A Framework For Kinematic Modeling of Cooperative Robotic Systems Based on Screw Theory

Carlos Rodrigues Rocha, carlos.rocha@riogrande.ifrs.edu.br
Cristiane Pescador Tonetto, cris.tonetto@emc.ufsc.br
Altamir Dias, altamir@emc.ufsc.br
Universidade Federal de Santa Catarina - Departamento de Engenharia Mecânica

Abstract.*This paper concerns the analysis of the screw theory-based kinematic modeling in order to ease the programming process. To do so, an object-oriented computational framework is developed from this analysis. Screw theory and related tools are well used in motion analysis of open and closed kinematic chains, typical of robotic manipulators, in an uniform and systematic way. Their use in the inverse kinematics resolution of redundant systems present advantages such as the dimensional consistency and the tendency to conserve movement. These are of interest in motion planning strategies for systems where redundancy is an inherent and necessary feature, such as parallel manipulators, cooperative robotic systems and vehicle-manipulator systems. These systems can present complex kinematic chains. However, these chains can be decomposed into simpler ones, which can be previously derived and stored in a sort of kinematic chain database. Thus, complex kinematic chains can be formed by composition of predefined kinematic chains connected to each other, considering also the necessary transformations to express all their screws in a common reference system. Besides facilitating the kinematic model definition of complex chains, this systematization is interesting to deal with systems whose kinematic model can vary over time, as in collision avoidance situations. Therefore, the study of the kinematic modeling modularization is of interest to make kinematic analysis easier and even to automate this process. It is also noted that, to the best knowledge of the authors, the mathematical software usually used in kinematic analysis do not have modules/libraries to deal with screw theory entities. To implement the computational system modularity features, object-oriented analysis techniques were used to design a framework to represent screw theory entities, kinematic chains and their composition. This framework is intended to make the implementation of kinematic chain databases and the future development of an automated system friendly. The first results of this analysis are presented in this paper. The modularization of kinematic chains, the possibility of chain databases and the object-oriented model resulting from the analysis of robotic systems are discussed. Some criteria and restrictions to the choice of the computational platform used and an example are presented.*

Keywords: *cooperative robotic systems, vehicle-manipulator systems, screw theory, kinematic modeling, computational framework*

1. INTRODUCTION

Kinematic analysis has great importance in robotic systems, since most robots are designed for motion. Recent applications require involvement of more than one robot to its execution. In these cases, a single robot could not execute satisfactorily a task or in some cases could not handle the execution of the operation assigned to it at all. These *cooperative robotic systems* are composed by two or more robots sharing a workspace and working toghether to execute a task. Vehicle-manipulator systems have some similarities to these systems, since they can be seen as composed by two different robots (the manipulator and the vehicle) which have to work toghether.

This analysis is usually a complex and difficult task, due to the kinematic chain size and structure. In most cases, the chain is kinematically redundant and have closed loops. Screw theory-based kinematic modeling is used as a tool for motion analysis, since it has a systematic and uniform way to deal with both open and closed kinematic chains, which is an advantage over the traditional modeling based on the Denavit-Hartenberg notation (Rocha *et al.*, 2011).

From the analysis of different robotic systems and its applications, it was noted that complex kinematic chains could be usually seen as being formed by composition of simpler chains. This appears to be obvious in cooperative robotic systems, since each manipulator that composes such system has its own kinematic chain. However, the use of modularization to build such models is not common. This could lead to a *kinematic chain database* that would allow to ease and to automatize the modeling of complex kinematic chains by composition of predefined chains.

This paper presents the initial development process of a computational framework to implement the screw-based kinematic modeling related entities and their behaviors. Differently from the usual numerical software employed in such cases, the framework design is based on the object-oriented paradigm. It aims to be used in interactive mode or in standalone program execution. This posed an interesting analysis to choose an computational platform where these features could be present.

The rest of the text is organized as follows. The fundamentals of the screw theory-based kinematic modeling and modularization issues are presented in the next section. The object-oriented design of a framework to represent this modeling process and its elements is discussed in the following. The choice of the computational platform used in this

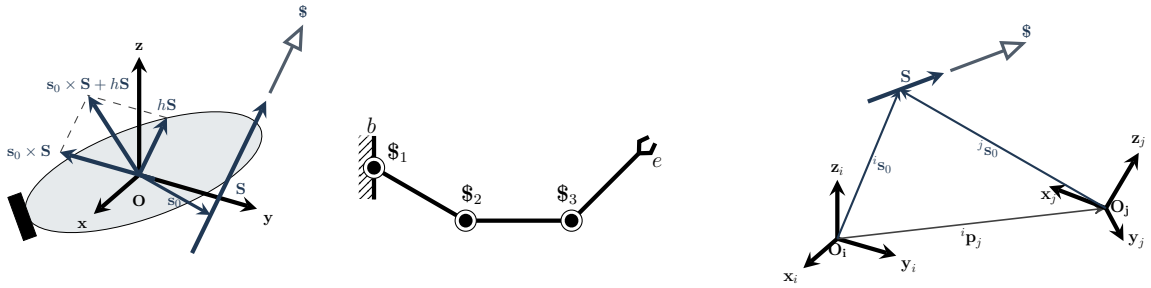
implementation is justified. A simple example is developed in order to illustrate the use of the framework. Then, final considerations are made.

