Path Planning for a Humanoid Robot Using NURBS Curves

Andreas J. Schmid and Heinz Woern
Institute for Process Control and Robotics
University of Karlsruhe
Engler-Bunte-Ring 8, 76131 Karlsruhe, Germany
{anschmid, woern}@ira.uka.de

Abstract – In this paper we present an algorithm for planning the path of a 3 DOF mobile platform for a humanoid robot. It is simple enough to be executed online and yet powerful enough to generate a smooth path from starting point and orientation to a destination point with a different orientation. Thus we strive to achieve a human-like appearance of our robot that encourages the humans in vicinity to interact with our humanoid robot and enhances the possibilities of human-robot collaboration. We opted to use NURBS curves as the basis for our path planning algorithm.

Index Terms – Humanoid robot, human-robot collaboration, NURBS, path planning.

I. INTRODUCTION

This work is part of a larger research effort that is our collaborative research center “Humanoid Robots” [1]. The goal of this research center is to come up with a humanoid service robot that is able to assist and collaborate with a human in a household environment. Our part of the research center focuses on the collaboration between robot and humans. The robot we are working with is a 7 degrees of freedom (DOF) humanoid manipulator, see Fig. 1.

Unlike the final demonstrator robot of the research effort this manipulator is built out of cubes with rotary actuators in a building block fashion. The important part is its similarity in dexterity to a human arm which is sufficient to test and experiment with our algorithms for the collaboration [2, 3]. The next step of our work is to add a mobile platform to our manipulator and extend our human – robot collaboration mechanisms accordingly. Before attaching such a mobile platform in hardware, however, we decided to simulate it with a CAD environment (Fig. 2).

Related work in the area of path planning with splines has been done by interpolating a given set of control points through B-spline trajectories [4, 5]. A method to avoid collisions on top of this has been proposed [6], as well as one for finding optimal B-spline-based trajectories that minimize given criteria [7]. The path of a moving target vehicle has been approximated by NURBS curves and used as input for a robot follower [8]. Furthermore, B-spline wavelets have been employed to transfer recorded human trajectories to a humanoid robot [9]. Our approach differs from the above significantly since we are concerned with the synthesis of a trajectory rather than with the approximation of a given one.

II. MOTIVATION

As a first milestone we had to conceive a concept for operating the mobile platform by itself. That is, the platform has to be able to start at a given point, the starting point $P_0$ with a given orientation $\alpha_0$, and travel along a path to a destination point $P_N$ and a final orientation of $\alpha_N$.

The path planning has to be performed on the fly as the destination point will be an input e.g. from an object recognition module capable of estimating the location of an object or human being in world coordinates. Thus there will be a need to react quickly on the sometimes unpredictable behavior of the human or objects in vicinity.

We assume for now that there are no obstacles in the way and that the arrival time does not matter. As to the mobile platform itself we assume that it has 3 DOF (2 position and 1 orientation coordinate) and operates in 2D space. It is either holonomic or nonholonomic with car-like holonomic constraints. From the latter assumption it follows that we have to take into account that the robot cannot turn in place. Instead a minimum radius of curvature has to be observed.
Since we want to give our robot a look and feel that is as much human-like as possible an algorithm for the movement is required that creates a smooth path without abrupt changes in direction or speed. After considering these requirements we opted to employ a NURBS curve as our path planning basis and searched for a suitable algorithm to determine the curve’s parameters.

III. NURBS CURVES

Non-Uniform Rational B-Splines (NURBS) [10] are a convenient way of generating curves and surfaces. They don’t take many parameters to be completely specified and yet can approximate arbitrary free forms and shapes with any desired accuracy. A NURBS curve is defined by a sum of B- (or Base-) splines \( N_{i,p} \) of degree \( p \) that serve as base functions of the curve, each weighted with a control point \( P_i \) and a weight \( w_i \). The parameter \( u \) determines the actual position on the curve, where \( u \in [0,1] \). Put together we obtain (1) as the equation of the curve.

\[
C(u) = \sum_{i=0}^{n} N_{i,p}(u) w_i P_i \tag{1}
\]

For many applications, the set of weights \( W = (w_0,w_1,\ldots,w_n) \) can simply be chosen to be uniformly 1. In that case the NURBS curve becomes non-rational.

The B-spline functions \( N_{i,p} \) require a set of knots \( U = (u_0,u_1,\ldots,u_n) \) with \( u_i \leq u_{i+1} \) to be specified, as well as the degree \( p \) of the curve. B-splines are defined recursively as follows:

\[
N_{i,p}(u) = \begin{cases} 
\frac{u-u_i}{u_{i+p}-u_i} N_{i,p-1}(u) + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} N_{i+1,p-1}(u) & \text{if } u_i \leq u < u_{i+p+1}, \\
0 & \text{otherwise}
\end{cases} \tag{2}
\]

The recursion terminates with

\[
N_{i,0}(u) = 1, \quad u_i \leq u < u_{i+1} \quad \text{and} \quad u < u_{i+1},
\]

and a factor of 0/0 equals 0 by convention.

