REDUNDANT INVERSE KINEMATICS SYSTEM FOR OBSTACLES AVOIDANCE

Aurel FRATU[*], Jean-François BRETHÉ[**], Mariana FRATU[*]
[*] Transilvania University of Brasov, Romania
[**] Université du Havre, France

Abstract. In this paper the authors relate to the robotic systems to avoid the obstacles while positioning end-effectors. A new strategy to on-line collision-avoidance of the redundant robots with obstacles is presented. The strategy allows the use of redundant degrees of freedom such that a manipulator can avoid obstacles while tracking the desired end-effectors trajectory. The strategy is based on the redundant inverse kinematics and leads to the favourable use of the abilities of redundant robots to avoid the collisions with obstacles. The effectiveness of the proposed method is discussed by theoretical considerations and illustrated by simulation of the motion of the four-joint planar manipulators between symmetric obstacles.

Keywords: redundant inverse kinematics, avoid collision, motion simulation

1. Introduction

The objective of the inverse kinematics problem is to compute joint-space variables corresponding to the assigned task-space variables. This is the objective of the inverse kinematics problem [1].

In this paper, the problem of redundant inverse kinematics is reviewed and obstacles avoid subtask for exploiting the self-motion are defined.

Cartesian position and orientation for the end-effectors task vector x_t can be described as a function of joint variables vector $\boldsymbol{q}$ of the manipulator:

$$\boldsymbol{x}_t = f(\boldsymbol{q}) \tag{1}$$

While equation (1) can be obtained easily with the help of Denawit-Hartenberg operators, the inverse problem is crucial. In the redundant case it is generally not possible to find an inverse mapping, f^{-1}.

As an alternative of constructing an inverse mapping, the problem is often reformulated in the velocities, utilizing the partial derivation of $f(q)$.

The end-effectors task velocity vector $\dot{\boldsymbol{x}}_t$ is:

$$\dot{\boldsymbol{x}}_t = \boldsymbol{J}_t(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{2}$$

with Jacobian matrix:

$$\boldsymbol{J}_t = \partial\, \boldsymbol{f}(\boldsymbol{q}) / \partial \boldsymbol{q} \tag{3}$$

Due to the fact that the inverse of the non-square (analytical) Jacobian $\boldsymbol{J}_t(q)$ does not exist in the redundant case, the well known generalized Moore–Penrose pseudo inverse $\boldsymbol{J}_t^+(q)$ is utilized. This proposed strategy often employs a special solution of equation (2).

Optimization criteria for the redundant self motion can be supplementary by e.g. null-space projection, which leads to the relation:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_t^{\,+}\dot{\boldsymbol{x}}_t + (\boldsymbol{I} - \boldsymbol{J}_t^{\,+}\boldsymbol{J}_t)\dot{\boldsymbol{q}}_0 \tag{4}$$

The expression $(\boldsymbol{I} - \boldsymbol{J}_t^{+}\boldsymbol{J}_t)$ represents the orthogonal projection matrix in the null space of $\boldsymbol{J}_t$, and $\dot{\boldsymbol{q}}_0$ is an arbitrary joint-space velocity; the second part of the solution is therefore a null-space velocity.

The vector $\dot{\boldsymbol{q}}_0$ can be chosen arbitrary and is used to force a desired behaviour for the null space motion. The null space of $\boldsymbol{J}_t$(q) is defined as [2]:

$$\boldsymbol{N}(\boldsymbol{J}_t(\boldsymbol{q})) = \{\dot{\boldsymbol{q}} \in \dot{\boldsymbol{Q}} : 0 = \boldsymbol{J}_t(\boldsymbol{q})\dot{\boldsymbol{q}}\} \tag{5}$$

In redundant directions joint velocities causes no motion at the end-effectors level. These are internal motions of the manipulator. Redundant joint velocities satisfy the equation:

$$\boldsymbol{J}_t\,(\boldsymbol{q})\;\; \dot{\boldsymbol{q}} \;=\; 0 \tag{6}$$

At each configuration, the null space of J_t is the set of joint-space velocities, which yield zero task velocity; these are thus called null-space velocities.

2. Augmented Jacobian for redundant robots

The redundant robots can operate in the Cartesian space with obstacles. The redundant self motion can satisfy both, the end-effector task and the additional constraint task cause of the obstacles, at the same time.

The concept of task-space augmentation introduces a constraint task to be fulfilled along with the end-effector task [3]. In that case, an augmented Jacobian matrix is set-up whose inverse gives the required joint velocity solution.

Let us consider the *p*-dimensional vector $\boldsymbol{x}_\mathrm{c} = (x_{\mathrm{c},1} \ . \ . \ . \ x_{\mathrm{c},p})$ which describes the additional tasks to be fulfilled in addition the *m*-dimensional end-effector task vector $\boldsymbol{x}_\mathrm{t}$, i. e., $p = n - m$, whereas *n* is the number of joints.