2. KINEMATIC ANALYSIS BASED ON SCREW THEORY

The *kinematic constraints method* is used for motion planning due to its uniformity, systematization and interesting features for inverse kinematics resolution. The method is based in screw theory, graph theory and derived tools such as Davies method and Assur virtual chains. These tools and their application are discussed and exemplified in several papers (Hunt, 2000; Tsai, 1999; Campos *et al.*, 2005; Santos *et al.*, 2006; Guenther *et al.*, 2008). In order to ease the comprehension and the development of the next sections, some of these fundamentals must be briefly presented here.

2.1 Motion Representation By Twists

A *screw* is a geometric entity which represents both rotational and translational quantities. These are defined according to an *axis* and they are related by an scalar *pitch* (Hunt, 2000). A *twist* is a screw representation for velocity, which is expressed in Plücker notation as $\$ = [\boldsymbol{\omega}; \mathbf{v}_p]^{\mathrm{T}}$. The angular velocity of the body is $\boldsymbol{\omega}$, while $\mathbf{v}_p$ is the linear velocity of a point of the rigid body which coincides instantaneously with the referential origin $\mathbf{O}$. Figure 1a illustrates the definition of a twist which represents the motion of in a rigid body (an underwater vehicle in the example).



(a) (b) (c)

Figure 1: Motion representation: (a)Twist representing the velocity state of an underwater vehicle; (b)3R planar manipulator; (c)Screw transformation

The twist is defined by the $\mathbf{S}$ and $\mathbf{s}_0$ vectors, which define the screw axis, and the scalar pitch h. It can be decomposed into a *normalized screw* $\hat{\$}$ (obtained by use of the $\mathbf{s}$ unitary vector of $\mathbf{S}$) and a magnitude $\dot{q}$, as shown in Eq. 1.

$$\$ = \begin{bmatrix} \boldsymbol{\omega} \\ \mathbf{v}_p \end{bmatrix} = \begin{bmatrix} \boldsymbol{\omega} \\ \mathbf{s}_0 \times \boldsymbol{\omega} + h\boldsymbol{\omega} \end{bmatrix} = \begin{bmatrix} \mathbf{s} \\ \mathbf{s}_0 \times \mathbf{s} + h\mathbf{s} \end{bmatrix} \dot{q} = \hat{\$}\dot{q} \tag{1}$$

In a kinematic chain, the relative velocity state between two links is obtained by the sum of the twists of the kinematic pairs between them. For instance, in a 3R planar manipulator such as the one shown in Fig. 1b, the velocity state of the end-effector $\$_e$ relative to the manipulator base b is expressed as in Eq. 2,

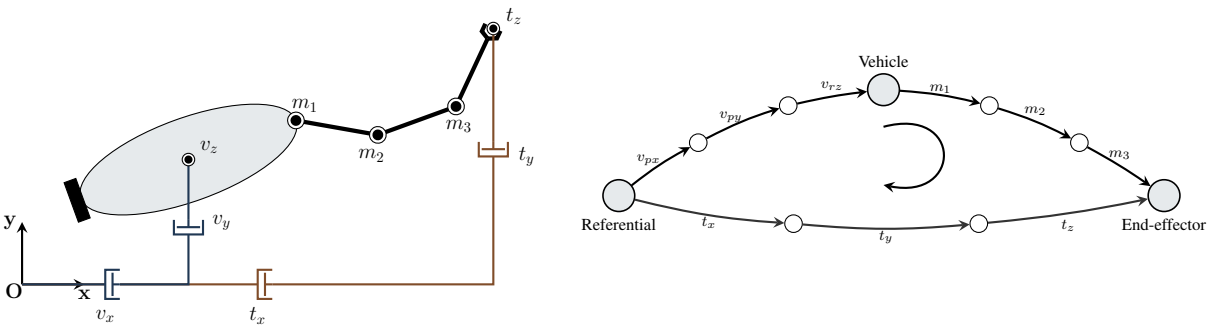
$$\$_e = \begin{bmatrix} \boldsymbol{\omega}_e \\ \mathbf{v}_{p_e} \end{bmatrix} = \sum_{i=b+1}^{e} \$_i = \sum_{i=b+1}^{e} \hat{\$}_i \dot{q}_i = \mathbf{J}\dot{\mathbf{q}} \tag{2}$$

where $\hat{\$}_i\dot{q}_i$ are the i twist components expressing the link i velocity relative to link $i-1$. The normalized screws column vectors compose the Jacobian matrix $\mathbf{J}$ and the column vector $\dot{\mathbf{q}} = [\dot{q}_{b+1} \cdots \dot{q}_e]^{\mathrm{T}}$ is formed by the magnitudes (Hunt, 2000).

