ROBOT CONTROL FOR AVOID SELF-COLLISIONS

Aurel FRATU, Mariana FRATU
Transilvania University of Brasov, Romania

Abstract. The present paper generally relates to robots, more specifically to robots acting autonomously and requiring a safety method to avoid collisions with other objects as well as collision between one moveable segments of the robot. The authors have tested an avoid collisions method on complex simulation scenarios involving a virtual robot and have estimated collision-free actions. The virtual robot is able to react to avoid the obstacles while executing the robotic tasks. This was shown experimentally with a virtual system. This kind of robot's behaviour simulation will be necessary in real applications, where interactions between robot and her environment are vital.

Keywords: self-collision prevention method, collision avoidance control, dynamic simulation protocol

1. Introduction

When a trajectory of a robot end-effector is controlled, a target state needs to be defined. The target state is, for example, defined by an object handled by a manipulating arm of a robot. In general, the position of the object can be described by three parameters. In addition to the object position, it is often necessary to describe a spatial orientation by Euler-angles for example,

To carry out the movement of an end-effector of a robot, the trajectory is usually generated by mapping increments from a control parameter space as characteristic point, on a configuration space. In robotics, the characteristic point is often defined as a reference point of the end-effector.

The control parameter space or task space is the space of the command elements. The control parameter space is composed of the command elements. The command (also "target" or "task") elements are the elements of a respective command vector. These elements define a useful description about what should be controlled, for example, the position and the orientation of an end-effector. The configuration space is the space of controllable degrees of freedom. The configuration space can be composed of individual joints of a robot and/or more complex kinematics mechanisms having controllable degrees of freedom.

Task coordinates refer to coordinates in which the movement of the effector is described. There are many ways to describe end-effector motions.

To describe the positions of the effectors, the x, y and z elements of a position vector are commonly chosen. For spatial orientations, the task is often described in Euler angles or quaternions. In many cases, special descriptions for a task are used.

Task space refers to the space that is described by the task coordinates. For example, if the hand position of a robot is controlled in x, y and z direction, the task space has a dimension of three and is described by these coordinates.

Trajectory is a continuous path describing the motion of a system. The trajectory can describe the path of the individual joints or a path represented in the task coordinates.

Conventionally, targets of robots were provided by operators and the robots simply tracked planned trajectories. The safety method in conventional robots is an emergency stop method that simply freezes motions of the actuators, if any motions were being performed.

Modern robots, in particular anthropomorphic robots, are expected to work outside the typical environment such as factories. The modern robots need to interact with dynamic environment that is less predictable. Thus, there is a need for a more advanced safety method, hereinafter referred to as a collision avoidance method instead of the emergency stop method.

The advantage of using the collision avoidance is not only safety. The collision avoidance does not necessarily stop the robot's target reaching motions and may expand the robot's working range.

2. Collision avoidance methods

The known collision avoidance methods may be divided into two categories. One category of collision avoidance method is a planning (non real-time) method which generates trajectories taking obstacles into account.

These methods are difficult to apply to interactive motions because the computation time becomes longer as the degrees of freedom of robots (e.g., anthropomorphic robots) increase.

Another category of the collision avoidance is reactive (real-time) collision avoidance methods.

The reactive collision avoidance modifies trajectories that are quite simple such as line segments connecting current positions and target positions [1].

In order to use this method, the direction of avoidance and how to switch the priority between target reaching motions and collision avoidance motions must be decided depending on the magnitude of danger of collisions in real-time. For instance, if the distance between segments is large enough, target reaching motions should have higher priority than collision avoidance motions.

2.1. Efficient collision avoidance technique

The motion control of the robot includes a collision avoidance module designed for calculating the two closest points of different segments of the robot connected to each other via at least one joint or two closest points of a segment of the robot and another object. The collision avoidance module also controls the collision avoidance action only in the dimension along the line connecting between the two closest points.

It is an objective of the present paper to provide an efficient collision avoidance technique for a robot which minimizes interferences with the task execution of the robot.

In the strategy for controlling a robot, a target for a motion of the robot is defined. A motion control signal adapted for the robot reaching the target is calculated.

A collision avoidance control signal based on the closest points of different segments of the robot connected to each other via at least one joint or a segment of the robot and another object is calculated.

The influence of the motion control signal and the influence of the collision avoidance control signal are combined.

The influence of the motion control output signal is higher when a calculated collision risk is lower. The influence of the collision avoidance control output signal is higher when the calculated collision risk is higher.

The motion of the robot is controlled according to the combination of the signal's influence. A soft task switching between target reaching by motion and collision avoidance is achieved by gradually changing the influence.

The influence of the collision avoidance output signal remains zero as long as the distance between the closest points is larger than a preset avoidance threshold distance.

The motion control module of the robot includes a collision avoidance module for calculating the two closest points of different segments of the robot connected to each other via at least one joint or a segment of the robot and another object.

The motion control module also controls avoidance action only in the dimension along a line connecting between the two closest points.

One characterization of the present paper also provides a computer program product embedded in a robot for implementing a motion control unit.

The motion control unit includes, among others, a distance computing module, a motion control unit, a collision avoidance module, and a blending control unit.

2.2. The distance computing module

The distance computing module calculates the two closest points of different segments of the robot connected to each other via at least one joint or a segment of the robot and another object.

The collision avoidance module is provided with an output signal from the distance computing module. The blending control unit combines the weighted output control signals of the motion control module and the collision avoidance control module. The weight of the motion control output signal is higher when a calculated collision risk is lower. The weight of the collision avoidance control output signal is higher when the calculated collision risk is higher.

The method can be implemented on an anthropomorphic robot such as the robot presented in Figure 1. For first example, the authors have used a virtual serial robot, which manipulate a cubical object. The authors have created all bodies and connect them if desired with proper joints.

For example, the 3DOF model is shown below integrated into 3D Cartesian space. Here, the multi-body library provides three-dimensional mechanical components to model rigid multi-body robot system. The robot system is built by connecting blocks representing parts of the robot like link bodies, joints, actuators and gripper.

In this section the authors will describe how to render a planning scenario in the form of constraints for the constraint-based planning framework.

Assume that the geometry representing the robots and obstacles is given, as well as prescribed motion, simulated for the obstacles over time [2]. Informatics system then defines constraints that will restrict the motion of the robots to meet the design

specifications, and also guide the robots to complete the planning tasks such as the collision will be avoided.

The simulation methodology uses a number of possible scenarios and involves some sequences of animation, as one can see in the Figure 1.

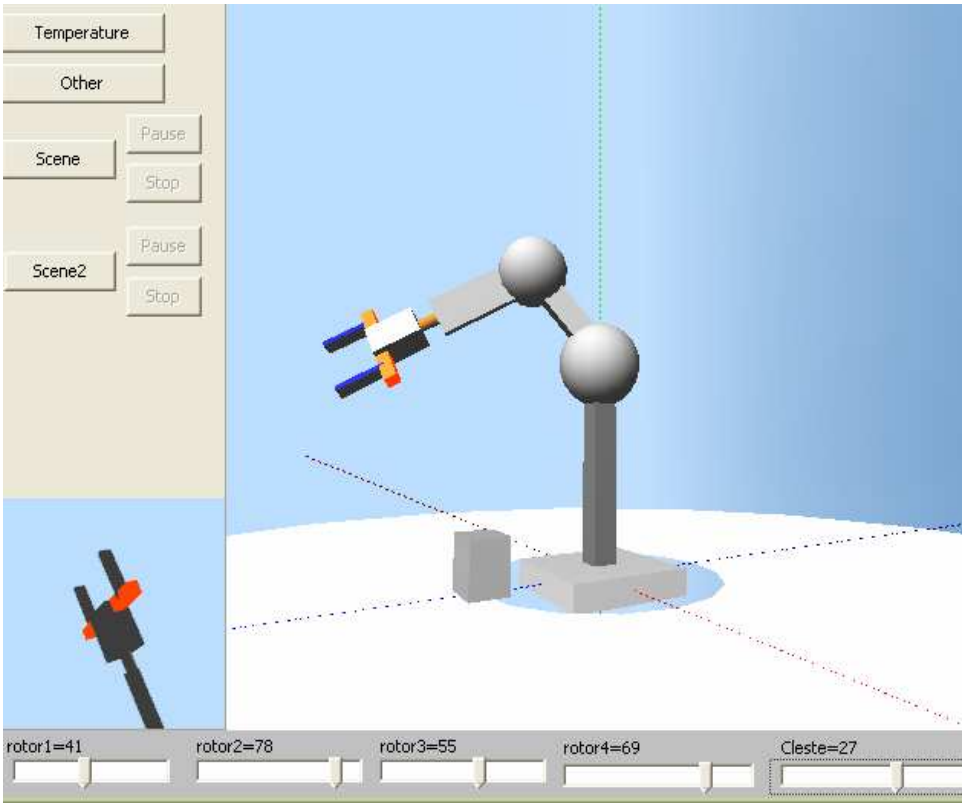

Figure 1. Anthropomorphic robotic arm

In Figure 1 is illustrated an anthropomorphic robotic arm having three segments where each segment comprises one or two spheres.

The basic essence of this approach is to describe each rigid object in the planning scene as a dynamical system, which is characterized by its state variables (i.e. position, orientation, linear and angular velocity). In this framework, a robot arm can be a collection of rigid bodies, subject to the influence of various forces in the workspace, and restricted by various motion constraints.

This transforms a motion planning problem into a problem of defining suitable constraints, and then simulating the rigid body dynamics of the scene with each constraint acting as a virtual force on the objects such as the collision will be avoided [3].

For this example, the authors use a virtual serial robot, which manipulate a virtual cubical object. The authors have created all bodies and connect them if desired with proper joints.

