Open Kinematics Interface

Introduction 2
Creating a template 4
Adding user code 6
  Internal presentation of Cartesian points 6
  Internal presentation of Joint points 7
  Robot Configuration Flags 7
  Kinematics functions 8
    Configuration Function 8
    Inverse Kinematics Function 8
  Direct Kinematics Function 8
  Working Envelope Test Function 8
Off-Line Computation Function 8
Inverse Jacobian Function 9
Linking the Kinematics Functions to the User template 10
Example 11
  Code: 14

Introduction

This document describes implementation of user-defined robot kinematics in the SoftMC system. The supported robot models are limited to 6 axis robots having unique solution of direct kinematics problem.

In order to handle basic kinematics mappings the following terms are defined:

World-Space: A fixed coordinate system referenced to the base of the robot. usually expressed in Cartesian coordinates for position (X, Y, Z) and another set of coordinates representing orientation, the set of orientation coordinates can differ between many available presentations (Euler angles, quaternion, rotation matrices, ...). Elements of this space we usually call **Cartesian points**.

Joint-Space. A coordinate system used to describe the state of the robot in terms of its joint states. Usually a set of joint angles (angles between robot segments) for rotary joints or displacement for linear joints. In many cases they directly represent the motor positions, but in some cases a non-diagonal coupling matrix can be used for translating motor angles into joint angles/displacements. Elements of this space we usually call **Joint points**.

User Kinematics is a set of user-defined algorithms representing the core kinematics functionality of a robot. Basically it consist of two main functions: Inverse Kinematics (**IK**) and Direct Kinematics (**DK**). Inverse Kinematics translate Cartesian or world points into joint coordinates. Usually it contains configuration description of the robot for each given world-space point. Direct Kinematics function translate the given joint point (or joint coordinates) into world point of any chosen coordinate system. For an arbitrary robot two auxiliary functions define its kinematics, first is checking the available world-space working envelope (**AF – accessibility function**), and the second is the auxiliary function **(CF- configuration function)** defining the configuration flags of any given joint point.

In cases of simple robot kinematics or extensive analytical models inverse velocity mapping can be defined (IJ - inverse Jacobian function). It is basically the matrix product of robot inverse Jacobian matrix at the given joint coordinates with the actual world velocity vector. As this function can be very complex and not always analytically available, having it is not mandatory. It can be omitted and then the SoftMC system takes the numeric derivative instead (which is slightly less accurate, depending on these elected sample time and robots velocity).

To summarize:

Direct Kinematics:: DK: JS → WS
Inverse Kinematics:: IK: WS x CS → JS
Configuration Function:: CF: JS → CS
Accessibility Function:: AF:WS → {true, false}
Inverse Jacobian Function ::: IJ: WS x JS→ JS

Creating a template

First of all a template element has to be created this is done using the special "model" value (5) identifier for user kinematics groups:

Common Shared <robot name> as group {axnm = <axis name>,} model = 5 of <pointtype>

This line defines a robotic group having undefined kinematic model (until it is linked to user functions) spawn on given axes and using the given point type for commands/queries. The **<pointtype>** is point descriptor and one of the following currently supported point types:

Point Type	Description	# coo
XY	XY table	2
XYZ	XYZ system	3
XYZR	XYZ + Roll	4
XYZRP	XYZ + Roll + Pitch	5
XYZPR	XYZ + Pitch + Roll	5
XYZYPR	XYZ + Yaw + Pitch + Roll	6

Additionally user-defined point types can be selected, there are 5 predefined user point types: **USER1, USER2, USER3, ..., USER5**.

If one of these is selected user need to provide conversion functions between user given Cartesian point coordinates and internal Cartesian point representation (See: Internal presentation of Cartesian points) This will be done using the function:

int **rbt_SetUsrPnt**(int usr, int size, utype set, utype get)

where:

- **usr** is the point index (1,2,3,4,5) denoting which one of the user-defined point types is taken (**USER1, ... USER5**).
- **size** is the number of coordinates used for this point type (1 .. 7).
- **set** is a user-provided function for converting user-given coordinates (#{...}) into internal Cartesian point.
- **get** is the user provided function converting internal Cartesian point presentation into user-given coordinates (#{...})
- **utype** is defined as:

typedef int (***utype**)(double *vector,double *car);

Where **vector** is a double floating point array of the user-provided (obtained) coordinates (exact copy of the list between "#{" and "}" and **car** is the double floating point array of internal Cartesian point representation.

