Task priority assignment with collision avoidance.

Stefano De Filippis & Marco Menchetti

October 2019

Abstract

In this paper we will face the problem of task priority resolution using a fast computation of the priority matrix (here *Flacco Matrix*) and the resulting joint velocities.

Collision avoidance for several control points has been taken as a high priority task in this case as well as trajectory tracking.

Introduction

Due to their high dexterity and the absence of non-holonomic constraints, manipulators has been used to perform a wide range of operation, and sometimes even more of them at the same time.

A handy yet practical example is the one considered below: a manipulator moving in a cluttered environment, trying to complete a trajectory tracking task and, at the same time, avoid collision with obstacles nearby.

1 Tasks definition

The tasks we used are four and they occupy all 7 manipulator's DOF:

- one carthesian positioning task occuping 3 DOF.
- one carthesian orientation task for the second link, which occupies 2 DOF.
- two control points' collision avoidance tasks that will occupy complessively 2 DOF.

1.1 Carthesian positioning

The carthesian positioning task is defined by the direct kinematics of the robot, $f(q)$, so the **ed-effector**'s position is $r = f(q)$. Hence it's straight-forward the expression of this task's velocity with respect to the joints' one:

$$\dot{r} = \frac{\partial f(q)}{\partial q}\dot{q} = J\dot{q} \tag{1}$$

1.1.1 Collision avoidance

In the formulation of our problem, where $\dot{r}$ is given (i.e. precomputed), we are left with finding the right value for $\dot{q}$.

As already said, in our approach, we have also to include the collision avoidance for the end-effector. Instead of treating it as a different task, as we will do for the other control points, we could handle it in a "tricky" way so as to not saturate other DOFs: we will use instead of $\dot{r}$, the **sum** between $\dot{r}$ and another carthesian velocity pushing the end-effector away from the obstacle. This carthesian velocity will be (or TODO: add citation i.e. as in [1])directed as the distance from the center of the obstacle to the tip of the manipulator, and will have a magnitude weighted by a non linear gain $v(P, O)$, where $P = f(q)$ is the end-effector position and O is the obstacle position. Hence:

$$\dot{r_o} = v(P, O)\frac{f(q) - O}{\|f(q) - O\|} \tag{2}$$

$$v(P, O) = \frac{V_{max}}{1 + e^{(\|f(q)-O\|(2/\rho)-1)\alpha}} \tag{3}$$

In the end we will have (1) in the form:

$$\dot{r} + \dot{r_o} = J\dot{q}$$

1.2 Carthesian orienting

TODO: all again and check if the equation is right

1.3 Control points' collision avoidance

When dealing with the collision avoidance task linked to the two control points we were left with only 2 DOFs for both so we had to use one for each point. We couldn't use the same approach we used for the end-effector but at the same time something rather similar has to be done.

To sqash the three DOFs into one, we projected the collision avoidance task velocity, $\dot{\mathbf{r}_{\mathbf{o},\mathbf{i}}}$ computed as in (2) but using the control point position as P, onto the direction of the velocity itself. In this way we get a task velocity which is a scalar (1 DOF) equal to the magnitude of the original one (i.e. (3)).

Defining

- $\eta = \frac{P-O}{\|P-O\|}$
- P as the control point's position
- O as the position of the obstacle
- J_i as the analytical jacobian associated to the i-th control point

we end up with:

$$\eta^T \dot{r}_{o,i} = v(P, O) = \eta^T J_i \dot{q} = J_{c,i}\dot{q}$$

Hence:

$$v(P, O) = J_{c,i}\dot{q} \tag{4}$$

2 Control architecture

Due to the high complexity of the task we divided our control scheme into 3 main blocks:

1. **Task priority matrix:** to compute in a fast way the joint velocities executing the task velocities coming from the prioritized stack of tasks.
2. **Priority resolution algorithm:** to organize the stack of tasks depending on the each ones' *generalized cost*. This concept will be further explained above.
3. **Control algorithm:** to generate the reference joint velocity that the manipulator should execute. TODO: fix this

2.1 Task priority matrix

TODO: stefano

2.2 Priority resolution algorithm

We defined the stack of tasks as: TODO

2.2.1 Cost definition

TTTTHE DISTANCE: TODO.

2.3 Control algorithm

We switch every ”n” so as: TODO

3 Code

4 Results

Contents