit_yaw                        : rotational_range_measure;
  lower_limit_pitch                      : rotational_range_measure;
  upper_limit_pitch                      : rotational_range_measure;
  lower_limit_roll                       : rotational_range_measure;
  upper_limit_roll                       : rotational_range_measure;
WHERE
  WR1: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (lower_limit_yaw))
        OR
        ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (upper_limit_yaw)))
       XOR
       (lower_limit_yaw < upper_limit_yaw);
  WR2: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (lower_limit_pitch))
        OR
        ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (upper_limit_pitch)))
       XOR
       (lower_limit_pitch < upper_limit_pitch);
  WR3: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (lower_limit_roll))
        OR
        ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (upper_limit_roll)))
       XOR
       (lower_limit_roll < upper_limit_roll);
END_ENTITY;
(*
```

Attribute definitions:

SELF\simple_pair_range.applies_to_pair: the **spherical_pair** to which the **spherical_-pair_range** applies.

lower_limit_yaw: the minimum value of the yaw angle for the **spherical_pair**.

upper_limit_yaw: the maximum value of the yaw angle for the **spherical_pair**.

lower_limit_pitch: the minimum value of the pitch angle for the **spherical_pair**.

upper_limit_pitch: the maximum value of the pitch angle for the **spherical_pair**.

lower_limit_roll: the minimum value of the roll angle for the **spherical_pair**.

upper_limit_roll: the maximum value of the roll angle for the **spherical_pair**.

Formal propositions:

WR1: The range of rotation about the z-axis of the **spherical_pair** shall be positive when both bounds are not given as unlimited.

WR2: The range of rotation about the y-axis of the **spherical_pair** shall be positive when both bounds are not given as unlimited.

WR3: The range of rotation about the x-axis of the **spherical_pair** shall be positive when both bounds are not given as unlimited.

5.4.37 universal_pair

The **universal_pair** constrains the motion between two adjacent links to two rotations about two intersecting axes.

NOTE – See figure 9.

To measure those two angles of rotation, a frame system is defined on each of the links such that the origins coincide in the intersection point of the axes, the first z-axis coincides with the first axis of rotation, and the second x-axis coincides with the second axis of rotation. A third constant rotation is inserted between these two variable rotations. In order to migrate from the first pair frame to the second pair frame the following sequence of rotations has to be performed:

1) rotation of the x_1,y_1-axes about the z_1-axis by the amount specified by the first angle yielding an intermediate frame denoted by $(x', y', z' = z_1)$;

2) rotation of the z', x'-axes about the y'-axis by the amount of the constant skew angle yielding a second intermediate frame denoted by $(x'', y'' = y', z'')$;

3) rotation of the y'', z''-axes about the x''-axis by the amount specified by the second angle, yielding the final frame $(x_2 = x'', y_2, z_2)$.

EXPRESS specification:

```
*)
ENTITY universal_pair
  SUBTYPE OF (kinematic_pair);
  input_skew_angle : OPTIONAL plane_angle_measure;
DERIVE
  skew_angle       : plane_angle_measure := NVL (input_skew_angle, 0.0);
WHERE
  WR1: COS (plane_angle_for_pair_in_radian (SELF\kinematic_pair, skew_angle))
       > 0.0;
END_ENTITY;
(*
```

Attribute definitions:

input_skew_angle: an optional attribute that, if present, defines the angle by which the intersection angle between the two axes of rotation deviates from a right angle.

skew_angle: the angle by which the intersection angle between the two axes of rotation deviates from a right angle. If **input_skew_angle** is given, the value of **skew_angle** becomes that of **input_skew_angle**; otherwise it becomes zero.

Formal propositions:

WR1: The absolute value of **skew_angle** shall represent an acute angle.

5.4.38 universal_pair_value

The **universal_pair_value** specifies the set of pair parameters for a **universal_pair**.

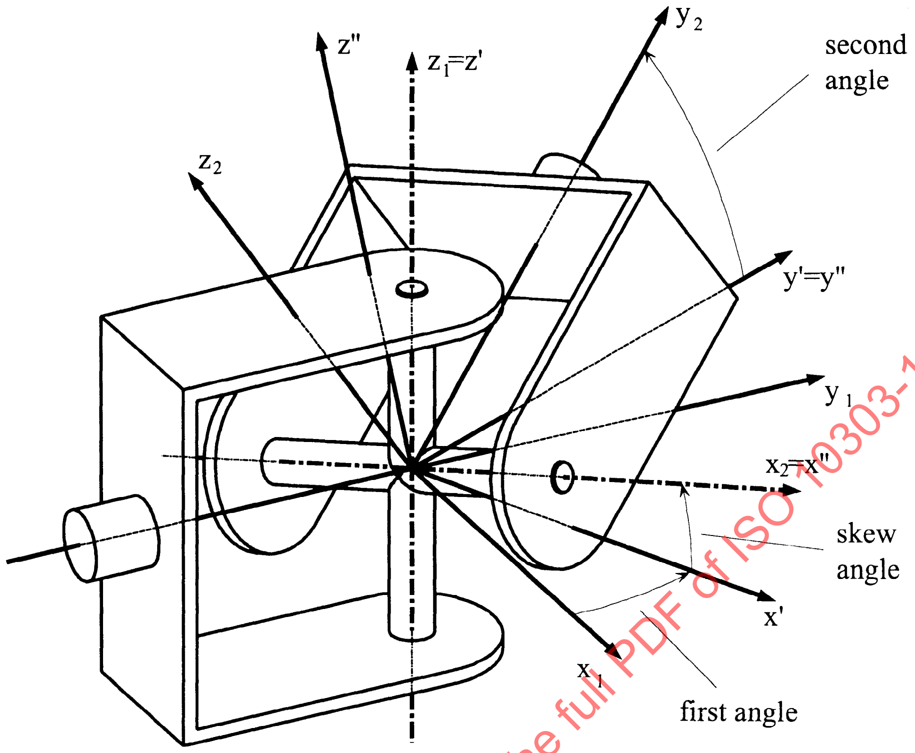


Figure 9 – Example of a joint representing a universal pair

EXPRESS specification:

```
*)
ENTITY universal_pair_value
  SUBTYPE OF (pair_value);
  SELF\pair_value.applies_to_pair : universal_pair;
  first_rotation_angle            : plane_angle_measure;
  second_rotation_angle           : plane_angle_measure;
END_ENTITY;
(*
```

Attribute definitions:

SELF\pair_value.applies_to_pair: the **universal_pair** to which the **universal_pair_value** applies.

first_rotation_angle: the angle of rotation around the first axis.

second_rotation_angle: the angle of rotation around the second axis.

5.4.39 universal_pair_range

The **universal_pair_range** specifies the lower bound and the upper bound of the parameter range for the **universal_pair**.

EXPRESS specification:

```
*)
ENTITY universal_pair_range
  SUBTYPE OF (simple_pair_range);
  SELF\simple_pair_range.applies_to_pair : universal_pair;
  lower_limit_first_rotation             : rotational_range_measure;
  upper_limit_first_rotation             : rotational_range_measure;
  lower_limit_second_rotation            : rotational_range_measure;
  upper_limit_second_rotation            : rotational_range_measure;
WHERE
  WR1: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (lower_limit_first_rotation))
        OR
        ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (upper_limit_first_rotation)))
       XOR
       (lower_limit_first_rotation < upper_limit_first_rotation);
  WR2: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (lower_limit_second_rotation))
        OR
        ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (upper_limit_second_rotation)))
       XOR
       (lower_limit_second_rotation < upper_limit_second_rotation);
END_ENTITY;
(*
```

Attribute definitions:

SELF\simple_pair_range.applies_to_pair: the **universal_pair** to which the **universal_pair_range** applies.

lower_limit_first_rotation: the minimum value of rotation around the first axis for the **universal_pair**.

upper_limit_first_rotation: the maximum value of rotation around the first axis for the **universal_pair**.

lower_limit_second_rotation: the minimum value of rotation around the second axis for the **universal_pair**.

upper_limit_second_rotation: the maximum value of rotation around the second axis for the **universal_pair**.

Formal propositions:

WR1: The range of rotation around the first axis of a **universal_pair** shall be positive when both bounds are not given as unlimited.

WR2: The range of rotation around the second axis of a **universal_pair** shall be positive when both bounds are not given as unlimited.

5.4.40 planar_pair

The **planar_pair** constrains the motion between two adjacent links to translations along the x-axis and y-axis and a rotation about the z-axis. To measure the translations and the angle of rotation, a frame is defined on each of the links such that the x-, y-, and z-axes coincide and the positive directions agree. The translations in x- and y-directions are defined as the displacements required to translate the origin of the first pair frame in positive direction of the x- and y-axes until it coincides with the origin of the second pair frame. The angle of rotation is defined as the angle required to rotate the x-axis of the first pair frame in positive direction around the common z-axis until its direction coincides with the direction of the x-axis of the second pair frame.

NOTE – See figure 10.

EXPRESS specification:

```
*)
ENTITY planar_pair
  SUBTYPE OF (kinematic_pair);
END_ENTITY;
(*
```

5.4.41 planar_pair_value

The **planar_pair_value** specifies the set of pair parameters for the **planar_pair**.

EXPRESS specification:

```
*)
ENTITY planar_pair_value
  SUBTYPE OF (pair_value);
  SELF\pair_value.applies_to_pair : planar_pair;
  actual_rotation                 : plane_angle_measure;
  actual_translation_x            : length_measure;
  actual_translation_y            : length_measure;
END_ENTITY;
(*
```

Attribute definitions:

SELF\pair_value.applies_to_pair: the **planar_pair** to which the **planar_pair_value** applies.

actual_rotation: the value of the angle of rotation for a **planar_pair**.

actual_translation_x: the value of translation in x-direction for a **planar_pair**.

actual_translation_y: the value of translation in y-direction for a **planar_pair**.

5.4.42 planar_pair_range

The **planar_pair_range** specifies the lower bound and the upper bound of the parameter range for the **planar_pair**.

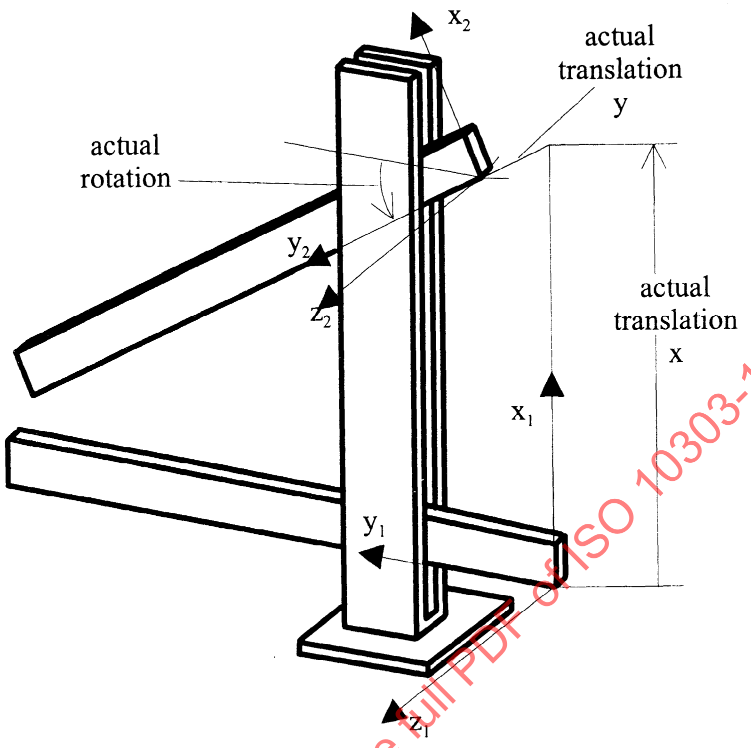


Figure 10 – Example of a joint representing a planar pair

EXPRESS specification:

```
*)
ENTITY planar_pair_range
  SUBTYPE OF (simple_pair_range);
  SELF\simple_pair_range.applies_to_pair : planar_pair;
  lower_limit_actual_rotation            : rotational_range_measure;
  upper_limit_actual_rotation            : rotational_range_measure;
  lower_limit_actual_translation_x       : translational_range_measure;
  upper_limit_actual_translation_x       : translational_range_measure;
  lower_limit_actual_translation_y       : translational_range_measure;
  upper_limit_actual_translation_y       : translational_range_measure;
WHERE
  WR1: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (lower_limit_actual_rotation))
        OR
        ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (upper_limit_actual_rotation)))
       XOR
       (lower_limit_actual_rotation < upper_limit_actual_rotation);
  WR2: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (lower_limit_actual_translation_x))
        OR
        ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (upper_limit_actual_translation_x)))
        XOR
        (lower_limit_actual_translation_x < upper_limit_actual_translation_x);
  WR3: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
          TYPEOF (lower_limit_actual_translation_y))
         OR
         ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
          TYPEOF (upper_limit_actual_translation_y)))
        XOR
        (lower_limit_actual_translation_y < upper_limit_actual_translation_y);
END_ENTITY;
(*
```

Attribute definitions:

SELF\simple_pair_range.applies_to_pair: the **planar_pair** to which the **planar_pair_range** applies.

lower_limit_actual_rotation: the minimum value of the angle of rotation for the **planar_pair**.

upper_limit_actual_rotation: the maximum value of the angle of rotation for the **planar_pair**.

lower_limit_actual_translation_x: the minimum value of translation in x-direction for the **planar_pair**.

upper_limit_actual_translation_x: the maximum value of translation in x-direction for the **planar_pair**.

lower_limit_actual_translation_y: the minimum value of translation in y-direction for the **planar_pair**.

upper_limit_actual_translation_y: the maximum value of translation in y-direction for the **planar_pair**.

Formal propositions:

WR1: The range of rotation of the **planar_pair** shall be positive when both bounds are not given as unlimited.

WR2: The range of translation in x-direction of the **planar_pair** shall be positive when both bounds are not given as unlimited.

WR3: The range of translation in y-direction of the **planar_pair** shall be positive when both bounds are not given as unlimited.

5.4.43 unconstrained_pair

The **unconstrained_pair** does not constrain the relative motion between two adjacent links.

NOTES

1 – The **unconstrained_pair** may be helpful in early design stages or for kinematic analysis systems, because it allows to eliminate the constraints between two links without changing the topological structure.

2 – The concept of pair range is not applicable to an **unconstrained_pair**.

EXPRESS specification:

```
*)
ENTITY unconstrained_pair
  SUBTYPE OF (kinematic_pair);
END_ENTITY;
(*
```

5.4.44 unconstrained_pair_value

An **unconstrained_pair_value** specifies the pair parameters for an **unconstrained_pair**.

EXPRESS specification:

```
*)
ENTITY unconstrained_pair_value
  SUBTYPE OF (pair_value);
  SELF\pair_value.applies_to_pair : unconstrained_pair;
  actual_placement                : axis2_placement_3d;
END_ENTITY;
(*
```

Attribute definitions:

SELF\pair_value.applies_to_pair: the **unconstrained_pair** to which the **unconstrained_pair_value** applies.

actual_placement: the placement of the second pair frame with respect to the first pair frame for the **unconstrained_pair**.

5.4.45 fully_constrained_pair

The **fully_constrained_pair** prevents any relative motion between two adjacent links. The two pair frames are always forced to coincide.

NOTES

1 – The **fully_constrained_pair** may be helpful in early design stages or for kinematic analysis systems, because it allows to fix two links each to the other without changing the topological structure.

2 – The concepts of pair value and pair range are not applicable to a **fully_constrained_pair**.

EXPRESS specification:

```
*)
ENTITY fully_constrained_pair
  SUBTYPE OF (kinematic_pair);
END_ENTITY;
(*
```

5.4.46 point_on_surface_pair

The **point_on_surface_pair** constrains the motion of two links such that a point defined on the second link always lies on a surface defined on the first link. The actual location of this point on the surface is called the contact point. The contact point specifies an origin of a frame called the contact frame. The local normal to the surface in the contact point coincides with the z-axis of the contact frame. The x-axis of the contact frame is parallel to the tangent to the iso-parameter line of the surface for the first surface parameter, which passes the contact point, and points into the direction where the first surface parameter increases.

NOTES

1 – Translational motion of the second link is constrained to the two-dimensional parameter space of the surface, while its rotational motion about the contact point is not constrained. Therefore, a **point_on_surface_pair** behaves like a **spherical_pair** that additionally may move along a surface on the first link.

2 – See figure 11.

To support the measure of motion, a pair frame is defined on each of the links. The representation $f_1(u,v)$ is defined relative to the pair frame on the first link. The translational motion is given by the actual surface parameters of the contact point, u_1 and v_1. The three angles of rotation are defined as the yaw, pitch, and roll angles. See 5.3.6. They are required to rotate the x-, y-, and z-axes of the contact frame until its directions coincide with the axis directions of the second pair frame. The origin of the pair frame on the second link coincides with the contact point. In order to migrate from the first pair frame to the second pair frame, the following sequence of transformations has to be performed:

1) Establish the contact frame ${}^{1}\boldsymbol{PF}_c$ relative to the first pair frame:

$$\overline{origin}_c^1 = f_1(u_1, v_1)$$

$$\overline{x\_vector}_c^1 = \left.\frac{\partial f_1(u,v)}{\partial u}\right|_{u=u_1, v=v_1} \quad ; \qquad \overline{x}_c^1 = \frac{\overline{x\_vector}_c^1}{|\overline{x\_vector}_c^1|}$$

$$\overline{w\_vector}_c^1 = \left.\frac{\partial f_1(u,v)}{\partial v}\right|_{u=u_1, v=v_1} \quad ; \qquad \overline{w}_c^1 = \frac{\overline{w\_vector}_c^1}{|\overline{w\_vector}_c^1|}$$

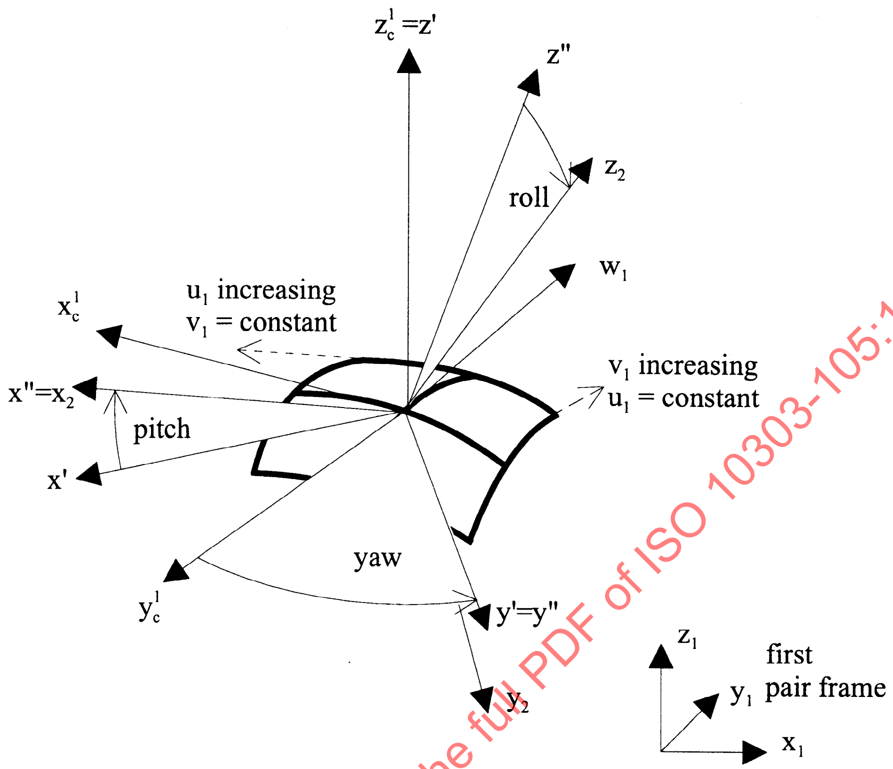


Figure 11 – Example of a point on surface pair

$$\overline{z\_vector}_c^1 = \overline{w}_c^1 \times \overline{x}_c^1 \; ; \qquad \overline{z}_c^1 = \frac{\overline{z\_vector}_c^1}{|\overline{z\_vector}_c^1|}$$

$$\overline{y\_vector}_c^1 = \overline{z}_c^1 \times \overline{x}_c^1 \; ; \qquad \overline{y}_c^1 = \frac{\overline{y\_vector}_c^1}{|\overline{y\_vector}_c^1|}$$

$\overline{origin}_c^1$ and the three direction vectors $\overline{x}_c^1$, $\overline{y}_c^1$, and $\overline{z}_c^1$ may be used to establish the rigid homogeneous matrix representation of the placement of the contact frame ${}^1\boldsymbol{PF}_c$ relative to the first pair frame.

2) Rotate the x_c^1,y_c^1-axes about the z_c^1-axis by the amount specified by the yaw angle. This yields an intermediate frame the axes of which are denoted by $(x', y', z' = z_c^1)$.

3) Rotate the z',x'-axes about the y'-axis by the amount specified by the pitch angle. This yields a second intermediate frame denoted by $(x'', y'' = y', z'')$.

4) Finally, rotate the y'',z''-axes about the x''-axis by the amount specified by the roll angle. The resulting frame $(x_2 = x'', y_2, z_2)$ now coincides with the pair frame on the second

link.

<u>*EXPRESS* specification:</u>

```
*)
ENTITY point_on_surface_pair
  SUBTYPE OF (kinematic_pair);
  pair_surface : surface;
WHERE
  WR1: frame_associated_to_background
         (SELF\kinematic_pair.pair_placement_in_first_link_context,
          pair_surface);
END_ENTITY;
(*
```

<u>Attribute definitions:</u>

pair_surface: the surface on the first link of the pair on which the point on the second link is sliding.

<u>Formal propositions:</u>

WR1: The **pair_surface** of a **point_on_surface_pair** shall be in the **items** set of a **kinematic_frame_background_representation** which, in turn, is related to the **kinematic_link_representation** of the first link of the **point_on_surface_pair** by a **kinematic_frame_background_representation_association**. The **transformer_frame** in the **transformation_operator** of this relationship shall be the **pair_placement_in_first_link_context** of the **point_on_surface_pair**. These requirements are checked by the function **frame_associated_to_background**.

<u>Informal propositions:</u>

continuous surface: The surface used as **pair_surface** shall be smooth in its entire domain, and its tangent vectors shall change their direction only in a steady manner, regardless of whether the surface is composed of multiple surfaces or not.

NOTE 3 – This is equivalent to requiring the **pair_surface** to be G1 continuous throughout and that, in the case of a **rectangular_composite_surface**, the **transition_code** shall be at least **cont_same_gradient** at each segment junction curve. (See ISO 10303-42.)

5.4.47 point_on_surface_pair_value

The **point_on_surface_pair_value** specifies the pair parameters for the **point_on_surface_pair**.

<u>*EXPRESS* specification:</u>

```
*)
ENTITY point_on_surface_pair_value
  SUBTYPE OF (pair_value);
  SELF\pair_value.applies_to_pair : point_on_surface_pair;
  actual_point_on_surface         : point_on_surface;
  input_orientation               : spatial_rotation;
DERIVE
```

```
  actual_orientation              : ypr_rotation
                                  := convert_spatial_to_ypr_rotation
                                     (SELF\pair_value.applies_to_pair,
                                      input_orientation);
WHERE
  WR1: SELF\pair_value.applies_to_pair\point_on_surface_pair.pair_surface :=:
       actual_point_on_surface.basis_surface;
END_ENTITY;
(*
```

Attribute definitions:

SELF\pair_value.applies_to_pair: the **point_on_surface_pair** to which the **point_on_surface_pair_value** applies.

actual_point_on_surface: the positional value for the point on the surface.

input_orientation: the input specification of the rotational pair parameter values from which the **actual_orientation** is derived. **input_orientation** is either a **ypr_rotation** or a **rotation_about_direction**.

actual_orientation: the set of rotational pair parameter values for a **point_on_surface_pair** according to the ypr notation. This is derived from **input_orientation** by the function **convert_spatial_to_ypr_rotation**.

Formal propositions:

WR1: actual_point_on_surface shall be defined as a point on the **pair_surface** of the **point_on_surface_pair** referenced by **applies_to_pair**.

5.4.48 point_on_surface_pair_range

The **point_on_surface_pair_range** specifies the lower bound and the upper bound of the parameter range for the **point_on_surface_pair**.

EXPRESS specification:

```
*)
ENTITY point_on_surface_pair_range
  SUBTYPE OF (simple_pair_range);
  SELF\simple_pair_range.applies_to_pair : point_on_surface_pair;
  range_on_pair_surface                  : rectangular_trimmed_surface;
  lower_limit_yaw                        : rotational_range_measure;
  upper_limit_yaw                        : rotational_range_measure;
  lower_limit_pitch                      : rotational_range_measure;
  upper_limit_pitch                      : rotational_range_measure;
  lower_limit_roll                       : rotational_range_measure;
  upper_limit_roll                       : rotational_range_measure;
WHERE
  WR1: SELF\simple_pair_range.applies_to_pair\point_on_surface_pair.pair_surface
       :=: range_on_pair_surface.basis_surface;
  WR2: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (lower_limit_yaw))
         OR
         ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
          TYPEOF (upper_limit_yaw)))
        XOR
        (lower_limit_yaw < upper_limit_yaw);
   WR3: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
          TYPEOF (lower_limit_pitch))
         OR
         ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
          TYPEOF (upper_limit_pitch)))
        XOR
        (lower_limit_pitch < upper_limit_pitch);
   WR4: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
          TYPEOF (lower_limit_roll))
         OR
         ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
          TYPEOF (upper_limit_roll)))
        XOR
        (lower_limit_roll < upper_limit_roll);
END_ENTITY;
(*
```

Attribute definitions:

SELF\simple_pair_range.applies_to_pair: the **point_on_surface_pair** to which the **point_on_surface_pair_range** applies.

range_on_pair_surface: the admissible range for the positional pair parameter value on the contact surface on the first link.

lower_limit_yaw: the minimum value of the yaw angle for the **point_on_surface_pair**.

upper_limit_yaw: the maximum value of the yaw angle for the **point_on_surface_pair**.

lower_limit_pitch: the minimum value of the pitch angle for the **point_on_surface_pair**.

upper_limit_pitch: the maximum value of the pitch angle for the **point_on_surface_pair**.

lower_limit_roll: the minimum value of the roll angle for the **point_on_surface_pair**.

upper_limit_roll: the maximum value of the roll angle for the **point_on_surface_pair**.

Formal propositions:

WR1: The underlying surface of **range_on_pair_surface** in the range specification shall be identical with the surface **pair_surface** in the **point_on_surface_pair** specification.

WR2: The range of rotation about the z-axis of the **point_on_surface_pair** shall be positive when both bounds are not given as unlimited.

WR3: The range of rotation about the y-axis of the **point_on_surface_pair** shall be positive when both bounds are not given as unlimited.

WR4: The range of rotation about the x-axis of the **point_on_surface_pair** shall be positive when both bounds are not given as unlimited.

5.4.49 surface_pair

A **surface_pair** constrains the motion of two links along a surface on each of the links such that they always have contact in at least one common point. This point lies on a surface of the links. The point is called the contact point. The contact point specifies an origin of a frame called a contact frame. The common tangential plane of the surfaces is the xy-plane of the contact frame. To measure motion, a pair frame is defined on each of the links. The surface representation $f(u,v)$ of each surface is given in the corresponding pair frame. For the subsequent specification both surface representations need to be expressed in a common coordinate system. The world coordinate system is used for this purpose.

The following notation is used:

u_p, v_p	parameters of the surface on the first link;
u_q, v_q	parameters of the surface on the second link;
${}^0f_p(u_p, v_p)$	representation of the surface on the first link relative to the world coordinate system;
${}^0f_q(u_q, v_q)$	representation of the surface on the second link relative to the world coordinate system;
x_p, y_p, z_p	axes of the first pair frame;
x_q, y_q, z_q	axes of the second pair frame;
x_c^p, y_c^p, z_c^p	axes of the contact frame, defined with respect to the first pair frame, but not taking into account the surface orientations and the respective rotation;
x_c^q, y_c^q, z_c^q	axes of the contact frame, defined with respect to the second pair frame;
x_c, y_c, z_c	axes of the contact frame, defined with respect to the first pair frame after the surface orientations and the respective rotation have been taken into account.

NOTE 1 – See figure 12 for an exploded view of a surface pair. There is a rotation angle of $-15°$ between x_c^p and x_c^q, and **orientation** is FALSE.

The following equations apply to **sliding_surface_pair** (see 5.4.52) and **rolling_surface_pair** (see 5.4.54).

The contact frame is determined from both sides of the pair as follows, beginning with the second link:

$$\overline{origin_c^q} = {}^0f_q(u_q, v_q)$$

$$\overline{x\_vector_c^q} = \left.\frac{\partial\, {}^0f_q(u,v)}{\partial u}\right|_{u=u_q, v=v_q} ; \qquad \overline{x_c^q} = \frac{\overline{x\_vector_c^q}}{|\overline{x\_vector_c^q}|}$$

$$\overline{w\_vector_c^q} = \left.\frac{\partial\, {}^0f_q(u,v)}{\partial v}\right|_{u=u_q, v=v_q} ; \qquad \overline{w_c^q} = \frac{\overline{w\_vector_c^q}}{|\overline{w\_vector_c^q}|}$$

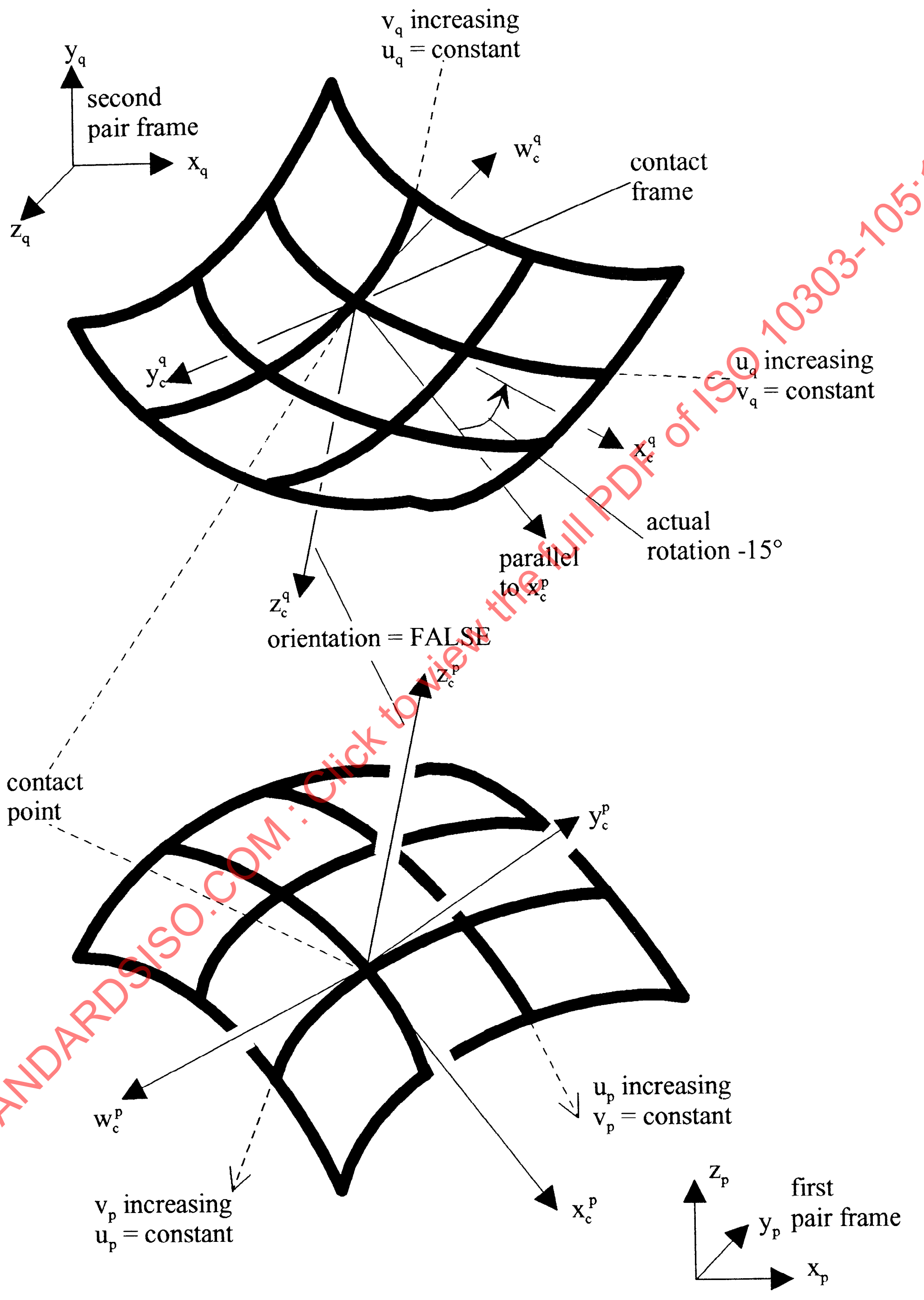


Figure 12 – Example of a surface pair.

$$\overline{z\_vector^q_c} = \overline{w^q_c} \times \overline{x^q_c} \; ; \qquad \overline{z^q_c} = \frac{\overline{z\_vector^q_c}}{|\overline{z\_vector^q_c}|}$$

$$\overline{y\_vector^q_c} = \overline{z^q_c} \times \overline{x^q_c} \; ; \qquad \overline{y^q_c} = \frac{\overline{y\_vector^q_c}}{|\overline{y\_vector^q_c}|}$$

$\overline{origin^q_c}$ and the three direction vectors $\overline{x^q_c}$, $\overline{y^q_c}$, and $\overline{z^q_c}$ may be used to establish the rigid homogeneous matrix representation of the placement of the contact frame ${}^q\boldsymbol{PF}_c$ relative to the second pair frame.

NOTE 2 – See figure 3 for an illustration of the relationships between the frames.

The same analysis is performed for the first link:

$$\overline{origin^p_c} = {}^0f_p(u_p, v_p)$$

$$\overline{x\_vector^p_c} = \left.\frac{\partial\, {}^0f_p(u,v)}{\partial u}\right|_{u=u_p, v=v_p} ; \qquad \overline{x^p_c} = \frac{\overline{x\_vector^p_c}}{|\overline{x\_vector^p_c}|}$$

$$\overline{w\_vector^p_c} = \left.\frac{\partial\, {}^0f_p(u,v)}{\partial v}\right|_{u=u_p, v=v_p} ; \qquad \overline{w^p_c} = \frac{\overline{w\_vector^p_c}}{|\overline{w\_vector^p_c}|}$$

$$\overline{z\_vector^p_c} = \overline{w^p_c} \times \overline{x^p_c} \; ; \qquad \overline{z^p_c} = \frac{\overline{z\_vector^p_c}}{|\overline{z\_vector^p_c}|}$$

$$\overline{y\_vector^p_c} = \overline{z^p_c} \times \overline{x^p_c} \; ; \qquad \overline{y^p_c} = \frac{\overline{y\_vector^p_c}}{|\overline{y\_vector^p_c}|}$$

The contact point shall be identically positioned, relative to the world coordinate system, irrespective of whether it is approached from the second or the first link.

$$\overline{origin_c} = \overline{origin^p_c} = \overline{origin^q_c}$$

Also the z-direction at the contact point shall be identical, relative to the world coordinate system, irrespective of whether it is approached from the second or the first link.

However, the direction vectors $\overline{z^q_c}$ and $\overline{z^p_c}$, as evaluated above, are either

1) $\overline{z_c} = \overline{z^q_c} = \overline{z^p_c}$ or

2) $\overline{z_c} = \overline{z^q_c} = -\overline{z^p_c}$

In case 1, **orientation** is TRUE and both z-directions are identical.

In case 2, **orientation** is FALSE and z_c^p (as well as y_c^p) is rotated by π about the x_c^p direction.

Finally, x_c^p has to be rotated about z_c until it coincides with x_c.

NOTE 3 – The rotation angle to achieve this is defined by the **actual_rotation** attribute of **sliding_surface_pair_value** (see 5.4.52) and of **rolling_surface_pair_value** (see 5.4.54), respectively.

EXPRESS specification:

```
*)
ENTITY surface_pair
  SUBTYPE OF (kinematic_pair);
  surface_1   : surface;
  surface_2   : surface;
  orientation : BOOLEAN;
WHERE
  WR1: frame_associated_to_background
         (SELF\kinematic_pair.pair_placement_in_first_link_context,
          surface_1);
  WR2: frame_associated_to_background
         (SELF\kinematic_pair.pair_placement_in_second_link_context,
          surface_2);
END_ENTITY;
(*
```

Attribute definitions:

surface_1: the contact surface on the first link, defined in the coordinate system specified by **SELF\kinematic_pair.pair_placement_in_first_link_context**.

surface_2: the contact surface on the second link, defined in the coordinate system specified by **SELF\kinematic_pair.pair_placement_in_second_link_context**.

orientation: an indication of whether the z-direction of the second surface agrees with the z-direction of the first surface.

Formal propositions:

WR1: surface_1 of a **surface_pair** shall be in the **items** set of a **kinematic_frame_background_representation** which, in turn, is related to the **kinematic_link_representation** of the first link of the **surface_pair** by a **kinematic_frame_background_representation_association**. The **transformer_frame** in the **transformation_operator** of this relationship shall be the **pair_placement_in_first_link_context** of the **surface_pair**. These requirements are checked by the function **frame_associated_to_background**.

WR2: surface_2 of a **surface_pair** shall be in the **items** set of a **kinematic_frame_background_representation** which, in turn, is related to the **kinematic_link_representation** of the second link of the **surface_pair** by a **kinematic_frame_background_representation_association**. The **transformer_frame** in the **transformation_operator** of this relationship shall be the **pair_placement_in_second_link_context** of the **surface_pair**. These requirements are checked by the function **frame_associated_to_background**.

Informal propositions:

continuous surfaces: The surfaces used as **surface_1** and **surface_2** shall be smooth in their entire domain, and their tangent vectors shall change their direction only in a steady manner, regardless of whether the surfaces are composed of multiple surfaces or not.

NOTE 4 – This is equivalent to requiring **surface_1** and **surface_2** to be G1 continuous throughout and that, in the case of a **rectangular_composite_surface**, the **transition_code** shall be at least **cont_same_gradient** at each segment junction curve. (See ISO 10303-42.)

orientation: If the directions z_q and z_c^p are identical, the **orientation** shall be set TRUE. If the directions z_q and z_c^p are oriented in the opposite direction to each other, the **orientation** shall be set FALSE.

5.4.50 surface_pair_range

The **surface_pair_range** specifies the lower bound and the upper bound of the parameter range for the **surface_pair**.

EXPRESS specification:

```
*)
ENTITY surface_pair_range
  SUBTYPE OF (simple_pair_range);
  SELF\simple_pair_range.applies_to_pair : surface_pair;
  range_on_surface_1                     : rectangular_trimmed_surface;
  range_on_surface_2                     : rectangular_trimmed_surface;
  lower_limit_actual_rotation            : rotational_range_measure;
  upper_limit_actual_rotation            : rotational_range_measure;
WHERE
  WR1: SELF\simple_pair_range.applies_to_pair\surface_pair.surface_1 :=:
       range_on_surface_1.basis_surface;
  WR2: SELF\simple_pair_range.applies_to_pair\surface_pair.surface_2 :=:
       range_on_surface_2.basis_surface;
  WR3: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (lower_limit_actual_rotation))
        OR
        ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (upper_limit_actual_rotation)))
       XOR
       (lower_limit_actual_rotation < upper_limit_actual_rotation);
END_ENTITY;
(*
```

Attribute definitions:

SELF\simple_pair_range.applies_to_pair: the **surface_pair** to which the **surface_pair_range** applies.

range_on_surface_1: the admissible range for the positional pair parameter value on the first contact surface.

range_on_surface_2: the admissible range for the positional pair parameter value on the second contact surface.

lower_limit_actual_rotation: the minimum value of the pair rotation parameter.

upper_limit_actual_rotation: the maximum value of the pair rotation parameter.

Formal propositions:

WR1: The underlying surface of **range_on_surface_1** in the range specification shall be identical with the surface **surface_1** in the **surface_pair** specification.

WR2: The underlying surface of **range_on_surface_2** in the range specification shall be identical with the surface **surface_2** in the **surface_pair** specification.

WR3: The range of rotation of the **surface_pair** shall be positive when both bounds are not given as unlimited.

5.4.51 sliding_surface_pair

A **sliding_surface_pair** specifies the sliding motion between the two contact surfaces of a **surface_pair**. See 5.4.49.

EXPRESS specification:

```
*)
ENTITY sliding_surface_pair
  SUBTYPE OF (surface_pair);
END_ENTITY;
(*
```

5.4.52 sliding_surface_pair_value

The **sliding_surface_pair_value** specifies the set of pair parameters for the **sliding_surface_pair**. See 5.4.49.

EXPRESS specification:

```
*)
ENTITY sliding_surface_pair_value
  SUBTYPE OF (pair_value);
  SELF\pair_value.applies_to_pair : sliding_surface_pair;
  actual_point_on_surface_1       : point_on_surface;
  actual_point_on_surface_2       : point_on_surface;
  actual_rotation                 : plane_angle_measure;
WHERE
  WR1: SELF\pair_value.applies_to_pair\surface_pair.surface_1 :=:
       actual_point_on_surface_1.basis_surface;
  WR2: SELF\pair_value.applies_to_pair\surface_pair.surface_2 :=:
       actual_point_on_surface_2.basis_surface;
END_ENTITY;
(*
```

Attribute definitions:

SELF\pair_value.applies_to_pair: the **sliding_surface_pair** to which the **sliding_surface_-pair_value** applies.

actual_point_on_surface_1: the contact point of the contact surface (**surface_1**) of the first link.

actual_point_on_surface_2: the contact point of the contact surface (**surface_2**) of the second link.

actual_rotation: the angle required to rotate the x_c^p-direction of the first link surface around the z_c-direction of that surface until it coincides with the x_c-direction of the second link surface.

Formal propositions:

WR1: actual_point_on_surface_1 shall be defined as a point on **surface_1** of the **surface_-pair** referenced by **applies_to_pair**.

WR2: actual_point_on_surface_2 shall be defined as a point on **surface_2** of the **surface_-pair** referenced by **applies_to_pair**.

5.4.53 rolling_surface_pair

A **rolling_surface_pair** specifies the rolling motion between the two contact surfaces of a **surface_pair**.

EXPRESS specification:

```
*)
ENTITY rolling_surface_pair
  SUBTYPE OF (surface_pair);
END_ENTITY;
(*
```

5.4.54 rolling_surface_pair_value

The **rolling_surface_pair_value** specifies the set of pair parameters for the **rolling_surface_-pair**. See 5.4.49.

EXPRESS specification:

```
*)
ENTITY rolling_surface_pair_value
  SUBTYPE OF (pair_value);
  SELF\pair_value.applies_to_pair : rolling_surface_pair;
  actual_point_on_surface         : point_on_surface;
  actual_rotation                 : plane_angle_measure;
WHERE
  WR1: SELF\pair_value.applies_to_pair\surface_pair.surface_1 :=:
       actual_point_on_surface.basis_surface;
END_ENTITY;
(*
```

Attribute definitions:

SELF\pair_value.applies_to_pair: the **rolling_surface_pair** to which the **rolling_surface_pair_value** applies.

actual_point_on_surface: the contact point of the contact surface (**surface_1**) of the first link.

actual_rotation: the angle required to rotate the x_c^p-direction of the first link surface around the z_c-direction of that surface until it coincides with the x_c-direction of the second link surface.

Formal propositions:

WR1: actual_point_on_surface shall be defined as a point on **surface_1** of the **surface_pair** referenced by **applies_to_pair**.

5.4.55 point_on_planar_curve_pair

The **point_on_planar_curve_pair** constrains the motion of two links such that a point defined on the second link always lies on a planar curve defined on the first link. The actual location of this point on the curve is called the contact point.

NOTE 1 – Translational motion of the second link is constrained to the one-dimensional parameter space of the curve, while its rotational motion about the contact point is not constrained. Therefore, a **point_on_planar_curve_pair** behaves like a **spherical_pair** that additionally may move along a curve on the first link.

The contact point specifies the origin of a frame called the contact frame. The x-axis of the contact frame is parallel to the tangent to the curve in the contact point and points into the direction where the curve parameter increases. The y-axis of the contact frame is normal to the plane in which the curve is defined. Hence, the z-axis of the contact frame coincides with a local normal to the curve which lies in the plane of the curve and passes the contact point.

NOTE 2 – See figure 13. In this case **orientation** is TRUE, as the y_c^1-axis has the same direction as the z_1-axis.

To support the measure of motion, a pair frame is defined on each of the links. The representation $f(u)$ of the two-dimensional curve is given in the xy-plane of the pair frame on the first link. Therefore, the z-axis of this pair frame always serves as the normal to the plane of the curve.

NOTE 3 – The definition of the curve plane and of its normal as specified here becomes important for cases where the curve is a straight line.

The translational motion is given by the actual curve parameter of the contact point, u_1. The three angles of rotation are defined as the yaw, pitch, and roll angles. See 5.3.6. They are required to rotate the x-, y-, and z-axes of the contact frame until its directions coincide with the axis directions of the second pair frame.

The origin of the pair frame on the second link coincides with the contact point, and the axes of this pair frame agree with the axes of the contact frame when all three rotations have been performed.

The following notation is used in the subsequent specification:

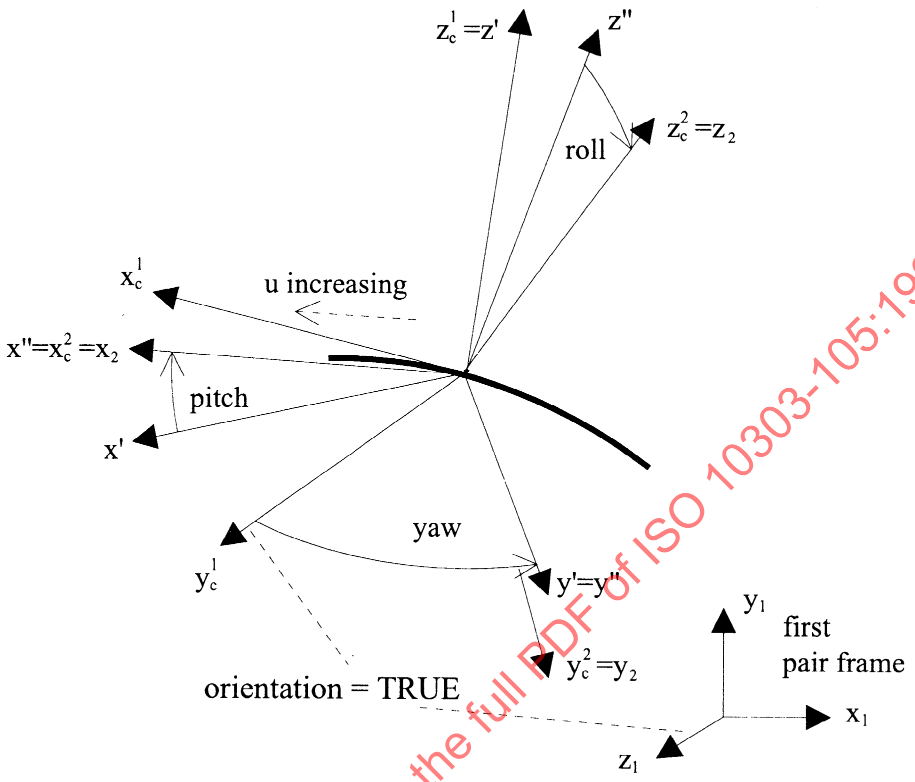


Figure 13 – Example of a point on planar curve pair

u	parameter of the curve on the first link;
u_1	parameter of the contact point on the curve on the first link;
$f(u)$	representation of the curve on the first link relative to the first pair frame;
x_1, y_1, z_1	axes of the first pair frame;
x_2, y_2, z_2	axes of the second pair frame;
x_c^1, y_c^1, z_c^1	axes of the contact frame, defined with respect to the first pair frame, but not taking into account the rotational motion;
x', y', z'	axes of the contact frame, defined with respect to the first pair frame, when the yaw rotation has been taken into account;
x'', y'', z''	axes of the contact frame, defined with respect to the first pair frame, when the yaw and pitch rotations have been taken into account;
x_c^2, y_c^2, z_c^2	axes of the contact frame, defined with respect to the first pair frame, when all the ypr rotations have been taken into account;

In order to migrate from the first pair frame to the second pair frame, the following sequence of transformations has to be performed:

1) Establish the contact frame ${}^1\boldsymbol{PF}_c$ relative to the first pair frame:

$$\overline{origin}_c^1 = f(u_1)$$

$$\overline{x\_vector}^1_c = \left.\frac{df(u)}{du}\right|_{u=u_1} \; ; \qquad \overline{x}^1_c = \frac{\overline{x\_vector}^1_c}{|\overline{x\_vector}^1_c|}$$

y^1_c is required to be parallel to z_1; hence, either

$$\overline{y}^1_c = \overline{z}_1$$

or

$$\overline{y}^1_c = -\overline{z}_1$$

In the first case, **orientation** is TRUE and the direction of y^1_c agrees with that of z_1. In the second case, **orientation** is FALSE and the direction of y^1_c is opposite to that of z_1.

$$\overline{z\_vector}^1_c = \overline{x}^1_c \times \overline{y}^1_c \; ; \qquad \overline{z}^1_c = \frac{\overline{z\_vector}^1_c}{|\overline{z\_vector}^1_c|}$$

$\overline{origin}^1_c$ and the three direction vectors $\overline{x}^1_c$, $\overline{y}^1_c$, and $\overline{z}^1_c$ may be used to establish the rigid homogeneous matrix representation of the placement of the contact frame ${}^1\boldsymbol{PF}_c$ relative to the first pair frame.

2) Rotate the x^1_c,y^1_c-axes about the z^1_c-axis by the amount specified by the yaw angle. This yields an intermediate frame the axes of which are denoted by $(x', y', z' = z^1_c)$.

3) Rotate the z',x'-axes about the y'-axis by the amount specified by the pitch angle. This yields a second intermediate frame denoted by $(x'', y'' = y', z'')$.

4) Finally, rotate the y'',z''-axes about the x''-axis by the amount specified by the roll angle. The resulting frame $(x^2_c = x'', y^2_c, z^2_c)$ now coincides with the pair frame on the second link, denoted by (x_2, y_2, z_2).

EXPRESS specification:

```
*)
ENTITY point_on_planar_curve_pair
  SUBTYPE OF (kinematic_pair);
  pair_curve  : curve;
  orientation : BOOLEAN;
WHERE
  WR1: frame_associated_to_background
         (SELF\kinematic_pair.pair_placement_in_first_link_context,
          pair_curve);
END_ENTITY;
(*
```

Attribute definitions:

pair_curve: the planar curve on the first link of the pair on which the point on the second link is sliding.

orientation: a flag indicating whether the y-axis of the unrotated contact frame agrees with the z-axis of the pair frame on the first link.

Formal propositions:

WR1: The **pair_curve** of a **point_on_planar_curve_pair** shall be in the **items** set of a **kinematic_frame_background_representation** which, in turn, is related to the **kinematic_link_representation** of the first link of the **point_on_planar_curve_pair** by a **kinematic_frame_background_representation_association**. The **transformer_frame** in the **transformation_operator** of this relationship shall be the **pair_placement_in_first_link_context** of the **point_on_planar_curve_pair**. These requirements are checked by the function **frame_associated_to_background**.

Informal propositions:

planar curve: The curve used as **pair_curve** shall be planar.

continuous curve: The curve used as **pair_curve** shall be smooth in its entire domain, and its tangent vector shall change its direction only in a steady manner, regardless of whether the curve is composed of multiple curves or not.

NOTE 4 – This is equivalent to requiring the **pair_curve** to be G1 continuous throughout and that, in the case of a **composite_curve**, the **transition_code** shall be at least **cont_same_gradient** at each segment junction point. (See ISO 10303-42.)

orientation: If the directions z_1 and y_c^1 are identical, the **orientation** shall be TRUE. If the directions z_1 and y_c^1 are oriented in the opposite direction, the **orientation** shall be FALSE.

5.4.56 point_on_planar_curve_pair_value

The **point_on_planar_curve_pair_value** specifies the pair parameters for the **point_on_planar_curve_pair**.

EXPRESS specification:

```
*)
ENTITY point_on_planar_curve_pair_value
  SUBTYPE OF (pair_value);
  SELF\pair_value.applies_to_pair : point_on_planar_curve_pair;
  actual_point_on_curve           : point_on_curve;
  input_orientation               : spatial_rotation;
DERIVE
  actual_orientation              : ypr_rotation
                                  := convert_spatial_to_ypr_rotation
                                     (SELF\pair_value.applies_to_pair,
                                      input_orientation);
WHERE
```

```
  WR1: SELF\pair_value.applies_to_pair\point_on_planar_curve_pair.pair_curve
       :=: actual_point_on_curve.basis_curve;
END_ENTITY;
(*
```

Attribute definitions:

SELF\pair_value.applies_to_pair: the **point_on_planar_curve_pair** to which the **point_on_planar_curve_pair_value** applies.

actual_point_on_curve: the positional value for the point on the curve.

input_orientation: the input specification of the rotational pair parameter values from which the **actual_orientation** is derived. **input_orientation** is either a **ypr_rotation** or a **rotation_about_direction**.

actual_orientation: the array of rotational pair parameter values for a **point_on_planar_curve_pair** according to the ypr notation. This is derived from **input_orientation** by the function **convert_spatial_to_ypr_rotation**.

Formal propositions:

WR1: actual_point_on_curve shall be defined as a point on the **pair_curve** of the **point_on_planar_curve_pair** referenced by **applies_to_pair**.

5.4.57 point_on_planar_curve_pair_range

The **point_on_planar_curve_pair_range** specifies the lower bound and the upper bound of the parameter range for the **point_on_planar_curve_pair**.

EXPRESS specification:

```
*)
ENTITY point_on_planar_curve_pair_range
  SUBTYPE OF (simple_pair_range);
  SELF\simple_pair_range.applies_to_pair : point_on_planar_curve_pair;
  range_on_pair_curve                     : trimmed_curve;
  lower_limit_yaw                         : rotational_range_measure;
  upper_limit_yaw                         : rotational_range_measure;
  lower_limit_pitch                       : rotational_range_measure;
  upper_limit_pitch                       : rotational_range_measure;
  lower_limit_roll                        : rotational_range_measure;
  upper_limit_roll                        : rotational_range_measure;
WHERE
  WR1: SELF\simple_pair_range.applies_to_pair\
            point_on_planar_curve_pair.pair_curve
       :=: range_on_pair_curve.basis_curve;
  WR2: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (lower_limit_yaw))
        OR
        ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (upper_limit_yaw)))
       XOR
```

```
      (lower_limit_yaw < upper_limit_yaw);
  WR3: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
        TYPEOF (lower_limit_pitch))
       OR
       ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
        TYPEOF (upper_limit_pitch)))
      XOR
      (lower_limit_pitch < upper_limit_pitch);
  WR4: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
        TYPEOF (lower_limit_roll))
       OR
       ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
        TYPEOF (upper_limit_roll)))
      XOR
      (lower_limit_roll < upper_limit_roll);
END_ENTITY;
(*
```

Attribute definitions:

SELF\simple_pair_range.applies_to_pair: the **point_on_planar_curve_pair** to which the **point_on_planar_curve_pair_range** applies.

range_on_pair_curve: the admissible range for the positional pair parameter value on the contact curve on the first link.

lower_limit_yaw: the minimum value of the yaw angle for the **point_on_planar_curve_pair**.

upper_limit_yaw: the maximum value of the yaw angle for the **point_on_planar_curve_pair**.

lower_limit_pitch: the minimum value of the pitch angle for the **point_on_planar_curve_pair**.

upper_limit_pitch: the maximum value of the pitch angle for the **point_on_planar_curve_pair**.

lower_limit_roll: the minimum value of the roll angle for the **point_on_planar_curve_pair**.

upper_limit_roll: the maximum value of the roll angle for the **point_on_planar_curve_pair**.

Formal propositions:

WR1: The underlying curve of **range_on_pair_curve** in the range specification shall be identical with the curve **pair_curve** in the **point_on_planar_curve_pair** specification.

WR2: The range of rotation about the z-axis of the **point_on_planar_curve_pair** shall be positive when both bounds are not given as unlimited.

WR3: The range of rotation about the y-axis of the **point_on_planar_curve_pair** shall be positive when both bounds are not given as unlimited.

WR4: The range of rotation about the x-axis of the **point_on_planar_curve_pair** shall be positive when both bounds are not given as unlimited.

5.4.58 planar_curve_pair

A **planar_curve_pair** constrains the motion of two links along a planar curve on each of the links. Both curves lie in a common plane whose normal is called $\overline{z}_c$. The point of contact lies on the curves. The point is called the contact point, and the contact point specifies the origin of a frame called a contact frame. The common plane is the xy-plane of the contact frame. To measure motion, a pair frame is defined on each of the links. The representation $f(u)$ of the two-dimensional curves is given in the xy-plane of the corresponding pair frame. For the subsequent specification both curve representations need to be expressed in a common coordinate system. The world coordinate system is used for this purpose.

The following notation is used:

u_p	parameter of the curve on the first link;
u_q	parameter of the curve on the second link;
${}^0f_p(u_p)$	representation of the curve on the first link relative to the world coordinate system;
${}^0f_q(u_q)$	representation of the curve on the second link relative to the world coordinate system;
x_p, y_p, z_p	axes of the first pair frame;
x_q, y_q, z_q	axes of the second pair frame;
x_c^p, y_c^p, z_c^p	axes of the contact frame, defined with respect to the first pair frame, but not taking into account the curve orientations;
x_c^q, y_c^q, z_c^q	axes of the contact frame, defined with respect to the second pair frame;
x_c, y_c, z_c	axes of the contact frame, defined with respect to the first pair frame after the curve orientations have been taken into account.

NOTE 1 – See figure 14 for an exploded view of a curve pair. In this case **orientation** is FALSE, as the x-axes have opposite directions.

The following equations apply to **sliding_curve_pair** and **rolling_curve_pair**.

The contact frame is determined from both sides of the pair as follows, beginning with the second link:

$$\overline{origin}_c^q = {}^0f_q(u_q)$$

$$\overline{z}_c^q = \overline{z}_c \quad \text{(normal of the common plane)}$$

$$\overline{y\_vector}_c^q = \left.\frac{d\,{}^0f_q(u)}{du}\right|_{u=u_q} ; \qquad \overline{y}_c^q = \frac{\overline{y\_vector}_c^q}{|\overline{y\_vector}_c^q|}$$

$$\overline{x\_vector}_c^q = \overline{y}_c^q \times \overline{z}_c^q ; \qquad \overline{x}_c^q = \frac{\overline{x\_vector}_c^q}{|\overline{x\_vector}_c^q|}$$

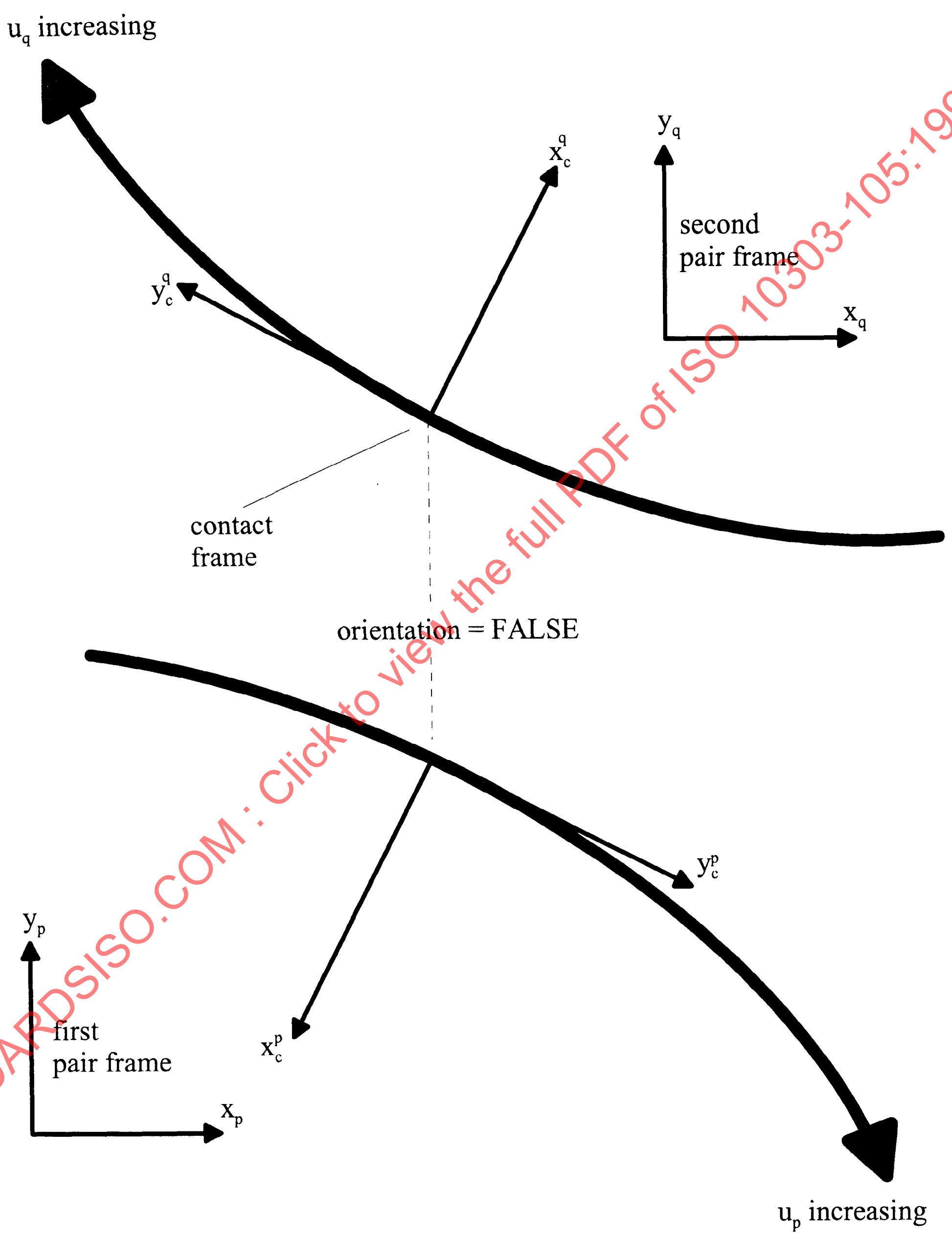


Figure 14 – Example of a planar curve pair.

$\overline{origin}_c^q$ and the three direction vectors $\overline{x}_c^q$, $\overline{y}_c^q$, and $\overline{z}_c^q$ may be used to establish the rigid homogeneous matrix representation of the placement of the contact frame ${}^q\boldsymbol{PF}_c$ relative to the second pair frame.

NOTE 2 – See figure 3 for an illustration of the relationships between the frames.

The same analysis is performed for the first link.

$$\overline{origin}_c^p = {}^0f_p(u_p)$$

$$\overline{z}_c^p = \overline{z}_c \quad \text{(normal of the common plane)}$$

$$\overline{y\_vector}_c^p = \left.\frac{d\,{}^0f_p(u)}{du}\right|_{u=u_p} ; \qquad \overline{y}_c^p = \frac{\overline{y\_vector}_c^p}{|\overline{y\_vector}_c^p|}$$

$$\overline{x\_vector}_c^p = \overline{y}_c^p \times \overline{z}_c^p ; \qquad \overline{x}_c^p = \frac{\overline{x\_vector}_c^p}{|\overline{x\_vector}_c^p|}$$

The contact point shall be identically positioned, relative to the world coordinate system, irrespective of whether it is approached from the second or the first link.

$$\overline{origin}_c = \overline{origin}_c^p = \overline{origin}_c^q$$

The z-directions are all normal to the same plane; hence

$$\overline{z}_c = \overline{z}_c^p = \overline{z}_c^q$$

At this stage, the direction vectors $\overline{x}_c^q$ and $\overline{x}_c^p$ are either

1) $\overline{x}_c = \overline{x}_c^q = \overline{x}_c^p$ or

2) $\overline{x}_c = \overline{x}_c^q = -\overline{x}_c^p$

In case 1, **orientation** is TRUE and both x-directions are identical.

In case 2, **orientation** is FALSE and x_c^p (as well as y_c^p) is rotated by π about the z_c direction.

EXPRESS specification:

```
*)
ENTITY planar_curve_pair
  SUBTYPE OF (kinematic_pair);
  curve_1     : curve;
  curve_2     : curve;
  orientation : BOOLEAN;
WHERE
  WR1: frame_associated_to_background
         (SELF\kinematic_pair.pair_placement_in_first_link_context,
          curve_1);
  WR2: frame_associated_to_background
         (SELF\kinematic_pair.pair_placement_in_second_link_context,
          curve_2);
END_ENTITY;
(*
```

Attribute definitions:

curve_1: the contact curve on the first link, defined in the coordinate system specified by **SELF\kinematic_pair.pair_placement_in_first_link_context**. This curve shall lie within a plane.

curve_2: the contact curve on the second link, defined in the coordinate system specified by **SELF\kinematic_pair.pair_placement_in_second_link_context**. This curve shall lie within a plane.

orientation: a flag indicating whether the x-axis-directions agree.

Formal propositions:

WR1: curve_1 of a **planar_curve_pair** shall be in the **items** set of a **kinematic_frame_background_representation** which, in turn, is related to the **kinematic_link_representation** of the first link of the **planar_curve_pair** by a **kinematic_frame_background_representation_association**. The **transformer_frame** in the **transformation_operator** of this relationship shall be the **pair_placement_in_first_link_context** of the **planar_curve_pair**. These requirements are checked by the function **frame_associated_to_background**.

WR2: curve_2 of a **planar_curve_pair** shall be in the **items** set of a **kinematic_frame_background_representation** which, in turn, is related to the **kinematic_link_representation** of the second link of the **planar_curve_pair** by a **kinematic_frame_background_representation_association**. The **transformer_frame** in the **transformation_operator** of this relationship shall be the **pair_placement_in_second_link_context** of the **planar_curve_pair**. These requirements are checked by the function **frame_associated_to_background**.

Informal propositions:

planar curves: The curves used as **curve_1** and **curve_2** shall be planar curves.

continuous curves: The curves used as **curve_1** and **curve_2** shall be smooth in their entire domain, and their tangent vectors shall change their direction only in a steady manner, regardless of whether the curves are composite or not.

NOTE 3 – This is equivalent to requiring **curve_1** and **curve_2** to be G1 continuous throughout and that, in the case of a **composite_curve**, the **transition_code** shall be at least **cont_same_gradient** at each segment junction point. (See ISO 10303-42.)

orientation: If the directions x_q and x_c^p are identical, the **orientation** shall be TRUE. If the directions x_q and x_c^p are oriented in the opposite direction, the **orientation** shall be FALSE.

5.4.59 planar_curve_pair_range

The **planar_curve_pair_range** specifies the lower bound and the upper bound of the parameter range of the **planar_curve_pair**.

EXPRESS specification:

```
*)
ENTITY planar_curve_pair_range
  SUBTYPE OF (simple_pair_range);
  SELF\simple_pair_range.applies_to_pair : planar_curve_pair;
  range_on_curve_1                       : trimmed_curve;
  range_on_curve_2                       : trimmed_curve;
WHERE
  WR1: SELF\simple_pair_range.applies_to_pair\planar_curve_pair.curve_1 :=:
       range_on_curve_1.basis_curve;
  WR2: SELF\simple_pair_range.applies_to_pair\planar_curve_pair.curve_2 :=:
       range_on_curve_2.basis_curve;
END_ENTITY;
(*
```

Attribute definitions:

SELF\simple_pair_range.applies_to_pair: the **planar_curve_pair** to which the **planar_-curve_pair_range** applies.

range_on_curve_1: the admissible range for the positional pair parameter value on the curve on the first link of the pair.

range_on_curve_2: the admissible range for the positional pair parameter value on the curve on the second link of the pair.

Formal propositions:

WR1: The underlying curve of **range_on_curve_1** in the range specification shall be identical with the curve **curve_1** in the **planar_curve_pair** specification.

WR2: The underlying curve of **range_on_curve_2** in the range specification shall be identical with the curve **curve_2** in the **planar_curve_pair** specification.

5.4.60 sliding_curve_pair

A **sliding_curve_pair** specifies the sliding motion between the curves on the two links of a **planar_curve_pair**.

EXPRESS specification:

```
*)
ENTITY sliding_curve_pair
  SUBTYPE OF (planar_curve_pair);
END_ENTITY;
(*
```

5.4.61 sliding_curve_pair_value

The **sliding_curve_pair_value** specifies the pair parameters for the **sliding_curve_pair**.

EXPRESS specification:

```
*)
ENTITY sliding_curve_pair_value
  SUBTYPE OF (pair_value);
  SELF\pair_value.applies_to_pair : sliding_curve_pair;
  actual_point_on_curve_1         : point_on_curve;
  actual_point_on_curve_2         : point_on_curve;
WHERE
  WR1: SELF\pair_value.applies_to_pair\planar_curve_pair.curve_1 :=:
       actual_point_on_curve_1.basis_curve;
  WR2: SELF\pair_value.applies_to_pair\planar_curve_pair.curve_2 :=:
       actual_point_on_curve_2.basis_curve;
END_ENTITY;
(*
```

Attribute definitions:

SELF\pair_value.applies_to_pair: the **sliding_curve_pair** to which the **sliding_curve_pair_value** applies.

actual_point_on_curve_1: the contact point, defined in the parameter space of the curve on the first link.

actual_point_on_curve_2: the contact point, defined in the parameter space of the curve on the second link.

Formal propositions:

WR1: actual_point_on_curve_1 shall be defined as a point on **curve_1** of the **planar_curve_pair** referenced by **applies_to_pair**.

WR2: actual_point_on_curve_2 shall be defined as a point on **curve_2** of the **planar_curve_pair** referenced by **applies_to_pair**.

5.4.62 rolling_curve_pair

A **rolling_curve_pair** specifies the rolling motion between the curves on the two links of a **planar_curve_pair**.

EXPRESS specification:

```
*)
ENTITY rolling_curve_pair
  SUBTYPE OF (planar_curve_pair);
END_ENTITY;
(*
```

5.4.63 rolling_curve_pair_value

The **rolling_curve_pair_value** specifies the pair parameter for the **rolling_curve_pair**.

EXPRESS specification:

```
*)
ENTITY rolling_curve_pair_value
  SUBTYPE OF (pair_value);
  SELF\pair_value.applies_to_pair : rolling_curve_pair;
  actual_point_on_curve_1         : point_on_curve;
WHERE
  WR1: SELF\pair_value.applies_to_pair\planar_curve_pair.curve_1 :=:
       actual_point_on_curve_1.basis_curve;
END_ENTITY;
(*
```

Attribute definitions:

SELF\pair_value.applies_to_pair: the **rolling_curve_pair** to which the **rolling_curve_-pair_value** applies.

actual_point_on_curve_1: the contact point, defined in the parameter space of the curve on the first link.

Formal propositions:

WR1: actual_point_on_curve_1 shall be defined as a point on **curve_1** of the **planar_-curve_pair** referenced by **applies_to_pair**.

5.4.64 gear_pair

The **gear_pair** constrains the motion between two adjacent links to a rolling (and potentially sliding) motion of the second link along the first link.

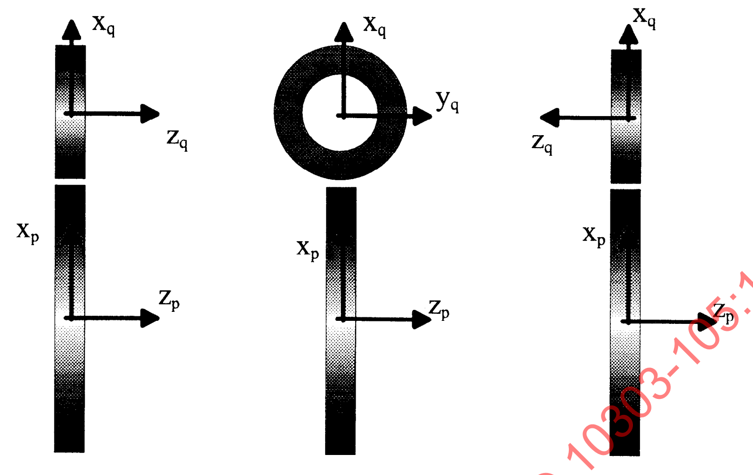
To measure the motion of the second link with respect to the first link, a frame is defined on each of the links such that the origin lies in the center of the rolling circle of the respective link.

NOTE 1 – See figure 15, where different values of the constants "bevel", "helical_angle", and "gear_ratio" are shown. The lower gear always represents the first link; the upper gear is the second link.

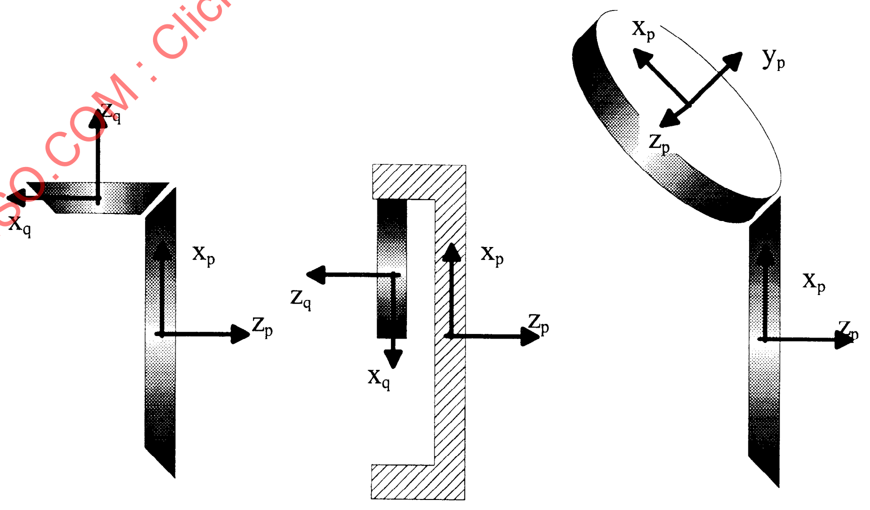
The direction of the z-axis is perpendicular to the plane of the circle. In a reference situation (with both rotation values being zero) the x-axis of the first link points from the origin towards the point of contact with the second link, while on the second link the x-axis points away from the point of contact.

Motion is measured in terms of the rotation of the contact point about the z-axis of the first link. The motion of the second link results from an analysis of the rolling process. At any instance the motion of the second link may be considered as a rotation of the second gear axis about the first gear axis while maintaining the distance (sum of the radii), the bevel, and the helical angle plus a rotation of the second gear about its own axis according to:

$$actual\_rotation\_2 = -gear\_ratio \cdot actual\_rotation\_1$$



bevel =	0	0	0
helical angle =	0	45°	90°
gear ratio =	2	2	2



bevel =	90°	180°	45°
helical angle =	0	0	30°
gear ratio =	2	2	1

Figure 15 – Examples of gear pairs in different situations.

Thus absolute bevel values between 0 and $\frac{\pi}{2}$ represent external gears while absolute bevel values between $\frac{\pi}{2}$ and π represent internal gears.

The angle between the two x-axis directions is given by the bevel according to

$$\overline{x}_p \cdot \overline{x}_q = \cos(bevel)$$

with $bevel \in]-\pi, \pi]$,

where in case of $\cos(bevel) = 0$, the sign of the bevel is defined according to:

$$\overline{y}_p \cdot (\overline{x}_p \times \overline{x}_q) = sign(bevel)$$

The angle between the two y-axis directions is given by the helical angle according to:

$$\overline{y}_p \cdot \overline{y}_q = \cos(helical\_angle)$$

with $helical\_angle \in]-\pi, \pi]$,

where in case of $\cos(helical\_angle) = 0$, the sign of the helical angle is defined according to:

$$\overline{x}_q \cdot (\overline{y}_p \times \overline{y}_q) = sign(helical\_angle)$$

NOTE 2 – In some cases the gear ratio depends on the radii of the rolling circles and also on the characteristics of the gear toothing such as in worm gears. For this reason the gear ratio is defined and not derived from the ratio of the two radii. Hence, the two radii are used to determine the geometrical arrangement of the two gears relative to each other in conjunction with bevel and helical angle, while the gear ratio determines the ratio of the second gear motion with respect to the rotation of the contact point about the first gear axis.

One way to construct the geometrical arrangement of the second gear relative to the first one is as follows:

An auxiliary frame is introduced at the point of contact with its axes x_a, y_a, and z_a all parallel to the respective axes of the first link frame. Now the auxiliary frame is rotated by the angle bevel about its y_a-axis followed by a rotation by the helical angle about the (previously rotated) x_a-axis. Now the x_a-y_a-plane represents the plane in which the second gear lies.

EXPRESS specification:

```
*)
ENTITY gear_pair
  SUBTYPE OF (kinematic_pair);
  radius_first_link  : length_measure;
  radius_second_link : length_measure;
  bevel              : plane_angle_measure;
  helical_angle      : plane_angle_measure;
  gear_ratio         : REAL;
END_ENTITY;
(*
```

Attribute definitions:

radius_first_link: the radius of the rolling circle of the first link.

radius_second_link: the radius of the rolling circle of the second link.

bevel: the angle between the two x-axes.

helical_angle: the angle between the two y-axes.

gear_ratio: the gear ratio of the pair.

5.4.65 gear_pair_value

The **gear_pair_value** specifies the pair parameters for the **gear_pair**.

NOTE – If the gear pair is part of a more complex gear mechanism, the gear pair values as defined here may not be the primary design targets. In the case of a planetary gear mechanism, for example, it is more important to know the rotation angles of the planetary gears with respect to an inertial coordinate system than the rotation angles between the planetary gears and the frames which support them. See [1] for a detailed description of this specific example.

For a given configuration of pairs within a mechanism, the more global motion parameters can be derived, in general, from the individual pair parameters and vice versa. This is not possible, however, generically without knowledge of the configuration. Therefore, the specification of functions which provide such derivations is left to an application protocol for which these derivations are required.

EXPRESS specification:

```
*)
ENTITY gear_pair_value
  SUBTYPE OF (pair_value);
  SELF\pair_value.applies_to_pair : gear_pair;
  actual_rotation_1               : plane_angle_measure;
DERIVE
  actual_rotation_2               : plane_angle_measure
                                  := - actual_rotation_1 *
                                     SELF\pair_value.applies_to_pair\
                                     gear_pair.gear_ratio;
END_ENTITY;
(*
```

Attribute definitions:

SELF\pair_value.applies_to_pair: the **gear_pair** to which the **gear_pair_value** applies.

actual_rotation_1: the value of the pair parameter of the first link.

actual_rotation_2: the value of the pair parameter of the second link.

5.4.66 gear_pair_range

The **gear_pair_range** specifies the lower bound and the upper bound of the parameter range of the **gear_pair**.

EXPRESS specification:

```
*)
ENTITY gear_pair_range
  SUBTYPE OF (simple_pair_range);
  SELF\simple_pair_range.applies_to_pair : gear_pair;
  lower_limit_actual_rotation_1          : rotational_range_measure;
  upper_limit_actual_rotation_1          : rotational_range_measure;
WHERE
  WR1: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (lower_limit_actual_rotation_1))
        OR
        ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (upper_limit_actual_rotation_1)))
       XOR
       (lower_limit_actual_rotation_1 <
        upper_limit_actual_rotation_1);
END_ENTITY;
(*
```

Attribute definitions:

SELF\simple_pair_range.applies_to_pair: the **gear_pair** to which the **gear_pair_range** applies.

lower_limit_actual_rotation_1: the minimum value of the pair parameter of the first link.

upper_limit_actual_rotation_1: the maximum value of the pair parameter of the first link.

Formal propositions:

WR1: The range of rotation of the **gear_pair** shall be positive when both bounds are not given as unlimited.

5.4.67 rack_and_pinion_pair

The **rack_and_pinion_pair** describes a pinion rolling on a rack where the rotation axis of the pinion is perpendicular to the direction of the rack.

NOTE – See figure 16. The rack and pinion pair may be considered as a special case of the **rolling_curve_pair** with one curve being a straight line and the other a circle.

The rack is always the first link; the pinion is the second link. The **rack_and_pinion_pair** constrains the motion between two adjacent links to a translation along a common axis and a rotation about an axis perpendicular to it.

To measure the motion of the second link with respect to the first link, a frame is defined on each of the links. On the pinion, the origin is the center of the rolling circle. The direction of the z-axis on both rack and pinion is perpendicular to the plane of the circle.

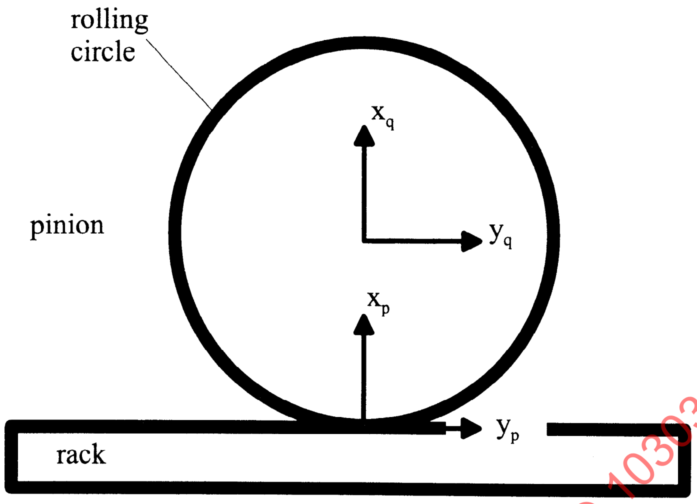


Figure 16 – Example of a rack and pinion pair in the reference situation

In a reference situation (with both pair parameter values being zero), the x-axis of the pinion points from the origin away from the point of contact with the rack. In the same reference situation, the x-axis of the rack points from the point of contact towards the center of the pinion. Motion is measured in terms of the translation of the contact point along the y-axis of the rack (first link). The motion of the second link results from an analysis of the rolling process. At any instance the motion of the second link is composed of a translation and a rotation where the rotation part is calculated from:

$$actual\_rotation = \frac{actual\_displacement}{pinion\_radius}$$

EXPRESS specification:

```
*)
ENTITY rack_and_pinion_pair
  SUBTYPE OF (kinematic_pair);
  pinion_radius : length_measure;
END_ENTITY;
(*
```

Attribute definitions:

pinion_radius: the radius of the rolling circle of the pinion.

5.4.68 rack_and_pinion_pair_value

The **rack_and_pinion_pair_value** specifies the pair parameters for the **rack_and_pinion_pair**.

EXPRESS specification:

```
*)
ENTITY rack_and_pinion_pair_value
  SUBTYPE OF (pair_value);
  SELF\pair_value.applies_to_pair : rack_and_pinion_pair;
  actual_displacement             : length_measure;
DERIVE
  actual_rotation : plane_angle_measure
                 := convert_plane_angle_for_pair_from_radian
                    (SELF\pair_value.applies_to_pair\kinematic_pair,
                     (- actual_displacement /
                      SELF\pair_value.applies_to_pair\
                      rack_and_pinion_pair.pinion_radius));
END_ENTITY;
(*
```

Attribute definitions:

SELF\pair_value.applies_to_pair: the **rack_and_pinion_pair** to which the **rack_and_pinion_pair_value** applies.

actual_displacement: the value of the pair parameter for the rack.

actual_rotation: the resulting rotation of the pinion.

5.4.69 rack_and_pinion_pair_range

The **rack_and_pinion_pair_range** specifies the lower bound and the upper bound of the parameter range for the **rack_and_pinion_pair**.

EXPRESS specification:

```
*)
ENTITY rack_and_pinion_pair_range
  SUBTYPE OF (simple_pair_range);
  SELF\simple_pair_range.applies_to_pair : rack_and_pinion_pair;
  lower_limit_rack_displacement          : translational_range_measure;
  upper_limit_rack_displacement          : translational_range_measure;
WHERE
  WR1: (('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (lower_limit_rack_displacement))
        OR
        ('KINEMATIC_STRUCTURE_SCHEMA.UNLIMITED_RANGE' IN
         TYPEOF (upper_limit_rack_displacement)))
       XOR
       (lower_limit_rack_displacement < upper_limit_rack_displacement);
END_ENTITY;
(*
```

Attribute definitions:

SELF\simple_pair_range.applies_to_pair: the **rack_and_pinion_pair** to which the **rack_and_pinion_pair_range** applies.

lower_limit_rack_displacement: the minimum value of the pair parameter for the rack.

upper_limit_rack_displacement: the maximum value of the pair parameter for the rack.

Formal propositions:

WR1: The range of motion of the **rack_and_pinion_pair** shall be positive when both bounds are not given as unlimited.

5.4.70 kinematic_substructure

A **kinematic_substructure** specifies the topological characteristics of a mechanism in terms of tree or network structures.

NOTES

1 – Using graph theory, the topology of a mechanism is represented by a directed graph, where the links correspond to the nodes, and the joints correspond to the edges. This graph can be derived from the definition of the links and joints. If the topology of a mechanism is defined only by this minimal information, it is referred to as "lower level topology" within this part of ISO 10303.

Graph theory provides a means to capture connectivity information which may be applied to the describe the topological structure of mechanisms. The geometrical and mechanical properties are not conveyed. If the full topological information including connectivity information is provided explicitly, this is referred to as "higher level topology" within this part of ISO 10303.

A general graph may be composed of two basic types of subgraphs:

— Trees;

— Networks.

If, in addition to the lower level of topology, the higher level topology is provided in a kinematic model, this higher level of topology is represented by kinematic substructures. These kinematic substructures are:

— a kinematic tree structure, whose graph representation is a single oriented tree and which consists only of oriented kinematic joints connecting the kinematic links;

— a kinematic network structure, whose graph representation consists of oriented kinematic loops.

2 – See [3] for methods that can be applied to analyze a **kinematic_substructure**.

EXPRESS specification:

```
*)
ENTITY kinematic_substructure
  SUPERTYPE OF (ONEOF (kinematic_tree_structure,
                       kinematic_network_structure));
  parent_structure : kinematic_structure;
END_ENTITY;
(*
```

Attribute definitions:

parent_structure: the **kinematic_structure** of which the **kinematic_substructure** forms a component.

5.4.71 kinematic_network_structure

The **kinematic_network_structure** represents a subgraph of network type; i.e., the mechanism is composed of closed kinematic chains.

NOTES

1 – See figure 17 for an illustration of a kinematic network structure.

2 – This entity is present in an instance of a kinematics model only if the kinematic structure takes the form of a directed graph.

EXPRESS specification:

```
*)
ENTITY kinematic_network_structure
  SUBTYPE OF (kinematic_substructure);
END_ENTITY;
(*
```

5.4.72 kinematic_tree_structure

The **kinematic_tree_structure** represents a subgraph of tree type; i.e., the mechanism is composed of open kinematic chains.

NOTES

1 – See figure 18 for an illustration of a kinematic tree structure.

2 – This entity is present in an instance of a kinematics model only if the kinematic structure takes the form of a directed graph.

EXPRESS specification:

```
*)
ENTITY kinematic_tree_structure
  SUBTYPE OF (kinematic_substructure);
END_ENTITY;
(*
```

5.4.73 kinematic_loop

The **kinematic_loop** represents a cycle in the subgraph. It is the representation of a closed kinematic chain in a mechanism, where the last link in the chain is again connected to the first link by a joint. The **kinematic_loop** is analogous to the loop in the topology model, where the joints correspond to the edges and the links correspond to the vertices of the topology model. The loop shall be pointed to by at least one **joint_logical_relationship**.

Every **kinematic_joint** shall be oriented such that the list of **kinematic_joint**s have a common orientation within the loop.

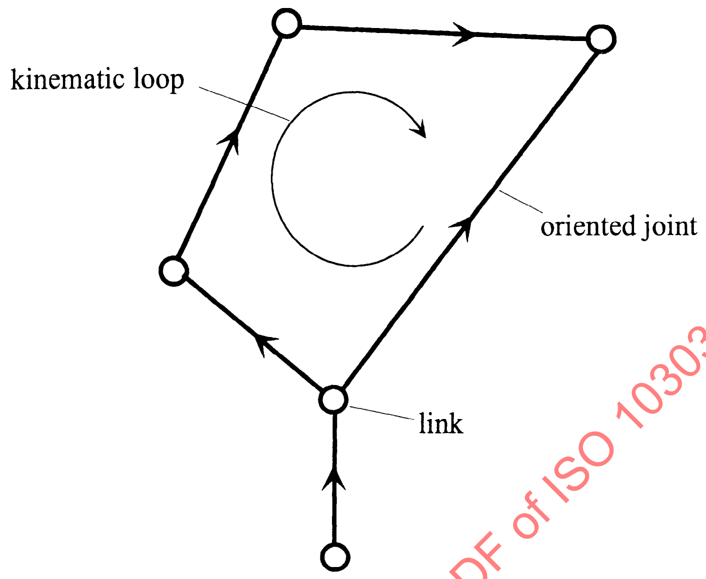


Figure 17 – Example of a kinematic network structure

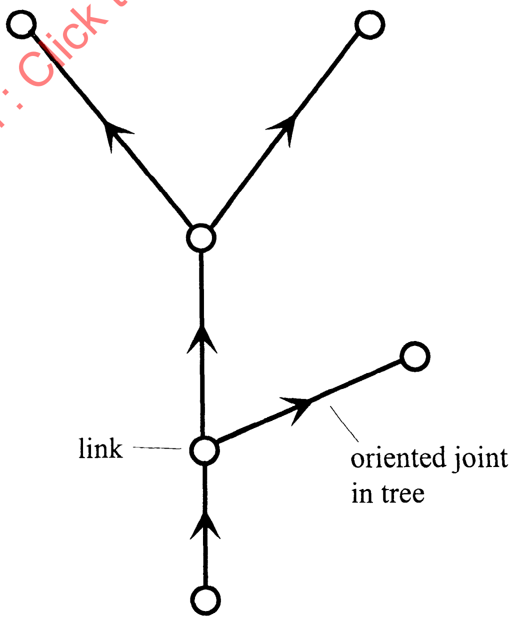


Figure 18 – Example of a kinematic tree structure

NOTE – This entity is present in an instance of a kinematics model only if the kinematic structure takes the form of a directed graph.

EXPRESS specification:

```
*)
ENTITY kinematic_loop;
  network : kinematic_network_structure;
WHERE
  WR1 : SIZEOF(USEDIN(SELF,
        'kinematic_structure_schema.joint_logical_relationship.loop'))>0;
  WR2 : SIZEOF (QUERY( relation_1 <* USEDIN(SELF,
         'kinematic_structure_schema.joint_logical_relationship.loop') |
          SIZEOF(QUERY( relation_2 <* (USEDIN (SELF,
           'kinematic_structure_schema.joint_logical_relationship.loop')
             - relation_1) |
               NOT(connected_in_simple_loop (relation_1,relation_2))))>0
                ))=0;
END_ENTITY;
(*
```

Attribute definitions:

network: the network structure that contains the loop.

Formal propositions:

WR1: The loop shall consist of at least one **joint_logical_relationship**, i.e., of at least two joints.

WR2: Every **oriented_joint** in a **kinematic_loop** shall be connected to every other **oriented_joint** in that kinematic loop, either directly through a single **joint_logical_relationship** entity, or indirectly through a number of **joint_logical_relationship** entities. This ensures that a **kinematic_loop** consists of exactly one loop.

5.4.74 joint_logical_relationship

The **joint_logical_relationship** is used to relate two **oriented_joints** that form part of a loop.

NOTE – This entity is present in an instance of a kinematics model only if the kinematic structure takes the form of a directed graph.

EXPRESS specification:

```
*)
ENTITY joint_logical_relationship;
  loop                          : kinematic_loop;
  previous_joint_logical_structure : oriented_joint;
  next_joint_logical_structure     : oriented_joint;
UNIQUE
  UR1: loop, previous_joint_logical_structure;
  UR2: loop, next_joint_logical_structure;
WHERE
  WR1 : previous_joint_logical_structure.exit_link :=:
        next_joint_logical_structure.advent_link;
```

```
END_ENTITY;
(*
```

Attribute definitions:

loop: the **kinematic_loop** of which the **previous_joint_logical_structure** and **next_joint_-logical_structure** form components.

previous_joint_logical_structure: the first of two adjacent **oriented_joints** in the loop.

next_joint_logical_structure: the second of two adjacent **oriented_joints** in the loop.

Formal propositions:

UR1: An **oriented_joint** shall be used by a **kinematic_loop** only once in the role of **previous_joint_logical_structure**.

UR2: An **oriented_joint** shall be used by a **kinematic_loop** only once in the role of **next_-joint_logical_structure**.

WR1: The exit link of the previous **oriented_joint** shall be the advent link of the next **oriented_joint**.

5.4.75 oriented_joint

The **oriented_joint** is used to indicate the direction of traversal of a joint.

NOTE – This entity is present in an instance of a kinematics model only if the kinematic structure takes the form of a directed graph.

EXPRESS specification:

```
*)
ENTITY oriented_joint;
  joint       : kinematic_joint;
  orientation : BOOLEAN;
DERIVE
  advent_link : kinematic_link
              := assign_directed_link (joint, orientation);
  exit_link   : kinematic_link
              := assign_directed_link (joint, NOT (orientation));
END_ENTITY;
(*
```

Attribute definitions:

joint: the joint whose **orientation** is to be indicated.

orientation: the direction of traversal of a joint. If **orientation** is true, the joint shall be entered from the first link and left via the second link of the joint. If **orientation** is false, the direction of traversal shall be opposite.

advent_link: the link entering the joint. This is either the first link or the second link, depending on **orientation**, which is evaluated by the function **assign_directed_link**.

exit_link: the link leaving the joint. This is either the second link or the first link, depending on **orientation**, which is evaluated by the function **assign_directed_link**.

5.4.76 oriented_joint_in_tree

The **oriented_joint_in_tree** is used to identify the **kinematic_tree_structure** to which the **oriented_joint** belongs.

NOTE – This entity is present in an instance of a kinematics model only if the kinematic structure takes the form of a directed graph.

EXPRESS specification:

```
*)
ENTITY oriented_joint_in_tree
  SUBTYPE OF (oriented_joint);
  parent_structure : kinematic_tree_structure;
END_ENTITY;
(*
```

Attribute definitions:

parent_structure: the **kinematic_tree_structure** of which the **oriented_joint** is a constituent.

5.4.77 advent_oriented_joint

An **advent_oriented_joint** indicates that the role of an **oriented_joint** is either an exit from one substructure, or an advent to the next. **advent_oriented_joint** distinguishes which **oriented_joints** are within a **kinematic_substructure** and which **oriented_joints** connect substructures.

NOTE – This entity is present in an instance of a kinematics model only if the kinematic structure takes the form of a directed graph.

EXPRESS specification:

```
*)
ENTITY advent_oriented_joint
  SUBTYPE OF (oriented_joint);
END_ENTITY;
(*
```

5.5 kinematic_structure_schema function definitions

5.5.1 ypr_index

The **ypr_index** function establishes an ordered sequence of angles of rotation for a linear transformation. The function returns 1 if the angle of rotation is of type **yaw**. The function returns 2 if the angle of rotation is of type **pitch**. The function returns 3 if the angle of rotation is of type **roll**. The type of the function is INTEGER.

EXPRESS specification:

```
*)
FUNCTION ypr_index (ypr : ypr_enumeration): INTEGER;
   CASE ypr OF
     yaw    : RETURN (1);
     pitch  : RETURN (2);
     roll   : RETURN (3);
   END_CASE;
END_FUNCTION;
(*
```

Argument definitions:

ypr: one axis of rotation as specified by the **ypr_enumeration** (see 5.3.6).

5.5.2 representation_of_link

This function returns the **kinematic_link_representation** that is related to **link** via a **kinematic_link_representation_relation**. If there is no such relation, an indeterminate value (?) is returned.

EXPRESS specification:

```
*)
FUNCTION representation_of_link (link : kinematic_link)
     : kinematic_link_representation;
  LOCAL
    link_rep_rel : BAG OF kinematic_link_representation_relation;
  END_LOCAL;

  link_rep_rel := USEDIN (link, 'KINEMATIC_STRUCTURE_SCHEMA.'+
                                'KINEMATIC_LINK_REPRESENTATION_RELATION.'+
                                'TOPOLOGICAL_ASPECTS');

  IF (SIZEOF (link_rep_rel) = 0) THEN
    RETURN (?);
  ELSE
    RETURN (link_rep_rel[1].geometric_aspects);
  END_IF;
END_FUNCTION;
(*
```

Argument definitions:

link: the **kinematic_link** for which the related **kinematic_link_representation** is wanted.

5.5.3 suitably_based_mechanism

This function checks whether the **base** of a **mechanism** is related to the context of the **kinematic_ground_representation** which is associated with the **containing_property** of the **mechanism**. If the **mechanism_base_placement** identified by **mbp** relates the **base** of **mech** to the corresponding context, directly or indirectly, in an acyclic way, the function returns TRUE. The function returns FALSE otherwise. The type of the function is BOOLEAN.

EXPRESS specification:

```
*)
FUNCTION suitably_based_mechanism (mbp  : mechanism_base_placement;
                                   mech : mechanism) : BOOLEAN;
  LOCAL
    kprop  : kinematic_property_definition;
    kgrep  : kinematic_ground_representation;
    klrep  : kinematic_link_representation;
    klnk   : kinematic_link;
    kjnts  : BAG OF kinematic_joint;
    nmechs : BAG OF mechanism;
    nmbps  : BAG OF mechanism_base_placement;
  END_LOCAL;

  kprop := mech.containing_property;

  IF ('KINEMATIC_STRUCTURE_SCHEMA.KINEMATIC_GROUND_REPRESENTATION' IN
      TYPEOF (mbp\representation_relationship.rep_1)) THEN
    kgrep := mbp\representation_relationship.rep_1;

    IF (kgrep.property\property_definition_representation.definition :=:
        kprop) THEN
      RETURN (TRUE);
    ELSE
      RETURN (FALSE);
    END_IF;
  ELSE
    klrep  := mbp\representation_relationship.rep_1;
    klnk   := klrep.link_representation_relation.topological_aspects;
    kjnts  := USEDIN (klnk,
                'KINEMATIC_STRUCTURE_SCHEMA.KINEMATIC_JOINT.FIRST_LINK') +
              USEDIN (klnk,
                'KINEMATIC_STRUCTURE_SCHEMA.KINEMATIC_JOINT.SECOND_LINK');
    nmechs := USEDIN (kjnts[1].structure,
                'KINEMATIC_STRUCTURE_SCHEMA.MECHANISM.STRUCTURE_DEFINITION');

    IF (nmechs[1] :=: mech) THEN
      RETURN (FALSE);
    ELSE
      IF (nmechs[1].containing_property :<>: kprop) THEN
        RETURN (FALSE);
      ELSE
        nmbps := USEDIN (nmechs[1], 'KINEMATIC_STRUCTURE_SCHEMA.'+
                                 'MECHANISM_BASE_PLACEMENT.BASE_OF_MECHANISM');

        IF (SIZEOF (nmbps) = 0) THEN
          RETURN (FALSE);
        ELSE
          RETURN (suitably_based_mechanism (nmbps[1], mech));
        END_IF;
      END_IF;
    END_IF;
  END_IF;
END_FUNCTION;
(*
```

Argument definitions:

mbp: the **mechanism_base_placement** that defines the placement of the **base** of **mech**.

mech: the **mechanism** whose **base** is being placed by **mbp**.

5.5.4 unique_link_usage

This function checks whether **link** belongs to only one **mechanism**. It returns TRUE if all **kinematic_joints** that use **link** as their **first_link** or as their **second_link** are in the **joints** set of the same **kinematic_structure**, and if this **kinematic_structure** serves as the **structure_-definition** of precisely one **mechanism**. The function returns FALSE otherwise. The type of the function is BOOLEAN.

EXPRESS specification:

```
*)
FUNCTION unique_link_usage (link : kinematic_link) : BOOLEAN;
  LOCAL
    mechs   : SET OF mechanism;
    joints  : SET OF kinematic_joint;
    struct  : kinematic_structure;
  END_LOCAL;

  joints := bag_to_set
            (USEDIN (link,
              'KINEMATIC_STRUCTURE_SCHEMA.KINEMATIC_JOINT.FIRST_LINK') +
             USEDIN (link,
              'KINEMATIC_STRUCTURE_SCHEMA.KINEMATIC_JOINT.SECOND_LINK'));
  struct := joints[1].structure;

  REPEAT i := 2 TO SIZEOF (joints);
    IF (joints[i].structure :<>: struct) THEN
      RETURN (FALSE);
    END_IF;
  END_REPEAT;

  mechs := bag_to_set
           (USEDIN (struct,
             'KINEMATIC_STRUCTURE_SCHEMA.MECHANISM.STRUCTURE_DEFINITION'));

  IF (SIZEOF (mechs) <> 1) THEN
    RETURN (FALSE);
  END_IF;

  RETURN (TRUE);

END_FUNCTION;
(*
```

Argument definitions:

link: the **kinematic_link** whose usage is to be checked.

5.5.5 coordinated_pair_link_representation

This function checks whether the definition of a **kinematic_pair** is coordinated with an existing **kinematic_link_representation** for a specified **kinematic_link** identified by **link**. The function returns TRUE if **link** is related to a **kinematic_link_representation**, and if **pair_placement** is in the **items** set of this **kinematic_link_representation**. The function returns FALSE otherwise. The type of the function is BOOLEAN.

EXPRESS specification:

```
*)
FUNCTION coordinated_pair_link_representation
    (link           : kinematic_link;
     pair_placement : rigid_placement) : BOOLEAN;
  LOCAL
    link_rep     : kinematic_link_representation;
  END_LOCAL;

  link_rep := representation_of_link (link);

  IF (link_rep = ?) THEN
    RETURN (FALSE);
  ELSE
    IF NOT (pair_placement IN link_rep\representation.items) THEN
      RETURN (FALSE);
    ELSE
      RETURN (TRUE);
    END_IF;
  END_IF;
END_FUNCTION;
(*
```

Argument definitions:

link: the **kinematic_link** whose representation is to be coordinated with the pair definition which uses **pair_placement**.

pair_placement: a **rigid_placement** that serves as **pair_placement_in_first_link_context** or as **pair_placement_in_second_link_context** of a **kinematic_pair**.

5.5.6 frame_associated_to_background

The function **frame_associated_to_background** checks whether the **kinematic_frame_background** identified by **background** is in the **items** set of a **kinematic_frame_background_representation**. It also checks whether this **kinematic_frame_background_representation** is related to the **kinematic_link_representation** that has the **rigid_placement** identified by **frame** in its **items** set. This relationship is through a **kinematic_frame_background_-**

representation_association, the **transformation_operator** of which uses **frame** as **transformer_frame**. The function returns TRUE if **frame** and **background** match these conditions; it returns FALSE otherwise. The type of the function is BOOLEAN.

EXPRESS specification:

```
*)
FUNCTION frame_associated_to_background
  (frame      : rigid_placement;
   background : kinematic_frame_background) : BOOLEAN;
  LOCAL
    rep_bag : BAG OF kinematic_frame_background_representation;
    trf_bag : BAG OF kinematic_frame_based_transformation;
    trm_bag : BAG OF kinematic_frame_based_transformation;
    ass_bag : BAG OF kinematic_frame_background_representation_association;
    rep     : kinematic_frame_background_representation;
    ass     : kinematic_frame_background_representation_association;
  END_LOCAL;

  rep_bag := USEDIN (background\representation_item,
                   'KINEMATIC_STRUCTURE_SCHEMA.' +
                   'KINEMATIC_FRAME_BACKGROUND_REPRESENTATION\' +
                   'REPRESENTATION.ITEMS');

  IF SIZEOF (rep_bag) = 0 THEN
    RETURN (FALSE);
  END_IF;

  trf_bag := USEDIN (frame,
                   'KINEMATIC_STRUCTURE_SCHEMA.' +
                   'KINEMATIC_FRAME_BASED_TRANSFORMATION.' +
                   'TRANSFORMER_FRAME');

  IF SIZEOF (trf_bag) = 0 THEN
    RETURN (FALSE);
  END_IF;

  REPEAT i := 1 TO HIINDEX (rep_bag);
    rep := rep_bag[i];

    ass_bag := USEDIN (rep\representation,
              'KINEMATIC_STRUCTURE_SCHEMA.' +
              'KINEMATIC_FRAME_BACKGROUND_REPRESENTATION_ASSOCIATION\' +
              'REPRESENTATION_RELATIONSHIP_WITH_TRANSFORMATION\' +
              'REPRESENTATION_RELATIONSHIP.REP_2');

    IF SIZEOF (ass_bag) > 0 THEN
      REPEAT j:= 1 TO HIINDEX (ass_bag);
        ass := ass_bag[j];

        trm_bag := QUERY (trm <* trf_bag |
          (trm\functionally_defined_transformation :=:
           ass\representation_relationship_with_transformation.
          transformation_operator));

        IF SIZEOF (trm_bag) > 0 THEN
          RETURN (TRUE);
        END_IF;

      END_REPEAT;
    END_IF;
  END_REPEAT;

  RETURN (FALSE);

END_FUNCTION;
(*
```

Argument definitions:

frame: a **rigid_placement** which is checked for its role as the **transformer_frame** in the relationship between the **kinematic_link_representation** which contains **frame** in its **items** set and the **kinematic_frame_background_representation** the **items** set of which contains **background**.

background: the **kinematic_frame_background** which is checked whether it is in the **items** set of a **kinematic_frame_background_representation** which is related to the **kinematic_link_representation** of **frame**, using **frame** as the **transformer_frame**.

5.5.7 plane_angle_for_pair_in_radian

The function **plane_angle_for_pair_in_radian** converts the specified **angle** to an equivalent **plane_angle_measure** whose unit is radian. The conversion factor is evaluated from the **plane_angle_unit** in the **units** set of a **global_unit_assigned_context** which is the **context_of_items** of the **first_link** of the **joint** of the specified **pair**. The type of the function is **plane_angle_measure**. If the appropriate units are found, the function returns the converted **plane_angle_measure**; the function returns an indeterminate value (?) otherwise.

EXPRESS specification:

```
*)
FUNCTION plane_angle_for_pair_in_radian
         (pair  : kinematic_pair;
          angle : plane_angle_measure) : plane_angle_measure;
  LOCAL
    converted_angle : plane_angle_measure := angle;
    link_rep        : kinematic_link_representation
                    := representation_of_link (pair.joint.first_link);
    link_cntxt      : representation_context;
    pa_units        : SET OF unit := [];
    pau             : unit;
  END_LOCAL;

  link_cntxt := link_rep\representation.context_of_items;
```

```
IF NOT ('MEASURE_SCHEMA.GLOBAL_UNIT_ASSIGNED_CONTEXT'
        IN TYPEOF (link_cntxt)) THEN
  RETURN (?);
END_IF;

pa_units := QUERY (unit <* link_cntxt\global_unit_assigned_context.units |
                   'MEASURE_SCHEMA.PLANE_ANGLE_UNIT' IN TYPEOF (unit));