The sum in Eq. 2 is possible if all twists are defined according to the same referential coordinate system. There are situations where some twists are defined in a different referential. In these cases, a *screw transformation* must be applied to these twists to express them in the desired referential. This transformation is also useful when the kinematic analysis must consider different reference points in the workspace. It has the form of Eq. 3, where ${}^i\mathbf{T}_j$ indicates that the transformation occurs from the j referential to the i referential and $\mathcal{S}\left({}^i\mathbf{p}_j\right)$ is the antisymmetrical matrix operator related to the $\mathbf{O}_j$ position vector expressed in the i frame (as seen in Fig. 1c). ${}^i\mathbf{R}_j$ is the rotation matrix expressing the orientation of the j frame relative to the i frame.

$${}^i\mathbf{T}_j = \begin{bmatrix} {}^i\mathbf{R}_j & 0 \\ \mathcal{S}\left({}^i\mathbf{p}_j\right){}^i\mathbf{R}_j & {}^i\mathbf{R}_j \end{bmatrix} \tag{3}$$

A kinematic chain can also be used to define or to monitor the motion of a single body, which is useful for motion analysis of vehicles. In this case, a *virtual kinematic chain* representing motion in the coordinate system is adopted. The use of virtual chains for this end was proposed in Santos *et al.* (2006) based on the work of Campos, who further developed the concept of Assur virtual kinematic chains (Campos *et al.*, 2005). Virtual chains can also be linked to real chains to monitor or to impose motion of particular links. Figure 2a illustrates both uses of virtual chains in the model of a planar underwater vehicle-manipulator system (UVMS). This robotic system is composed of a submarine vehicle and one manipulator attached to it for intervention tasks. The motion of the vehicle is represented by a PPR virtual chain v, while the manipulator corresponds to the real chain m. To monitor the end-effector motion, a PPR virtual chain t is defined. This chain is also used to define a task for the intervention system. It must be observed this forms a closed kinematic chain (Rocha and Dias, 2010).



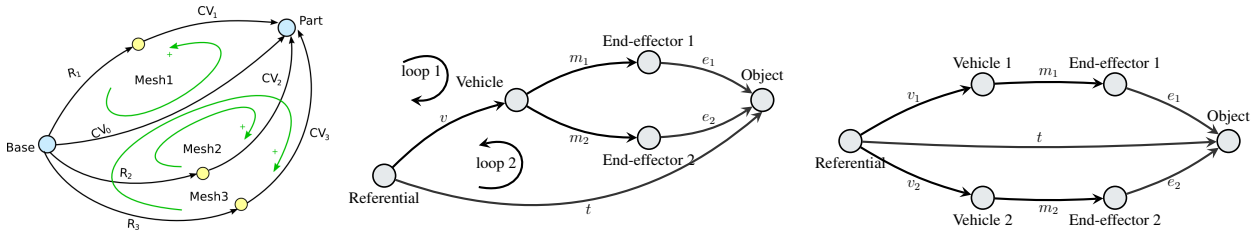
(a) (b)

Figure 2: Kinematic modeling of an UVMS: (a)Using kinematic chains; (b)Corresponding motion graph

Kinematic chains can be very complex to analyze, particularly in the case of cooperative systems. Graph theory is used to systematize and to simplify this analysis. To do so, *motion graphs* are used to represent chains. In these graphs, each vertex corresponds to a link of the chain, while each edge corresponds to a one degree-of-freedom (dof) joint (a n dof joint is replaced by a subchain formed by n single dof joints) (Tsai, 2000). The motion graph of the UVMS shown in Fig. 2a is depicted in Fig. 2b.

A generalized graph representation of a typical cooperative robotic industrial system (CRIS) is shown in Fig. 3a. The scenario corresponds to an object which is simultaneously manipulated by the robotic arms. This is a contracted graph, where each edge represents a complete subchain. The manipulators correspond to the R_i real subchains. The motion of the object is represented by the CV_0 virtual chain. CV_i virtual chains ($i \neq 0$) represent the i manipulator end-effector motion relative to the part manipulated (Tonetto *et al.*, 2010).

For cooperative vehicle-manipulator systems, there are two basic cases. The system can be composed of a single vehicle with more than one manipulator executing the task, such as the motion graph depicted in Fig. 3b. The other possibility is of more than one UVMS in cooperation, as shown in Fig. 3c, where it is considered that each UVMS has only one arm attached (Rocha and Dias, 2010).



(a) (b) (c)

Figure 3: Contracted motion graphs: (a)Cooperative robotic industrial system; (b)Multiple arm cooperating in a single UVMS; (c)Cooperation between UVMS

2.2 Davies Method

Davies adapted the Kirchhoff's circulation law for electric circuits to the case of kinematic chain circuits. In his method it is stated that *the algebraic sum of the kinematic pairs relative velocities in a closed kinematic chain equals zero*

(Davies, 1981). This leads to Eq. 4, known as *constraint equation*,

$$\sum_{i=1}^{n} \$_i = \sum_{i=1}^{n} \hat{\$}_i \dot{q}_i = \mathbf{N}\dot{\mathbf{q}} = \mathbf{0} \tag{4}$$

where $\mathbf{N}$ is the *network matrix* relating the joints motions to the independent circuits of the kinematic chain and $\dot{\mathbf{q}}$ is the magnitudes vector. The network matrix is obtained from the motion graph analysis as shown in Eq. 5,

$$\mathbf{N} = \begin{bmatrix} \mathbf{D}\mathrm{diag}\left\{\mathbf{B}_1\right\} \\ \vdots \\ \mathbf{D}\mathrm{diag}\left\{\mathbf{B}_l\right\} \end{bmatrix} \tag{5}$$

where $\mathbf{D}$ is a matrix composed by the normalized screws of the system and $\mathbf{B}$ is the circuit matrix of the motion graph, whose elements have values $+1$, -1 or 0 depending on the circuits and edges directions. Operator $\mathrm{diag}\left\{\mathbf{B}_i\right\}$ forms a diagonal matrix from the elements of $\mathbf{B}_i$ line of the circuit matrix, which has l lines (one for each independent circuit).

