GeDeE – An Application for Generating, Deriving and Evaluating Mathematical Expressions

RĂZVAN TUDOR TĂNASIE*, DORIAN COJOCARU**
* Software Engineering Department, ** Mechatronics Department
University of Craiova
Bvd. Decebal, Nr. 107, 200440, Craiova, Dolj
ROMANIA

Abstract: - GeDeE is an application for generating expressions and handling them, for deriving expressions and interpreting them numerically. In this paper, the GeDeE structure has been presented and its using way has been exemplified for a 3D inverse kinematics application for hyper-redundant robots. This is only one example, the GeDeE structure allowing easy adaptability for any other type of expressions application. An expressions’ generator has been built in order to store the values of the operational coordinates and then to determine an expressions’ Jacobian. Algorithms for generation and deriving expressions have been created. Derivation and decryption have been achieved using an interpreter.

Key-Words: - expressions’ generator, computation, hyper-redundant robots, control.

1 Introduction
A tentacle manipulator is a hyper-redundant or hyper-degree-of-freedom manipulator and there has been a rapidly expanding interest in their study and construction lately [1, 6]. The snake-arm robots and elephant’s trunk robots are also described as continuum robots, although these descriptions are restrictive in their definitions and cannot be applied to all snake-arm robots. A continuum robot is a continuously curving manipulator, much like the arm of an octopus. The control of these systems is very complex and a great number of researchers have tried to offer solutions for this difficult problem. Various solutions have been tried in order to obtain a fast and accurate control algorithm [7].

In a paper presented at RAAD 2007, the problem of a class of tentacle arms of variable length was discussed [8, 10]. In order to avoid the difficulties associated with the dynamic model, the control law was based only on the gravitational potential and a new artificial potential. Servoing was based on binocular vision, a continuous measure of the arm parameters derived from the real-time computation of the binocular optical flow over the two images, and is compared with the desired position of the arm.

One of the most complex aspects of the control of a hyper-redundant robot is the inverse kinematics algorithm [2, 3]. In order to solve the inverse kinematics model, we use the differential model. It is essential to create the Jacobian using the partial derivatives in respect to the generalized coordinates. The solution for such a system is much more complex than in the 2D case. Because two axes rotations are involved, thus complicating the determination of a recurrence equation is difficult [9]. For this reason, the creation of an utility application capable of generating expressions, deriving and computing the numeric values of these expressions called GeDeE. This tool is used in order to generate and estimate the Jacobian. In order to make this process even more efficient, the process parallelization technique by means of execution wires has been used – threads. This paper presents the structure and the implementation of a GeDeE (Generate Derive Evaluate).

GeDeE is an application for generating expressions and handling them, for deriving expressions and interpreting them numerically. In this paper, the GeDeE structure has been presented and its using way has been exemplified for a 3D inverse kinematics application for hyper-redundant robots. This is only one example, the GeDeE structure allowing easy adaptability for any other type of expressions application.

2 Expression Generator
The structure of the Jacobian for 3D applications is very complex and cannot be evaluated manually. For this reason, a whole mathematic system capable of generating mathematic expressions, operations using these expressions and derivates of these expressions is created. The first step in building such a mathematic system is to determine the necessary elementary structure [4]. The expressions of the operational coordinates used in determining the Jacobian for the 3D inverse kinematics algorithm for a hyper-redundant robot are:
\[
\frac{\text{Global}}{g} = \frac{m + \frac{m}{n}}{\text{total}} \prod_{i=0}^{n-1} \frac{\text{Rot}_i(t)}{a_{i,v}} \prod_{i=0}^{n-1} \text{Rot}_i(t) \left( \alpha_{arcuire} \right) + \text{Trans}_i(t) + \text{Rot}_i(t) \left( a_{arcuire} \right) \tag{1}
\]

**Definition:** It is called atom the indivisible elementary structure, in respect to the used operations, necessary for an expression generator.

In order to ease the implementation inside the application, we only used mathematical values. This is why, some mathematical functions such as sin, cos, tan will have to be illustrated by mapping numerical values. Thus, the process of generating expressions must have a new stage called numeric encryption.

**Definition:** It is called type, the integer type variable from the structure of an atom that stores the mapping code of the mathematic function expressed by that atom.

Leaving from the above definitions the structure of an atom is the following:

```
struct atom
    int type;
    float value;
```

In order to generate the expressions for the Jacobian five types of functions in expressing the operational coordinates have been identified:

- cosinus – in the rotation arrays, in handling they are given the 0 value;
- sinus – in rotation arrays and in the expression of the thread; when mapping it is given the value 1;
- the inverse of an arching angle – in the expression of the thread; when mapping it is given the value 2;
- constant – in rotation arrays and in the expression of the translation array; when mapping it is given the value 3;
- the length of the robotic segment – in the expression of the thread; when mapping, it is give the value 4.

