2008 8th IEEE-RAS International Conference on Humanoid Robots
December 1 ~ 3, 2008 / Daejeon, Korea

Inverse Kinematics of a Humanoid Robot based on Conformal Geometric Algebra using optimized Code Generation

Joachim Pitt [1], Dietmar Hildenbrand [1], Maximilian Stelzer [2], Andreas Koch [3]

[1,2,3]*Technische Universität Darmstadt (University of Technology Darmstadt), Germany*

[1]*Faculty of Interactive Graphics Systems Group dietmar.hildenbrand@gris.informatik.tu-darmstadt.de*

[2]*Simulation & Systems Optimization & Robotics stelzer@sim.tu-darmstadt.de*

[3]*Embedded Systems & Applications Group koch@esa.informatik.tu-darmstadt.de*

***Abstract*— This paper presents a solution to solve the inverse kinematics for the legs of a humanoid robot using conformal geometric algebra. We geometrically intuitively develop the algorithm with the freely available CLUCalc software and optimize it with the help of the computer algebra system Maple and the Clifford package for geometric algebras. We describe our Gaalop code generator which produces executable C code with just elementary expressions leading to a very efficient implementation.**

I. Introduction

The main contribution of this paper are the high-level description of the inverse kinematics of the leg of a humanoid robot using conformal geometric algebra.

'Mr. DD Junior 2' (see figure 1) is a humanoid robot of about 38 cm total height. It was used for the 2005 RoboCup competition [2] where robots play soccer completely autonomously. Its legs have six degrees of freedom each: from hip to foot, the first joint rotates about the forward oriented axis, the next three joints about the sidewards oriented axis, and the last two joints about the forward and upward oriented axis. This is different from standard configuration of humanoid robot legs, where the joint that rotates about the upward oriented axis usually is located in the hip. The robot is equipped with a camera for vision, a pocket PC for computation and servo motors for actuation. The robot must localize itself on the field which has color-coded landmarks, identify other players, the location of the ball and the goal. For walking, Mr. DD Junior 2 uses the following inverse kinematics approach: The motion of the hips and feet are given by smooth trajectories, that are described by several parameters and the joint angle trajectories of the legs are computed from the hips and feet trajectories by inverse kinematics. This computation is done online on the pocket PC, which also is used for image processing.

Even for more advanced methods of generating offline reference trajectories like model based optimal control [1], inverse kinematics is used for computation of starting trajectories in the iterative process of optimizing the motion trajectories and to reduce the number of degrees of freedom in the optimization model.

In this paper, we develop the inverse kinematics algorithm in section III using the geometrically very intuitive mathematical language of conformal geometric algebra as briefly introduced in section II. In section IV we port it to the computer algebra system Maple with the Clifford package for geometric algebras. In section V we describe our Gaalop code generator generating executable C code and CLUCalc [10] code with just elementary expressions.

For the foundations of conformal geometric algebra and its application to kinematics please refer to [8] for instance, [5] and to the tutorials [6] and [4]. Runtime performance for inverse kinematics algorithms using conformal geometric algebra has been discussed in the paper [7].
Another code generator for geometric algebra with a different approach is Gaigen 2 [3]. While Gaigen 2 uses optimizations on the level of geometric objects and operations our code generator optimizes parts of algorithms combining steps of the algorithm to optimized statements.

II. Foundations of Conformal Geometric Algebra

Blades are the basic computational elements and the basic geometric entities of the geometric algebra. For example, the 5D conformal geometric algebra provides a great variety of basic geometric entities to compute with. It consists of blades with **grades** 0, 1, 2, 3, 4 and 5, whereby a scalar is a **0-blade** (blade of grade 0). There exists only one element of grade five in the conformal geometric algebra. It is therefore also called the pseudoscalar. Table II lists all the 32 blades of conformal geometric algebra. The indices indicate 1: scalar, 2..6: vector 7..16: bivector, 17..26: trivector, 27..31: quadvector, 32: pseudoscalar. A linear combination of blades is called a **k-vector**. So a bivector is a linear combination of blades with grade 2. Other k-vectors are vectors (grade 1), trivectors (grade 3) and quadvectors (grade 4). Furthermore, a linear combination of blades of different grades is called a multivector. Multivectors are the general elements of a Geometric Algebra.