IF SIZEOF (pa_units) <> 1 THEN
  RETURN (?);
END_IF;

pau := pa_units[1];

IF (NOT ('MEASURE_SCHEMA.SI_UNIT' IN TYPEOF (pau)) AND
    NOT ('MEASURE_SCHEMA.CONVERSION_BASED_UNIT' IN TYPEOF (pau))) THEN
  RETURN (?);
END_IF;

REPEAT WHILE ('MEASURE_SCHEMA.CONVERSION_BASED_UNIT' IN TYPEOF (pau));
  converted_angle := converted_angle *
                     pau\conversion_based_unit.conversion_factor.
                     value_component;
  pau := pau\conversion_based_unit.conversion_factor.unit_component;

  IF ((NOT ('MEASURE_SCHEMA.SI_UNIT' IN TYPEOF (pau)) AND
       NOT ('MEASURE_SCHEMA.CONVERSION_BASED_UNIT' IN TYPEOF (pau))) OR
      (NOT ('MEASURE_SCHEMA.PLANE_ANGLE_UNIT' IN TYPEOF (pau)))) THEN
    RETURN (?);
  END_IF;
END_REPEAT;

IF (pau\si_unit.name <> si_unit_name.radian) THEN
  RETURN (?);
END_IF;

CASE pau\si_unit.prefix OF
  si_prefix.exa     : RETURN (1.E18 * converted_angle);
  si_prefix.peta    : RETURN (1.E15 * converted_angle);
  si_prefix.tera    : RETURN (1.E12 * converted_angle);
  si_prefix.giga    : RETURN (1.E9 * converted_angle);
  si_prefix.mega    : RETURN (1.E6 * converted_angle);
  si_prefix.kilo    : RETURN (1.E3 * converted_angle);
  si_prefix.hecto   : RETURN (1.E2 * converted_angle);
  si_prefix.deca    : RETURN (1.E1 * converted_angle);
  si_prefix.deci    : RETURN (1.E-1 * converted_angle);
  si_prefix.centi   : RETURN (1.E-2 * converted_angle);
  si_prefix.milli   : RETURN (1.E-3 * converted_angle);
  si_prefix.micro   : RETURN (1.E-6 * converted_angle);
  si_prefix.nano    : RETURN (1.E-9 * converted_angle);
  si_prefix.pico    : RETURN (1.E-12 * converted_angle);
  si_prefix.femto   : RETURN (1.E-15 * converted_angle);
```

```
    si_prefix.atto      : RETURN (1.E-18 * converted_angle);
  OTHERWISE             : RETURN (converted_angle);
  END_CASE;

END_FUNCTION;
(*
```

Argument definitions:

pair: the **kinematic_pair** the context of whose **pair_placement_in_first_link_context** is to be evaluated for the conversion factor.

angle: the angle that is converted to an angle in radians.

5.5.8 convert_plane_angle_for_pair_from_radian

The function **convert_plane_angle_for_pair_from_radian** converts an angle expression, specified as **angle_expr**, the angle unit of which is radian, to an equivalent **plane_angle_measure** with a unit as specified in the respective context. The conversion factor is evaluated from the **plane_angle_unit** in the **units** set of a **global_unit_assigned_context** which is the **context_of_items** of the **first_link** of the **joint** of the specified **pair**. The type of the function is **plane_angle_measure**. If the appropriate units are found, the function returns the converted **plane_angle_measure**; the function returns an indeterminate value (?) otherwise.

EXPRESS specification:

```
*)
FUNCTION convert_plane_angle_for_pair_from_radian
         (pair       : kinematic_pair;
          angle_expr : REAL) : plane_angle_measure;
  LOCAL
    link_rep      : kinematic_link_representation
                  := representation_of_link (pair.joint.first_link);
    link_cntxt    : representation_context;
    pa_units      : SET OF unit := [];
    pau           : unit;
    conv_factor   : REAL := 1.0;
    result        : plane_angle_measure;
  END_LOCAL;

  link_cntxt := link_rep\representation.context_of_items;

  IF NOT ('MEASURE_SCHEMA.GLOBAL_UNIT_ASSIGNED_CONTEXT'
          IN TYPEOF (link_cntxt)) THEN
    RETURN (?);
  END_IF;

  pa_units := QUERY (unit <* link_cntxt\global_unit_assigned_context.units |
                     'MEASURE_SCHEMA.PLANE_ANGLE_UNIT' IN TYPEOF (unit));

  IF SIZEOF (pa_units) <> 1 THEN
    RETURN (?);
  END_IF;

  pau := pa_units[1];

  IF (NOT ('MEASURE_SCHEMA.SI_UNIT' IN TYPEOF (pau)) AND
      NOT ('MEASURE_SCHEMA.CONVERSION_BASED_UNIT' IN TYPEOF (pau))) THEN
    RETURN (?);
  END_IF;

  REPEAT WHILE ('MEASURE_SCHEMA.CONVERSION_BASED_UNIT' IN TYPEOF (pau));
    conv_factor := conv_factor *
                   pau\conversion_based_unit.conversion_factor.
                   value_component;
    pau := pau\conversion_based_unit.conversion_factor.unit_component;

    IF ((NOT ('MEASURE_SCHEMA.SI_UNIT' IN TYPEOF (pau)) AND
         NOT ('MEASURE_SCHEMA.CONVERSION_BASED_UNIT' IN TYPEOF (pau))) OR
        (NOT ('MEASURE_SCHEMA.PLANE_ANGLE_UNIT' IN TYPEOF (pau)))) THEN
      RETURN (?);
    END_IF;
  END_REPEAT;

  IF (pau\si_unit.name <> si_unit_name.radian) THEN
    RETURN (?);
  END_IF;

  CASE pau\si_unit.prefix OF
    si_prefix.exa      : conv_factor := 1.E18 * conv_factor;
    si_prefix.peta     : conv_factor := 1.E15 * conv_factor;
    si_prefix.tera     : conv_factor := 1.E12 * conv_factor;
    si_prefix.giga     : conv_factor := 1.E9 * conv_factor;
    si_prefix.mega     : conv_factor := 1.E6 * conv_factor;
    si_prefix.kilo     : conv_factor := 1.E3 * conv_factor;
    si_prefix.hecto    : conv_factor := 1.E2 * conv_factor;
    si_prefix.deca     : conv_factor := 1.E1 * conv_factor;
    si_prefix.deci     : conv_factor := 1.E-1 * conv_factor;
    si_prefix.centi    : conv_factor := 1.E-2 * conv_factor;
    si_prefix.milli    : conv_factor := 1.E-3 * conv_factor;
    si_prefix.micro    : conv_factor := 1.E-6 * conv_factor;
    si_prefix.nano     : conv_factor := 1.E-9 * conv_factor;
    si_prefix.pico     : conv_factor := 1.E-12 * conv_factor;
    si_prefix.femto    : conv_factor := 1.E-15 * conv_factor;
    si_prefix.atto     : conv_factor := 1.E-18 * conv_factor;
  OTHERWISE            : ;
  END_CASE;

  result := angle_expr / conv_factor;
  RETURN (result);
END_FUNCTION;
(*
```

Argument definitions:

pair: the **kinematic_pair** the context of whose **pair_placement_in_first_link_context** is to be evaluated for the conversion factor.

angle_expr: the angle expression in radian that is converted to an angle with the unit which is required in the context of **pair**.

5.5.9 convert_spatial_to_ypr_rotation

The function **convert_spatial_to_ypr_rotation** converts the specified **rotation**, which is either a **ypr_rotation** or a **rotation_about_direction**, to its equivalent representation as a **ypr_rotation**. The units for the yaw, pitch, and roll angles of the **ypr_rotation** are evaluated from the **plane_angle_unit** in the **units** set of a **global_unit_assigned_context** which is the **context_of_items** of the **first_link** of the **joint** of the specified **pair**. This evaluation is done by a call to the function **plane_angle_for_pair_in_radian**.

If **rotation** is already of type **ypr_rotation**, it is returned immediately. Otherwise, **convert_spatial_to_ypr_rotation** returns either a **ypr_rotation** that is equivalent to the **rotation_about_direction** which has been used as the **rotation** argument, or an indeterminate value (?), if the unit evaluation has not been successful. The type of the function is **ypr_rotation**.

EXPRESS specification:

```
*)
FUNCTION convert_spatial_to_ypr_rotation (pair     : kinematic_pair;
                                          rotation : spatial_rotation)
                                                   : ypr_rotation;

 LOCAL
   axis       : direction;
   angle      : plane_angle_measure;   -- rotation angle in application
                                       -- specific units
   conv_angle : plane_angle_measure;   -- rotation angle in radians
   ya, pa, ra : plane_angle_measure;   -- yaw, pitch, and roll angle
   ucf        : REAL;                  -- unit conversion factor
   dx, dy, dz : REAL;                  -- components of direction vector
   s_a, c_a   : REAL;                  -- sine and cosine of rotation angle
   rotmat     : ARRAY [1 : 3] OF
                ARRAY [1 : 3] OF REAL; -- rotation matrix
   cm1        : REAL;
   s_y, c_y   : REAL;
   s_r, c_r   : REAL;
 END_LOCAL;

 -- If rotation is already a ypr_rotation, return it immediately
 IF 'KINEMATIC_STRUCTURE_SCHEMA.YPR_ROTATION' IN TYPEOF (rotation) THEN
   RETURN (rotation);
 END_IF;

 -- rotation is a rotation_about_direction

 axis  := normalise (rotation\rotation_about_direction.direction_of_axis);
```

```
angle := rotation\rotation_about_direction.rotation_angle;

-- a zero rotation is converted trivially
IF (angle = 0.0) THEN
  RETURN ([0.0, 0.0, 0.0]);
END_IF;

dx := axis.direction_ratios[1];
dy := axis.direction_ratios[2];
dz := axis.direction_ratios[3];

-- provide angle measured in radian

conv_angle := plane_angle_for_pair_in_radian (pair, angle);

IF (conv_angle = ?) THEN
  RETURN (?);
END_IF;

ucf := angle / conv_angle;
s_a := SIN (conv_angle);
c_a := COS (conv_angle);

-- axis parallel either to x-axis or to z-axis?
IF (dy = 0.0) AND (dx * dz = 0.0) THEN
  REPEAT WHILE (conv_angle <= - PI);
    conv_angle := conv_angle + 2.0 * PI;
  END_REPEAT;
  REPEAT WHILE (conv_angle > PI);
    conv_angle := conv_angle - 2.0 * PI;
  END_REPEAT;

  ya := ucf * conv_angle;
  IF (conv_angle <> PI) THEN
    ra := - ya;
  ELSE
    ra := ya;
  END_IF;

  IF (dx <> 0.0) THEN
    -- axis parallel to x-axis - use x-axis as roll axis
    IF (dx > 0.0) THEN
      RETURN ([0.0, 0.0, ya]);
    ELSE
      RETURN ([0.0, 0.0, ra]);
    END_IF;
  ELSE
    -- axis parallel to z-axis - use z-axis as yaw axis
    IF (dz > 0.0) THEN
      RETURN ([ya, 0.0, 0.0]);
    ELSE
      RETURN ([ra, 0.0, 0.0]);
      END_IF;
    END_IF;
  END_IF;

  -- axis parallel to y-axis - use y-axis as pitch axis
  IF ((dy <> 0.0) AND (dx = 0.0) AND (dz = 0.0)) THEN
    IF (c_a >= 0.0) THEN
      ya := 0.0;
      ra := 0.0;
    ELSE
      ya := ucf * PI;
      ra := ya;
    END_IF;

    pa := ucf * ATAN (s_a, ABS (c_a));
    IF (dy < 0.0) THEN
      pa := - pa;
    END_IF;

    RETURN ([ya, pa, ra]);
  END_IF;

  -- axis not parallel to any axis of coordinate system
  -- compute rotation matrix

  cm1 := 1.0 - c_a;

  rotmat := [ [ dx * dx * cm1 + c_a,
                dx * dy * cm1 - dz * s_a,
                dx * dz * cm1 + dy * s_a ],
              [ dx * dy * cm1 + dz * s_a,
                dy * dy * cm1 + c_a,
                dy * dz * cm1 - dx * s_a ],
              [ dx * dz * cm1 - dy * s_a,
                dy * dz * cm1 + dx * s_a,
                dz * dz * cm1 + c_a ] ];

  -- rotmat[1][3] equals SIN (pitch_angle)
  IF (ABS (rotmat[1][3]) = 1.0) THEN
    -- |pa| = PI/2
    BEGIN
      IF (rotmat[1][3] = 1.0) THEN
        pa := 0.5 * PI;
      ELSE
        pa := -0.5 * PI;
      END_IF;

      -- In this case, only the sum or difference of roll and yaw angles
      -- is relevant and can be evaluated from the matrix.
      -- According to IP 'rectangular pitch angle' for ypr_rotation,
      -- the roll angle is set to zero.
```

```
        ra := 0.0;
        ya := ATAN (rotmat[2][1], rotmat[2][2]);

        -- result of ATAN is in the range [-PI/2, PI/2].
        -- Here all four quadrants are needed.

        IF (rotmat[2][2] < 0.0) THEN
          IF ya <= 0.0 THEN
            ya := ya + PI;
          ELSE
            ya := ya - PI;
          END_IF;
        END_IF;
      END;
    ELSE
      -- COS (pitch_angle) not equal to zero
      BEGIN
        ya := ATAN (- rotmat[1][2], rotmat[1][1]);

        IF (rotmat[1][1] < 0.0) THEN
          IF (ya <= 0.0) THEN
            ya := ya + PI;
          ELSE
            ya := ya - PI;
          END_IF;
        END_IF;

        ra := ATAN (-rotmat[2][3], rotmat[3][3]);

        IF (rotmat[3][3] < 0.0) THEN
          IF (ra <= 0.0) THEN
            ra := ra + PI;
          ELSE
            ra := ra - PI;
          END_IF;
        END_IF;

        s_y := SIN (ya);
        c_y := COS (ya);
        s_r := SIN (ra);
        c_r := COS (ra);

        IF ((ABS (s_y) > ABS (c_y)) AND
            (ABS (s_y) > ABS (s_r)) AND
            (ABS (s_y) > ABS (c_r))) THEN
          cm1 := - rotmat[1][2] / s_y;
        ELSE
          IF ((ABS (c_y) > ABS (s_r)) AND (ABS (c_y) > ABS (c_r))) THEN
            cm1 := rotmat[1][1] / c_y;
          ELSE
            IF (ABS (s_r) > ABS (c_r)) THEN
              cm1 := - rotmat[2][3] / s_r;
        ELSE
          cm1 := rotmat[3][3] / c_r;
        END_IF;
      END_IF;
    END_IF;

    pa := ATAN (rotmat[1][3], cm1);

  END;
END_IF;

ya := ya * ucf;
pa := pa * ucf;
ra := ra * ucf;

RETURN ([ya, pa, ra]);

END_FUNCTION;
(*
```

Argument definitions:

pair: the **kinematic_pair** the context of whose **pair_placement_in_first_link_context** is to be evaluated with regard to the **plane_angle_unit**.

rotation: the **spatial_rotation** that is to be converted by the function.

5.5.10 assign_directed_link

The function **assign_directed_link** checks the **orientation** of a **kinematic_joint**. This function returns **joint.first_link**, if **orientation** is true and **joint.second_link** otherwise. The type of the function is **kinematic_link**.

EXPRESS specification:

```
*)
FUNCTION assign_directed_link
   (joint : kinematic_joint; orientation : BOOLEAN ) : kinematic_link;
   IF (orientation) THEN
      RETURN (joint.first_link);
   ELSE
      RETURN (joint.second_link);
   END_IF;
END_FUNCTION;
(*
```

Argument definitions:

joint: a **kinematic_joint** for which the **kinematic_link** is determined.

orientation: a boolean indicating the direction of traversal of a joint.

5.5.11 connected_in_simple_loop

This function checks if two given **joint_logical_relationships** are related to the same loop, and if the orientation of the **oriented_joints** which they relate is consistent. The function returns TRUE if **relation_1** and **relation_2** match these conditions. The function returns FALSE otherwise. The type of the function is BOOLEAN.

EXPRESS specification:

```
*)
FUNCTION connected_in_simple_loop( relation_1 : joint_logical_relationship;
                                   relation_2 : joint_logical_relationship)
  : BOOLEAN;
  LOCAL
    next_jlr_in_loop_set : SET [1:?] OF joint_logical_relationship;
  END_LOCAL;

  IF ((relation_1.loop:<>: relation_2.loop) OR
      (relation_1 :=: relation_2)) THEN
        RETURN (FALSE);
  ELSE
    IF (relation_1.next_joint_logical_structure :=:
        relation_2.previous_joint_logical_structure) THEN
      RETURN (TRUE);
    ELSE
      next_jlr_in_loop_set :=
          QUERY (relation <* USEDIN (relation_1.next_joint_logical_structure,
          'kinematic_structure_schema.joint_logical_relationship.' +
          'previous_joint_logical_structure')
           | relation.loop :=: relation_1.loop);
      IF (SIZEOF(next_jlr_in_loop_set) <> 1) THEN
          RETURN (FALSE);
      ELSE
          RETURN (connected_in_simple_loop(next_jlr_in_loop_set [1],
                  relation_2));
      END_IF;
    END_IF;
  END_IF;
END_FUNCTION;
(*
```

Argument definitions:

relation_1: a **joint_logical_structure_relationship** which is checked whether it is connected with **relation_2**.

relation_2: the second **joint_logical_structure_relationship** which is checked whether it is connected with **relation_1**.