Also, in establishing the necessary functions we must take into account the functions used after deriving the operational coordinates. Thus:

\[
\frac{\partial \cos(x)}{\partial x} = -\sin(x) \tag{2}
\]

\[
\frac{\partial \sin(x)}{\partial x} = \cos(x) \tag{3}
\]

\[
\frac{\partial 1/\sqrt{x}}{\partial x} = -\frac{1}{2x^2} \tag{4}
\]

\[
\frac{\partial a}{\partial x} = 0 \tag{5}
\]

\[
\frac{\partial \text{segment}}{\partial x} = 0 \tag{6}
\]

Derivation of any of the five function in respect to any other variable is obviously a constant, respectively 0.

From the equations (2) – (6) it results that the five functions are enough because the results obtained after derivation are either one of these functions \((0, \sin)\) or may be expressed as the product of two or three of such functions \((-\sin=-1\sin, -1 + 1 = -1, 1 = 1, x = x, x = x)\).

Once the functions established, we must also establish what each means for the assigned value:

- cosinus – the value indicates the index in the general coordinates vector of the angle whose cosinus is computed \( (1..n - arching angle, n+1..2n - rotation angle) \);
- sinus – the value indicates the index in general coordinates’ value whose sinus is computed;
- the inverse of the arching angle – the value indicates the index in general coordinates value of the used angle;
- constant – indicates the actual value of the constant;
- the length of the robotic segment – the value indicates the index of the segment whose length is used \((1..n)\).

For this class an overload of some operators was necessary in order to allow expressing the atoms operations. In this respect, the following operators have been overloaded:

- the attribution operator = - attributes both the type and the value of an atom to an other atom;
- equality operator == - checks if two atoms are equal;
- different operator != checks if two atoms are different.

**Axiom:** It is said two atoms \(a\) and \(b\) are equal if and only if they fulfill the following conditions:

- \(a\).type = \(b\).type;
- \(a\).value = \(b\).value.

**Axiom:** It is said that two atoms \(a\) and \(b\) are different if and only if at least one of the conditions in the equality Axiom is not fulfilled.

After building an elementary structure, the architecture must be developed. The form of the expressions for the inverse kinematics algorithm is one of sum of atoms’ products. As a consequence, the next structure to be built is one that will illustrate the product of \(k\) atoms. Prior knowing the fact that atoms are connected through the product operator, we chose to use a list type structure with an iterator in order to illustrate he new structure called term.
**Definition:** It is called term a simply linked list of atoms.

In order to allow working with lists, three type template classes have been implemented (Nod, List, List_Iterator). This allows the reuse of the classes even for superior levels of the architectures of the application generating expressions. Having reviewed the way of implementation using lists, we will present the structure of a term.

Thus, a term is a list of atoms, can be defined as following:

```cpp
struct termen
{
    List<atom> lista_termen;
}
```

for the term class it is necessary to verload some operators and to implement some optimization functions.

The overloaded operators for the term class are:
- the binary multiplication operator *, as friend operator of the term calss – concatenates two terms; is achieved by adding the two terms, using the binary summing operator overloaded for lists and calles the optimization function;
- equality operator == - checks if the two terms are equal;
- different operator != - checks if two terms are different.

**Axiom:** It is said about two terms a and b that they are equal if and only if they have the same number of atoms and the same atoms.

**Axiom:** It is said that two terms a and b are different if and only if the following conditions are fulfilled: they don’t have the same number of atoms; there is at least one atom in the list of one of the terms, that does not exist in list of the other.

It is considered that in a product of atoms, one of these atoms can be zero. If it is the case, the whole term becomes zero and it is no longer necessary to retain a whole list of atoms for those terms, but only the atom constant zero. In this respect, there has been implemented an optimization function checkForZeros that does exactly that thing, thus easing ulterior calculations.

This being the structure and the implementation of a term, we shall present the last level of the architecture. As previously presented, an expression is a sum of products, in other words, a sum of terms. As a consequence, the next structure that must be built is one to illustrate the sum of \( P \) terms and which will be called element.

Prior knowing that the operator linking these terms is the sum, we may chose to represent such a structure in using a list of terms.

Aiming this, we shall use three template classes already projected, implemented and used in order to represent the atoms’ lists inside the terms.

An element is defined as following:

```cpp
struct element
{
    List<termen> lista_element;
}
```

For the element class it is necessary to overload some operators and to implement some optimization function.

The overloaded operators for the element class are:
- the binary sum operator +, as friend operator of the element class – concatenates two elements; is achieved by adding the lists of the two elements;
- the binary product operator *, as friend operator of the element class – combines two elements.