The relation between the joint-space coordinate vector $\boldsymbol{q}$ and the constraint-task vector $\boldsymbol{x}_\mathrm{c}$ can be considered as a direct kinematics equation:

$$\boldsymbol{x}_\mathrm{c} = \boldsymbol{k}_\mathrm{c}(\boldsymbol{q}) \qquad (7)$$

where $\boldsymbol{k}_\mathrm{c}$ is a continuous nonlinear vector function. By differentiating (7) one can be obtained (8):

$$\dot{\boldsymbol{x}}_\mathrm{c} = \boldsymbol{J}_\mathrm{c}(\boldsymbol{q})\dot{\boldsymbol{q}} \qquad (8)$$

In (8) $\dot{\boldsymbol{x}}_\mathrm{c}$ is the constraint-task velocity vector, and the mapping $\boldsymbol{J}_\mathrm{c}(\boldsymbol{q}) = \partial \boldsymbol{k}_\mathrm{c}/\partial \boldsymbol{q}$ is the (p x n) constraint-task Jacobian matrix.

At this moment, an augmented-task vector $\boldsymbol{x}_\mathrm{a}$, can be defined by stacking the end-effector task vector with the constraint-task vector as:

$$\boldsymbol{x}_a = [\, \boldsymbol{x}_t \ \ \boldsymbol{x}_c \,]^T = [\ \boldsymbol{k}_t(\boldsymbol{q}) \ \ \boldsymbol{k}_c(\boldsymbol{q}) \ ]^T \qquad (9)$$

According to this definition [3], finding a joint configuration $\boldsymbol{q}$ that result, in some desired value for $\boldsymbol{x}_\mathrm{a}$, means satisfying both the end-effector task and the constraint task, at the same time.

At the differential level, Cartesian velocities are given by:

$$\dot{\boldsymbol{x}}_\mathrm{a} = \boldsymbol{J}_\mathrm{a}(\boldsymbol{q})\dot{\boldsymbol{q}} \qquad (10)$$

To finding a joint velocity $\dot{\boldsymbol{q}}$ in some desired value for Cartesian velocities $\dot{\boldsymbol{x}}_\mathrm{a}$ can be used:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_\mathrm{a}{}^{+}\dot{\boldsymbol{x}}_\mathrm{a} + (\boldsymbol{I} - \boldsymbol{J}_\mathrm{a}{}^{+}\boldsymbol{J}_\mathrm{a})\dot{\boldsymbol{q}}_0 \qquad (11)$$

That means the inverting of the augmented Jacobian, $\boldsymbol{J}_\mathrm{a}(\boldsymbol{q}) = [\ \boldsymbol{J}_\mathrm{t}(\boldsymbol{q}) \quad \boldsymbol{J}_\mathrm{c}(\boldsymbol{q}) \ ]^\mathrm{T}$ to obtain the pseudo-inverse matrix, $\boldsymbol{J}_\mathrm{a}{}^{+}$.

3. Task priority strategy

With reference to solution (11), the task-priority strategy consists of computing $\dot{\boldsymbol{q}}_0$ so as to correctly achieve the *p*-dimensional constraint-task velocity, $\dot{\boldsymbol{x}}_\mathrm{c}$.

In the typical case, an end-effector task is considered as the primary task, even if examples may be given in which it becomes the secondary task [4]. The idea is that, when an exact solution does not exist, the reconstruction error should only affect the lower-priority task.

Conflicts between the end-effector task and the constraint task are handled in the framework of the task-priority strategy by correctly assigning an order of priority to the given tasks and then satisfying the lower priority task only in the null space of the higher-priority task. Remarkably, the projection of $\dot{\boldsymbol{q}}_0$ onto the null space of $\boldsymbol{J}_\mathrm{t}$ ensures lower priority of the constraint task with respect to the end-effector task because this results in a null-space velocity for the higher-priority task .

When the secondary task $\dot{\boldsymbol{x}}_\mathrm{c}$ is orthogonal to the primary task $\dot{\boldsymbol{x}}_\mathrm{t}$ the joint velocity:

$$\dot{\boldsymbol{q}}_0 = \boldsymbol{J}_\mathrm{c}{}^{+}(\boldsymbol{q})\dot{\boldsymbol{x}}_\mathrm{c} \qquad (12)$$

would easily solve the problem, being in addition already a null-space velocity for the primary-task velocity mapping (2).

