Inverse kinematics computation in computer graphics and robotics using conformal geometric algebra

Dietmar Hildenbrand, Julio Zamora and Eduardo Bayro-Corrochano

ABSTRACT We focus on inverse kinematics applications in computer graphics and robotics based on **Conformal Geometric Algebra**. Here, geometric objects like spheres and circles that are often needed in inverse kinematics algorithms are simply represented by algebraic objects.
We present algorithms for the inverse kinematics of a human arm like kinematic chain and for the grasping of robots and virtual humans. The main benefits of using geometric algebra in the virtual reality software Avalon are the easy, compact and geometrically intuitive formulation of the algorithms and the immediate computation of quaternions.

Keywords: Clifford algebras, Inverse Kinematics.

1 Introduction

After a short introduction of Conformal Geometric Algebra we present an algorithm for the inverse kinematics of a virtual human with the focus on the immediate computation of quaternions.

We also show how to perform certain basic robot object manipulation tasks like solving the problem of positioning the gripper in a certain position of space disregarding the grasping plane or the gripper's alignment. Then, we will illustrate how the robotic arm can follow linear and spherical paths.

After a brief introduction of two software packages for the development and the implementation of algorithms based on Geometric Algebra we present a grasping algorithm based on CLUCalc that you are able to download and to make your own experiments.

AMS Subject Classification: 15A66, 17B37, 20C30, 81R25.

2 Conformal Geometric Algebra

We use the 5D **Conformal Geometric Algebra** which is an extension of the 4D Projective Geometric Algebra.
While points and vectors are normally used as basic geometric entities, in Conformal Geometric Algebra we have a wider variety of basic objects. For

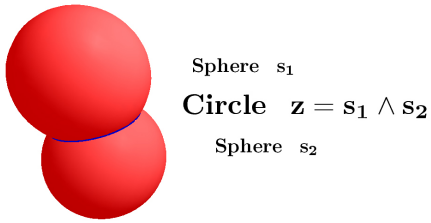


FIGURE 1. Intersection of two spheres

example, spheres and circles are simply represented by algebraic objects. To represent a circle you only have to intersect two spheres, which can be done with a basic algebraic operation. Alternatively you can simply combine three points to obtain the circle through these three points.

Some more detailed introductions to the Conformal Geometric Algebra will be found in [8, 2] and [12], some Geometric Algebra tutorials will be found in [3], [4], [9], [11] [14], [16] and some applications in [5], [6] and [17].

Besides the construction of algebraic entities, kinematics can also be expressed in Geometric Algebra. We present algorithms for the inverse kinematics of a virtual human and for the grasping of a robot as well as the software tools to develop and implement the applications.

3 The basic geometric entities in Conformal Geometric Algebra

Table 1.1 lists the two representations of the geometric entities in Conformal Geometric Algebra. Please find details in [9] and [13].

In this table $\mathbf{x}$ and $\mathbf{n}$ are marked bold to indicate that they represent 3D entities as linear combination of the 3D base vectors e_1, e_2 and e_3.

$$\mathbf{x} = x_1 e_1 + x_2 e_2 + x_3 e_3 \tag{3.1}$$

The additional two base vectors are indicated by

- $\mathbf{e}_0$ representing the 3D origin
- $\mathbf{e}_\infty$ representing the point at infinity

TABLE 1.1. list of the conformal geometric entities

entity	**representation 1**	**representation 2**
Point	$P = \mathbf{x} + \frac{1}{2}\mathbf{x}^2\mathbf{e}_\infty + e_0$	
Sphere	$s = P - \frac{1}{2}r^2\mathbf{e}_\infty$	$s^* = x_1 \wedge x_2 \wedge x_3 \wedge x_4$
Plane	$\pi = \mathbf{n} + d\mathbf{e}_\infty$	$\pi^* = x_1 \wedge x_2 \wedge x_3 \wedge \mathbf{e}_\infty$
Circle	$z = s_1 \wedge s_2$	$z^* = x_1 \wedge x_2 \wedge x_3$
Line	$l = \pi_1 \wedge \pi_2$	$l^* = x_1 \wedge x_2 \wedge \mathbf{e}_\infty$
Point Pair	$Pp = s_1 \wedge s_2 \wedge s_3$	$Pp^* = x_1 \wedge x_2$