**Axiom:** It is said about an element \( c \) that it is the result of the combination of two elements \( a \) and \( b \), if and only if the terms of the \( c \) element are the result of the Cartesian product between the lists of terms of the \( a \) and \( b \) elements.

In an element there may come up even constant zero terms that do not influence the summation operation. That is why they can be eliminated. In this respect, an optimization function deleteZeros has been created. It deletes all the terms containing the constant zero atom from the terms’ list of an element. This function is called for each of the two friend operators of the element class.

The expressions must then be derived in respect to the generalized coordinates in order to obtain the Jacobian. For this reason, it is necessary to built an other function that derivates an element in respect to a certain generalized coordinate – derive. This function returns the result of the derivation. Its manner of projection and implementation will be presented in the next paragraphs. The general structure of an expression crated using the system architecture is presented in Fig 1. Having this system generating expressions, we can present its usage and the steps of an algorithm for the determination of the Jacobian for 3D inverse kinematics applications. The implemented algorithm consists in three steps:

1. the structure of the hyper-redundant robot, its initial position and the target position of the gripper are read.
2. for each iteration, the mathematical model built for direct kinematics applications is used in order to determine the rotation and translation arrays and the position vector.
3. each array and vector are encrypted, thus generating a type element expression for each element of the array or vector, obtaining arrays and vectors of elements.
4. computations are done using the expressions generator and obtaining the expressions of the operational coordinates.
5. the expressions are derived in respect to each generalized coordinate, thus generating the Jacobian under the shape of an expression.
6. the Jacobian is decrypted “translating” each expression and evaluating it numerically.
As it can be easily depicted, this algorithm is a highly computational one. For this reason, we chose to implement it using execution wires, fact that will make parallel many processes, allowing a much shorter execution time. The parallel processes have been possible and achieved in such a manner as two allow the user to choose how many execution wires to be created, according to the computing power at his disposal.

Obviously, the efficiency of a parallelization process depends on the hardware available. For a single processor computer that does not allow multi-threading, the efficiency of such parallelization is null.

### 3 Expression Derivator and Evaluator

The first stage is building the expression Jacobian. In this respect, each element of the array or of the vector, must be transformed in a type element variable. We shall treat separately the array generation from the vector generation.

The arrays used are either rotation arrays or translation arrays. The translation arrays illustrate a translation along the OY axes of the coordinate system with the length of the thread determined by the segment to whom the coordinate system is attached. Thus, the values that may come up in such an array are:

- 1 and 0;
- sinus – for expressing the lengths of the thread;
- the inverse of the arching angle – for expressing the thread length;
- the length of a segment – for expressing the length of the thread.

Values 1 and 0 determine the elements of the array. Thus, for each of them, we shall create one element that contains one term which, in turn contains only one constant atom with the value 0 or 1.

For the case of the thread length, we shall create an element that contains three atoms, one for sinus, the arching angle inverse and the segment length. In the exemplification, we shall illustrate the way a translation array expressed for the 3 segment of hyper-redundant robot looks like. Taking into account that on each position of the array there is only one element, we shall only present the lists of atoms.

\[
\begin{align*}
\text{type} & \quad \text{value} \\
\text{type} & \quad \text{value} \\
\text{type} & \quad \text{value} \\
\text{type} & \quad \text{value} \\
\text{type} & \quad \text{value} \\
\text{type} & \quad \text{value} \\
\text{type} & \quad \text{value} \\
\text{type} & \quad \text{value} \\
\end{align*}
\]

Rotation arrays illustrate a rotation around the OY axes of the actual coordinates system with the rotation angle of the segment to whom the coordinates’ system is...
attached, or a rotation around the OZ axes of the actual coordinate system with an arching angle of the segment to whom the coordinates’ system is attached. Consequently, the values that may come up in such an array are:

- 1 and 0;
- sinus – in expressing the rotation;
- cosinus – in expressing the rotation.

The cosinus, sinus, 1 and 0 values determine the elements of the array. Thus, for each of them, we shall create a term that, in turn, contains only one constant atom with the value 0 or 1, or a cosinus or sinus atom, according to case.

For -sinus we shall create an element with a term that contains two atoms, one for sinus and one for -1. For exemplification we shall illustrate the way a rotation array around the OY axes expressed in expressions for the 3 segment of a 5 segments’ hyper-redundant robot looks like. Taking into account the fact that for each position in the array, there is only one element with only one term, we shall only present the atoms’ lists.