Table I presents the basic geometric entities of conformal geometric algebra, points, spheres, planes, circles, lines and point pairs. The s_i represent different spheres and the π_i

978-1-4244-2822-9/08/$25.00 ©2008 IEEE

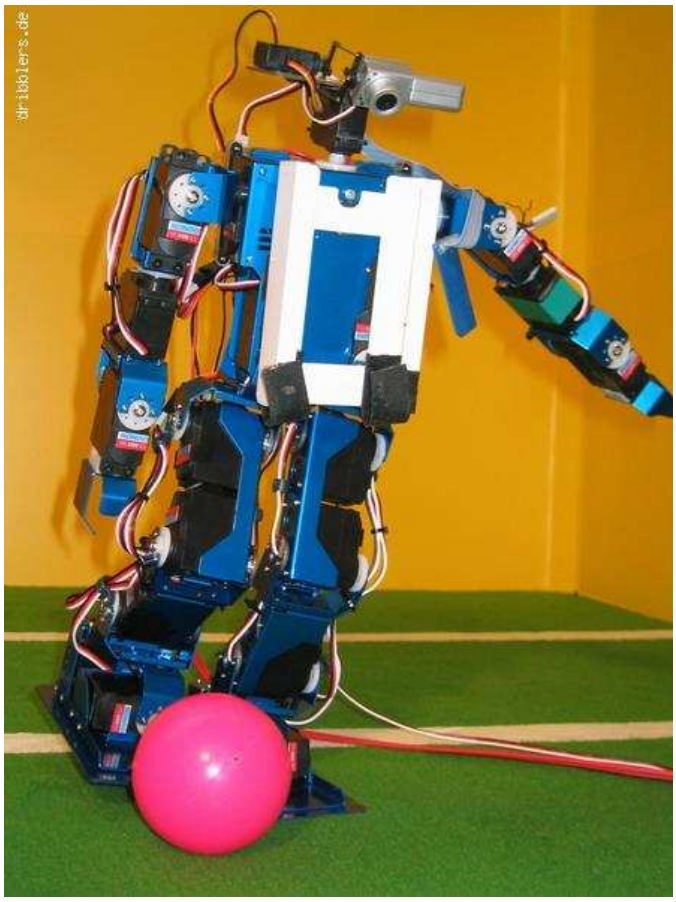

Fig. 1. The robot Mr. DD Junior 2 of the RoboCup team, the Darmstadt Dribbling Dackels (DDD)

TABLE I

REPRESENTATIONS OF THE CONFORMAL GEOMETRIC ENTITIES

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

different planes. The two representations are dual to each other. In order to switch between the two representations, the dual operator which is indicated by '*', can be used. For instance in standard representation a sphere is represented with the help of its center point P and its radius r, while in the direct representation it is constructed by the outer product '$\wedge$' of four points x_i that lie on the surface of the sphere ($x_1 \wedge x_2 \wedge x_3 \wedge x_4$). In standard representation the dual meaning of the outer product is the intersection of geometric entities. For instance a circle is defined by the intersection of two spheres ($s_1 \wedge s_2$).

TABLE II

THE 32 BLADES OF THE 5D CONFORMAL GEOMETRIC ALGEBRA

Index	blade	grade
1	1	0
2	e_1	1
3	e_2	1
4	e_3	1
5	e_∞	1
6	e_0	1
7	$e_1 \wedge e_2$	2
8	$e_1 \wedge e_3$	2
9	$e_1 \wedge e_\infty$	2
10	$e_1 \wedge e_0$	2
11	$e_2 \wedge e_3$	2
12	$e_2 \wedge e_\infty$	2
13	$e_2 \wedge e_0$	2
14	$e_3 \wedge e_\infty$	2
15	$e_3 \wedge e_0$	2
16	$e_\infty \wedge e_0$	2
17	$e_1 \wedge e_2 \wedge e_3$	3
18	$e_1 \wedge e_2 \wedge e_\infty$	3
19	$e_1 \wedge e_2 \wedge e_0$	3
20	$e_1 \wedge e_3 \wedge e_\infty$	3
21	$e_1 \wedge e_3 \wedge e_0$	3
22	$e_1 \wedge e_\infty \wedge e_0$	3
23	$e_2 \wedge e_3 \wedge e_\infty$	3
24	$e_2 \wedge e_3 \wedge e_0$	3
25	$e_2 \wedge e_\infty \wedge e_0$	3
26	$e_3 \wedge e_\infty \wedge e_0$	3
27	$e_1 \wedge e_2 \wedge e_3 \wedge e_\infty$	4
28	$e_1 \wedge e_2 \wedge e_3 \wedge e_0$	4
29	$e_1 \wedge e_2 \wedge e_\infty \wedge e_0$	4
30	$e_1 \wedge e_3 \wedge e_\infty \wedge e_0$	4
31	$e_2 \wedge e_3 \wedge e_\infty \wedge e_0$	4
32	$e_1 \wedge e_2 \wedge e_3 \wedge e_\infty \wedge e_0$	5

III. THE INVERSE KINEMATICS OF THE LEG OF A HUMANOID ROBOT

The following inverse kinematics algorithm has been developed using conformal geometric algebra to solve the 6 DOF kinematic chain for the leg of the humanoid robot 'Mr. DD Junior 2' (see figure 2). The leg consists of joints with one degree of freedom each. The hip (P_1) defines the first joint and lies in the origin, rotating about the x-axis. Three joints rotating about the y-axis, one rotating about the x-axis and a final one rotating about the z-axis follow, leading to the footpoint (P_7). The coordinates of the foot (P_x, P_y, P_z), the normal of the foot (n) and the length of the links ($l_1, l_2, l_3, l_4, l_5, l_6$) are needed to solve the inverse kinematics chain. Please refer to table III for a list of the input and output parameters of the algorithm.

A. Computation of the positions of link 5 and 6 in the kinematics chain

Since there is only a rotation about the z-axis, link 5 and 6 (P_5, P_6) are on the normal (n) of the foot. The translator T is needed to translate P_7 about l_6 in direction of n.

$$\begin{aligned} T &= 1 - \left(\frac{1}{2}\, n\, l_6\right) e_\infty \\ P_6 &= T\, P_7\, \tilde{T} \end{aligned} \tag{1}$$

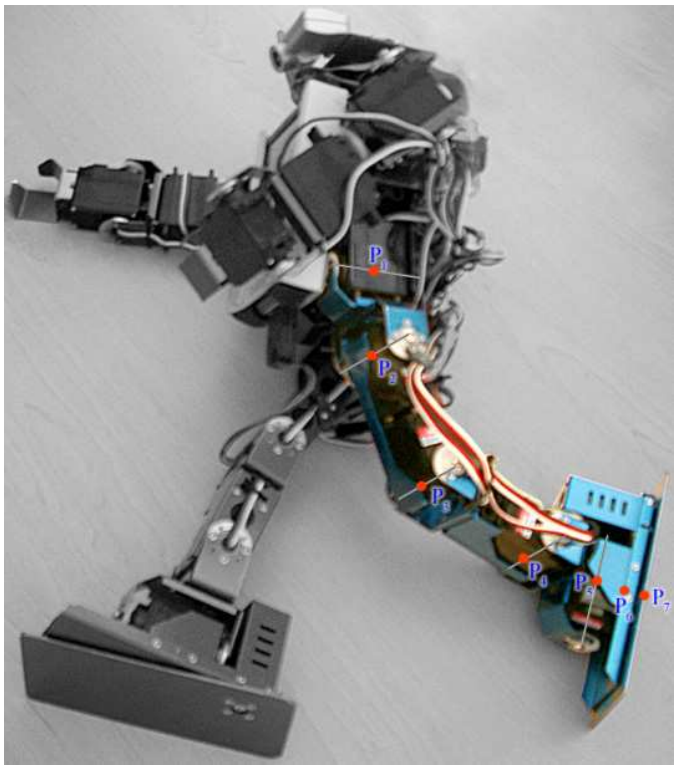

Fig. 2. The leg of the robot 'Mr. DD Junior 2' with the indication of the points P_1 to P_7