The constraint equation relates all joints velocities. Thus, it is possible to calculate the velocity of some of the joints if the other velocities are known. So, the constraint equation can be partitioned into two sets: the *primary* partition formed by the completely defined twists, and the *secondary* partition formed by the twists whose magnitudes are to be determined. This results in Eq. 6, where subscripts p and s stand for primary and secondary respectively. Secondary magnitudes are determined after rearranging the partitioned constraint equation as shown in Eq. 7.

$$\mathbf{N}\dot{\mathbf{q}} = \mathbf{N}_p\dot{\mathbf{q}}_p + \mathbf{N}_s\dot{\mathbf{q}}_s = \mathbf{0} \tag{6}$$

$$\dot{\mathbf{q}}_s = -\mathbf{N}_s^{-1}\mathbf{N}_p\dot{\mathbf{q}}_p \tag{7}$$

The secondary magnitudes are determined if $\mathbf{N}_s$ is invertible. Though $\mathbf{N}_s$ should be square to apply the inverse operation, there are cases where the number of secondary variables is greater than the workspace dimension, which turns $\mathbf{N}_s$ a retangular matrix. The pseudoinverse operation can be applied in these situations (Guenther *et al.*, 2008).

2.3 The Kinematic Constraints Method

The kinematic constraints method was originally proposed by Santos to solve the UVMS inverse kinematics (Santos *et al.*, 2006). It systematizes the application of the Davies method in order to determine the vehicle-manipulator motion from a specified task consisting of the end-effector motion and complementary goals. In essence, it consists of the following steps: insert virtual chains to impose desired motions and restrictions to the real system; define the primary and secondary sets according to the task definition and the desired actuated joints; apply Davies method to find secondary velocities; integrate velocities over time to obtain desired positions.

According to this method, a task consists of velocity profiles to be applied to the primary joints of the kinematic chain. The partitioning of the constraint equation depends on the task to be executed and can vary over time to satisfy complementary goals, such as avoiding joint limits in a CRIS manipulator. Eventually even the kinematic model and consequently the constraint equation can change to solve an unforeseen event or a different subtask specification, such as the presence of obstacles in an UVMS workspace. These different variations may be grouped in hybrid state models as proposed in Santos (2006).

Regarding the determination of joint positions by integration methods, virtual chains can also be used to minimize drift errors commonly generated by the use of numerical methods. The concept of *error virtual chains* is developed in Guenther *et al.* (2008).

2.4 Modularization of The Kinematic Model

The kinematic modeling of the previous discussed systems is often a difficult process. The components of a twist can be defined by complex mathematical expressions depending on the location of the respective joint in the chain. This fact also influences the definition of the constraint equation.

However, it is observed that the kinematic chain of the robotic system can be decomposed into simpler chains which are interconnected to each others. In the case of cooperative robotic industrial systems, each manipulator is a kinematic chain which can be previously modeled. Besides, there are few different types of virtual chains, which can be easily modeled. These simpler chains generally have also simpler definitions for the twists components.

So, a kinematic chain can be seen as composed by links and *kinematic components* which can be pure joints (and associated twists) or subchains (composed of more than one joint). In terms of the motion graph, each edge represents a kinematic component.

It should be noted that all screws in a kinematic chain must be defined relative to a common referential. In the case of a composed kinematic chain, a potential problem is the fact that the models of the subchains can be defined according to

different referentials. In this case, each subchain must have a screw transformation associated to it, from the referential of the subchain to the referential of the entire system chain.

In this work, it is assumed that the each kinematic chain has a base and an end-effector links (a mechanism, according to Davies (1995)) and its twists are defined according to a referential coincident with the base. The end-effector posture is used to define a screw transformation between the end-effector and the base which is associated to the kinematic chain (the rotation matrix and the position vector for instance, as shown in Eq. 3).

Another convention used is that two chains can be connected by their extremities (the end-effector of the first chain attaches to the base of the second chain). In this way, it is possible to express the twists of the second chain according to the first chain referential by applying the first chain screw transformation. In the case of a series of chains linked in sequence, screw transformations can be linked by premultiplication, as described by Eq. 8, where the transformation to be applied to the screws of chain i is composed by the premultiplication of the transformations of the chains b to $i-1$.

$$ {}^{b}\mathbf{T}_{i} = {}^{b}\mathbf{T} \dots {}^{i-1}\mathbf{T} \tag{8} $$

Using these conventions, it must be observed that if there is an isolated joint between two subchains it must also have a screw transformation associated.

The planar UVMS described in Sec. 2.can be modeled by the modularization described. The contracted graph in Fig. 4 identifies the system subchains. Both virtual subchains v and t are PPR and have the same kinematic model (differing only by their chain parameters values). The m subchain is RRR, which is commonly used for planar manipulators. So, if this two types of kinematic chains are part of a *chain database* they can be surely used to compose the entire chain of the underwater system and they can also be used to compose other systems.

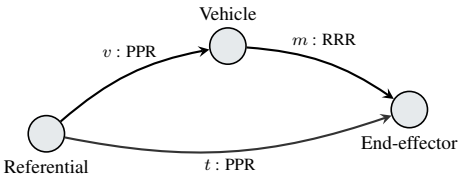


Figure 4: Contracted motion graph of the planar UVMS described of Fig. 2a

3. FRAMEWORK DESIGN BASED ON THE OBJECT-ORIENTED PARADIGM

The development of a framework for screw-based kinematic modeling was motivated by the need to ease the creation of cooperative/vehicle-manipulator systems simulations. For such a framework, modularity and reconfigurability were desired features, along with the possibility to define and systematize a database of kinematic chains. Besides, to the best knowledge of the authors, there is no similar work based on screw theory in literature, despite several work presented simulations and real implementations using this theory.

From the problem analysis, the following requirements were devised:

- To be able to represent screws and their actions;
- to be able to represent screw-based models of kinematic chains, whether single (formed only by single joints/twists) or composed (formed by subchains and twists);
- to be modular and extensible;
- to be possible to use it in interactive simulation environments and as part of programs;
- to have the ability to provide velocity information using task trajectory generators;
- to be able to store/load definitions of kinematic chains and also specific instances of them;

The adoption of an object-oriented approach to the framework design was based on the identified requirements, the vast knowledge base available, the diversity of development platforms and the authors experience. The Unified Modeling Language (UML) was used in the design to describe the framework elements (Booch *et al.*, 1999). In the initial analysis and modeling of the framework classes, it was observed that some of the components were similar to the design patterns described in literature (Metsker, 2004). This allowed to solve some design issues in a fast and systematic way, by the use of the design patterns suggested implementations or some simplification where possible.

The class diagram in Fig. 5 depicts the framework main classes. The representation of kinematic chains, screws and twists was modeled as a case of the *composite* design pattern. According to this pattern, both `Screw` and `KinematicChain`

classes are seen as kinematic chain components (descendents of the `KCComponent` class). Both class instances can compose an instance of `KinematicChain`. They share some attributes and methods which are implemented in the base class.

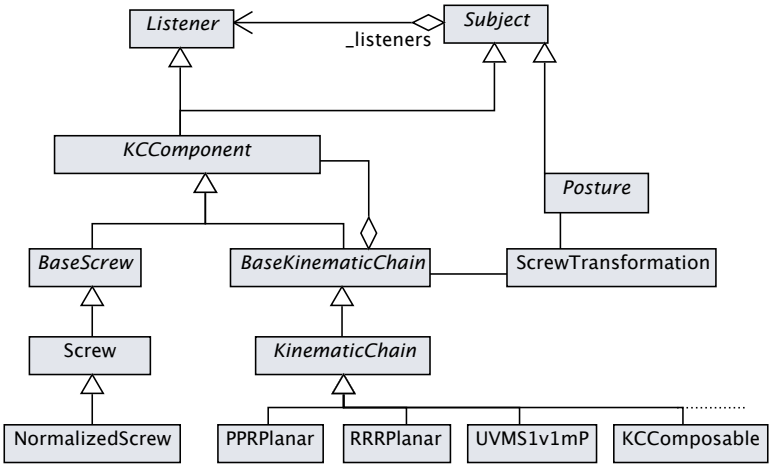


Figure 5: UML class diagram of the kinematic modeling framework

Screws and twists are modeled by the `Screw` class. A `Screw` instance is defined by its components, which can be directly assigned (as constants or mathematical expressions related to predefined attributes) or computed by a custom defined method. There is also a derived `NormalizedScrew` class in which the magnitude is always equal to one.

Kinematic chains are descendent of the `BaseKinematicChain` and `KinematicChain` classes. Besides having inherited properties from the `KCComponent` class, these classes have additional properties to define the chain composition. `KinematicChain` also has attributes to define the number of loops, the network matrix and its partitioning, along with a method to solve Davies method to calculate the secondary variables magnitudes. `BaseKinematicChain` is intented to be a base for single chains (open chains for instance) while the `KinematicChain` class goal is to be a base for classes where Davies method is used, whether single (all screws defined in the class) or composite (screws defined in different chains which compose the main kinematic chain).

The `KCComposable` class derived from `KinematicChain` allows to dynamically assemble a kinematic chain from predefined screws and kinematic chains using graphs to represent its structure. This class is intended to automatize the modeling process and the constraint equation resolution. In this way, it would be possible to use the class as a backend of a graphical user interface to interactively build the desired system model and also to deal with the storage and retrieving of chains in a database. This class is also of interest to represent hybrid event-based models where the constraint equation can be modified during the execution of a motion planning strategy.

Though flexible to create kinematic chains, a `KCComposable` instance can have worse performance than direct modeling of kinematic chains by deriving classes from `KinematicChain`. So, it is encouraged to create specific classes to well-known kinematic chains instead of using the `KCComposable` when possible.

All these classes are supposed to be data-aware (to notice when an important attribute is changed and automatically update their data/behaviors). So, the *observer* design pattern is also applicable to them (in a simplified way) by inheritance of the `Listener` and `Subject` abstract classes. Auxiliary classes, such as the `ScrewTransformation` class, also uses a *façade* pattern to define a generic way to deal with different types of posture definitions. Also, the *factory method* pattern is intended to be used to manage families of kinematic chain classes.

4. FRAMEWORK IMPLEMENTATION

The choice of computational platform used in the framework implementation was made parallel to its initial design process. The framework desired features and the evaluation of modules/libraries necessary to the implementation guided the decision process. Other desired features were also taken into account, such as the ability to interface with other software and the existence of graphical resources to visualize data. The licensing type and availability for different operating systems were also considered, since it is desired that the framework be available to as many platforms as possible and licensed as free software.

A comparison between possible implementation platforms is made in Tab. 1. These options were considered because of their use in robotics simulation, their availability for the operational systems commonly used in simulations (Microsoft Windows®, Linux and Mac OS®) and the authors experience. Each line corresponds to a characteristic to be evaluated.

After comparison and research for available resources in each platform (three numerical software systems and three

Table 1: Comparison between possible platforms for the framework implementation

Platform	Matlab®	Scilab	Octave	C/C++	Java	Python
Matrices / linear algebra libraries	Yes	Yes	Yes	Yes	Yes	Yes
Graph library	Yes	Yes	Yes	Yes	Yes	Yes
2D/3D graphics	Yes	Yes	Yes	Yes	Yes	Yes
Graphic modeling and animation	Yes	Yes	Yes	Yes	Yes	Yes
Extensible	Yes	Yes	Yes	Yes	Yes	Yes
Object-oriented	No	No	No	Yes	Yes	Yes
Interactive environment	Yes	Yes	Yes	No	No	Yes
Standalone program execution	MEX/C compiler	No	No	Yes	Runtime environment	Yes
Interface with other applications	Yes	Yes	Yes	Yes	Yes	Yes
Creation of embedded applications	Yes	No	No	Yes	Yes	Yes
Type	Commercial	Free Software	Free Software	Free Software	Free Software	Free Software

programming languages), the Python platform was chosen (Python.org, 2011). The capabilities of a full programming environment allied to the flexibility of use, similar to the numerical software analyzed, were the main reasons for this choice. Python implements the object-oriented features desired for the framework and has various modules for scientic computing. There are simulation softwares which use Python as an interactive interface language, allowing to implement motion planning strategies into a simulated robotic system (Diankov and Kuffner, 2008).

The classes modeled in the design process were implemented using the *numpy* and *scipy* libraries for matrices and linear algebra (Scipy.org, 2011). *Networkx* is the module used to deal with graph representation and analysis (NetworkX, 2011). Numpy arrays were used to store the kinematic chains, screws and auxiliary classes vectors/matrices. In cases where numerical manipulation is not essential, Python lists and tuples were used. A very interesting feature of the numpy array is the *slicing*, where views of an array can be used as an array itself. It differs from a partial array copy in the sense that it shares the values between the original array and the view, which implies that modifications in the view cells affect the original array. This allows to create matrices shared between the screws of a kinematic chain or even between chains.

The framework is used in scripts as any other Python module. The proper classes must be imported into a script, which then allow the necessary objects instantiation. New kinematic chain classes can be created from inheritance of `BaseKinematicChain` or `KinematicChain`. `KCComposable` instances are used when it is not desired to create classes for a specific chain structures. In this case, the chain structure and components must be manually assigned to the graph using the proper class methods/properties.