However, in the general case the two tasks may be compatible but not orthogonal or may conflict and there not exist a joint velocity solution that ensures the achievement of both $\dot{\boldsymbol{x}}_\mathrm{t}$ and $\dot{\boldsymbol{x}}_\mathrm{c}$. Coherently with the defined order of priority between the two tasks, a reasonable choice is then to guarantee exact tracking of the primary-task velocity, while minimizing the constraint-task velocity reconstruction error, $\boldsymbol{\varepsilon} = \dot{\boldsymbol{x}}_\mathrm{c} - \boldsymbol{J}_\mathrm{c}(\boldsymbol{q})\dot{\boldsymbol{q}}$; this gives [3]:

$$\dot{\boldsymbol{q}}_0 = [\ \boldsymbol{J}_\mathrm{c}(\boldsymbol{I} - \boldsymbol{J}_\mathrm{t}{}^{+}\boldsymbol{J}_\mathrm{t}) \ ]^{+}\left(\dot{\boldsymbol{x}}_\mathrm{c} - \boldsymbol{J}_\mathrm{c}\boldsymbol{J}_\mathrm{t}{}^{+}\dot{\boldsymbol{x}}_\mathrm{t}\right) \qquad (13)$$

The solution given by (4) and (13) can be simplified to [3]:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_\mathrm{t}{}^{+}\boldsymbol{x}_\mathrm{t} + [\ \boldsymbol{J}_\mathrm{c}(\boldsymbol{I} - \boldsymbol{J}_\mathrm{t}{}^{+}\boldsymbol{J}_\mathrm{t}) \ ]^{+}\left(\dot{\boldsymbol{x}}_\mathrm{c} - \boldsymbol{J}_\mathrm{c}\boldsymbol{J}_\mathrm{t}{}^{+}\dot{\boldsymbol{x}}_\mathrm{t}\right) \qquad (14)$$

Another approach is to relax minimization of the secondary-task velocity reconstruction constraint and simply pursue tracking of the components of (12) that do not conflict with the primary task [3], namely

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_t^{+} \dot{\boldsymbol{x}}_t + (\boldsymbol{I} - \boldsymbol{J}_t^{+} \boldsymbol{J}_t) \boldsymbol{J}_c^{+} \dot{\boldsymbol{x}}_c \qquad (15)$$

An intuitive justification of this solution can be given as follows: The pseudo inverses $\boldsymbol{J}_t^{+}$ and $\boldsymbol{J}_c^{+}$ are used to solve separately for the joint velocities associated with the respective task velocities. The joint velocity associated with the (secondary) constraint task is then projected onto the null space of $\boldsymbol{J}_t$ to remove the components that would interfere with the (primary) end-effector task. Finally the joint velocity associated with the constraint task is added to the joint velocity associated with the end-effector task.

By construction, the solution (15) leads to larger constraint-task reconstruction errors than solution (14); this is the price paid to give smooth and feasible trajectories for the joint velocity in tracking conflicting tasks.

4. Delphi simulators for collision avoidance problem

The authors propose the Delphi informatics environment, for the qualitative simulation of the robotic systems, using the visual programming. The prototype graphical simulation system has been developed to support and demonstrate different aspects of robots behaviour.

Delphi is a visual programming tool, which makes possible to carry out, in a practically instantaneous way, the interface with Windows applications. Having a very intuitive access, this extremely rich tool offers with effortlessness the management of an environment as sophisticated as Windows.

In traditional robotics, programmers attempted to anticipate and explicitly control every aspect of the action of the robot. However, these systems often failed when are placed in a changing environment, where unanticipated situations appear.

Systems using autonomous agents, with de-centralized control of robot motion, enable robots to learn and adapt to changing environments.

Advanced robotic systems involve large agent-based simulators aimed at understanding "action selection". By this mechanism, the robot selects the behaviour to execute from much possible behaviour.

Delphi simulators are generally smaller programs, and they are written in order to extend research in Robotics. They usually involve algorithms that allow exposing the behaviour of the virtual robots and their environment. This technique can also be used to provide a method of robotic system command.

All of these programs enable the manifestation of global behaviours. They also attempt to simulate more aspects of robot behaviour - not just movement. As such they include the movement of robot manipulators, the contact with manipulated objects and the collision with unexpected objects into the work environment.

The information regarding the collision are given about the shape of each body and then it figures out which bodies touch each other.

The Delphi programs are written with an object-oriented approach. The simulation is driven by a special object, the Object Inspector.

The Object Inspector maintains a list of the active object and sends step messages when it is time for objects to update themselves.

All components of the artificial world are objects, as is the environment itself. The interface consists of a collection of tools for analysis and visualization.

Data analysis objects can communicate with the user interface objects to present displays of data, as well as for storage to files. The prototype allows the user to create, modify, and destroy objects. Each object has a few standard attributes managed by Object Inspector, as well as user-specified private data [6].

In this paper, the graphical simulation prototype system has been developed to support and demonstrate different aspects of robots behaviour when met the obstacles in his work environment. All of these programs are write in Delphi language and enable the manifestation of global behaviours.