TABLE III
INPUT/OUTPUT OF THE INVERSE KINEMATICS ALGORITHM

Input	
var	**description**
P_x	foot x-value
P_y	foot y-value
P_z	foot z-value
	normal of foot x-value
n	normal of foot y-value
	normal of foot z-value
l_1	length of 1^{st} link
l_2	length of 2^{nd} link
l_3	length of 3^{rd} link
l_4	length of 4^{th} link
l_5	length of 5^{th} link
l_6	length of 6^{th} link

Output	
var	**description**
ang_1	angle at P_1
ang_2	angle at P_2
ang_3	angle at P_3
ang_4	angle at P_4
ang_5	angle at P_5
ang_6	angle at P_6

Please notice that a translator is defined by the expression $T = 1 - \frac{1}{2}\, t_{vec}\, e_\infty$ with t_{vec} being the 3D translation vector and that a translation is defined by a multiplication of the translator from the left and of its reverse from the right. Another translator (T) is necessary to compute P_5, where the distance to P_7 is $l_5 + l_6$.

$$\begin{aligned} T &= 1 - \left(\frac{1}{2}\, n\, (l_5 + l_6)\right) e_\infty \\ P_5 &= T\, P_7\, \tilde{T} \end{aligned} \tag{2}$$

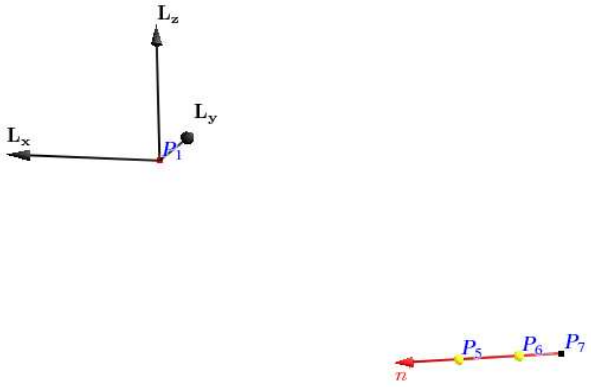

Fig. 3. Step A. - translating P_7 by l_6 and l_5 in direction of n to get P_6 and P_5

B. Computation of the position of link 4

By taking a closer look at the kinematic chain, one will notice, that P_1, P_2, P_3, P_4, P_5 define a plane π_3, which includes the x-axis. Since the joints in P_2, P_3 and P_4 all rotate about the y-axis, these and the joints directly connected to them (P_1 and P_5), are all on one plane. Every plane can be defined by 3 points, which here are P_1, P_5 and the auxiliary point on the x-axis P_{H2}. Another plane, defined by the points P_4, P_5, P_6, P_7, is orthogonal to plane π_3. So the projection of the line through P_5 and P_7 onto the plane π_3 yields L_{Proj}, which intersects the sphere S_5 (center: P_5, radius: l_4), resulting in a point pair. Function `pp_get2nd` selects link 4 (P_4) from it.

$$\begin{aligned} S_5 &= \text{Sphere}(P_5, l_4) \\ \pi_3 &= (P_{H2} \wedge P_1 \wedge P_5 \wedge e_\infty)^* \\ \pi_3 &= \frac{\pi_3}{|\pi_3|} \\ L_{P5P7} &= (P_5 \wedge P_7 \wedge e_\infty)^* \\ L_{Proj} &= \frac{(\pi_3 \cdot L_{P5P7})}{\pi_3} \\ P_4 &= \text{pp\_get2nd}((S_5 \wedge L_{Proj})^*) \end{aligned} \tag{3}$$

The used function `Sphere` generates a sphere around parameter 1 with the radius of parameter 2.

$$\text{Sphere}(x, r) = x - \frac{1}{2}\, r^2\, e_\infty \tag{4}$$

The functions `pp_get1st` and `pp_get2nd` each pick one point out of a point pair.

$$\text{pp\_get1st}(x) = \frac{\sqrt{|x \cdot x|} - x}{e_\infty \cdot x} \tag{5}$$

$$\text{pp\_get2nd}(x) = \frac{-\sqrt{|x \cdot x|} + x}{e_\infty \cdot x} \tag{6}$$

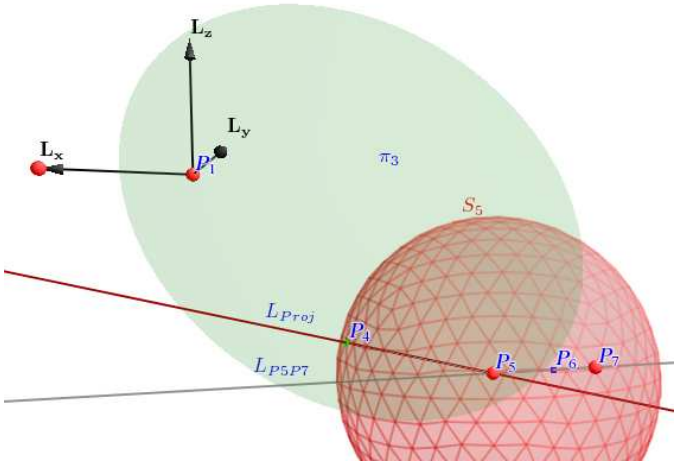

Fig. 4. Step B. - Projection of the line through P_7 and P_5 onto the green plane defined by P_1, P_5 and an auxiliary point on the x-axis. Intersection of the Sphere around P_5 with radius l_4 and the projected line returns P_4

C. Computation of the position of link 2

Link 1 and 2 are located on the yz-plane π_1 respectively, so the intersection of planes π_1 and π_3 results in a line, with P_1 and P_2 on it. The distance between P_2 and P_1 is l_1, hence the intersection of the sphere S_1 around P_1 with radius l_1 results in a point pair, from which P_2 can be selected.

$$\begin{aligned} \pi_1 &= e_1 \\ S_1 &= \text{Sphere}(P_1, l_1) \\ L_1 &= \pi_1 \wedge \pi_3 \\ P_2 &= \text{pp\_get1st}((L_1 \wedge S_1)^*) \end{aligned} \tag{7}$$

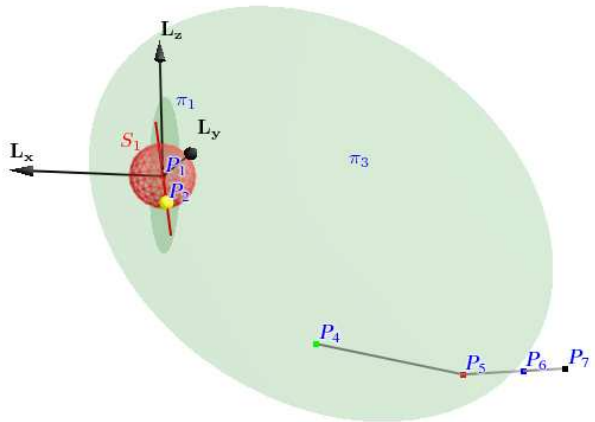

Fig. 5. Step C. - Intersecting the sphere around P_1 with radius l_1 with the intersection of the plane π_3 and the yz-plane returns P_2

D. Computation of the position of link 3

The intersection of the two spheres S_2 and S_4 results in a circle Z_3. P_3 must be located on circle Z_3 and on plane π_3 as well, thus the intersection of Z_3 and π_3 results in a point pair again, from which P_3 can be selected.

$$\begin{aligned} S_2 &= \text{Sphere}(P_2, l_2) \\ S_4 &= \text{Sphere}(P_4, l_3) \\ Z_3 &= S_2 \wedge S_4 \\ P_3 &= \text{pp\_get1st}((Z_3 \wedge \pi_3)^*) \end{aligned} \tag{8}$$

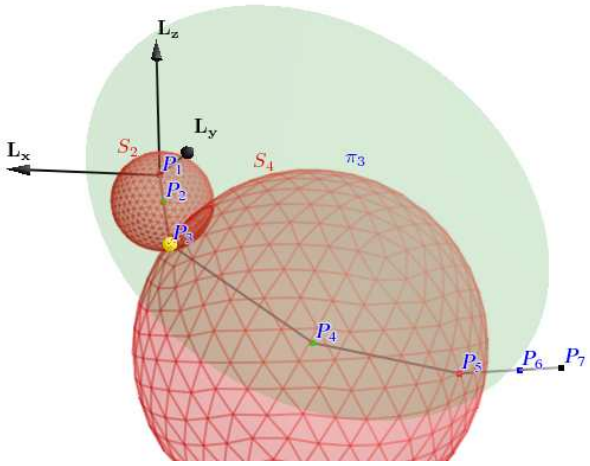

Fig. 6. Step D. - The intersection of the spheres around P_2 with radius l_2 and around P_4 with radius l_3 results in the red circle. Intersecting the circle with the plane π_3 returns P_3

E. Compute the angles of the links

Since a line is defined by 2 points, 3 points are necessary to generate 2 intersecting lines and to compute the angle in between. To compute the angle of the first link, a point above the origin (0,0,1) is used as a parameter.

$$\text{angle}(x,y,z) = \pi - \arccos\left(\frac{(x \wedge y \wedge e_\infty) \cdot (z \wedge y \wedge e_\infty)}{|x \wedge y \wedge e_\infty| \; |z \wedge y \wedge e_\infty|}\right) \tag{9}$$

IV. Geometric Algebra Computations with Maple

The most important feature of Maple [9] we use, is its symbolic calculation functionality. In order to deal with the computation of conformal geometric algebra, the Clifford package is used, which was developed by Rafal Ablamowicz and Bertfried Fauser. The most important operations of the Clifford package are presented in table IV. For the inner product we use the left contraction LC operation.

TABLE IV
Notation of GA-Operations

Notation	Maple Notation	Meaning
ab	a &c b	geometric product
$a \wedge b$	a &w b	outer product
$a \cdot b$	LC(a,b)	inner product
a^*	-(a) &c e12345	dualization
$\tilde{a}$	reversion(a)	reversion

Besides the main operations in table IV, the functions `scalarpart()` and `vectorpart()` for extracting the scalar or the vector part of a multivector are also needed. In order to use conformal geometric algebra computation we have to load the Clifford package, set the metric of Clifford algebra, set aliases to basic blades and define e_0 and e_∞.

```
> with(Clifford);
> B:=linalg[diag](1, 1, 1, 1, -1);
> eval(makealiases(5, "ordered"));
> e0 := -0.5*e4+0.5*e5;
> einf := e4+e5;
```

We define some often needed functions:

```
> dual := proc(x)
>   local dual;
>   global e12345;
>   dual:= -x &c e12345;
>   RETURN(dual);
> end:

> angle := proc(x,y,z)
>   local angle,a,b;
>   global einf;
>   a := x &w y &w einf;
>   b := z &w y &w einf;
>   angle := Pi-arccos(scalarpart(a &c b)
          / (sqrt(scalarpart(a &c a))
          * sqrt(scalarpart(b &c b))));
>   RETURN(angle);
> end:

> VecN3 := proc(x,y,z)
>   global e0,e1,e2,e3,einf;
>   local vector;
>   vector := x*e1 + y*e2 + z*e3
           + 0.5*(x^2+y^2+z^2)*einf+e0;
>   RETURN(vector);
> end:
```

The function `dual` computes the dual representation using the geometric product as described in table IV. The function `angle` used in (9), computes the angle between two lines, that are created each with 2 points on it and e_∞ according to the direct representation in table I. Since both lines share one point, the angle between the lines is computed in this point. The function `VecN3` generates a conformal point according to the standard representation in table I and is conform to the correspondent CLUCalc instruction.

```
> P1 := VecN3(0,0,0);
> P7 := VecN3(-px,py,pz);

> PH1 := VecN3(0,1,0);
> PH2 := VecN3(-1,0,0);

> normal := FNorm[1] &c e1
          - FNorm[2] &c e2
          - FNorm[3] &c e3;
> normal := normal / mul_abs(normal);
```

Hip (P_1), foot (P_7) and two auxiliary points (P_{H1} and P_{H2}) are set. The normal of the foot is generated and normalized.

```
> T := 1 - ((normal * (len[6]))/2) &c einf;
> P6 := vectorpart(T &c P7
      &c reversion(T),1);
```

A translator T is created to compute the position of P_6, which is located on the normal of the foot (1).

```
> T := 1 - ((normal * (len[5]+len[6]))/2)
     &c einf;
> P5 := vectorpart(T &c P7
      &c reversion(T),1);
```

P_5 is computed (2).

```
> S5 := Sphere(P5,len[4]);
> Plane3 := dual(PH2 &w e0 &w P5 &w einf);
> Plane3 := Plane3 / mul_abs(Plane3);
> LP5P7 := dual(P5 &w P7 &w einf);
> LProj := LC(Plane3,LP5P7)
         &c inverses(Plane3);
> S5LProj := S5 &w LProj;
> dual_S5LProj := dual(S5LProj);
> P4 := pp_get2nd(dual_S5LProj);
```

According to (3) S_5, π_3 (`Plane3`), L_{P5P7}, L_{Proj} and P_4 are computed. For maple calculation issues, we fragment the equation:

$$P_4 = \text{pp\_get2nd}((S_5 \wedge L_{Proj})^*) \tag{10}$$

```
> Plane1 := e1;
> S1 := Sphere(P1,len[1]);
> Line1 := Plane1 &w Plane3;
> Line1S1 := Line1 &w S1;
> dual_Line1S1 := dual(Line1S1);
> P2 := pp_get1st( dual_Line1S1 );
```

Corresponding to (7) π_1 (`Plane1`), S_1, L_1 (`Line1`) and P_2 are computed. Again for calculation issues, we fragment the following equation

$$P_2 = \text{pp\_get1st}((L_1 \wedge S_1)^*) \tag{11}$$

```
> S2 := Sphere(P2,len[2]);
> S4 := Sphere(P4,len[3]);
> Z3 := S2 &w S4;
> pp_P3 := Z3 &w Plane3;
> dual_pp_P3 := dual(pp_P3);
> P3 := pp_get1st( dual_pp_P3 );
```

In accordance with (8) S_2, S_4, Z_3 and P_3 are computed. Again for calculation issues, we fragment the following equation

$$P_3 = \text{pp\_get1st}((Z_3 \wedge \pi_3)^*) \tag{12}$$

```
> ang[1] := angle(PH1,P1,P2);
> ang[2] := angle(P1,P2,P3);
> ang[3] := angle(P2,P3,P4);
> ang[4] := angle(P3,P4,P5);
> ang[5] := angle(P4,P5,P6);
```

and compute the angles at the links according to (9).

V. Code Generator

Our Gaalop [11] code generator is a tool that communicates with Maple to simplify expressions symbolically. Its input is a Maple or CLUCalc algorithm like the one just described, producing executable C code, CLUCalc code or LaTeX formulas as output.

The main data structure of our code generator is the multivector using the basic blades $blade_1, \cdots, blade_{32}$ as listed in table II. For performance measures and future plans concerning hardware, our Gaalop tool splits up the calculations for the resulting expressions to their components of the 32 blades they contain. These blades only consist of elementary expressions with no multivector operations.

Every dissected expression is rebuilt of its 32 blades. Regard that only few coefficients have a value other than zero. The optionally following symbolic simplifications use the rebuilt expressions instead of the original ones, which then lead to shorter expressions.

The algorithm now only consists of elementary expressions and does not have to deal with (multi)vectors.

As an example, lets have a look at the code generated for the following equation, which is part of of (3)

$$L_{P5P7} = (P_5 \wedge P_7 \wedge e_\infty)^* \tag{13}$$

This line through the points P_5 and P_7 is represented by the following line of Maple code

```
> LP5P7 := dual(P5 &w P7 &w einf);
```

as an assignment to the multivector `LP5P7`. We automatically generate the following code with assignments to the corresponding array `LP5P7`:

```
LP5P7[7] = P5[4]-pz*P5[6];
LP5P7[8] = P5[3]-py*P5[6];
LP5P7[9] = P5[4]*py-P5[3]*pz;
LP5P7[11] = px*P5[6]+P5[2];
LP5P7[12] = P5[4]*px+P5[2]*pz;
LP5P7[14] = -P5[2]*py-P5[3]*px;
```

Only blades within the range $blade_7, \cdots, blade_{16}$ appear, which are all of grade 2. This matches the definition of lines, since these are bivectors. `P5[2]`, `P5[3]`, `P5[4]` and `P5[6]` were already computed in a previous step, `px`, `py` and `pz` are the foot values according to table III.

Please notice that, with these optimization results, equation (13) could also be written as follows

$$\begin{aligned} L_{P5P7} = {} & LP5P7[7] \cdot (e_1 \wedge e_2) + LP5P7[8] \cdot (e_1 \wedge e_3) \\ & + LP5P7[9] \cdot (e_1 \wedge e_\infty) + LP5P7[11] \cdot (e_2 \wedge e_3) \\ & + LP5P7[12] \cdot (e_2 \wedge e_\infty) + LP5P7[14] \cdot (e_3 \wedge e_\infty) \end{aligned}$$

by using the computed coefficients and their corresponding blades.

VI. Results and Future Work

We optimized the previously used inverse kinematics algorithm by hand resulting in approximately 200 additions, 300 multiplications and about 30 other operations like square roots. The comparison with the automatically generated implementation based on the described code generator resulted in the same magnitude of basic operations.

Compared with the original robot algorithm, our visually developed inverse kinematics algorithm, based on optimized code, turned out to be about ten times slower concerning multiplications and additions. This is due to several still optimizable intermediate steps and redundancy in some of our expressions. Since our focus lies on moving towards parallel architectures like FPGAs and graphic cards, the redundant computations do not matter to us. We still see a lot of potential in what is possible with future automatic optimization to get the same magnitude of basic operations or even better.

VII. Conclusion

Inverse kinematics is an important topic in robotics. It does not have analytical solutions for all joint configurations, and also if it has, efficient computation often is difficult. Conformal geometric algebra gives an geometrically intuitive approach for solving it. Furthermore, our Gaalop code generator is able to automatically generate an implementation that is able to compete with optimizations done by hand.

References

[1] M. Buss, M. Hardt, J. Kiener, M. Sobotka, M. Stelzer, O. von Stryk, and D. Wollherr. Towards an autonomous, humanoid, and dynamically walking robot: modeling, optimal trajectory planning, hardware architecture, and experiments. Third IEEE International Conference on Humanoid Robots, SeptemberOctober 2003.

[2] T. R. Federation. Robocup official site. http://www.robocup.org.

[3] D. Fontijne, T. Bouma, and L. Dorst. Gaigen 2: A geometric algebra implementation generator. http://staff.science.uva.nl/ fontijne/gaigen2.html.

[4] D. Hildenbrand. Geometric computing in computer graphics using conformal geometric algebra. Computers & Graphics, 29(5):802–810, 2005.

[5] D. Hildenbrand, E. Bayro-Corrochano, and J. Zamora-Esquivel. Advanced geometric approach for graphics and visual guided robot object manipulation. In proceedings of ICRA conference, Barcelona, Spain, 2005.

[6] D. Hildenbrand, D. Fontijne, C. Perwass, and L. Dorst. Tutorial geometric algebra and its application to computer graphics. In Eurographics conference Grenoble, 2004.

[7] D. Hildenbrand, D. Fontijne, Y. Wang, M. Alexa, and L. Dorst. Competitive runtime performance for inverse kinematics algorithms using conformal geometric algebra. In Eurographics conference Vienna, 2006.

[8] S. M. L.Dorst, D. Fontijne and M. Kaufman. Geometric Algebra for Computer Science, An Object-Oriented Approach to Geometry. Morgan Kaufman, 2005.

[9] The homepage of Maple. http://www.maplesoft.com/products/maple. 615 Kumpf Drive, Waterloo, Ontario, Canada N2V 1K8.

[10] C. Perwass. The CLU home page. HTML document http://www.clucalc.info, 2005.

[11] J. Pitt and D. Hildenbrand. The Gaalop homepage. HTML document http://www.gaalop.de, 2008.