Here, the multi-body library provides three-dimensional mechanical components to model rigid multi-body robot system. The robot system is built by connecting blocks representing parts of the robot like link bodies, joints, actuators and gripper.

The simulation methodology uses a number of possible scenarios and involves some sequences of animation, as one can see in the Figure 1.

The dynamic simulation of multi-body systems becomes very interested when the robot manipulator must interact with the mobile objects, where the success will depend only on the capabilities of the robots. In this example, the self-collision detection is analyzed as well as test robot control algorithms [4].

Proposed scheme for dynamic simulation or animation, using the distance computation algorithm is an iterative process which continuously inserts and deletes the object pairs from a stack according to their approximate time to collision, as the objects move in a dynamic environment.

Although many fast methods of animation have been proposed, few techniques are currently able to dynamically animate.

The virtual system was implemented on a programming platform, using Delphi object-oriented programming language [5].

The Delphi source code for this application (only some procedures) is shown below.

```
{$R *.dfm}
procedure TForm1.TrackBar1Change(Sender: TObject);
begin
  glCylinder1.TurnAngle:=trackbar1.Position;
  Valuex.Caption:='rotor1='+floattostr(trackbar1.Position);
end;

procedure TForm1.TrackBar2Change(Sender: TObject);
begin
  glCylinder2.TurnAngle:=trackbar2.Position;
  Valuey.Caption:='rotor2='+floattostr(trackbar2.Position);
end;

  procedure TForm1.TrackBar3Change(Sender: TObject);
begin
  glCylinder3.TurnAngle:=trackbar3.Position;
  Valuez.Caption:='rotor3='+floattostr(trackbar3.Position);
end;

  procedure TForm1.TrackBar4Change(Sender: TObject);
begin
  glCylinder4.TurnAngle:=trackbar4.Position;
  Valuex2.Caption:='rotor4='+floattostr(trackbar4.Position);
end;

 procedure TForm1.TrackBar5Change(Sender: TObject);
begin
  pensa1.Position.Z:=trackbar5.Position/100;
  pensa2.Position.Z:=-trackbar5.Position/100;
  Valuecleste.Caption:='Cleste='+floattostr(trackbar5.Position);
end;

procedure TForm1.GLSceneViewer1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
soarece:=true;
end;

procedure TForm1.GLSceneViewer1MouseUp(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
soarece:=false;
end;

procedure TForm1.GLSceneViewer1MouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
  if(soarece)then
  begin
    if(x1-X<0)then
      glcube1.TurnAngle:=glcube1.TurnAngle+2
    else
      glcube1.TurnAngle:=glcube1.TurnAngle-2;
//glcamera1.MoveAroundTarget(x1-x,y1-y);
    x1:=x;
    y1:=y;
  end;
end.
```

The strategy for dynamic simulation or animation, using the distance computation algorithm is an iterative process which continuously inserts and deletes the object pairs from a heap according to their approximate time to collision, as the objects move in a dynamic environment. The simulation method purely exploits the spatial arrangement of the two end-effectors without any other information.

The end-effectors pair which has a small separation is likely to have an impact within the next few time instances, and those virtual pairs which are far apart from each other cannot possibly come to interfere with each other until certain time.

For spatial tests to reduce the number of virtual pairs judicious comparisons, the authors assume that the environment is quite free and the end-effectors move in such a way that the geometric coherence can be preserved, i.e. the assumption that the motion is essentially continuous in time domain.

3. Conclusion

This paper gives an application of the collision detection algorithm for a virtual system with a virtual robot. The authors have applied this algorithm to perform collision detection. This algorithm has been utilized in dynamic simulation as well as in a virtual environment. This application attests for the practicality of the algorithms and the importance of the problem natures. This algorithm and the distance computation method have been used in the dynamics simulator written in Delphi language.

The essential performance of this dynamic simulator is the ability to simulate of small mechanical parts of a robot arm in real time. It reduces the frequency of the collision checks significantly and helps to speed up the calculations of the dynamic simulator considerably.

References

1. Fernandez-Madrigal, J-A. et al. (2008) *A software engineering approach for the development of heterogeneous robotic applications*. Proceeding of Robotics and Computer-Integrated Manufacturing, no. 24, p.150-166
2. Fleury, G., Lacomme, Ph., Tanguy, A. (2006) *Simulation an eveniments discrets.* EYROLLES, 2006
3. Yared, R. et al. (2007) *Collision prevention using group communication for asynchronous cooperative mobile robots.* Proceeding of the 21st IEEE International Conference on Advanced Information Networking and Applications (AINA'07), ISBN: 0-7695-2846-5, p. 244-249
4. Zucker, M., Kuffner, J., Branicky, M: *Multipartite RRTs for rapid replanning in dynamic environments.* Proceeding of the IEEE Int. Conf. on Robotics and Automation, p. 1603-1609, 2007
5. Fratu, A., Fratu, M. (2011) *Visual programming in Delphi environment - with Application in Robotics.* Second edition, Transilvania University Press, ISBN 978-973-598-963-7

Received in January 2012