The $\{s_i\}$ represent different spheres and the $\{\pi_i\}$ different planes.
The two representations are **dual** to each other. In order to switch between the two representations you can use the dual operator which is indicated by '*'. Depending on the application and convenience, one of these two sets of representations is selected as standard representation. We use representation 1 as standard representation and representation 2 as dual representation.
In representation 2 the outer product '$\wedge$' indicates the construction of geometric objects with the help of points x_i that lie on it. E. g. a sphere is defined by 4 points ($x_1 \wedge x_2 \wedge x_3 \wedge x_4$) determining the sphere.
In representation 1 the dual meaning of the outer product is the intersection of geometric entities. E. g. a circle is defined by the intersection of two spheres ($s_1 \wedge s_2$). Please refer to figure 1.

4 The Inverse Kinematics of a human-arm-like kinematic chain

Our model of the human arm is a 7 degrees of freedom (DOF) kinematic chain according to [18] with 3 degrees of freedom ($\theta_1, \theta_2, \theta_3$) at the shoulder, 1 degree of freedom at the elbow (θ_4) and 3 degrees of freedom at the wrist ($\theta_5, \theta_6, \theta_7$).
While in former analytic algorithms a lot of mathematical knowledge about trigonometry, rotation matrices etc. has to be available, in our approach only some basic operations with basic geometric entities like planes and spheres are needed.

4.1 Computing the joint angles

Our goal is to reach the chosen point p_t with the wrist.
The geometrically intuitive algorithm can be separated in the following steps

- compute the swivel plane

- compute the elbow point $\mathbf{p_e}$
- compute the elbow angle θ_4
- elevate to the rotation plane including the elbow point $\mathbf{p_e}$ and compute the angle θ_1
- rotate until the elbow position matches and compute the angle θ_2
- rotate until the wrist location is reached and compute the angle θ_3

that are described in detail in [10].

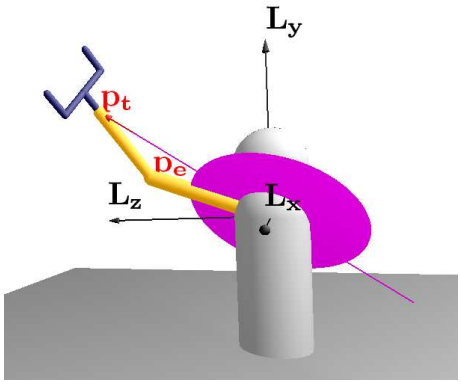


FIGURE 2. Model of the arm of a virtual human with 3 DOF of freedom at the shoulder (origin of the coordinate system), 1 DOF at the elbow and 3 DOF at the wrist

4.2 Computing the quaternions

Until now, we computed all the relevant angles needed to describe the transformation of the human-arm-like kinematic chain in order to reach its goal. But, the virtual reality software Avalon needs quaternions in order to perform the SLERP motion between two gestures.
In order to handle quaternions we reduce the dimensionality and map to the Euclidean geometric (sub)algebra based on e_1, e_2 and e_3.

We identify the imaginary components $\mathbf{i}, \mathbf{j}$ and $\mathbf{k}$, representing a dual coordinate system, with bivectors. Please refer to [16] for details about quaternions in Euclidean geometric algebra.

In our application we have to compute the quaternions $q_{shoulder}$ and q_{elbow} based on the angles $\theta_1, \theta_2, \theta_3$ and θ_4 and the relevant coordinate axes.

For instance, the elbow quaternion can be computed based on the angle θ_4 according to the following equation

$$q_{elbow} = cos(\frac{\theta_4}{2}) - \mathbf{i}sin(\frac{\theta_4}{2}) \tag{4.1}$$

The shoulder quaternion can be expressed as the geometric product of the 3 quaternions based on the angles $\theta_1, \theta_2, \theta_3$

$$q_{shoulder} = q_1 q_2 q_3 \tag{4.2}$$

5 Following Geometric Primitives

In this section we will show how to perform certain basic object manipulation tasks now in the context of conformal geometric algebra. First, we will solve the problem of positioning the gripper of the arm in a certain position of space disregarding the grasping plane or the gripper's alignment. Then, we will illustrate how the robotic arm can follow linear and spherical paths.