Adding user code

Internal presentation of Cartesian points

In the softMC system all Cartesian points (independently of the selected model or point type) are represented by the following structure:

Component	**coordinates**	**description**
Position	X	X in mm
	Y	Y in mm
	Z	Z in mm
Orientation Quaternion	Cos (Phi/2)	denoted as "Ro" also
	Nx*Sin (Phi/2)	
	Ny*Sin(Phi/2)	
	Nz*Sin(Phi/2)	

Note that all internal softMC orientations are represented by quaternions of the form: ***[Cos(phi/2), n*Sin(phi/2)]*** where ***b = (Nx,Ny,Nz)*** and ||***n***|| =***1***. Therefore all Cartesian points will be handled as double floating point arrays of 7 elements, for all types of robots

Internal presentation of Joint points

Joint points are internally represented as double floating point arrays. The units are radians or millimeters depending on axis type (rotary, linear).

Robot Configuration Flags

Robot configuration flags will be represented as a bit filed with following order:

Flag	**Name**	**Bits**
Arm	Lefty(1), Righty(2)	B0-B1
Elbow	Below(1), Above(2)	B2-B3
Wrist	NoFlip(1), Flip(2)	B4-B5

The flags will be transferred as a long integer (32 bit) value. Note that the filed names of the configuration flags are appropriate for general open-kinematics structures. However more than these three configuration flags are not supported in the system (language constrain). Also the filed names could be inappropriate for different kinematics types. If a kinematics that is implemented demands more than three configuration flags they must be implemented via user functions.

Kinematics functions

Once the kinematics equations have been written the following user functions have to be codded.

Configuration Function

int Config(double *jnt)

Where jnt is the joint coordinates array and the returned value is long integer representing the robot configuration flags for the given joint coordinates.

Inverse Kinematics Function

int InverseKinematics(int cfg, double *cpnt, double *jnt)

Where:
cfg – desired robot configuration
cpnt – artesian point
jnt – joint point

Function returns 0 if everything is OK and nonzero number (1,2,3, ...) of axis where a problem is detected.

Direct Kinematics Function

int DirectKinematics(double *cpnt, double *jnt)

Where:
cpnt – Cartesian point
jnt – joint point

Function returns 0 if everything is OK and nonzero number (1,2,3, ...) of axis where a problem is detected.

Working Envelope Test Function

int Accessible(double *cpnt)

Check if the given point is in the working envelope of the robot. If yes returned 0. If the point is on the outer side of the working envelope 1 is returned. If it is too close to the zero origin point -1 is returned.

Off-Line Computation Function

int Setup()

returns 0 if all is OK. This function may be not needed, it depends on user implementation.

Inverse Jacobian Function

If Inverse Jacobian Matrix can be analytically computed it can be provided together with all user kinematics functions, but if not this function can be omitted.

int InvJacobian (double *JointPoint,double *JointVel,double *CartesianPoint,double *CartesianVel)

Linking the Kinematics Functions to the User template

Once the user kinematics functions are available they can be linked to template group (model=5) using the following function:

int **rbt_SetUserKin**(int rbt_id, RBT_UFUN &user);

Where **rbt_id** is the template group id, obtainable querying ElementId of a group. And "user" a structure (class) defined in **rbtUkin.typ** header file containing pointers to all user functions.

Interface user structure (class):

```
class RBT_UFUN
{
        public:

        int (*setup)();
        int (*ikin)(int, double *, double *);
        int (*ijac)(double *, double *,double *, double *);
        int (*dkin)(double *, double *);
        int (*acces)(double *);
        int (*cfg)(double *);
        RBT_UFUN()
        {
                ikin = 0;
                ijac = 0;
                dkin = 0;
                acces = 0;
                cfg = 0;
                setup = 0;
        }
};
```

The class should be first filled with appropriate functions and then transferred to rbt_SetUserKin function. Once the function is called there is no further relevance of this structure and it can be deleted.

Example

An example of user-defined kinematics will be given using the SCORBOT robot.
Attached files: ***User.cpp, TUKIN.PRG***.

In the Scorbot robot, the second, third, and fourth joint axes are parallel to one another and point into the paper at points A, B, and P, respectively. The first joint axis points up vertically, and the fifth joint axis intersects the fourth perpendicularly. Find the overall transformation matrix for the robot.