```
*)
END_SCHEMA;          -- end kinematic_structure_schema
(*
```

6 Kinematic motion representation

The following *EXPRESS* declaration begins the **kinematic_motion_representation_schema** and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA kinematic_motion_representation_schema;

REFERENCE FROM geometry_schema
    (cartesian_point,
     curve);

REFERENCE FROM measure_schema
    (length_measure,
     measure_with_unit,
     parameter_value,
     plane_angle_measure);

REFERENCE FROM representation_schema
    (functionally_defined_transformation,
     representation_item);

REFERENCE FROM kinematic_structure_schema
    (rotation_about_direction,
     spatial_rotation,
     ypr_rotation);
(*
```

NOTES

1 – The schemas referenced above can be found in the following parts of ISO 10303:

kinematic_structure_schema	Clause 5 of this part of ISO 10303
geometry_schema	ISO 10303-42
measure_schema	ISO 10303-41
representation_schema	ISO 10303-43

2 – See annex D for a graphical representation of this schema using the *EXPRESS-G* notation.

6.1 Introduction

The subject of the **kinematic_motion_representation_schema** is the description of motion. It specifies different types of paths that comprise the motion for a kinematic model.

6.2 Fundamental concepts and assumptions

Motion in this part of ISO 10303 is defined in terms of a parameter t. t is a non-decreasing parameter as motion progresses. Several motions may be synchronised by the requirement that the actual state of all motion in a kinematic model is controlled by a single parameter t which appears as an attribute in the **kinematic_control** and in the **kinematic_result**.

6.3 kinematic_motion_representation_schema type definition: motion_parameter_measure

The **motion_parameter_measure** is a mechanism by which different poses in the course of a motion are distinguished. The motion parameter may be defined with or without units.

NOTE – To introduce time aspects, the motion parameter may be defined with time units.

EXPRESS specification:

```
*)
TYPE motion_parameter_measure = SELECT
  (parameter_value,
   measure_with_unit);
END_TYPE;
(*
```

6.4 kinematic_motion_representation_schema entity definitions

6.4.1 translation

A **translation** specifies a displacement in Cartesian coordinate space R^3.

EXPRESS specification:

```
*)
ENTITY translation;
  x : length_measure;
  y : length_measure;
  z : length_measure;
END_ENTITY;
(*
```

Attribute definitions:

x: displacement in x-direction.

y: displacement in y-direction.

z: displacement in z-direction.

6.4.2 transform

A **transform** specifies a linear transformation of a frame along a path, composed of a rotation and a **translation**. First the rotation shall be performed and then the **translation**.

EXPRESS specification:

```
*)
ENTITY transform
  SUBTYPE OF (functionally_defined_transformation);
  rotation_component    : spatial_rotation;
  translation_component : translation;
END_ENTITY;
(*
```

Attribute definitions:

rotation_component: the rotational part of the **transform**.

translation_component: the displacement part of the **transform**.

6.4.3 path_node

A **path_node** describes a constituent of a path definition. It consists of a linear transformation and its associated motion parameter.

EXPRESS specification:

```
*)
ENTITY path_node;
  control_transform : transform;
  t_parameter       : motion_parameter_measure;
END_ENTITY;
(*
```

Attribute definitions:

control_transform: a linear transformation that specifies a local frame on a path.

t_parameter: the parameter t of a path which corresponds to the control transform.

6.4.4 kinematic_path

A **kinematic_path** describes a path or a segment of a path along which motion is to take place.

EXPRESS specification:

```
*)
ENTITY kinematic_path
  SUPERTYPE OF (ONEOF (path_element, composite_path))
  SUBTYPE OF (representation_item);
END_ENTITY;
(*
```

6.4.5 path_element_connection

A **path_element_connection** is the connection of two adjacent **path_elements**. The parameter t at the end of the previous **path_element** is the same as the parameter t at the start of the next **path_element**.

EXPRESS specification:

```
*)
ENTITY path_element_connection;
  previous_element     : path_element;
  next_element         : path_element;
WHERE
  WR1: previous_element.node_to = next_element.node_from;
  WR2: compare_unit_components (previous_element.node_to.t_parameter,
                                next_element.node_from.t_parameter);
END_ENTITY;
(*
```

Attribute definitions:

previous_element: the first element in a sequence of two **path_element**s.

next_element: the second element in a sequence of two **path_element**s.

Formal propositions:

WR1: The **path_node** at the end of the previous **path_element** shall be identical with the **path_node** at the start of the next **path_element**.

WR2: The **unit_component**s of the parameter values of the two **path_element**s shall be identical.

6.4.6 composite_path

A **composite_path** is a collection of **path_element**s joined end to end. The **composite_path** has its own t parameter range; the **path_element**s are re-parametrised in such a way that the sum of the lengths of their individual t parameter ranges fits the length of the parameter range of the **composite_path**, and the ratio between the lengths of the individual parameter ranges remains unchanged.

EXPRESS specification:

```
*)
ENTITY composite_path
  SUBTYPE OF (kinematic_path);
  elements : SET [1 : ?] OF path_element_connection;
  t_start  : motion_parameter_measure;
  t_end    : motion_parameter_measure;
WHERE
  WR1: increasing_measure_value (t_start, t_end);
  WR2: compare_unit_components (t_start, t_end);
  WR3: connected_in_simple_path (elements);
END_ENTITY;
(*
```

Attribute definitions:

elements: the **path_element_connection**s that define the **path_element**s, their sequence and connectivity.

t_start: the parameter value at the start of the **composite_path**.

t_end: the parameter value at the end of the **composite_path**.

Formal propositions:

WR1: The parameters shall be increasing.

WR2: The **unit_components** of the parameter values at the start and end of a **composite_path** shall be identical.

WR3: Every **path_element** in a **composite_path** shall be connected to every other **path_element** in that **composite_path**, either directly through a single **path_element_connection**, or indirectly through a number of **path_element_connections**, in such a way that the **path_elements** form a single open coherent chain without gaps or branches. This is checked by the function **connected_in_simple_path**.

6.4.7 path_element

A **path_element** is a **kinematic_path** that describes either a point-to-point, a linear, a circular, or a general curve-based type of motion.

EXPRESS specification:

```
*)
ENTITY path_element
  SUPERTYPE OF (ONEOF (point_to_point_path,
                       circular_path,
                       linear_path,
                       curve_based_path))
  SUBTYPE OF (kinematic_path);
  node_from : path_node;
  node_to   : path_node;
WHERE
  WR1: compare_unit_components (node_from.t_parameter,
                                node_to.t_parameter);
  WR2: increasing_measure_value (node_from.t_parameter,
                                 node_to.t_parameter);
END_ENTITY;
(*
```

Attribute definitions:

node_from: the frame where the **path_element** starts.

node_to: the frame where the **path_element** ends.

Formal propositions:

WR1: The **unit_components** of the parameter values of the two **path_nodes** shall be identical.

WR2: The parameter t shall be increasing from the beginning to the end of the **path_element**.

6.4.8 point_to_point_path

A **point_to_point_path** defines the frames which are to be traversed. It does not have a prescribed interpolation.

NOTE – The interpolation is left to the software that uses the data.

EXPRESS specification:

```
*)
ENTITY point_to_point_path
  SUBTYPE OF (path_element);
END_ENTITY;
(*
```

6.4.9 circular_path

A **circular_path** describes a portion of a complete **kinematic_path** such that the interpolation rule gives a circle for the translational part of the motion of the origin of the frame of the **kinematic_link**. The interpolation for the orientation is performed in synchronous fashion with the interpolation of translation.

EXPRESS specification:

```
*)
ENTITY circular_path
  SUBTYPE OF (path_element);
  via_point : cartesian_point;
WHERE
  WR1: SELF\path_element.node_to.control_transform.translation_component <>
       SELF\path_element.node_from.control_transform.translation_component;
  WR2: non_coincident_coordinates (via_point,
         SELF\path_element.node_from.control_transform.translation_component)
       AND
       non_coincident_coordinates (via_point,
         SELF\path_element.node_to.control_transform.translation_component);
END_ENTITY;
(*
```

Attribute definitions:

via_point: an intermediate point through which the **circular_path** passes.

Formal propositions:

WR1: The endpoint of a **circular_path** shall not coincide with its startpoint.

WR2: the **via_point** of a **circular_path** shall not coincide with either its startpoint or its endpoint.

6.4.10 linear_path

A **linear_path** is an arcwise connected bounded linear path element defined by the two **transforms** of the **path_nodes** at the beginning and the end of the path element. **transforms** between the values of the motion parameters given in these **path_nodes** are obtained by linear interpolation between the **transforms** specified in **node_from** and **node_to** of the **linear_path**.

EXPRESS specification:

```
*)
ENTITY linear_path
  SUBTYPE OF (path_element);
END_ENTITY;
(*
```

6.4.11 curve_based_path

A **curve_based_path** describes a portion of a complete **kinematic_path** such that the interpolation rule follows a curve for the translational part of the motion of the origin of the frame of the **kinematic_link**. The interpolation for the orientation is performed in synchronous fashion with the interpolation of translation.

EXPRESS specification:

```
*)
ENTITY curve_based_path
  SUBTYPE OF (path_element);
  path_curve : curve;
WHERE
  WR1: SELF\path_element.node_to.control_transform.translation_component <>
       SELF\path_element.node_from.control_transform.translation_component;
END_ENTITY;
(*
```

Attribute definitions:

path_curve: the curve which defines the translational interpolation between the **path_nodes** that are connected by the **curve_based_path**.

Formal propositions:

WR1: The endpoint of a **curve_based_path** shall not coincide with its startpoint.

Informal propositions:

startpoint on curve: The startpoint of a **curve_based_path** shall lie on the curve which is referenced as **path_curve**.

endpoint on curve: The endpoint of a **curve_based_path** shall lie on the curve which is referenced as **path_curve**.

6.5 kinematic_motion_representation_schema function definitions

6.5.1 connected_in_simple_path

The **connected_in_simple_path** function checks whether a **composite_path** is simply connected. The function returns TRUE if the **connections** connect all the **path_elements** of the path in such a way that they form an open (i.e., acyclic) coherent chain without gaps or branches. The function returns FALSE otherwise. The type of the function is BOOLEAN.

EXPRESS specification:

```
*)
FUNCTION connected_in_simple_path (connections : SET OF path_element_connection)
                                               : BOOLEAN;
  LOCAL
     connection_set : SET [0 : ?] OF path_element_connection;
     nec0           : INTEGER;
     pec0           : INTEGER;
     necbranch      : INTEGER;
     pecbranch      : INTEGER;
  END_LOCAL;

  IF SIZEOF (connections) > 1 THEN
    connection_set := QUERY (pec1 <* connections |
                        SIZEOF (QUERY (pec2 <* connections - pec1 |
                          pec1.next_element :=: pec2.previous_element)) = 0);
    nec0 := SIZEOF (connection_set);

    connection_set := QUERY (pec1 <* connections |
                        SIZEOF (QUERY (pec2 <* connections - pec1 |
                          pec2.next_element :=: pec1.previous_element)) = 0);
    pec0 := SIZEOF (connection_set);

    connection_set := QUERY (pec1 <* connections |
                        SIZEOF (QUERY (pec2 <* connections - pec1 |
                          pec1.next_element :=: pec2.previous_element)) > 1);
    necbranch := SIZEOF (connection_set);

    connection_set := QUERY (pec1 <* connections |
                        SIZEOF (QUERY (pec2 <* connections - pec1 |
                          pec2.next_element :=: pec1.previous_element)) > 1);
    pecbranch := SIZEOF (connection_set);

    IF ((nec0 <> 1) OR (pec0 <> 1) OR (necbranch > 0) OR (pecbranch > 0)) THEN
      RETURN (FALSE);
    ELSE
      RETURN (TRUE);
    END_IF;
  ELSE
    RETURN (TRUE);
  END_IF;
END_FUNCTION;
(*
```

Argument definitions:

connections: the set of **path_element_connections** which has to be checked.

6.5.2 compare_unit_components

The **compare_unit_components** function checks the units of two given path parameters of the type **motion_parameter_measure**. The function returns TRUE if the units are identical. The function returns FALSE if the units are different. The type of the function is BOOLEAN.

EXPRESS specification:

```
*)
FUNCTION compare_unit_components (parm1 : motion_parameter_measure;
                                  parm2 : motion_parameter_measure) : BOOLEAN;
   IF (('MEASURE_SCHEMA.PARAMETER_VALUE' IN TYPEOF (parm1)) AND
       ('MEASURE_SCHEMA.PARAMETER_VALUE' IN TYPEOF (parm2))) THEN
         RETURN (TRUE);
   ELSE
     IF (('MEASURE_SCHEMA.MEASURE_WITH_UNIT' IN TYPEOF (parm1)) AND
         ('MEASURE_SCHEMA.MEASURE_WITH_UNIT' IN TYPEOF (parm2))) THEN
        IF (parm1.unit_component = parm2.unit_component) THEN
            RETURN (TRUE);
        ELSE
            RETURN (FALSE);
        END_IF;
     ELSE
       RETURN (FALSE);
     END_IF;
  END_IF;
END_FUNCTION;
(*
```

Argument definitions:

parm1: the first path parameter which is checked for identical units with the second path parameter.

parm2: the second path parameter which is checked for identical units with the first path parameter.

6.5.3 increasing_measure_value

The **increasing_measure_value** function compares the values of two given path parameters of the type **motion_parameter_measure**. The function returns TRUE if the value of the first parameter is less than the value of the second parameter. The function returns FALSE if the value of the second parameter is less than or equal to the value of the first parameter. The type of the function is BOOLEAN. This function assumes that both parameters are either **parameter_values** or given in the same units.

EXPRESS specification:

```
*)
FUNCTION increasing_measure_value (parm1 : motion_parameter_measure;
                                   parm2 : motion_parameter_measure) : BOOLEAN;
  IF ('MEASURE_SCHEMA.PARAMETER_VALUE' IN TYPEOF (parm1)) THEN
    IF (parm1 < parm2)  THEN
```

```
      RETURN (TRUE);
    ELSE
      RETURN (FALSE);
    END_IF;
  ELSE
    IF ('MEASURE_SCHEMA.MEASURE_WITH_UNIT' IN TYPEOF (parm1)) THEN
      IF (parm1.value_component < parm2.value_component) THEN
        RETURN (TRUE);
      ELSE
        RETURN (FALSE);
      END_IF;
    ELSE
      RETURN (FALSE);
    END_IF;
  END_IF;
END_FUNCTION;
(*
```

Argument definitions:

parm1: the first path parameter whose value is compared with the value of the second path parameter.

parm2: the second path parameter whose value is compared with the value of the first path parameter.

6.5.4 non_coincident_coordinates

This function checks whether a given **translation** coincides with a given **cartesian_point** or not. It returns TRUE if **trltn** does not coincide with **crtpt**. It returns FALSE otherwise. The type of the function is BOOLEAN.

EXPRESS specification:

```
*)
FUNCTION non_coincident_coordinates (crtpt : cartesian_point;
                                     trltn : translation) : BOOLEAN;
  IF ((crtpt.coordinates[1] = trltn.x) AND
      (crtpt.coordinates[2] = trltn.y) AND
      (crtpt.coordinates[3] = trltn.z)) THEN
    RETURN (FALSE);
  ELSE
    RETURN (TRUE);
  END_IF;
END_FUNCTION;
(*
```

Argument definitions:

crtpt: the **cartesian_point** whose non-coincidence with **trltn** has to be checked.

trltn: the **translation** whose non-coincidence with **crtpt** has to be checked.

```
*)
END_SCHEMA;   -- end kinematic_motion_representation_schema
(*
```

7 Kinematic analysis control and result

The following *EXPRESS* declaration begins the **kinematic_analysis_control_and_result_schema** and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA kinematic_analysis_control_and_result_schema;

REFERENCE FROM kinematic_motion_representation_schema
    (kinematic_path,
     motion_parameter_measure);

REFERENCE FROM kinematic_structure_schema
    (kinematic_joint,
     kinematic_link_representation,
     mechanism,
     pair_value,
     rigid_placement);

REFERENCE FROM geometry_schema
    (geometric_representation_context);

REFERENCE FROM representation_schema
    (representation,
     representation_relationship);
(*
```

NOTES

1 – The schemas referenced above can be found in the following parts of ISO 10303:

kinematic_motion_representation_schema	Clause 6 of this part of ISO 10303
kinematic_structure_schema	Clause 5 of this part of ISO 10303
geometry_schema	ISO 10303-42
representation_schema	ISO 10303-43

2 – See annex D for a graphical representation of this schema using the *EXPRESS-G* notation.

7.1 Introduction

The subject of the **kinematic_analysis_control_and_result_schema** is the identification and analysis of configurations of **kinematic_structure**s and the interpolation between configurations. The schema covers the prescribed paths and the paths resulting from analysis.

7.2 Fundamental concepts and assumptions

A kinematic analysis is a specific presentation of the kinematic behaviour of a mechanical product. There are two possible ways to check the kinematic behaviour of a kinematic structure:

– forward analysis;

– backward analysis.

The forward analysis specifies the motion of a kinematic structure in terms of a sequence of configurations and of the interpolations between them. The result of a forward analysis is the path of specified frames on specified links.

The backward analysis specifies the motion of a kinematic structure with a prescribed path defined by frames. The result of a backward analysis is the sequence of configuration definitions in the form of the pair parameter values for a sufficiently large number of pairs to control the motion.

A configuration of a kinematic structure is defined by the specification of pair values for a set of pairs of the kinematic structure.

7.3 kinematic_analysis_control_and_result_schema type definitions

7.3.1 interpolation_type

The **interpolation_type** describes the method of interpolation between two configuration definitions of a kinematic structure.

EXPRESS specification:

```
*)
TYPE interpolation_type = ENUMERATION OF
  (undefined,
   synchronous,
   linear);
END_TYPE;
(*
```

Enumerated item definitions:

undefined: an interpolation in which all pair parameters shall match the set of pair parameter values defined in the next configuration definition.

synchronous: an interpolation in which all pair parameters are interpolated such that the pair parameters pass from one value to the next one at the same rate. The rate may vary proportionally among the pairs.

linear: an interpolation in which all pair parameters are interpolated such that the pair parameters pass from one value to the next one at the same rate. The rate remains constant.