They also attempt to simulate more aspects of robot behaviour, not just movement.

For itself they include the movement of virtual robot manipulator, the contact with virtual objects and the collision avoid, into the work environment, with expected or unexpected virtual objects.

The information regarding the collision are given about the shape of each face of the body and then it figures out which bodies face touch each other and passes, the resulting contact point visual information, to designer. The designer can then take the proper decisions concerning the real prototype.

The simulation methodology uses a number of possible scenarios and involves some sequences of animation, as one can see in the figures 1, 2 and 3, for a virtual system.

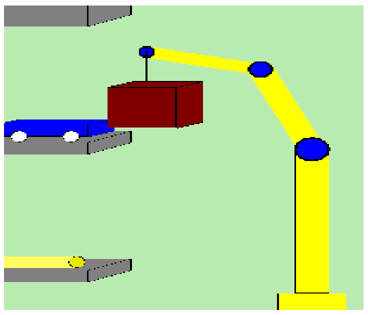
Figure 1. Beginning pose of the animation sequence

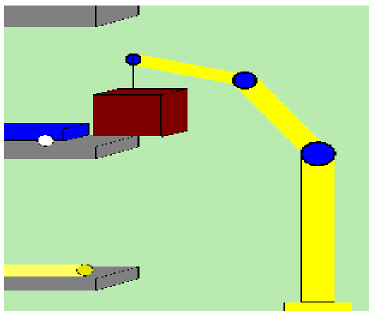
Figure 2. Middle pose of the animation sequence

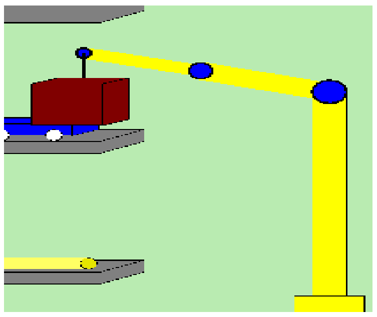
Figure 3. Ending pose of the animation sequence

5. Conclusions

This paper has considered a redundant inverse kinematics model based on the augmented Jacobian that enables the use of self-motion of the manipulator to perform as sub-task the obstacles avoidance.

According to the task-splitting concept in (primary) end-effector task and (secondary) constraint task, the concept was demonstrated on a virtual system. Interaction between the virtual robot and virtual obstacle was demonstrated, while performing end-effector tasks on the robot application. The virtual robot is able to react to avoid the obstacles while executing the primary task. This kind of robot's behaviour simulation will be necessary in real applications, where interactions between robots and his environment are vital.

If we combine the command strategy of virtual robot with the command strategy of real robot, one can avoid the collision with the obstacles. We can apply a command strategy from virtual system to real system without any complicated control theory or position sensors. Moreover, if we use different shaped obstacles, it's very difficult to find proper control rules, but we can use the virtual robot learning by applying this strategy to get the satisfactory result with the real robot arm. The real robot will imitate the virtual robot.

We wrote a program that allow to the end-effector of the virtual robot to wander around without bumping into obstacles in its path.

In this paper, we implemented a very simple program that worked effectively. In the virtual environment not all simulation programs need to be complex to be effective. It is easy to imagine that the software algorithms for real robot are highly complexity.

References

1. Siciliano, B., Khatib, O.: *Handbook of Robotics*. Springer-Verlag, ISBN: 978-3- 540-23957-4, Berlin Heidelberg, 2008
2. Shengwu, L., Shaheen, A.: *Mathematical Analysis of the Joint Motion of Redundant Robots Under Pseudo-Inverse Control*. ECE Technical Reports, Purdue Libraries, March 1992, Purdue University West Lafayette, IN 47907, USA
3. Siciliano, B.: *Kinematic Control of Redundant Robot Manipulators: A Tutorial*. Journal of Intelligent and Robotic Systems, ISSN 0921-0296, Vol. 3, no. 3, p. 201-212, 1990
4. Chiacchio, P., Chiaverini, S, Sciavicco, L, Siciliano, B: *Closed-loop inverse kinematics schemes for constrained redundant manipulators with task space augmentation and task priority strategy*. The International Journal of Robotics Research, Vol. 10, no. 4, p. 410-425, 1991
5. Schreiber, G., Ott, Ch., Hirzinger, G.: *Interactive Redundant Robotics: Control of the Inverted Pendulum with Nullspace Motion*. Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, Maui, Hawaii, USA, Oct. 29 - Nov. 03, 2001
6. Fratu, A., Vermeiren, L.: *Adaptive control architecture for redundant robots*. RECENT, Vol. 8, no. 2(20), 2007, p. 97-100, ISSN 1582-0246, Brasov, Romania

Received in February 2010