The ordered set of knots can be chosen at will. It is common, however, to choose values between 0 and 1. Uniformly distributed knots make for a uniform curve, otherwise it is non-uniform. To achieve that the curve meets the control points at the beginning and at the end of a curve, \( p + 1 \) knots at the beginning and at the end each must be equal. Combined with the definition (2), this leads to the requirement

\[
m = n + p + 1, \quad (4)
\]

where \( m \) is the number of knots and \( n \) the number of control points (and weights). In the special case of \( n = p + 1 \) and uniform weights \( w_i = 1 \), the NURBS curve is identical to a Bézier curve.

The degree \( p \) influences the number of knots that a particular B-spline \( N_{i,p} \) stretches over, namely from knots \( u_i \) to \( u_{i+p} \). The lower the degree, the less knots are involved. This way the curve has a more local behavior, as each \( N_{i,p} \) stretches a shorter section of \( u \) and thus the influence of its weight, the control point \( P_i \), is reduced. A higher degree in turn implies that moving a control point affects the shape of a larger part of the curve. In the case of a Bézier curve each control point influences the shape of the complete curve.

Another property of the curve that is influenced by \( p \) is its differentiability at the knots. For a single knot at some \( u = u_i \), the curve \( C(u) \) can be continuously differentiated \( p-1 \) times. For \( n \) knots at \( u = u_i \), this is reduced to \( p-n \) times.

IV. OUR ALGORITHM

Defining our curve means to choose a degree \( p \), a set of control points and a set of the knots. We chose \( p = 3 \) to obtain a cubic curve. This keeps enough locality for the control points. It also allows for a smooth curve since \( C(u) \) can be continuously differentiated \( p-1 \) times which is enough for the human eye to perceive it as smooth.

As to the control points, two of them are obviously necessary – the start point \((P_0)\) and the end point \((P_N)\). The next two control points need to be chosen such that they fulfill the requirement of starting and ending at the predefined angles \( \alpha_0 \) and \( \alpha_N \). We achieve this by placing them in the direction \((P_1)\) and opposite direction \((P_{N-1})\) of these angles at a distance \( r \) of \( P_0 \) and \( P_N \), see Fig. 3. The desired orientations specified by the angles are shown by the arrows. As an equation this can be written as

\[
\vec{P}_h = \vec{P}_0 + r \left( \begin{array}{c} \cos \alpha_h \\ \sin \alpha_h \end{array} \right), \tag{5}
\]

where \( \vec{P}_h \) is the vector from the origin to \( P_h \). Similarly for \( P_{N-1} \).

Fig. 2 Simulation of the Humanoid Manipulator with a Mobile Platform.
If we use this minimal configuration to create our path with a NURBS curve, we need a set of knots with $4 + 3 + 1 = 8$ members, see (4). Since we want the first and the last four knots each be equal, we obtain the knot vector $(0, 0, 0, 0, 1, 1, 1, 1)$. The four B-splines $N_{i,3}$ ($i = 0 \ldots 3$), calculated using (2), are shown in Fig. 4. You can see the resulting NURBS curve for some arbitrary points $P_0$ and $P_3$ in Fig. 5. It is quite obvious that the result can not be satisfactory. The radius of the curvature around $P_0$ is too narrow and in fact becomes zero for $\alpha_0 = 180^\circ$ (given the current positions of $P_0$ and $P_N$). The same problem exists for $P_N$ and $\alpha_N$.

Furthermore, we do not appreciate the fact that with this configuration we have a Bézier curve where moving one control point means an alteration of the whole path and not only a locally limited section.

Consequently we improved this setting by adding two more control points that shape the curve to our requirements of (a) smooth path, and (b) turns with radii no smaller than a given minimum value. Fig. 6 illustrates how these additional points $P_3$ and $P_4$ are determined. Mathematically this can be formulated

$$\tilde{p}_i = \tilde{p}_0 \pm r \left( \cos \alpha_i \sin \alpha_i \right)$$

If $\alpha_i = 180^\circ$ (given the current positions of $P_0$ and $P_N$), the sign of the second addend depends on the angles, since we always want to lay $P_3$ and $P_4$ in between the smaller one of the two angles in the argument of the cosine or sine, respectively. This can be achieved for (7) by choosing a "+" if the condition

$$\angle(\tilde{p}_1 - \tilde{p}_0) + \angle(\tilde{p}_2 - \tilde{p}_0) > 180^\circ$$

is true, otherwise a "−". It works likewise for (8).