5.1 Touching a point

In order to reconstruct the point of interest, we back-project two rays extending from two views of a given scene (see Figure 3). These rays will not intersect in general, due to noise. Hence, we compute the directed distance between these lines and use the the middle point as target. Once the 3D point p_t is computed with respect to the cameras' framework, we transform it to the arm's coordinate system.

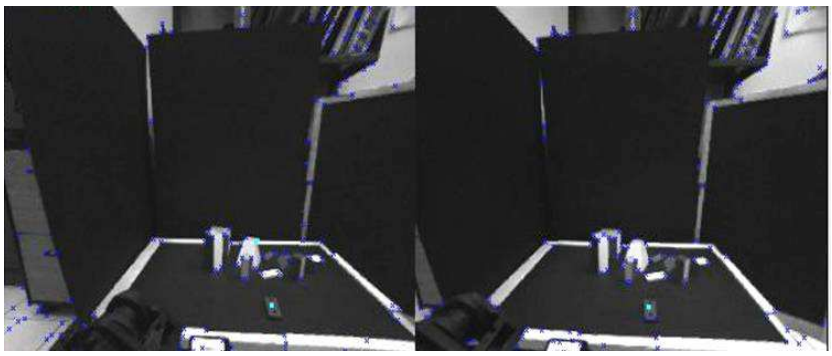


FIGURE 3. Point of interest in both cameras (p_t).

Once we have a target point with respect to the arm's framework, there are three cases to consider. There might be several solutions (see Figs. 4.a and 5.a), a single solution (see Figure 4.b), or the point may be impossible to reach (Figure 5.b).

In order to distinguish between these cases, we create a sphere $S_t = p_t - \frac{1}{2}d_3^2 e_\infty$ centered at the point p_t and intersect it with the bounding sphere $S_e = p_0 - \frac{1}{2}(d_1 + d_2)^2 e_\infty$ of the other joints (see Figures 4.a and 4.b), producing the circle $z_s = S_e \wedge S_t$.

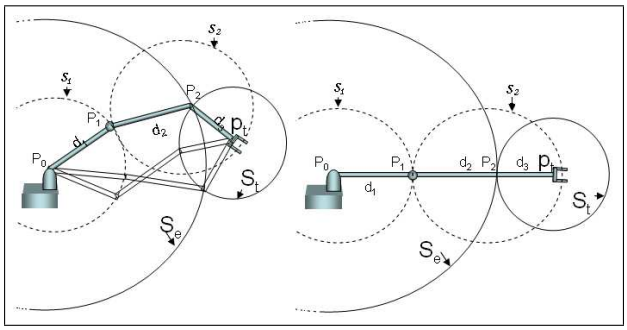


FIGURE 4. a) S_e and S_t meet (infinite solutions) b) S_e and S_t are tangent (single solution).

If the spheres S_t and S_e intersect, then we have a solution circle z_s which represents all the possible positions the point p_2 (see Figure 4) may have in order to reach the target. If the spheres are tangent, then there is only one point of intersection and a single solution to the problem as shown in Figure 4.b.

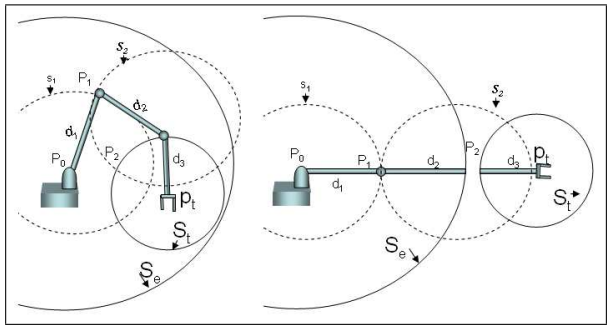


FIGURE 5. a) S_t inside S_e produces infinite solutions, b) S_t outside S_e, no possible solution.

If the spheres do not intersect, then there are two possibilities. The first case is that S_t is outside the sphere S_e. In this case, there is no solution since the arm cannot reach the point p_t as shown in Figure 5.b. On the other hand, if the sphere S_t is inside S_e, then we have a sphere of solutions. In other words, we can place the point p_2 anywhere inside S_t as shown in Figure 5.a. For this case, we arbitrarily choose the upper point of the sphere S_t.

In the experiment shown in Figure 6.a, the sphere S_t is placed inside the bounding sphere S_e, therefore the point selected by the algorithm is the upper limit of the sphere as shown in Figures 6.a and 6.b. The last joint is completely vertical.

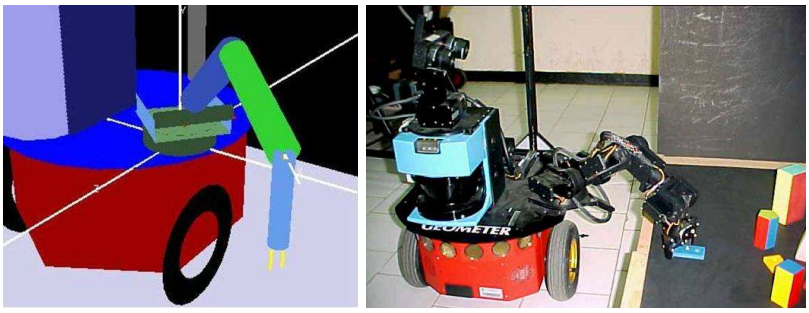


FIGURE 6. a) Simulation of the robotic arm touching a point. b) Robot "Geometer" touching a point with its arm.

5.2 Line of intersection of two planes

In the industry, mainly in the sector dedicated to car assembly, it is often required to weld pieces. However, due to several factors, these pieces are not always in the same position complicating this task and making this process almost impossible to automate. In many cases the requirement is to weld pieces of straight lines when no points on the line are available. This is the problem to solve in the following experiment.

If we do not have points on the line of interest, then we find this line via the intersection of two planes (the welding planes). In order to determine each plane, we need three points. The 3D coordinates of the points are triangulated using the stereo vision system of the robot yielding a configuration like the one shown in Figure 7.

Once the 3D coordinates of the points in space have been computed, we can find each plane with $\pi^* = x_1 \wedge x_2 \wedge x_3 \wedge e_\infty$, and $\pi'^* = x'_1 \wedge x'_2 \wedge x'_3 \wedge e'_\infty$. The line of intersection is computed via the *meet* operator $l = \pi' \cap \pi$. In Figure 8.a we show a simulation of the arm following the line produced by the intersection of these two planes.

Once the line of intersection l is computed, it suffices with translating it on the plane $\psi = l^* \wedge e_2$ (see Figure 8.b) using the translator $T_1 = 1 + \gamma e_2 e_\infty$, in the direction of e_2 (the y axis) a distance γ. Furthermore, we build the translator $T_2 = 1 + d_3 e_2 e_\infty$ with the same direction (e_2), but with a separation d_3 which corresponds to the size of the gripper. Once the translators have been computed, we find the lines l' and l'' by translating the line l with $l' = T_1 l T_1^{-1}$, and $l'' = T_2 l' T_2^{-1}$.

The next step after computing the lines, is to find the points p_t and p_2

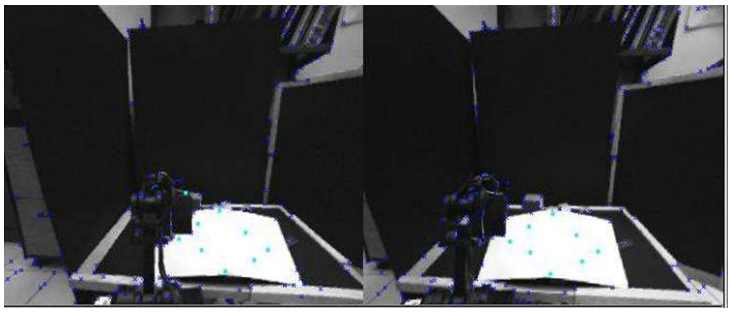


FIGURE 7. Images acquired by the binocular system of the robot "Geometer" showing the points on each plane.

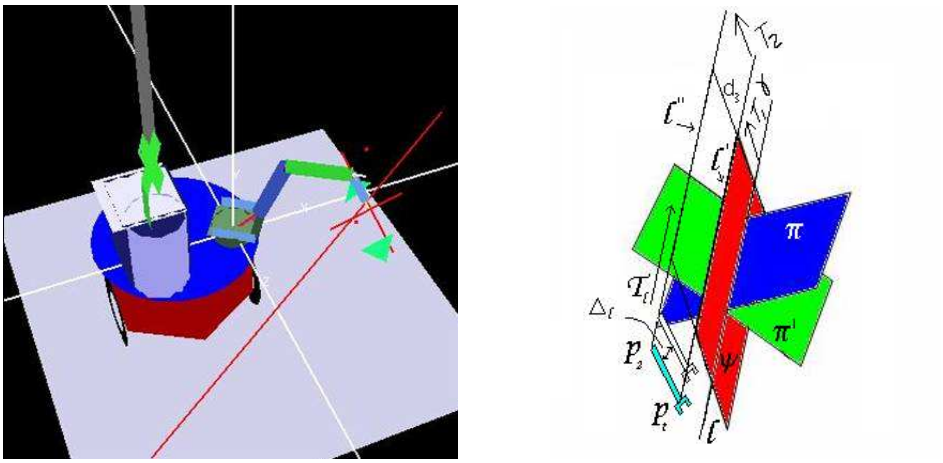


FIGURE 8. a) Simulation of the arm following the path of a line produced by the intersection of two planes. b) Guiding lines for the robotic arm produced by the intersection (meet) of planes and vertical translation.

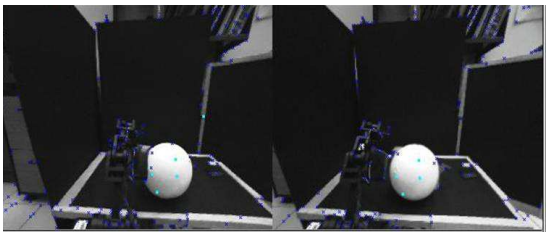
which represent the places where the arm will start and finish its motion, respectively. These points were given manually, but they may be computed with the intersection of the lines l' and l'' with a plane that defines the desired depth. In order to make the motion over the line, we build a translator $T_L = 1 - \Delta_L l e_\infty$ with the same direction as l as shown in Figure 8.b. Then, this translator is applied to the points $p_2 = T_L p_2 T_L^{-1}$ and $p_t = T_L p_t T_L^{-1}$ in an iterative fashion to yield a displacement Δ_L on the robotic arm.

By placing the end point over the lines and p_2 over the translated line, and by following the path with a translator in the direction of l we get a motion over l as seen in the image sequence of Figure 9.

FIGURE 9. Image sequence of a linear-path motion.

5.3 Following a spherical path

This experiment consists in following the path of a spherical object at a certain fixed distance from it. For this experiment, only four points on the object are available (see Figure 10.a).



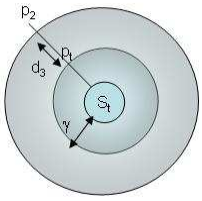


FIGURE 10. a) Points over the sphere as seen by the robot "Geometer". b) Guiding spheres for the arm's motion.

After acquiring the four 3D points, we compute the sphere $S^* = x_1 \wedge x_2 \wedge x_3 \wedge x_4$. In order to place the point p_2 in such a way that the arm points towards the sphere, the sphere was expanded using two different dilators. This produces a sphere that contains S^* and ensures that a fixed distance between the arm and S^* is preserved, as shown in Figure 10.b.

The dilators are computed as follows

$$D_\gamma = e^{-\frac{1}{2}\ln(\frac{\gamma+\rho}{\rho})E}, \tag{5.1}$$

$$D_d = e^{-\frac{1}{2}\ln(\frac{d_3+\gamma+\rho}{\rho})E}. \tag{5.2}$$

The spheres S_1 and S_2 are computed by dilating S_t:

$$S_1 = D_\gamma S_t D_\gamma^{-1}, \tag{5.3}$$

$$S_2 = D_d S_t D_d^{-1}. \tag{5.4}$$

We decompose each sphere in its parametric form as

$$p_t = M_1(\varphi)M_1(\phi)p_{s_1}M_1^{-1}(\phi)M_1^{-1}(\varphi), \tag{5.5}$$
$$p_2 = M_2(\varphi)M_2(\phi)p_{s_2}M_2^{-1}(\phi)M_2^{-1}(\varphi). \tag{5.6}$$

Where p_s is any point on the sphere. In order to simplify the problem, we select the upper point on the sphere. To perform the motion on the sphere, we vary the parameters φ and ϕ and compute the corresponding p_t and p_2 using equations (5.5) and (5.6). The results of the simulation are shown in Figure 11.a, whereas the results of the real experiment can be seen in Figures 11.b and 11.c.