Table 9.1 D-H parameters of the Scorbot robot

	α_i	a_i	d_i	θ_i
1	$-\pi/2$	a_1	d_1	θ_1
2	0	a_2	0	θ_2
3	0	a_3	0	θ_3
4	$-\pi/2$	0	0	θ_4
5	0	0	d_5	θ_5

$$ {}^{0}A_1 = \begin{bmatrix} c\theta_1 & 0 & -s\theta_1 & a_1 c\theta_1 \\ s\theta_1 & 0 & c\theta_1 & a_1 s\theta_1 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(a)} $$

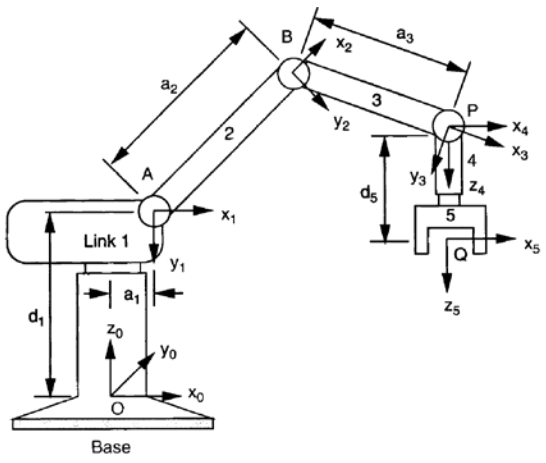

Fig. E4. Schematic diagram of the Scorbot robot

$$^{1}A_2 = \begin{bmatrix} c\theta_2 & -s\theta_2 & 0 & a_2c\theta_2 \\ s\theta_2 & c\theta_2 & 0 & a_2s\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(b)}$$

$$^{2}A_3 = \begin{bmatrix} c\theta_3 & -s\theta_3 & 0 & a_3c\theta_3 \\ s\theta_3 & c\theta_3 & 0 & a_3s\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(c)}$$

$$^{3}A_4 = \begin{bmatrix} c\theta_4 & 0 & -s\theta_4 & 0 \\ s\theta_4 & 0 & c\theta_4 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(d)}$$

$$^{4}A_5 = \begin{bmatrix} c\theta_5 & -s\theta_5 & 0 & 0 \\ s\theta_5 & c\theta_5 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{(e)}$$

Multiplying (b), (c), (d) yields

$$^{1}A_4 = \begin{bmatrix} c\theta_{234} & 0 & -s\theta_{234} & a_3c\theta_{23} + a_2c\theta_2 \\ s\theta_{234} & 0 & c\theta_{234} & a_3s\theta_{23} + a_2s\theta_2 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(f)}$$

where $\theta_{ij} = \theta_i + \theta_j$, and $\theta_{ijk} = \theta_i + \theta_j + \theta_k$

Treat θ_2, θ_{23}, and θ_{234} as new variables. In this way, the rotation matrix contains only one variable, θ_{234}, while the position submatrix contains two variables, θ_2 and θ_{23}. Multiplying (a), (f), and (e) yields the overall transformation matrix $^{0}A_5$ as:

$$u_x = c\theta_1 c\theta_{234} c\theta_5 + s\theta_1 s\theta_5 \quad , \quad u_y = s\theta_1 c\theta_{234} c\theta_5 - c\theta_1 s\theta_5 \quad , u_z = -s\theta_{234} c\theta_5$$

$$v_x = -c\theta_1 c\theta_{234} s\theta_5 + s\theta_1 c\theta_5 \text{ , } v_y = -s\theta_1 c\theta_{234} s\theta_5 - c\theta_1 c\theta_5 \quad , \quad v_z = s\theta_{234} s\theta_5$$

$$w_x = -c\theta_1 s\theta_{234} \quad , \quad w_y = -s\theta_1 s\theta_{234} \quad , \quad w_z = -c\theta_{234}$$

$$q_x = c\theta_1(a_1 + a_2c\theta_2 + a_3c\theta_{23} - d_5s\theta_{234}),$$

$$q_y = s\theta_1(a_1 + a_2c\theta_2 + a_3c\theta_{23} - d_5s\theta_{234}),$$

$$q_z = d_1 - a_2s\theta_2 - a_3s\theta_{23} - d_5c\theta_{234} \quad \text{(g)}$$

Since this is a 5-dof manipulator, only five of the six parameters of the end-effector can be described. Very often, the desired position of a point and the direction of a line in the end-effector are specified.

(a) Direct Kinematics

For the direct kinematic problem, we simply substitute the given joint angles into Eq.(g) to obtain the end-effector position and the orientation in terms of $\bar{u}$, $\bar{v}$, and $\bar{w}$.