In order to exemplify, we shall illustrate a position vector expressed using the expressions for hyper-redundant robot with 4 segments. Taking into account the fact that for each position in the vector, there is only one element with one term, we shall only present the atoms lists.

\[
\begin{pmatrix}
\text{type}0 & \text{type}3 & \text{value}8 & \text{value}0 & \text{type}4 & \text{value}4 \\
\text{type}3 & \text{type}3 & \text{value}3 & \text{value}8 & \text{type}3 & \text{value}0 \\
\text{value}0 & \text{value}1 & \text{type}3 & \text{type}0 & \text{type}3 & \text{value}8 \\
\text{type}1 & \text{value}0 & \text{value}0 & \text{value}8 & \text{value}0 & \text{type}3 \\
\end{pmatrix}
\] (8)

Once formulated all the matrices and the position vector, they are multiplied according to the mathematical model. In achieving these products, the functions of the element, term and atom classes are used. The final result obtained is a vector expressing the operational coordinates under the shape of expressions.

Using the operational coordinates formulated using expressions, we may pass to computing the Jacobian. As each operational coordinate is an element, we shall use for the derivation the derive function in the element class.

The derivation of an element is done giving the indices of the generalized coordinate after derivation is achieved. Then, the terms in the list are skimmed through one by one, generating one term for each atom of each term.

The derivation algorithm is the following:

1. the indices of the generalized coordinates according to which derivation is made, are read;
2. a new element is created, which momentarily contains only one constant 0 atom;
3. a list iterator is initialized on the first position of the list of terms of the element to be derived;
4. the current element in the list is read;
5. a list iterator on the first position of the atoms’ list of the current element is initialized;
6. if the value of the current atom is not equal to the indices in respect to which derivation is done, it results that it is not dependent of the angle, and thus the result of the derivation is 0. consequently, there is no added term to the term list;
7. if the value of the current atom is equal to the indices of the generalized coordinate in respect to which the derivation is done and its type illustrates that there is a cosinus, a sinus or the inverse of an angle, the mathematical function is derived, anew term is built which is equal to the current term and where the current atom is substituted through the atom or atoms obtained after derivation (cosinus derived generates two atoms, a constant atom -1 and a sinus atom). This term is added to the terms’ list of the element;
8. if the value of the current is equal to the indices in respect to which derivation is done and its type illustrates that a constant or a segment length is involved, it means that it does not depend on this angle.
and the result of the derivation is 0. Consequently, there is no added term in the element list;

9. we check if there are any atoms in the list of the current term. If there are any, we move forward one position in the atoms’ list and we return to step 6;
10. we check if there are any terms in the term list. If there are any, we move forward one position in the terms’ list and we return to step 5;
11. the optimization function of the element deleteZeros is called.

It can be easily noticed the fact that the algorithm is double iterative and the exit from the cycle is only possible after all the atoms of all the terms in the term list have been skimmed through. The final step, step 11 brings optimization through the elimination of the terms formed only of the constant 0 atom.

Based on the expression Jacobian we may determine the numerical Jacobian necessary in determining the generalized coordinates. In order to decrypt these expressions an expression interpreter has been implemented. This interpreter has a relatively poor language (five types of atoms, \(2n\) angles as values, \(n\) segment length and constant values) and two mapping maps.

The interpretation algorithm skims through the lists of terms and atoms and for each atom, it interprets based on mapping maps and numerically evaluates the decrypted functions, and sums up the result to the value of the current position of the numerical Jacobian.

Once the numerical Jacobian created, the numerical pseudo-inverse of the Jacobian can be calculated and then, the generalized coordinates of the hyper-redundant robot can be computed.

4 Conclusion

GeDeE is and application for generating expressions and handling them, for deriving expressions and interpreting them numerically. In this paper, the GeDeE structure has been presented and its using way has been exemplified for a 3D inverse kinematics application for hyper-redundant robots. This is only one example, the GeDeE structure allowing easy adaptability for any other type of expressions application.

An expressions’ generator has been built in order to store the values of the operational coordinates and then to determine an expressions’ Jacobian. The used structures are illustrated in Figure 1. these structures (atom, term, element) have used template classes in order to store the lists of objects or in order to access them using iterators. Algorithms for generation and deriving expressions have been created. Derivation and decryption have been achieved using an interpreter.

Aiming to achieve all these operations, the overload of several operators and the creation of some optimization functions have been necessary.

The projected and developed algorithms are highly computational and, for this reason, we chose to implement them using execution threads, fact which will parallelize most of the processes, thus allowing a shorter execution time [5]. Parallelization is possible and achieved in such a manner as two allow the user to choose how many execution wires to be created, according to the computing power at his disposal.

In order to determine the values of the Jacobian an interpreter is used in order to decrypt the expressions and to obtain the numerical values of the Jacobian. This interpreter has a relatively poor language (five types of atoms, \(2n\) angles as values, \(n\) segment length and constant values) and two mapping maps that allow the “translation” into a mathematical format of the atoms, leaving from their types and values.

References: