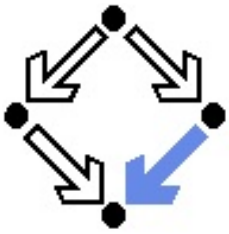
JOHANNES KEPLER
UNIVERSITÄT LINZ | JKU

TNF

Technisch-Naturwissenschaftliche
Fakultät



Flexible solutions of polynomial systems and their applications in robotics

(Flexible Lösungen von polynomen Systemen und deren Anwendung in der Robotik)

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Masterstudium

Computermathematik

Eingereicht von:
Michael Steglehner, Bakk.techn.

Angefertigt am:
Research Institute for Symbolic Computation

Beurteilung:
Univ.-Prof. DI. Dr. Franz Winkler

Linz, 20. Sep. 2014

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.
Die vorliegende Masterarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, 26. Sep. 2014

Michael Steglehner

Abstract

This thesis is concerned with the kinematics of an robot arm, more precisely with the flexibility of the 3-dimensional robot arm for a fixed robot hand position. The main question of this thesis is: How can we move the robot arm if we fix the two ends of the robot arm in a 3-dimensional space?

We want to solve the mathematical model of the kinematics of the robot arm in an algebraical way. This mathematical model will be a system of polynomial equations. Therefore we take a look at the theory of polynomial ideals, elimination theory, Gröbner basis and resultants. Especially the relations between ideals and algebraic sets are essential in elimination theory, which in turn is essential for solving systems of polynomial equations. In elimination theory we have to consider two steps, the elimination step and the extension step. Gröbner basis and resultants are two common elimination steps.

Before we can solve the system of polynomial equations we have to model it. So we have to take a look how we can put the robot arm in a mathematical model. Then we have to derive a polynomial model of the general model. Then we construct a polynomial ideal of the polynomials in the model. On this ideal we can use the elimination steps of the theory. Afterwards we should be able to solve simple univariate polynomials and extend the solutions in the extension step.

The thesis also contains a maple program to calculate and simulate a given robot arm for a fix hand position. For a better perception the thesis is supported by examples of a Stanford Manipulator.

Acknowledgements

I would like to thank my supervisor Franz Winkler for his patience, motivation, enthusiasm and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for writing my master thesis.

I also thank Manfred Minimayr who hosted me for one semester at Seton Hall University, started with me the idea to this thesis and introduced me to resultants.

Finally I want to thank my family and friends for supporting me during my studies.

Contents

1. General definitions

The first chapter of this thesis gives attention to the theory we need to solve systems of polynomial equations. Therefore we start with an explanation of polynomials and their properties.

1.1. Polynomials

Before we start with the details we have to clear some definitions. What are polynomials? What is a polynomial ring?

The ideas for the following definitions are from the books [CLO04], [CLO96] by Cox, Little and O'Shea and [Win96] by F. Winkler.

Definition 1.1

*Let $X = \{x_1, \ldots, x_n\}$ be a set of n variables. A **power product** or **term** in X is a product of the form:*

$$x^\alpha = x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdots x_n^{\alpha_n}$$

where $\alpha = (\alpha_1, \ldots, \alpha_n)$ is a multi-exponent of non-negative integers, i.e., $\alpha_i \in \mathbb{N}$.

For the **set of all terms** or power products in n variables we write $[X]$.

We can multiply two terms $x^\alpha \cdot x^\beta$ by adding the associate exponents $\alpha_i + \beta_i$. With this multiplication and the neutral element $1 = x_1^0 \cdot x_2^0 \cdots x_n^0$ w.r.t. multiplication, $[X]$ is a monoid.

Definition 1.2

*Consider the term $x^\alpha = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$. Then the **total degree** of this term (or power product) is $deg(x^\alpha) = |\alpha| = \alpha_1 + \cdots + \alpha_n$ and the **partial degree** in x_i is α_i, we write $deg_{x_i}(x^\alpha) = \alpha_i$.*

We can divide two terms, x^α and x^β, if and only if all partial degrees $\alpha_i \geq \beta_i$. Then the result is

$$\frac{x^\alpha}{x^\beta} = x^{\alpha_1-\beta_1} \cdots x^{\alpha_n-\beta_n}.$$

Now we can define polynomials over a commutative ring.

Definition 1.3

*A **polynomial** a in $X = \{x_1, \ldots, x_n\}$ with coefficients in a commutative ring R (over R) is a finite sum of the form*

$$a = \sum_{\alpha \in \mathbb{N}^n} a_\alpha x^\alpha$$

where the $a_\alpha \in R$ and for all but finitely many α we have $a_\alpha = 0$.

*The summands $a_\alpha x^\alpha$ in a are denoted as **monomials** in a.*
By $R[X]$ we denote the set of all polynomials in X over R.

Definition 1.4

Let $a = \sum_\alpha a_\alpha x^\alpha$ be a polynomial in $R[X]$.

1. *a_α is called the **coefficient** of the term x^α in a.*
2. *The **total degree** of f, $deg(f)$, is the maximal degree of the terms in f with a non-zero coefficient.*

At the moment a polynomial is just an algebraic expression. If we want to do calculations with polynomials, we need to define polynomial functions.

Definition 1.5

Let a be a polynomial over R in n variables, $A \subset \mathbb{N}^n$ the finite set of multi-exponents with a non-zero coefficient in a and $(b_1, \ldots, b_n)$ a vector in R^n. Then

the function

$$f_a : R^n \longrightarrow R$$
$$f_a(b_1, \ldots, b_n) = \sum_{\alpha \in A} a_\alpha b_1^{\alpha_1} \ldots b_n^{\alpha_n}$$

is called the ***polynomial function*** *of* a.

Now we define some expressions and definitions of terms and polynomials.

Definition 1.6

Let s *be a power product in* $[X]$, f *a non-zero polynomial in* $R[X]$ *and* $F \subset R[X]$. *By*

coeff(f, s) *we denote the coefficient of* s *in* f

$lpp(f) := max_<\{t \in [X] |$ *coeff*$(f, t) \neq 0\}$

$lc(f) :=$ *coeff*$(f, lpp(f))$

$lm(f) := lc(f)lpp(f)$

$red(f) := f - lm(f)$

$lpp(F) := \{lpp(f) | f \in F \setminus \{0\}\}$

$lc(F) := \{lc(f) | f \in F \setminus \{0\}\}$

$lm(F) := \{lm(f) | f \in F \setminus \{0\}\}$

$red(F) := \{red(f) | f \in F \setminus \{0\}\}$

lpp is the acronym of "leading power product", lc of the "leading coefficient", lm of "leading monomial" and red is the acronym of "reductum".

Ring properties of polynomials

If we want to add two polynomials p, q we simply add the coefficients of the associated terms.

Definition 1.7

Let $a, b \in R[X]$ so that,

$$a(x) = a_{\alpha^{(n)}} x^{\alpha^{(n)}} + \cdots + a_{\alpha^{(1)}} x^{\alpha^{(1)}}$$
$$b(x) = b_{\alpha^{(m)}} x^{\alpha^{(m)}} + \cdots + b_{\alpha^{(1)}} x^{\alpha^{(1)}}.$$

Then the sum of the two polynomials is

$$a(x) + b(x) = (a_{\alpha^{(n)}} + b_{\alpha^{(n)}}) x^{\alpha^{(n)}} + \cdots + (a_{\alpha^{(1)}} + b_{\alpha^{(1)}}) x^{\alpha^{(1)}}.$$

So the degree of the new polynomial is $max(deg(p), deg(q))$, if the leading monomials are not cancelled each other, $lm(p) = -lm(q)$.

The zero polynomial 0, the polynomial where all coefficients are $0 \in R$, is the neutral element w.r.t. addition. $-a$,the polynomial we get if we replace all non-zero coefficients with their inverse in R, is the inverse polynomial of a w.r.t. addition. Thus we see $a + (-a) = 0$. If R is commutative w.r.t. addition then also the addition of two polynomials is commutative.

If we want to multiply two polynomials p, q we have to multiply each monomial of p with each monomial of q.

Definition 1.8

Let $p, q \in R[X]$.

$$p(x) = a_{\alpha^{(n)}} x^{\alpha^{(n)}} + \cdots + a_{\alpha^{(1)}} x^{\alpha^{(1)}}$$
$$q(x) = b_{\alpha^{(m)}} x^{\alpha^{(m)}} + \cdots + b_{\alpha^{(1)}} x^{\alpha^{(1)}}$$

Then the product of the two polynomials

$$p(x) \cdot q(x) = \sum_{i=1}^{n} \sum_{j=1}^{m} a_{\alpha^{(i)}} \cdot b_{\alpha^{(j)}} \cdot x^{\alpha^{(i)} + \alpha^{(j)}}$$

So the degree of our new polynomial is $deg(p) + deg(q)$.

The 1 polynomial where all coefficients of $x_\alpha \in [X] \setminus 1$ are $0 \in R$ and the coefficient of $1 \in [X]$ is $1 \in R$ is the neutral element w.r.t. multiplication.

The multiplication in $R[X]$ inherits the properties of the multiplication in R.

Therefore we can derive the lemma:

Lemma 1.9

Let R be a commutative Ring with identity and $[X]$ the set of n variables. Then the set of all polynomials $R[X]$ is a commutative ring with identity w.r.t. to the addition and multiplication defined above.
*Therefore $R[X]$ is the **polynomial ring** in the variables X over R.*

Properties of polynomials

Because we work over fields in this thesis we mainly write $K[X]$ for the polynomial ring over the field K.

Proposition 1.10

Let K be an infinite field, and let $f \in K[X]$. Then $f = 0$ in $K[X]$ if and only if $f : K^n \to K$ is the zero function.

One important property of the polynomial ring is the noetherian property. We call a relation **Noetherian** if it fulfils an ascending or decreasing chain condition. For instance in the natural numbers there is no infinite strictly decreasing sequence.

Definition 1.11

*Let R be a commutative ring. Then a subset $I \subseteq R$ is called an **ideal** of R if and only if*

- $0 \in I$,
- $a, b \in I \Rightarrow a + b \in I$,
- $a \in I, r \in R \Rightarrow a \cdot r \in I$.

Definition 1.12

*We say that the **basis condition** holds for a commutative ring with identity R, if every ideal $I \subset R$ has a finite basis. If the basis condition holds, we call R a **Noetherian ring**.*

Lemma 1.13

A commutative ring with identity R is a Noetherian Ring if and only if there are no infinite strictly ascending chains of ideals in R. I.e., if

$$I_1 \subset I_2 \subset \cdots \subset R,$$

then there is an index k such that

$$I_k = I_{k+1} = \ldots.$$

Proof: Assume R is a Noetherian Ring and let $I_1 \subset I_2 \subset I_3 \subset \ldots$ be an ascending chain of ideals in R.

$$I := \bigcup_{i=0}^{\infty} I_i$$

We know that I is an ideal in R, therefore it has a finite basis. This basis must be contained in some I_k, so $I_k = I_{k+1} = \ldots$.

The other way round, assume that an ideal I is not finitely generated. Choose a non-zero element $r_1 \in I$, then $I_1 := \langle r_1 \rangle \neq I$. Extend I_1 by an element $r_2 \in I \setminus I_1$; $I_2 := \langle r_1, r_2 \rangle \neq I$. This step can be repeated indefinitely to get an infinite strictly ascending chain of ideals in R. ■

Theorem 1.14 (*Hilbert's basis theorem*)

If R is a Noetherian ring, then also the univariate polynomial ring $R[x]$ is Noetherian.

Proof: A proof of the Hilbert's basis theorem can be found in van der Waerden's book [vdW67, Ch. 15] ■

We can extend this theorem to the polynomial ring $R[X]$. To show this we just have to do an induction on the number of variables n.

Because the only two ideals of a field K, $\langle 0 \rangle, \langle 1 \rangle$, are finitely generated, the following corollary is true.

Corollary 1.15

For any finite set of variables X, $K[X]$ is a Noetherian ring.

Now let us define a concept similar to prime numbers, the reducibility or rather irreducibility of polynomials.

Definition 1.16

*Let $K \subseteq L$ be fields. A polynomial $p \in K[X]$ is **reducible** over L if and only if there are two non-constant polynomials $f, g \in L[X]$ so that $p = f \cdot g$.*
*Otherwise p is **irreducible** over L.*

Note that irreducibility depends on the field. For instance $x^2 + 4$ is irreducible over the field $\mathbb{R}$, but you can reduce it to $x^2 + 4 = (x - 2i) \cdot (x + 2i)$ over the field $\mathbb{C}$.

Similar to prime factorization in the integers there is an irreducible factorization of polynomials.

Theorem 1.17

Every non-constant $p \in K[X]$ can be written as a product $p = p_1 \cdots p_r$ where all $p_i \in K[X]$ are irreducible polynomials over K. Furthermore, this factorization is unique up to permutation and multiplication by constants.

Proof: The first part of the proof we can do with induction on the degree of p. From Definition 1.16 we see that polynomials of degree 0 are irreducible. Assume all polynomials of degree $\leq n$ can be written as a product of irreducibles.

Then for a polynomial p of degree $n+1$, we have to consider two cases.

- p is irreducible. Then we are done.
- $p = f \cdot g$ where $f, g \in K[X]$ with degree $\leq n$. By the induction hypothesis f, g can be written as product of irreducible polynomials. And therefore also p can be written as product of irreducible polynomials.

Thus every polynomial can be factorized into irreducible polynomials.

For the uniqueness we first need an additional theorem. ■

Proposition 1.18

Assume $p \in K[X]$ is an irreducible polynomial over K. Furthermore p divides the product $g \cdot h$ with $g, h \in K[X]$. Then p divides g or h.

Proof: A proof for this proposition can be found in [CLO96, p.147 Thm. 3.]. ■

Proof CONTINUATION Thm. 1.17: To proof the uniqueness of the factorization we assume that p has two irreducible factorizations $f_1 \cdots f_r = p = g_1 \cdots g_s$ for non-constant f_l and g_k.
g_1 divide p, thus by Thm. 1.18 has to divide some f_j. Because g_1 and f_j are irreducible they have to be equal up to a non-zero constant, $g_1 = c_1 f_j$.
We can apply the same approach to $g_2 \cdots g_s$ and $f_1 \cdots f_{j-1} f_{j+1} \cdots f_r$.

Thus there is a f_j for every g_i with $g_i = c_i f_i$. It is easy to see that $r = s$, otherwise the degree of the two products would not be the same. ■

You can also show that the product $c_1 \cdots c_s = 1$. This is left to the reader.

Algorithms to factorize polynomials over $\mathbb{Q}$ into irreducibles are well known and implemented in common computer algebra systems. Factorizing polynomials $p \in \mathbb{Q}[X]$ over $\mathbb{R}$ and $\mathbb{C}$ is also possible but much more difficult.

For every polynomial ring $K[X]$ we can consider the quotient field $K(X)$. Quotient fields are the smallest polynomial fields where it is possible to divide polynomials, regardless if the divisor is a factor of the dividend or not. For the definition of quotient fields see [Win96, Ch. 2.3].

At last we consider a corollary to Thm. 1.18 and Gauss's Lemma, which will be useful if we work with resultants.

Corollary 1.19

Assume $f, g \in K[X]$ have positive partial degrees in x_1. Then f and g have a common factor in $K[X]$ of positive degree in x_1 if and only if they have a common factor in $K(x_2, \ldots, x_n)[x_1]$.

1.2. Term orderings

We know that polynomials are sums of monomials, or more precisely terms. So to arrange polynomials or compare two polynomials we have to compare terms. Thus we have to introduce an ordering on the terms $[X]$.

If we have polynomials in just one variable ($K[x]$) we order the term by the degree, $x^{t+1} > x^t$. If we have linear polynomials in $K[x_1, \ldots, x_n]$ we will choose an ordering like $x_1 > x_2 > \cdots > x_n$. Of course we can permute the order of the variables.

For general polynomials in $K[X]$ the terms have the form $x^\alpha = x_1^{\alpha_1} \cdot \dots \cdot x_n^{\alpha_n}$, so the exponent α is a n-tuple in $\mathbb{N}^n$. This means that there is an isomorphism from the set of terms in n variables to $\mathbb{N}^n$. So we can choose an ordering on $\mathbb{N}^n$ and say $x^\alpha > x^\beta$ if and only if $\alpha > \beta$.

After some operations on polynomials the terms in the resulting polynomial are not arranged. So it would be nice if we could rearrange this polynomial. To arrange a polynomial in the correct order two arbitrary terms have to be comparable. In other words, the ordering has to be total (or linear). An ordering is **total** if for every pair of terms x^α and x^β exactly one of the three statements

$$x^\alpha < x^\beta$$
$$x^\alpha = x^\beta$$
$$x^\alpha > x^\beta$$

is true.

Therefore we define a term ordering in the following way:

Definition 1.20

*A **term ordering** on $[X]$ is a relation $<$ on the set of terms $x^\alpha, \alpha \in \mathbb{N}^n$ (or equivalently, a relation on $\mathbb{N}^n$) , satisfying:*

1. $1 = x_1^0 \dots x_n^0 < t$ *for all* $t \in [X] \setminus \{1\}$,
2. $s < t \Rightarrow su < tu$ *for all* $s, t, u \in [X]$.
3. $<$ *is a total ordering.*

We see that a term ordering is compatible with the monoid structure. Therefore a term ordering is also called admissible ordering.

The following lemma will be important when we want to show the termination property of our algorithms, which uses a term ordering.

Definition 1.21

An ordering $>$ on a set A is a well-ordering if and only if every strictly decreasing sequence in A

$$a_1 > a_2 > a_3 > \ldots \qquad a_i \in A$$

eventually terminates.
In other words, every non-empty subset of $\mathbb{N}^n$ has a smallest element under $<$.

It is easy to see that the term ordering is a well-ordering. The following Lemma provides some properties of orderings on $[X]$. We also connect the mathematical terms noetherian and well-ordered.

Lemma 1.22 (*[Win96, L. 8.2.3]*)

Let $<$ be a term ordering on $[X]$.

1. *If $s, t \in [X]$ and s divides t then $s \leq t$.*
2. *$<$ is Noetherian and consequently every subset of $[X]$ has a smallest element.*

There are many different term orderings on $\mathbb{N}^n$ but not all of them are effective. The most common ones are the lexicographic ordering and the graduated lexicographic ordering.

Definition 1.23 (*Lexicographic ordering*)

Let $\alpha, \beta \in \mathbb{N}^n$ and π be a permutation on $[1..n]$. We say $\alpha >_{lex,\pi} \beta$ if

$$\exists i \in [1..n]\ \forall j \in [1..i-1] : \alpha_{\pi(j)} = \beta_{\pi(j)} \text{ and } \alpha_{\pi(i)} > \beta_{\pi(i)}$$

With $\pi = id$ we get the usual lexicographic ordering.

Definition 1.24 (*Graduated lexicographic ordering*)

Let $\alpha, \beta \in \mathbb{N}^n$, π be a permutation on $[1..n]$ and $w : [1..n] \to \mathbb{R}_+$ a weight

function. We say $\alpha >_{glex} \beta$ *if*

$$\sum_{i=1}^{n} w(i)\alpha_i > \sum_{i=1}^{n} w(i)\beta_i \qquad \textit{or} \qquad \sum_{i=1}^{n} w(i)\alpha_i = \sum_{i=1}^{n} w(i)\beta_i \textit{ and } \alpha >_{lex,\pi} \beta$$

For the usual graduated lexicographic ordering we choose $w(i) = 1$.

Definition 1.25 (*Graduated reverse lexicographic ordering*)

Let $\alpha, \beta \in \mathbb{N}^n$, π. *We say* $\alpha >_{grlex} \beta$ *if*

$$|\alpha| = \sum_{i=1}^{n} \alpha_i > |\beta| = \sum_{i=1}^{n} \beta_i, \quad \textit{or} \quad |\alpha| = |\beta| \textit{ and } \alpha >_{lex,\pi} \beta$$

where $\pi(i) = n - j - 1$.

There are other term orderings. One also can connect different orderings on a set of variables. We can do that by partitioning the set of variables and putting an ordering on each partition. Then we define a sequence of the different orderings.

We summarize the previous definitions to introduce an ordering on polynomials.

Definition 1.26

Any term ordering $<$ *on* $[X]$ *induces a partial ordering* $\ll$ *on* $R[X]$, *the* ***induced ordering***, *for* $g \neq 0$ *in the following way:*

$$\begin{aligned} f \ll g \textit{ iff} \quad & f = 0 \textit{ and } g \neq 0 && \textit{or} \\ & lpp(f) < lpp(g) && \textit{or} \\ & lpp(f) = lpp(g) \textit{ and } red(f) \ll red(g). \end{aligned}$$

Lemma 1.27

$\gg$ *is a Noetherian partial ordering on* $R[X]$.

Later we will need to divide a polynomial in $K[X]$ by a set of other polynomials

in $K[X]$. Thus, we define division in $K[X]$.

Theorem 1.28 (*Division in $K[X]$, [CLO96, p. 61]*)

Fix a term ordering $>$ on $[X]$ and let $F = (f_1, \ldots, f_s)$ be an ordered s-tuple of polynomials in $K[X]$. Then every $f \in K[X]$ can be written as

$$f = a_1 f_1 + \cdots + a_s f_s + r,$$

*where $a_i, r \in K[X]$, and either $r = 0$ or r is a linear combination, with coefficients in K, of terms which are not divisible by any of $lpp(f_1), \ldots, lpp(f_s)$. r is called a **remainder** of f divided by F.*

2. Algebraic sets and ideals

In this section we define algebraic sets, varieties and polynomial ideals. We will show how they are generated by polynomials and the correlation between them.

The main ideas are from the first chapter of the book *Ideals,varieties and algorithms* [CLO96].

2.1. Definitions

Definition 2.1

Let K be a field and n a positive integer. Then we define the n-dimensional ***affine space*** *over K as the set*

$$K^n = \{(a_1, \ldots, a_n) \,|\, a_1, \ldots, a_n \in K\}.$$

One specific $(a_1, \ldots, a_n) \in K^n$ is called a ***point*** *in the affine space of K^n.*

In other literature K^n is also written as $\mathbb{A}^n(K)$. We call K^1 the affine line and K^2 the affine plane.

Subsets of the affine space, which can be described by polynomials, are called algebraic sets.

Definition 2.2

Let K be a field and $f_1, \ldots, f_s$ polynomials in $K[X]$. Then

$$\mathbf{V}(f_1, \ldots, f_s) = \{(a_1, \ldots, a_n) \in K^n \,|\, f_i(a_1, \ldots, a_n) = 0 \text{ for all } i \in [1 \ldots s]\}$$

is called the ***algebraic set*** *defined by $f_1, \ldots, f_s$.*

EXAMPLE:
Let $K = \mathbf{R}$. The algebraic set defined by $f_1 = x_1^2 + x_2^2 + x_3^2 - 1, f_2 = x_1 - x_2$ is a circle in $\mathbf{R}^3$. See Figure 1.

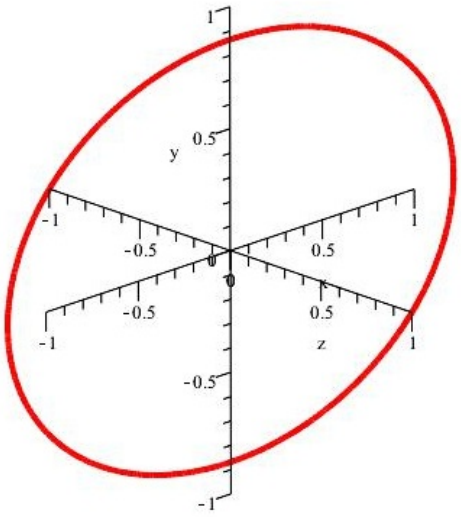


Figure 1: Example

Lemma 2.3

Assume $V, W \subseteq K^n$ are algebraic sets. Then also $V \cup W$ and $V \cap W$ are algebraic sets in K^n.

Proof: Let $V = \mathbf{V}(f_1, \ldots, f_s)$, $W = \mathbf{V}(g_1, \ldots, g_t)$ and $a = (a_1, \ldots, a_n)$ be a point in K^n. We set:

$$V \cap W = \mathbf{V}(f_1, \ldots, f_s, g_1, \ldots, g_t)$$
$$V \cup W = \mathbf{V}(f_i \cdot g_j \mid s \in \{1..s\}, j \in \{1..t\}).$$

If we look at the definition of algebraic sets we see that for the intersection all $f_1, \ldots, f_s$ and $g_1, \ldots, g_t$ have to be 0. And by definition $\mathbf{V}(f_1, \ldots, f_s, g_1, \ldots, g_t)$ is this algebraic set.

For the union either all f_i or all g_i have to be 0 for a point $a \in V \cup W$. So if all $f_i(a) = 0$ then of course $f_i \cdot g_j = 0$ for all g_i, therefore $a \in \mathbf{V}(f_i \cdot g_j)$ which implies $V \subseteq \mathbf{V}(f_i \cdot g_j)$. To show $W \subseteq \mathbf{V}(f_i \cdot g_j)$ we proceed the same way.
On the other hand, to show $\mathbf{V}(f_i \cdot g_j) \subset V \cup W$ we assume $a \notin V$. So if $a \in \mathbf{V}(f_i \cdot g_j)$ then $g_j(a) = 0$ for $j \in [1 \ldots t]$ or in other words $a \in W$. We conclude that $\mathbf{V}(f_i \cdot g_j) \subset V \cup W$. ■

Definition 2.4

*An algebraic set $V \subseteq K^n$ is **reducible** if and only if there are two algebraic sets $V_1, V_2 \subset V$, so that $V = V_1 \cup V_2$. Otherwise V is irreducible. An irreducible algebraic set is called a **variety**.*

A subset $I \subseteq R[X]$ which fulfils the ideal conditions ($0 \in I$, closed under addition of ideal elements and multiplication with ring elements) is called **polynomial ideal**.

Now the question rises, how we can represent or generate a polynomial ideal.

Lemma 2.5

Let $f_1, \ldots, f_s \in R[X]$. Then

$$I = \langle f_1, \ldots, f_n \rangle = \{ \, p_1 f_1 + \cdots + p_s f_s \, | \, p_i \in R[X] \text{ for } i \in [1 \ldots s] \}.$$

is a polynomial ideal.

Proof: We have to show that all 3 conditions of 1.11 are fulfilled.

1. If we set all $p_i = 0$ we get 0.
2. Let $p = p_1 f_1 + \cdots + p_s f_s$ and $q = q_1 f_1 + \cdots + q_s f_s$.
   Then $p + q = (p_1 + q_1) f_1 + \cdots + (p_s + q_s) f_s$ is in I because $p_i + q_i \in R[X]$ for $i = 1, \ldots, s$.
3. Let $f = p_1 f_1 + \cdots + p_s f_s$ and $h \in R[X]$.
   Then $hf = (hp_1) f_1 + \cdots + (hp_s) f_s$ is in I because $hp_i \in R[X]$ for $i = 1, \ldots, s$. ■

Definition 2.6

*We call $I = \langle f_1, \ldots, f_n \rangle$ the **ideal generated by** $\{f_1, \ldots, f_s\}$.*

Definition 2.7

*Let $I = \langle f_1, \ldots, f_n \rangle$. Then we call $\{f_1, \ldots, f_n\}$ a **basis** of the ideal I.*

The next lemma will show us that a non-zero constant generates the whole polynomial ring $K[X]$.

Lemma 2.8

The ideal which is generated by a non-zero constant, is the whole polynomial ring $K[X]$.

Proof: Let c be in I then $c \cdot c^{-1} = 1$ is also in the ideal. Then by 1.11 (3) all $r \in K[X]$ are also in the ideal. Thus $I = K[X]$. ■

2.2. Correlation between ideals and algebraic sets

It also would be nice if we could find a connection between ideals and algebraic sets. In the following we create such a connection.

Proposition 2.9

If $\{f_1, \ldots, f_s\}$ and $\{g_1, \ldots, g_t\}$ are bases of the same ideal in $K[X]$, $\langle f_1, \ldots, f_s\rangle = \langle g_1, \ldots, g_t\rangle$, then $\mathbf{V}(f_1, \ldots, f_s) = \mathbf{V}(g_1, \ldots, g_t)$.

Proof: Assume a point $a = (a_1, \ldots, a_n) \in \mathbf{V}(f_1, \ldots, f_s)$. Then $f_i(a) = 0$ for $i \in [1..s]$. $g_1, \ldots, g_t$ are in the ideal generated by $\langle f_1, \ldots, f_s\rangle$. Therefore we can write each g_j as

$$g_j = \sum_{i=1}^{s} c_i f_i \qquad \text{for } c_i \in K[X].$$

It is easy to see that then $g_j(a) = 0$ for $j \in [1..t]$ and thus $a \in \mathbf{V}(g_1, \ldots, g_t)$.
We proceed the same way to show the other way round. ■

Definition 2.10

Let $V \subseteq K^n$ be a subset of the affine space. We denote

$$\mathbf{I}(V) = \{f \in K[X] \mid f(a_1, \ldots, a_n) = 0 \text{ for all } (a_1, \ldots, a_n) \in V\}$$

*as the **ideal of** V.*

It is easy to show that $\mathbf{I}(V)$ is actually an ideal.

Lemma 2.11

Let $V \subset K^n$ then $\mathbf{I}(V) \subset K[X]$ is an ideal.

Proof: It is easy to see that $\mathbf{I}(V) \subset K[X]$ and $0 \in \mathbf{I}(V)$, because every point vanishes on the zero polynomial. Let $f, g \in K[X]$ be polynomials which vanish on V and an arbitrary $h \in K[X]$. Then for a $(a_1, \ldots, a_n) \in V$:

$$f(a_1, \ldots, a_n) + g(a_1, \ldots, a_n) = 0 + 0 = 0.$$
$$h(a_1, \ldots, a_n) \cdot g(a_1, \ldots, a_n) = h(a_1, \ldots, a_n) \cdot 0 = 0.$$

Thus $\mathbf{I}(V)$ is an ideal ■

For an algebraically closed field K, we can ask the question: What is the ideal of $V = K^n$ or $V = \emptyset$?

For $V = K^n$ we need the polynomials which vanish on the whole algebraic space K^n. By Prop. 1.10 we know that just the zero polynomial satisfies this condition for infinite fields. We know algebraically closed fields are infinite, so $\mathbf{I}(K^n) = \langle 0 \rangle = \{0\}$.

If $V = \emptyset$, we need the polynomials which do not have a zero. Constant polynomials different from 0 fulfil this. But if we have a constant in the generator of an ideal the whole polynomial ring is the ideal. So $\mathbf{I}(\emptyset) = \langle 1 \rangle = K[X]$.

This is a theorem called the Weak Nullstellensatz.

Theorem 2.12

Let K be an algebraic closed field and $I \subseteq K[X]$ be an ideal satisfying $\mathbf{V}(I) = \emptyset$. Then $I = K[X]$

Proof: In the Book [CLO96, P. 168] you find a detailed proof of this theorem. ■

Another question is: Is an ideal I equal to the ideal $\mathbf{I}(\mathbf{V}(I))$? The next lemma gives us the answer.

Lemma 2.13

If $f_1, \ldots, f_s \in K[X]$, then $\langle f_1, \ldots, f_s \rangle \subseteq \mathbf{I}(\mathbf{V}(f_1, \ldots, f_s))$. Note they can be different.

That inequality can occur because there are maybe smaller polynomials than the f_i where the algebraic set vanishes. For instance, this will occur if the f_i's are not square-free. Take the ideal $I = \langle (x^2 + 2y^2 - 2)^2 \rangle$, then $I' = \mathbf{I}(\mathbf{V}(I)) = \langle x^2 + 2y^2 - 2 \rangle$, because of course all points which vanish on I will also vanish on I'. We see that the second ideal contains more polynomials than the first one. I' is called the radical ideal of I. But more on that later.

The following proposition shows some properties of the correspondence of ideals and algebraic sets.

Proposition 2.14

Let V and W be algebraic sets in K^n. Then:

- $V \subset W$ *if and only if* $\mathbf{I}(V) \supset \mathbf{I}(W)$.
- $V = W$ *if and only if* $\mathbf{I}(V) = \mathbf{I}(W)$.

Lemma 2.15

Let I, J be ideals and $\{I_\alpha\}_{\alpha \in A}$ be an arbitrary family of ideals. Then

- $V(\bigcup_{\alpha \in A} I_\alpha) = \bigcap_{\alpha \in A} V(I_\alpha)$.
  In words, the intersection of arbitrary many algebraic sets is an algebraic set.
- $V(I) \cup V(J) = V(I \cap J) = V(\{f \cdot g \mid f \in I \text{ and } g \in I)$.
  In words, every finite union of algebraic sets is an algebraic set

By the last lemma and the facts that $V(0) = K^n$ and $V(1) = \emptyset$ we get a topology on K^n by letting the algebraic sets be the closed sets. The necessary open sets for a topology are the complements of the algebraic sets.

Definition 2.16

*We define a topology on K^n by choosing the closed sets of the topology as the the algebraic sets in K^n. This topology is called the **Zariski topology**.*

A further important question is:

Can we represent every ideal by a finite basis/generating set?

Theorem 2.17

Let X be a finite set of variables. Then every ideal $I \subset K[X]$ has a finite basis. So every ideal can be written as $I = \langle g_1, \ldots, g_t \rangle$ for $g_i \in K[X]$.

Proof: The theorem is a direct consequence of the Hilbert's basis theorem (1.14). This can easily be shown by induction on the number of variables. K is a field and therefore K is finitely generated. Then by the Hilbert's basis theorem $K[x_1]$ is also finitely generated. If we apply the Hilbert's basis theorem again we get that $K[x_1][x_2] = K[x_1, x_2]$ is finitely generated.
So if we apply the Hilbert's basis theorem n times, we see that every ideal in $K[X]$ has a finite basis. ■

Now we know that every polynomial ideal, over a field K in finite variables, has a finite basis. We still have to clarify the relation between I and $\mathbf{I}(\mathbf{V}(I))$. From Lemma 2.13 we know,

$$I \subseteq \mathbf{I}(\mathbf{V}(I)).$$

In some cases the two ideals are equal. But which conditions must an ideal I fulfil to be equal with the ideal of the algebraic set defined by the ideal I?

To answer this question we have to define radical ideals.

Definition 2.18

*Let $k \geq 1$ be a positive integer and $I \in K[X]$ be an ideal. If $f^k \in I$ implies that $f \in I$ then we call I **radical**.*

Let $I \in K[X]$ be an ideal. The ***radical of*** *I, denoted by $\sqrt{I}$, is the set*

$$\{f : f^k \in I \text{ for some integer } k \geq 1\}.$$

So we see that if we decompose polynomials $f \in I$ into m irreducible factors f_i,

$$f = f_1^{e_1} f_2^{e_2} \cdots f_m^{e_m}.$$

Then the radical of I, $\sqrt{I}$, contains the polynomials

$$f' = f_1 f_2 \cdots f_m.$$

For completeness we have to show that $\sqrt{I}$ is an ideal. Moreover $\sqrt{I}$ is radical.

Lemma 2.19

Let $\sqrt{I}$ be as in the definition above. Then $\sqrt{I}$ is an ideal in $K[X]$ containing I and I is radical.

Proof: We can find a proof for this lemma in the book [CLO96, P. 174]. ■

With this theory we can answer the question.

Theorem 2.20 (*Strong Nullstellensatz*)

Let K be an algebraically closed field and $I \subseteq K[X]$ be an ideal. Then

$$\sqrt{I} = \mathbf{I}(\mathbf{V}(I))$$

Proof: It is easy to see that $\sqrt{I} \subseteq \mathbf{I}(\mathbf{V}(I))$. Assume $f \in \sqrt{I}$, then by definition f^k vanishes on $\mathbf{V}(I)$ for some $k \geq 1$. f has the same zeros as f^k therefore it also vanishes on $\mathbf{V}(I)$. Thus $f \in \mathbf{I}(\mathbf{V}(I))$.

Let $f \in \mathbf{I}(\mathbf{V}(I))$. Then by the Hilbert's Nullstellensatz there is a k so that $f^k \in I$. By definition of the radical of I we see $f \in \sqrt{I}$. We can follow $\mathbf{I}(\mathbf{V}(I)) \subseteq \sqrt{I}$. ■

So if I is radical then there is a bijection to an algebraic set.

By this theorem we can show the ideal-algebraic set correspondence.

Theorem 2.21

Let K be an algebraically closed field. Then there is a bijection between algebraic sets and radical ideals. Moreover this bijections is exact $\mathbf{I}$ *and* $\mathbf{V}$*.*

algebraic sets $\longleftrightarrow$ *radical ideals*

$$\sqrt{I} \underset{\mathbf{V}}{\overset{\mathbf{I}}{\rightleftarrows}} V$$

3. Elimination theory

In this section we will consider how to solve systems of polynomial equations with elimination theory. Therefore we study two important concepts, the elimination and the extension, and take a look at the geometric interpretation of elimination and the Closure Theorem. The ideas, which are presented in this chapter, are mainly from [CLO96, Ch. 3].

So the problem we want to solve is to develop an easy algorithm to calculate the algebaric set $\mathbf{V}(I)$ to a corresponding ideal I.

3.1. Elimination step and extension step

When we talk about elimination, we want to reduce the number of variables or rather the number of equations. For linear systems of polynomial equations there is the comparatively simple Gaussian elimination algorithm. For general systems of polynomial equations in $K[X]$ elimination is a bit harder.

When an elimination algorithm is finished and just one equation or one variable is left we can solve it and extend the the solutions to the polynomial equations of more variables. That we call extension.

Thus we have to do two steps to solve polynomial equation systems:

- **Elimination step**: Find simpler equations in fewer variables which have partial solutions of the original system of polynomial equations.
- **Extension step**: Extend this partial solutions to solutions of the original system.

First we have a look at the elimination step. Therefore we need to clarify the term elimination ideal.

Definition 3.1

*Let $I = \langle f_1, \ldots, f_s \rangle \subseteq K[X]$, then the j-th **elimination ideal** I_j is the ideal*

defined by

$$I_j = I \cap K[x_{j+1}, \ldots, x_n]$$

So I_j just consists of the polynomials in I which are independent of the variables $x_1, \ldots, x_j$. Note that the elimination ideal depends on the ordering of the variables.

One way to get the elimination ideals are Gröbner Bases. Thus in the chapter Gröbner Bases we will see a simple way to calculate elimination ideals with Gröbner Bases.

Another approach to elimination theory is the theory of resultants. We will also have a look on them in a later chapter.

If we have a dimensional smallest, non empty elimination ideal, we can calculate the partial solutions, the algebraic set of this ideal.

So the basic idea is to construct the solutions coordinate by coordinate. So if we have the so called **partial solutions** $(a_{j+1}, \ldots, a_n)$ of the j-th elimination ideal I_j we maybe can extend this solution to a greater elimination ideal I_l ($l < j$). The next theorem gives a mathematical description of this extension.

Theorem 3.2 (*The extension theorem*)

Let K be algebraically closed and $I = \langle f_1, \ldots, f_s \rangle \subseteq K[X]$. We can write each $f_i, i \in [1..s]$, in the form

$$f_i = g_i(x_2, \ldots, x_n)x_1^{N_i} - \text{ monomials of degree } < N_i \text{ in } x_1,$$

where $N_i \geq 0$ and $g_i \in K[x_2, \ldots, x_n]$ is non-zero.
Suppose that we have a partial solution of the first elimination ideal I_1 of I, $(a_2, \ldots, a_n) \in \mathbf{V}(I_1)$. If $(a_2, \ldots, a_n) \notin \mathbf{V}(g_1, \ldots, g_s)$, then there exists a $a_1 \in K$ such that $(a_1, a_2, \ldots, a_n) \in \mathbf{V}(I)$.

Proof: A proof for the extension theorem by resultants is given in [CLO96, Ch. 3, §6] ■

Lets have a look at the properties of the extension theorem.

K has to be algebraically closed. If not we use the algebraic closure $\bar{K}$ of K in the theorem because every equation has to be solvable. For instance if we have the equations $f_1 = x^2 - y, f_2 = y + 1 \in \mathbb{R}[x, y]$, the partial solution of I_1 is (-1) in $\mathbb{R}[y]$. If we extend it to I we get $x^2 + 1 = 0$, what is unsolvable in $\mathbb{R}$. But if we use the algebraic closed field $\mathbb{C}$ instead of $\mathbb{R}$ we can extend the partial solution (-1) to the solution $(\pm i, -1)$.

We also assume that the leading coefficients g_i are not zero in the partial solutions. The need of this condition will be clear if we have a look at the ideal $I = \langle xy - 1, xz - 1\rangle \subset \mathbb{C}[x, y, z]$. $I_1 = \langle y - z\rangle$, thus we have the partial solution (a, a). When we extend this partial solution we get $(\frac{1}{a}, a, a)$ except for the partial solution $(0, 0)$ where the leading coefficients y and z of I vanish.

So from the Extension theorem we see that the extension step can only fail when the leading coefficients g_i vanish simultaneously.

Sometimes we can change the variety $\mathbf{V}(g_1, \ldots, g_s)$ by changing to a different basis of the original ideal I. And we should also note, that if we work in the projective space we can extend all partial solutions.

3.2. Geometric interpretation

In this section we will consider the geometric interpretations of the elimination theorem and the extension theorem. We will work over the field $K = \mathbb{C}$ to simplify this problem. In the geometrical way the elimination step is a projection from the n-dimensional space to the $n - 1$-dimensional space.

Definition 3.3

*Let $V \subseteq \mathbb{C}^n$ be an affine variety. The **projection map** π_k is defined by*

$$\begin{array}{rccc} \pi_k : & \mathbb{C}^n & \mapsto & \mathbb{C}^{n-k} \\ & (a_1, \ldots, a_n) & \to & (a_{k+1}, \ldots, a_n) \end{array}$$

If we apply $\pi_k(V)$ we get the projection of the variety V into the affine space $\mathbb{C}^{n-k}$. So $\pi_k(V) \subseteq \mathbb{C}^{n-k}$.

The following lemma links the projection $\pi_j(V)$ and the j-th elimination ideal I_j.

Lemma 3.4

Let I_j be the j-th elimination ideal of $I = \langle f_1, \ldots, f_s \rangle \subseteq \mathbb{C}[X]$. Then in $\mathbb{C}^{n-1}$, we have

$$\pi_j(V) \subseteq \mathbf{V}(I_j).$$

By the lemma we also can write $\pi_j(V)$ in the following way

$$\pi_j(V) = \{(a_{j+1}, \ldots, a_n) \in \mathbf{V}(I_j) : \exists a_1, \ldots, a_j \in \mathbb{C} \text{ so that } (a_1, \ldots, a_j, a_{j+1}, \ldots, a_n) \in V\}$$

In other words π_j is exactly composed of the partial solutions that extend to the complete solutions.

EXAMPLE:
Now we have a look at the example from above $I = \langle xy-1, xz-1 \rangle$. As we already know $I_1 = \langle y - z \rangle$. Thus $\mathbf{V}(I_1)$ is the line $\{(a, a) | \, a \in \mathbb{R}\}$ on the yz-plane though the origin.

The projection of $\{(1/a, a, a) | \, a \in \mathbb{R} \setminus 0\}$ by eliminating the first coordinate x is the line $\{(a, a) | \, a \in \mathbb{R} \setminus 0\}$ on the yz-plane excluding the origin $(0, 0)$.

So we see that $\pi_j(V)$ can be smaller than the algebraic set $\mathbf{V}(I_j)$. The following theorem gives us a concrete picture of the missing parts.

Theorem 3.5

Let $I = \langle f_1, \ldots, f_s \rangle \subseteq \mathbb{C}[X]$, and $V = \mathbf{V}(I) \subseteq \mathbb{C}^n$, with g_i be as in the Extension Theorem 3.2. If I_1 is the first elimination ideal of I, then we have the equality in $\mathbb{C}^{n-1}$:

$$\mathbf{V}(I_1) = \pi_1(V) \cup (\mathbf{V}(g_1, \ldots, g_s) \cap \mathbf{V}(I_1)),$$

where $\pi_1 : \mathbb{C}^n \to \mathbb{C}^{n-1}$ is the projection onto the last $n-1$ components.

As we saw in the extension theorem we have to exclude the solutions of $\langle g_1, \ldots, g_s \rangle$ from $\mathbf{V}(I_1)$ to get the projection of $\mathbf{V}(I)$

We can show some important connections between the projection and the elimination ideals.

Theorem 3.6 (*The closure theorem*)

Let $I = \langle f_1, \ldots, f_s \rangle \subseteq \mathbb{C}[X]$, $V = \mathbf{V}(I) \subseteq \mathbb{C}^n$ and let I_j be the j-th elimination ideal of I. Then:

1. *$\mathbf{V}(I_j)$ is the smallest affine variety containing $\pi_j(V)$.*
2. *When $V \neq 0$, there is an affine variety $W \subset \mathbf{V}(I_j)$ such that $\mathbf{V}(I_j) - W \subseteq \pi_j(V)$.*

Therefore we see that $\pi_j(V)$ maybe omits some points of $\mathbf{V}(I_j)$ which are located in a strictly smaller variety.

We showed the geometric interpretations and the Closure Theorem for the complex numbers $\mathbb{C}$. But we can show that this theorems and what follows from them is true for any algebraically closed field.

4. Gröbner bases

In a previous chapter we describe ideals and algebraic sets and see how they are correlated. In this chapter we have a look at Gröbner basis. Gröbner bases are special bases of ideals $I \subseteq K[X]$, which make it easy to decide if a polynomial in $K[X]$ is in the ideal or not. Gröbner bases were introduced by Bruno Buchberger who named them after his mentor W. Gröbner. [Buc65]

Gröbner bases are also useful for deciding many other problems in an ideal. Amongst other things one is able to calculate the algebraic set to a corresponding ideal in an easy way. In other words one can solve a polynomial equation system with little effort.

Buchberger also developed an algorithm to calculate a Gröbner basis of an ideal, given any finite basis in the polynomial ring $K[X]$. We will have a look at this algorithm later in this chapter.

The ideas, definitions and theorems in this chapter are from Franz Winkler's book "Polynomial Algorithms in Computer Algebra" [Win96, Ch. 8].

4.1. Reduction of polynomials

To calculate a Gröbner basis we have to learn how to "reduce" polynomials. For this purpose we have a look at reduction relations.

Definition 4.1

*Let M be a set. Then we call a binary relation $\longrightarrow$ on M a **reduction relation** on M and write $a \longrightarrow b$ if a can be reduced to b, or rather $(a, b) \in \longrightarrow$.*
Let $\longrightarrow$ and $\longrightarrow'$ be two reduction relations on M, then we can define new reduction relations on $M \times M$ in the following way

- *$\longrightarrow \circ \longrightarrow'$, the composition of two reduction relations. $a \longrightarrow \circ \longrightarrow' b$ if and only if there exists a $c \in M$ such that $a \longrightarrow c \longrightarrow' b$.*

- $\longrightarrow^{-1}$ *(or* $\longleftarrow$*) the inverse relation, is defined as* $a \longleftarrow b$ *if and only if* $b \longrightarrow a$*.*
- $\longrightarrow_{\text{sym}}$ *(or* $\longleftrightarrow$*) the symmetric closure of* $\longrightarrow$*, is defined as* $\longrightarrow \cup \longleftarrow$*,i.e.* $a \longleftrightarrow b$ *if and only if* $a \longrightarrow b$ *or* $a \longleftarrow b$*.*
- $\longrightarrow^i$*, the i-th power of* $\longrightarrow$*, is the reduction relation defined inductively for* $i \in \mathbb{N}_{\not 0}$ *as*

  $\longrightarrow^0 :=$ *id (* $a \longrightarrow^0 b$ *if and only if* $a = b$*).*

  $\longrightarrow^i := \longrightarrow \circ \longrightarrow^{i-1}$ *for* $i > 0$

  So $a \longrightarrow^i b$ *if and only if there exists* $c_1, \ldots, c_{i-1}$ *such that* $a \longrightarrow c_1 \longrightarrow \ldots \longrightarrow c_{i-1} \longrightarrow b$*. We say* a *reduces to* b *in* i *steps.*
- $\longrightarrow^+ := \bigcup_{i=1}^{\infty} \longrightarrow^i$*, the transitive closure of* $\longrightarrow$*.*
- $\longrightarrow^* := \bigcup_{i=0}^{\infty} \longrightarrow^i$*, the reflexive-transitive closure of* $\longrightarrow$*.*
- $\longleftrightarrow^*$*, the reflexive-transitive-symmetric closure of* $\longrightarrow$*.*

Henceforth we assume that $\longrightarrow$ is decidable, in other words one can decide whether $x \longrightarrow y$ or not.

When we take a look at the previous definition we see that $\longleftrightarrow^*$ is an equivalence relation on M. By $M_{/\longleftrightarrow^*}$ we denote the set of equivalence classes modulo $\longleftrightarrow^*$. The basic problem for any equivalence relation is to decide if two objects are equivalent. This problem is called the *equivalence problem* for the reduction relation. In general, the equivalence problem might be hard to solve.

Now we define some properties of reduction relations to describe them.

Definition 4.2

Let $x, y, z \in M$*.*

- $\underline{x}_{\longrightarrow}$ *means* x *is* ***irreducible*** *or* ***in normal form*** *w.r.t.* $\longrightarrow$*, if and only if there is not any* y *so that* $x \longrightarrow y$*.*

If the reduction relation $\longrightarrow$ is clear from the context we just write $\underline{x}$.

- *$x \downarrow y$ represents that x and y have a **common successor**, i.e. $x \longrightarrow z \longleftarrow y$ for some z.*
- *$x \uparrow y$ represents that x and y have a **common predecessor**, i.e. $x \longleftarrow z \longrightarrow y$ for some z.*
- *x is a $\longrightarrow$-**normal form** of y if and only if $y \longrightarrow^* x$ and x is irreducible.*

Note that in general there are more than one normal form for an element in M. So usually the normal form is not unique.

Hereafter we assume that we are able to decide if $x \in M$ is reducible and if so, we are able to find a $y \in M$ so that $x \longrightarrow y$.

For the sake of usability we have to study some nice properties of reduction relation. One of them is the termination property. A reduction relation has the termination property if you have to get a normal form in finitely many steps. The termination property of reduction relations is important if we use them in algorithms. The uniqueness of the normal forms is another desirable property of reduction relation and is also necessary in some applications of reduction relation. The following definition defines the terms of these two properties.

Definition 4.3

a. *$\longrightarrow$ is **Noetherian** or **has the termination property** if and only if every reduction sequence ends, i.e., there is no infinite sequence $x_1, x_2, \ldots$ in M so that $x_1 \longrightarrow x_2 \longrightarrow \ldots$.*

b. *$\longrightarrow$ is **Church-Rosser** if and only if $a \longleftrightarrow^* b \Rightarrow a \downarrow_* b$.*

Theorem 4.4

If $\longrightarrow$ is Noetherian and Church-Rosser, then the equivalence problem for $\longrightarrow$ is decidable.

Proof: Assume $x, y \in M$ and $\tilde{x}, \tilde{y}$ be normal forms relating to $\longrightarrow$. Apparently $x \longleftrightarrow^* y$ if and only if $\tilde{x} \longleftrightarrow^* \tilde{y}$. Because $\longrightarrow$ is Church-Rosser, $\tilde{x} \longleftrightarrow^* \tilde{y}$ if and only if $\tilde{x} \downarrow_* \tilde{y}$. We know that $\tilde{x}$ and $\tilde{y}$ are irreducible, so by $\tilde{x} \downarrow_* \tilde{y}$ we can imply that $\tilde{x} = \tilde{y}$.

So x and y are in the same equivalence class modulo $\longleftrightarrow^*$ if and only if their normals forms are equal. ■

So if the reduction relation is Noetherian and Church-Rosser then the normal forms are unique.

The previous theorem does not work in both directions. It is possible that the equivalence problem for $\longrightarrow$ is decidable though it is not Church-Rosser or Noetherian.

We can show the Church-Rosser property for a reduction relation if we can connect two common predecessors below the parent element.

Definition 4.5

*Assume $>$ a partial ordering on M, $x, y, z \in M$ and $\longrightarrow$ is a reduction relation on the set M. x and y are **connected below** z (w.r.t. $\longrightarrow$ and $>$) if and only if there are $w_1, \ldots, w_n \in M$ such that $x = w_1 \longleftrightarrow \ldots \longleftrightarrow w_n = y$ and $w_i < z$ for $i \in [1..n]$. We denote this by $x \longleftrightarrow^*_{(<z)} y$.*

Theorem 4.6

Let $\longrightarrow$ be a reduction relation on M and $>$ a partial Noetherian ordering on M such that $\longrightarrow$ is a subset of $>$ ($\longrightarrow \subset >$). Then $\longrightarrow$ is Church Rosser if and only if for all $x, y, z \in M$:

$$x \longleftarrow z \longrightarrow y \quad \Rightarrow \quad x \longleftrightarrow^*_{(<z)} y$$

Proof: see [Win96, Thm 8.1.2, 8.1.3] ■

Eventually we define the polynomial reduction relation:

Definition 4.7

*Let $f, g, h \in K[X], F \subseteq K[X]$ and $<$ a term ordering on $[X]$. We say that g **reduces to** h w.r.t. f ($g \longrightarrow_f h$) if and only if there are power products $s, t \in [X]$ such that s has a non-vanishing coefficient c in g, $s = lpp(f) \cdot t$ and*

$$h = g - \frac{c}{lc(f)} \cdot t \cdot f. \tag{1}$$

*We say that g **reduces to** h w.r.t F ($g \longrightarrow_F h$) if and only if there are $f_1, \ldots, f_k \in F$ such that $g \longrightarrow_{f_1} \ldots \longrightarrow_{f_k} h$.*

In general the polynomial reduction do not have the Church-Rosser property. To see the termination property we do an example.