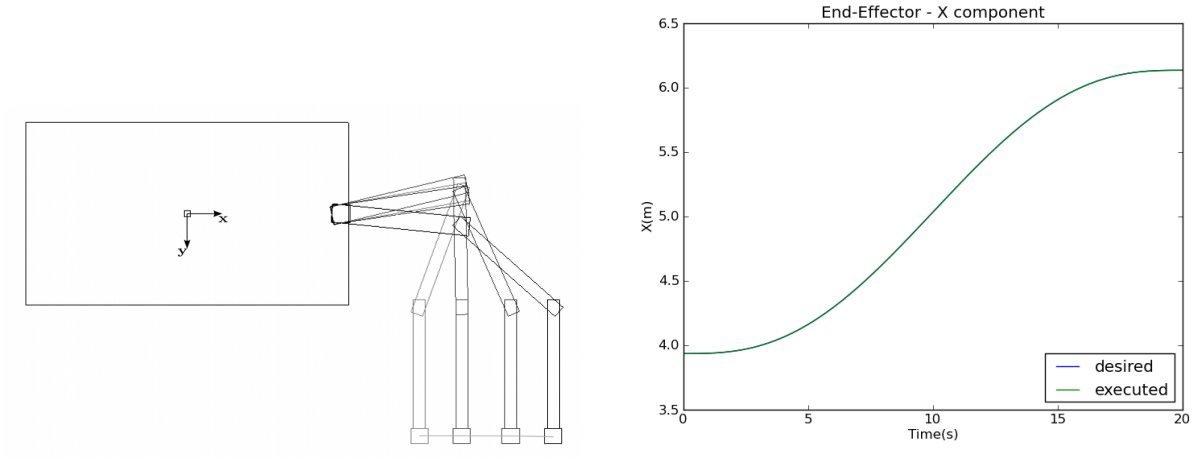
5. EXAMPLE

A simple case demonstrates the use of the framework in simulations. It consists in the execution of a simple task by the planar UVMS analyzed in Sec. 2. The system parameters are described in Santos (2006). A new class, named `UVMS1v1mP`, is derived from `KinematicClass` to represent this model and its behavior. The constructor and the `_update` methods were overloaded to implement the model definitions obtained from the screw based method in order to calculate the normalized screws when necessary.

The task is to move the manipulator end-effector in a linear path from a point P_1 to a point P_2 with a constant orientation of 90° while the vehicle remains stationary. This motion takes 20s to be completed. To guarantee smooth movement, the trajectory between P_1 and P_2 is supplyed by a class which implements a 5^{th} degree polynomial trajectory generator.

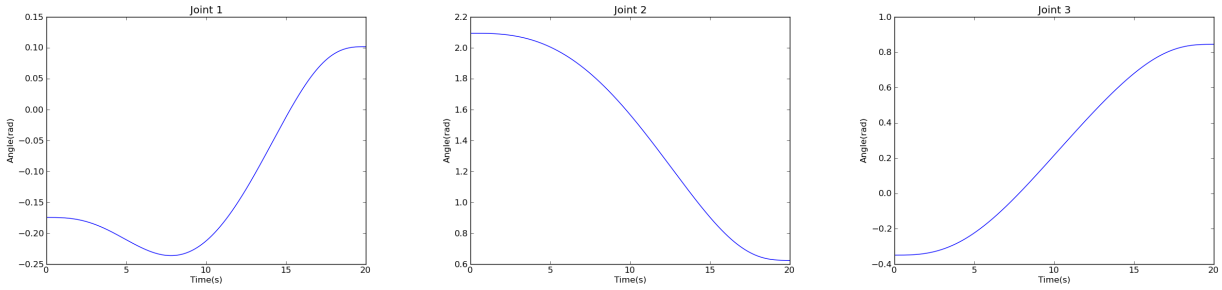
The simulation script first instantiates the trajectory generator and the kinematic chain, passing to their constructors the necessary attributes and definitions. In the simulator loop, the trajectory generator supplies position/velocity references at each iteration, which are used to define the primary variables magnitudes. Then, the `solveDavies` method is invoked to determine the secondary magnitudes (manipulator joints velocities) and after this velocities are integrated using Euler method to generate the positional information, which is used to update the chain configuration.

The simulation script interfaces with an OpenRAVE environment, where an UVMS model was build. This environment uses the simulation results to define the manipulator joint positions over time. Figure 6a shows a composition of different animation time frames. In Fig. 6b the end-effector X position component is plotted. The Y component and the orientation were constant. Joint positions are plotted in Fig 7.



(a) Motion in XY plane (b) X axis displacement

Figure 6: Task execution results - end-effector motion



(a) (b) (c)

Figure 7: Task execution results - joints positions

6. IMPLEMENTATION ANALYSIS

The simulation example presented in the last section was used as a reference for testing the framework implementation and its different possible modeling approaches. The `UVMS1v1mP` had different versions where the twists representation and the network matrix determination were implemented in alternative ways. In all these versions the simulation results were the same. This indicates that the framework has the flexibility to allow the users to decide how to define the desired modeling according to their needs.

The example kinematic chain was also used to test the `KCComposable` class implementation. Different model implementations were also considered in the simulations: kinematic chain definition in the simulation code (instantiating the kinematic chain object and using its methods to define links and joints); definition of all joints and links sharing the same attributes/variables data structure; definition of each joint and link independently and unifying them only when the kinematic chain was used in the simulation; and using a textual XML definition stored in an external file. This last variant eases the model reuse in different simulations. Also, it turns the construction of kinematic chains databases a straightforward process. The simulations using `KCComposable` instances provided similar results, and they were also similar to the simulations using the `UVMS1v1mP` class results as expected. Differences were due to imprecisions in real number computational representation.

All simulations used single kinematic chains, since the modifications to use subchains instead of single joints were not implemented so far.

Concerning the `KCComposable` class, additional tests were made to verify its flexibility to define different chain structures and its ability to obtain the constraint equation data structures automatically from these definitions. Different kinematic structures were created using the class methods. Though not used in simulations, it was observed that `KCComposable` instances were capable to build the desired kinematic chain representation, defining the necessary data structures from the motion graph created by the user and building the corresponding network matrix. These tests were

conducted in interactive mode in a Python shell.

Except for the composition of kinematic chains from subchains, it is considered that the initial requirements were achieved. The remaining requirements are in process of further analysis for implementation. Documentation is also being elaborated along with some use examples to aid the interested user.