FIGURE 11. a) Simulation of the motion over a sphere. b) and c) Two of the images in the sequence of the real experiment.

6 Software support

Here we briefly describe two software tools that are convenient to develop and implement algorithms based on Conformal Geometric Algebra.

6.1 Visual development of algorithms

The OpenSource **CLUCalc** software can be used advantageously to **calculate with Geometric Algebra** and to **visualize the results** of these calculations. CLUCalc is freely available for download at [15]. With the help of the CLUCalc Software you are able to edit and run Scripts called **CLUScripts**. A screenshot of CLUCalc can be seen in figure 12.

CLUCalc provides the following three windows

- script editor window
- visualization window
- output window

There is almost a one to one correspondence between formulae and code. For example the computations of the intersection of two spheres can easily be done as follows

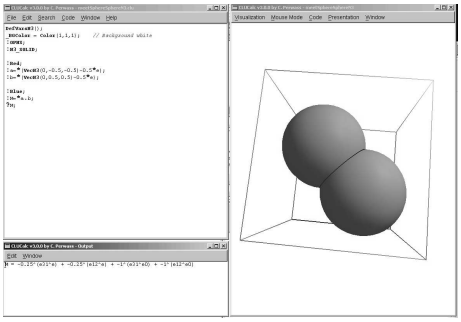


FIGURE 12. Screenshot of the CLUCalc windows

```
S1 = p0 - 0.5*d2*d2*einf;
S2 = p2 - 0.5*d3*d3*einf;
Circle = S1^S2;
```

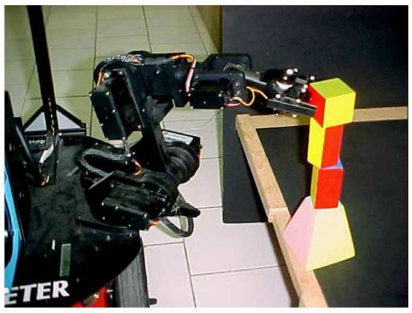
For details regarding CLUScript please refer to the CLUCalc online help [15].

The grasping algorithm, described in the next section, has been implemented using CLUCalc.

6.2 Implementation of the algorithms

With the use of the Gaigen toolkit it is possible to implement algorithms and integrate them inside of a desired target platform. Gaigen is a code generator for geometric algebras. Its focus is to optimize the resulting C++ code as much as possible for applications. Details can be found in [7].

7 Virtual grasping of an object



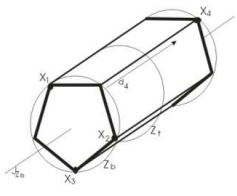


FIGURE 13. grasping an object

The following grasping algorithm of a computer graphics application is based on the robotics algorithm of the paper [10].
It can be downloaded as a CLUScript (Grasping.clu is the main file) from the homepage

`http://www.gris.informatik.tu-darmstadt.de/~dhilden/`

The goal of our grasping algorithm is to

- compute the grasping plane π_t
- compute the point of contact p_t

in order to be able to compute the inverse kinematics of a virtual human or a robot.

7.1 Extract points

For our grasping algorithm, first of all, we need 4 points, identifying the object to be grasped. We assume in our computer graphics application that these 4 points are already stored in the data structure of the object.

Recall that the following steps are performed by the robot "Geometer" to extract these 4 points

- Take a calibrated stereo pair of images of the object
- Extract four non-coplanar points from these images
- Compute the corresponding 3D points $x_i\,,\ i = 1, .., 4$ using triangulation

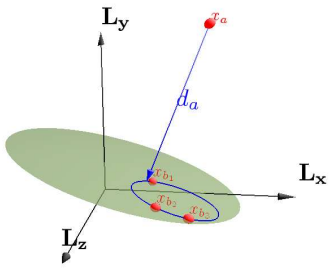


FIGURE 14. The 4 points, identifying the object to be grasped

7.2 Assign points

First of all, we need 4 points, identifying the object to be grasped. In the real application they are taken from a calibrated stereo pair of images of the object. In order to assign the four points we at first compute the distances between all the 4 points and the plane spanned by the 3 other points. The point with the greatest distance d_a will be called apex point x_a (see CLUCalc visualization in FIGURE 14). The other 3 points are called base points $x_{b_1}, x_{b_2}, x_{b_3}$.