(b) Inverse Kinematics

For the inverse kinematic problem, only 5 of the 12 parameters can be specified at will. This is because the manipulator has only 5-dof. It is obvious that the position vector $\vec{q}$ and the approach vector $\vec{w}$ cannot be specified simultaneously, due to the fact that $\vec{q}$ and $\vec{w}$ together depend only on 4 degrees of freedom of the manipulator. In this example, $\vec{q}$ and $\vec{u}$ are specified. A more straightforward approach by multiplying both sides of the loop-closure equation by $({}^{o}A_1)^{-1}$; that is

$$({}^{o}A_1)^{-1}\,{}^{o}A_5 = {}^{1}A_2\,{}^{2}A_3\,{}^{3}A_4\,{}^{4}A_5 \qquad \text{(h)}$$

Equating the first column of the (h),

$$u_x c\theta_1 + u_y s\theta_1 = c\theta_{234} c\theta_5 \qquad \text{(i)}$$

$$-u_z = s\theta_{234} c\theta_5 \qquad \text{(j)}$$

$$-u_x s\theta_1 + u_y c\theta_1 = -s\theta_5 \qquad \text{(k)}$$

Similarly, equating the fourth column of Eq.(h)

$$q_x c\theta_1 + q_y s\theta_1 - a_1 = a_2 c\theta_2 + a_3 c\theta_{23} - d_5 s\theta_{234} \qquad \text{(l)}$$

$$-q_z + d_1 = a_2 s\theta_2 + a_3 s\theta_{23} + d_5 c\theta_{234} \qquad \text{(m)}$$

$$-q_x s\theta_1 + q_y c\theta_1 = 0 \qquad \text{(n)}$$

The first joint angle θ_1 is obtained by (n) :

$$\theta_1 = \tan^{-1} q_y / q_x$$

There are two solutions for $\theta_1 = \theta_1^*$ or $(\theta_1^* + \pi)$. Once θ_1 is found, two solutions for θ_5 are obtained from (k):

$$\theta_5 = \sin^{-1}(u_x s\theta_1 - u_y c\theta_1) = \theta_5^*, \quad \text{or} \quad \theta_5 = \pi - \theta_5^*.$$

Corresponding to each solution of (θ_1, θ_5), Eqs.(i) and (j) produce a unique solution of θ_{234}

$$\theta_{234} = \text{ATAN2}[-u_z / c\theta_5, (u_x c\theta_1 + u_y s\theta_1)/c\theta_5]$$

Now, solving Eqs.(l) and (m) for θ_2 and θ_3

$$a_2 c\theta_2 + a_3 c\theta_{23} = k_1 \qquad \text{(o)}$$

$$a_2 s\theta_2 + a_3 s\theta_{23} = k_2 \qquad \text{(p)}$$

Where $k_1 = q_x c\theta_1 + q_y s\theta_1 - a_1 + d_5 s\theta_{234}$ and $k_2 = -q_z + d_1 - d_5 c\theta_{234}$

Summing squares of the above equations yields

$$a_2^2 + a_3^3 + 2a_2 a_3 c\theta_3 = k_1^2 + k_2^2$$

$$\Rightarrow \theta_3 = \cos^{-1}[(k_1^2 + k_2^2 - a_2^2 - a_3^2)/2a_2 a_3]$$

and there are two solutions for θ_3. If $\theta_3=\theta_3^*$ is also a solution, $\theta_3=-\theta_3^*$ is also a solution. Once θ_3 is known, θ_2 can be solved by expanding Eqs.(o) and (p) as follows:

$$(a_2 + a_3 c\theta_3)c\theta_2 - (a_3 s\theta_3)s\theta_2 = k_1$$

$$(a_3 s\theta_3)c\theta_2 + (a_2 + a_3 c\theta_3)s\theta_2 = k_2$$

$$\Rightarrow \begin{cases} c\theta_2 = A \\ s\theta_2 = B \end{cases}$$

Hence, corresponding to each solution of (θ_1, θ_3, θ_5, θ_{234}), we obtain a unique solution of θ_2:

$$\theta_2 = A\tan 2(B, A).$$

Finally, θ_4 is obtained by

$$\theta_4 = \theta_{234} - \theta_2 - \theta_3$$

We conclude that corresponding to each given end-effector location, there are at most eight inverse kinematic solutions.

Code:

See attached User.cpp