In these initial tests, it was observed that the simulation code is more concise and systematized than similar simulation codes observed in other platforms. It was also observed that it is easier to reuse the kinematic models. Though, further comparison must be conducted with metrics to validate these observations.

Since the implementation is in its initial stages, no performance evaluations were made yet, except comparison of execution times between the specific modeling classes (`UVMS1v1mP`) and the generic `KCComposable` class where it was observed that the last case is slightly slower, as expected. As soon the framework is considered mature enough to be used for more users, these tests will be conducted and a general implementation review will be made.

7. CONCLUSION

This work presented the initial development of a computational framework to implement the screw-based kinematic modeling of robotic systems in simulations and possibly in future real implementation of motion planning estrategies. The Python platform chosen to develop the framework proved to be reliable and computationally efficient. Other works, like the OpenRAVE platform, show that Python is also a very useful interface language between applications and that there is possiblity to embed the estrategies implemented in real robotic systems. The example of a planar UVMS was used to show the use of the features implemented so far. Though simple, it demonstrated the efectiveness of the implementation. It must be noted that this is an work in progress. New features and implementations will be presented in future work.

8. ACKNOWLEDGEMENTS

This work was partially supported by Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

Booch, G., Rumbaugh, J. and Jacobson, I., 1999. *The unified modeling language user guide*. Addison Wesley Longman, Redwood City. ISBN 0201571684.

Campos, A., Guenther, R. and Martins, D., 2005. "Differential kinematics of serial manipulators using virtual chains". *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, Vol. 27, pp. 345–356.

Davies, T.H., 1981. "Kirchhoff's circulation law applied to multi-loop kinematic chains". *Mechanism and Machine Theory*, Vol. 16, No. 3, pp. 171–183.

Davies, T., 1995. "Couplings, coupling networks and their graphs". *Mechanism and machine theory*, Vol. 30, No. 7, pp. 991–1000.

Diankov, R. and Kuffner, J., 2008. "Openrave: A planning architecture for autonomous robotics". Technical Report CMU-RI-TR-08-34, Robotics Institute. URL `http://openrave.programmingvision.com`.

Guenther, R., Simas, H., da Cruz, D. and Martins, D., 2008. "A new integration method for differential inverse kinematics of closed-chain robots". In *ACBM Symposium Series In Mechatronics*, ACBM, Rio de Janeiro, Brasil, Vol. 3 of *ACBM Symposium Series*, pp. 225–235.

Hunt, K., 2000. "Don't cross-thread the screw". In *A Symposium Commemorating The Legacy, Works and Life of Sir Robert Stawell Ball Upon the 100th Anniversary of A Treatise on The Theory of Screws*. University of Cambridge - Trinity College, Cambridge University Press, Cambridge, pp. 1–37.

Metsker, S.J., 2004. *Padrões de Projeto em Java*. Bookman, Porto Alegre. ISBN 85-363-0411-1.

NetworkX, 2011. "Networkx v1.4 documentation". Available in http://networkx.lanl.gov. Last access in Feb 12nd, 2011.

Python.org, 2011. "Python programming language - official website". Available in http://www.python.org. Last access in Feb 2nd, 2011.

Rocha, C.R., Tonetto, C.P. and Dias, A., 2011. "A comparison between the denavit-hartenberg and the screw-based methods used in kinematic modeling of robot manipulators". *Robotics and Computer-Integrated Manufacturing*, Vol. In Press, Corrected Proof, pp. –. ISSN 0736-5845. doi:doi:10.1016/j.rcim.2010.12.009.

Rocha, C. and Dias, A., 2010. "Kinematic modeling of underwater vehicle-manipulator systems by use of screw theory methods". In *Proceedings of the 12th Mechatronics Forum Biennial International Conference*. IMechE, IWF - Institute of Machine Tools and Manufacturing, Zurich, Vol. 2, pp. 336–343.

Santos, C.H.F., 2006. *Movimento Coordenado de Sistemas Veículo-Manipulador Submarinos Utilizando Técnicas de Inteligência Artificial e Sistemas Híbridos*. Ph.D. thesis, Universidade Federal de Santa Catarina, Florianópolis.

Santos, C., Guenther, R., Martins, D. and De Pieri, E., 2006. "Virtual kinematic chains to solve the underwater vehicle-manipulator systems redundancy". *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, Vol. 28, pp. 354–361.

Scipy.org, 2011. "Scipy website". Available in http://www.scipy.org/SciPy. Last access in Feb 2nd, 2011.

Tonetto, C., Simas, H. and Dias, A., 2010. "Sistemática procedural para o processo de cálculo da cinemática de sistemas multirrobos cooperativos". In *Anais do VI Congresso Nacional de Engenharia Mecânica*. ABCM, Campina Grande - Paraíba.

Tsai, L.W., 2000. *Mechanism Design: Enumeration of Kinematic Structures According to Function*. CRC-Press, Boca Raton, Florida.

Tsai, L., 1999. *Robot Analysis: The Mechanics of Serial and Parallel Manipulators*. Wiley-Interscience, New York.

9. Responsibility notice

The authors are the only responsible for the printed material included in this paper.