7.3 Compute grasping plane π_t

To compute the grasping plane π_t we compute the circle

$$z_b^* = x_{b_1} \wedge x_{b_2} \wedge x_{b_3} \tag{7.1}$$

based on the base points.

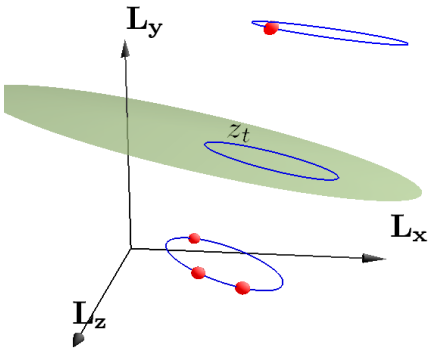


FIGURE 15. The grasping plane π_t including the grasping circle z_t

Then, we translate the circle z_b in the direction and magnitude of $\frac{d_a}{2}$ to produce the grasping plane π_t with the following steps :

With the help of the translator

$$T = 1 + \frac{1}{4} d_a e_\infty \tag{7.2}$$

we compute the grasping circle

$$z_t = T z_b \tilde{T} \tag{7.3}$$

and the grasping plane

$$\pi_t^* = z_t^* \wedge e_\infty \tag{7.4}$$

7.4 Compute the point of contact p_t

We already know the orientation for our grasping process, but we still need the point of contact p_t. Therefore we compute the closest point of the grasping circle to the y-axis with the help of the following steps :

The y-axis can be expressed as the intersection of the two planes with the normal vectors e_1 and e_3

$$L_y = e_1 \wedge e_3 \tag{7.5}$$

The center of the grasping circle z_t can be computed based on the following sandwich product

$$c_a = z_t e_\infty z_t \tag{7.6}$$

Then, we compute the plane through L_y and c_a

$$\pi_y = (L_y^* \wedge c_a)^* \tag{7.7}$$

and intersect with the grasping circle

$$Pp_t = \pi_y \wedge c_t \tag{7.8}$$

and select the point closer to the y-axis of the resulting point pair

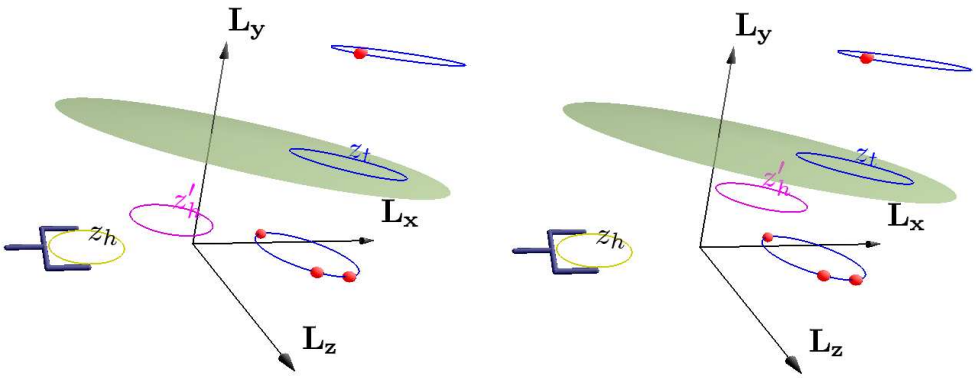


FIGURE 16. Moving the gripper circle z_h towards the grasping circle z_t

7.5 Move gripper to the target

In order to be able to move the gripper to the target we have to compute the transformation (rotation R and translation T) between the gripper circle and the grasping circle. Then we are able to move the gripper circle z_h step by step (z_h') towards the grasping circle z_t (see CLUCalc visualization in FIGURE 16).

$$z_h' \quad = \quad TRz_h\widetilde{R}\widetilde{T}. \tag{7.9}$$

8 Conclusion

Here, we solved our application problems using Conformal Geometric Algebra. We transferred our 3D problem with the help of geometric intuition into the Conformal Geometric Algebra, performed the geometric computing in 5D and got our results back into the real 3D world.

Geometric Algebra is applicable in many different engineering scenarios and provides a straightforward and intuitive problem solving approach. This is why this kind of algorithms is easier to understand, easier to develop and to maintain compared to algorithms not using Geometric Algebra.