Thus we have a total of six control points in the order $P_0 \rightarrow P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_2 \rightarrow P_N$ since the indices denote the order of calculation of these points rather than how the NURBS curve passes along them. We choose the corresponding knot vector to be $(0, 0, 0, 0, 1/3, 2/3, 1, 1, 1, 1)$, thus clamping the curve to $P_0$ and $P_N$ and distributing the remaining knots evenly. The B-splines that are associated with these knots are plotted in Fig. 7. Note that solely $N_{3,3}$ and $N_{2,3}$ span the whole range of the parameter $u$. The other four are limited to a section of the range, making for the locality behavior of the curve. A typical NURBS curve with such a configuration is depicted in Fig. 8.
V. DISCUSSION AND EVALUATION

In order to evaluate our algorithm quickly and graphically we implemented the algorithm in C code and used OpenGL commands to display the result. Keyboard inputs allow to change the angles $\alpha_0$ and $\alpha_N$ to any desired value, and you can drag the start and end points with the mouse pointer. The control points and the curve are then computed on the fly as it will be the case when used with real hardware, and displayed on the screen subsequently.

It turns out that the algorithm successfully manages to place the control points in such a way that the radii of the turns are not narrower than a minimum value. This holds true for all angle settings, especially the difficult ones where the starting angle $\alpha_0$ is 180° opposed to the location of the end point $P_N$. Fig. 9 shows some of these interesting configurations.

For a more thorough analysis that would give us precise numbers we evaluated the algorithm analytically and numerically. Judging the configuration shown in Fig. 9a) to be the one with the narrowest radii, we determined the radius of the curvature analytically for these values of $\alpha_0$ and $\alpha_N$. It is defined as the inverse of the curvature. Denoting a point on the curve with $x(u)$ and its derivatives with respect to $u$ with dots on top, i.e. $\dot{x}(u) = \delta x(u)/\delta u$, the radius of the curvature is defined according to this equation:

$$\text{radius} = \left| \frac{\dot{x}(u)}{\ddot{x}(u) \times \dddot{x}(u)} \right|$$  \hfill (10)

where the product in the denominator constitutes a cross product. Due to the rather lengthy result of this equation, we present the graph of the result instead of the result itself, plotted over the whole range of $u$; see Fig. 10.
From the construction of the algorithm, however, there arises a clear limitation as to how large the parameter \( r \) can be chosen with respect to the distance of the start and end point. Namely \( r \) cannot be set to a value larger than a quarter of the distance between \( P_0 \) and \( P_N \). Otherwise the curve gets deformed and bulges out in a manner that was not intended.

To test the validity of the selection of \( \alpha_0 = \alpha_N = 180^\circ \) for the examinations shown in Figs. 10 and 11, we plotted the minimum radius of curvature over the starting angle \( \alpha_0 \) in Fig. 12. We see that we have a global minimum at 180° and a local one at 0°. Therefore we have proven that our approach was justified.

Lastly we will show how the speed of our robot looks like when travelling along the curve at constant steps of the parameter \( u \). The speed is defined as

\[
\text{speed} = |\dot{u}(u)|. \tag{11}
\]

An example for the values \( \alpha_0 = \alpha_N = 180^\circ \) is displayed in Fig. 13. It turns out that the speed is largely dependent on the radius of the curvature – a small radius corresponds to a slow speed, a larger radius to a higher speed. This correspondence is an important property because we need to ensure that our robot doesn’t tip over by taking a turn too fast.

VI. CONCLUSION

We came up with an algorithm to compute control points for creating a NURBS curve on the fly. This enables us to flexibly travel from a starting point \( P_0 \) and an orientation of \( \alpha_0 \) to a destination point \( P_N \) and a desired orientation of \( \alpha_N \) in that point. The algorithm is simple to ease the requirements on compute power yet powerful enough to allow for a smooth transition from start to destination. This is an important point as our robot is designed to interact with human beings; a smooth and organic movement is therefore highly desirable.
Another asset of this algorithm is its ability to accommodate limitations of the robot’s mobile platform as far as the minimum radius of turns is concerned. Non-holonomic platforms like e.g. cars do not allow to move in an arbitrary direction at any given time. Rather they have to follow the direction of their wheels. As a consequence, one needs to travel along a path that takes this restriction into account.

VII. FUTURE WORK

In the future we want to achieve an even further humanoid movement of our robot by incorporating the findings of our research program’s group that studies the properties of human motion. With the help of markers that are attached to a person’s body one can record the trajectory of human movement and subsequently extract its characteristics.

We are also striving to come up with algorithms to take care of configurations that were out of the scope of this research, namely very close start and destination points, the presence of obstacles in the path and strategies for handling exceptions that might be encountered along the traveling path.

Last but not least it is our goal to find an algorithm that can adequately distribute our movement over the redundant number of DOFs. After all, the point will not be that the platform reaches a certain point on the ground, but rather that the endeffector of the manipulator attached to the mobile platform reaches a certain position in 3D space with a 3D orientation. Having a mobile platform with 3 DOF and a humanoid manipulator with 7 DOF there is a redundancy of 4 DOF in the system. We want to use this redundancy at our advantage to achieve a movement that is as human-like as possible.