References

[1] Bayro-Corrochano E., and Zamora-Esquivel J. [2004] Inverse Kinematics, Fixation and Grasping Using Conformal Geometric Algebra. In Proc. of the Int. Conf. on Intelligent Robots and Systems, IROS2004, Senday, Japan, pp. 3841-3846.

[2] Bayro-Corrochano Eduardo, [2005]. Robot perception and action using conformal geometry. In: the Handbook of Geometric Computing. Applications in Pattern Recognition, Computer Vision, Neurocomputing and Robotics. Eduardo Bayro-Corrochano (Ed.), chap. 13, pp. 405-458, Springer Verlag, Heidelberg.

[3] Dorst L. , Mann S. and Bouma Tim , GABLE: A Matlab Tutorial for Geometric Algebra, available at http://carol.wins.uva.nl/ leo/GABLE/

[4] Dorst L. , Mann S. , Geometric Algebra: A Computational Framework for Geometrical Applications, IEEE Computer Graphics and Applications 22(3):24-31, May/June 2002

[5] Leo Dorst, Chris Doran, Joan Lasenby, editors. Applications of Geometric Algebra in Computer Science and Engineering. Birkhaeuser, 2002

[6] Daniel Fontijne and Leo Dorst, Modeling 3D Euclidean Geometry, IEEE Computer Graphics and Applications, 23(2):68-78, March/April 2003

[7] D. Fontijne and T. Bouma and L. Dorst, Gaigen: A Geometric Algebra Implementation Generator, available at http://www.science.uva.nl/ga/gaigen

[8] Hestenes D. 2001. Old Wine in New Bottles: A New Algebraic Framework for Computational Geometry, in Geometric Algebra with applications in science and engineering Editors Bayro-Corrochano E. and Sobczyk G. Birkhauser, Boston, 2001

[9] Hildenbrand D., Fontijne D., Perwass Ch., Dorst L. , Geometric Algebra and its Application to Computer Graphics. Tutorial notes of the EUROGRAPHICS conference 2004 in Grenoble.

[10] D. Hildenbrand, E. Bayro-Corrochano and J. Zamora, Advanced geometric approach for graphics and visual guided robot object manipulation, proceedings of ICRA 2005 International Conference on Robotics and Automation in Barcelona, ISBN 0-7803-8915-8

[11] D. Hildenbrand, Geometric Computing in Computer Graphics using Conformal Geometric Algebra , Computers & Graphics, vol. 29, no. 5, October 2005

[12] H. Li, D. Hestenes and A. Rockwood. Generalized homogeneous coordinates for computational geometry, in [17] pages 27-52

[13] E.M.S. Hitzer. Euclidean Geometric Objects in the Clifford Geometric Algebra of Origin, 3-Space, Infinity, **11**(5) pp. 653-662,
Bulletin of the Belgian Mathematical Society - Simon Stevin (2004)

[14] Naeve and Rockwood, Geometric Algebra Siggraph 2001 course # 53

[15] Perwass C., the CLUCalc homepage,
HTML document http://www.CluCalc.info,
Cognitive Systems Group, University Kiel, 2004

[16] Christian Perwass and Dietmar Hildenbrand, Aspects of Geometric Algebra in Euclidean, Projective and Conformal Space, An Introductory Tutorial, September 2003, Download
http://www.informatik.uni-kiel.de/reports/2003/2003_tr10.pdf
http://www.gris.informatik.tu-darmstadt.de/ dhilden/

[17] Sommer G., editor. Geometric Computing with Clifford Algebra. Springer Verlag Heidelberg, 2001

[18] Tolani D., Goswami A., Badler N. , Real-time inverse kinematics techniques for anthopomorphic limbs. University of Pennsylvania, 2000

8.1 Information about the Authors

Dietmar Hildenbrand
Interactive Graphics Systems Group
University of Technology Darmstadt, Germany
E-mail: dhilden@gris.informatik.tu-darmstadt.de

Julio Zamora and Eduardo Bayro-Corrochano
Computer Science Department
Centro de Investigación y de Estudios Avanzados
Guadalajara, Jalisco 44550, Mexico
E-mail: edb,jzamora@gdl.cinvestav.mx

Submitted: Nov. 30, 2005; Revised: TBA.