EXAMPLE:

Let

$$F = \{x_1^2 + 4x_2x_3 - x_3, x_2x_3^2 + 2x_2\} \subset \mathbb{R}[x_1, x_2, x_3] \tag{2}$$

$$g = -x_1^2x_2x_3 - 2x_2^2x_3^2 + x_2x_3^2 + 4x_2^2 + x_2 \tag{3}$$

and $<$ be the graduated lexicographic ordering with $x_1 > x_2 > x_3$. Then we can reduce g with the first polynomial in F, ($c = -1, t = x_2x_3$).

$$g \longrightarrow_{f_1} h_1 := 2x_2^2x_3^2 + 4x_2^2 + x_2$$

Afterwards we reduce h_1 by f_2, ($c = -2, t = x_2$).

$$h_1 \longrightarrow_{f_2} h_2 := x_2$$

h_2 is irreducible by F because the leading power products of F are bigger than the leading power product of h_2.

So $g \longrightarrow_F x_2$, where x_2 is a normal form of g w.r.t. $\longrightarrow_F$.

From Definition 4.7 we follow that this reduction relation is almost compatible with the operations in a polynomial ring.

$$g_1 \longrightarrow_F g_2 \quad \Longrightarrow \quad \begin{array}{l} s \cdot g_1 \longrightarrow_F s \cdot g_2 \\ g_1 + h \downarrow_F^* g_2 + h \end{array}, \tag{4}$$

for $s \in K[X] \setminus \{0\}, F \subseteq K[X]$ and $g_1, g_2, h \in K[X]$.

Furthermore we can observe that if we reduce $g \longrightarrow_F h$ then h is smaller than g, w.r.t. $>$. Thus we can follow that the reduction relation $\longrightarrow_F$ is Noetherian.

Additionally we can proof the equality of the reflexive-transitive-symmetric-closure of the reduction relation $\longrightarrow_F$ and the congruence modulo the ideal generated by F.

Theorem 4.8

Let $F \subseteq K[X]$. The ideal congruence modulo $\langle F \rangle$ equals the reflexive-transitive-symmetric closure of $\longrightarrow_F$.

$$x \equiv_{\langle F \rangle} y \quad \Longleftrightarrow \quad x \longleftrightarrow_F^* y$$

for all $x, y \in K[X]$.

Proof: A proof for this theorem you can find in the solution of the exercise 8.2.3. in the book *Polynomial Algorithms in Computer Algebra* [Win96, p. 247]. ■

So by Thm. 4.4 and 4.8 the congruence $\equiv_F$ can be decided if $\longrightarrow_F$ has the Church-Rosser property. A set of polynomials F which fulfil the Church-Rosser property is called Gröbner basis.

4.2. Theory of Gröbner bases

From Hilbert's basis theorem we derive that the multivariate polynomial ring over a field, $K[X]$, is Noetherian or rather every ideal I in $K[X]$ has a finite

basis. Now we want to know if there are also finite nice bases with Church-Rosser property, so called Gröbner bases.

Definition 4.9

Let F be a subset of $K[X]$,$\longrightarrow_F$ a polynomial reduction w.r.t. term ordering $<$ on $[X]$.

F is a Gröbner basis of $\langle F\rangle$ if and only if $\longrightarrow_F$ is Church-Rosser.

Literature sometimes define different but equivalent definition for Gröbner bases . One of them is:

Definition 4.10

For a fixed monomial ordering $<$, a finite subset $G = \{g_1, \ldots, g_s\}$ of an ideal I is a Gröbner basis if and only if

$$\langle lm(g_1), \ldots, lm(g_s)\rangle = \langle lm(I)\rangle.$$

Such a Gröbner basis is not unique. If we have an ideal $I \in K[X]$, (K infinite) then actually there are infinitely many Gröbner bases. Because if there is a Gröbner basis, one can add any $f \in I$ and get another Gröbner basis.

To check if a basis F is a Gröbner basis we first have to define the notion of subtraction polynomials, also called S-polynomials.

Definition 4.11

Let $f, g \in K[X]^*, t = lcm(lpp(f), lpp(g))$. Then

$$cp(f,g) = (t - \frac{t}{lm(f)} \cdot f, t - \frac{t}{lm(g)} \cdot g)$$

*is called the **critical pair** of f and g.*

And the difference of the two elements of a critical pair,

$$Spol(f,g) = \frac{t}{lm(f)} \cdot f - \frac{t}{lm(g)} \cdot g$$

*is denoted as the **S-polynomial** of f and g.*

With the definition of S-polynomials we can implement Buchberger's theorem.

Theorem 4.12 (*Buchberger's theorem*)

Let F be a subset of $K[X]$ and $\longrightarrow_F$ the polynomial reduction w.r.t. to a term ordering $<$.

1. *F is a Gröbner basis (of $\langle F\rangle$) if and only if $g_1 \downarrow_F^* g_2$ for all critical pairs (g_1, g_2) of elements in F.*
2. *F is a Gröbner basis if and only if $spol(f, g) \longrightarrow_F^* 0$ for all $f, g \in F$.*

Proof: A proof for Buchberger's theorem you can find in Winkler's book "Polynomial Algorithms in Computer Algebra" [Win96, Thm 8.3.1]. ■

As I mentioned at the beginning there are algorithms to find a Gröbner bases G for an ideal I. With the previous theorem we have enough theory to introduce such an algorithm. Buchberger implemented this algorithm by using the S-polynomials. A modern version of Buchbergers algorithm is given by the algorithm GRÖBNERBASIS. As input for the algorithm we need an arbitrary basis F for the ideal I and an ordering on the polynomials.

The polynomials, which are constructed in this algorithm, are elements of the ideal $\langle F\rangle$. So G is a basis of $\langle F\rangle$ at any step of the algorithm. We also see that this algorithm has to terminate, by applying Hilbert's basis theorem to the leading term ideal.

The Gröbner basis algorithm is a constructive proof of the following theorem:

Theorem 4.13

Every ideal I in $K[X]$ has a finite Gröbner basis.

EXAMPLE: Let $F = \{x_1^2x_3 + 4x_2x_3 - x_3, x_1x_3^2 + 2x_2\} \subset \mathbb{R}[x_1, x_2, x_3]$, and $<$ be the graduated lexicographic ordering with $x_3 < x_2 < x_1$.

Algorithm 1 GRÖBNERBASIS $(Input : F, <\ Output : G)$

▷ where F is a finite subset of $K[X]^*$, $<$ is a term ordering on $[X]$ and G is a Gröbner basis of $\langle F \rangle$ w.r.t. $<$.

$G := F$;

$C := \{\{g_1, g_2\} \mid g_1, g_2 \in G, g_1 \neq g_2\}$;

while C is not empty **do**

choose an element $\{g_1, g_2\}$ from C;

$C := C \setminus \{g_1, g_2\}$

$h := Spol(g_1, g_2)$

$\underline{h} :=$ normal form of h w.r.t. $\longrightarrow_G$;

if $\underline{h} \neq 0$ **then**

$C := C \cup \{\{g, \underline{h}\} \mid g \in G\}$;

$G := G \cup \{\underline{h}\}$;

end if

end while

Now we use the Gröbner basis algorithm to F:

1. $f_3 := spol(f_1, f_2) = zf_1 - xf_2 = 4x_2x_3^2 - 2x_1x_2 - x_3^2$
   f_3 is irreducible so $G = \{f_1, f_2, f_3\}$
2. $f_4 := spol(f_2, f_3) = 4yf_2 - xf_3 = 2x_1^2x_2 + x_1x_3^2 + 8x_2^2$
   we reduce f_4 by f_2 and divide by 2 to $f_4 := x_1^2x_2 + 4x_2^2 - x_2$, so $G = \{f_1, f_2, f_3, f_4\}$
3. $f_5 := spol(f_1, f_3) = 4x_2x_3f_1 - x_2f_3 = 2x_1^3x_2 + x_1^2x_3^2 + 16x_2^2x_3^2 - 4x_2x_3^2$
   we can reduce f_5 to 0 by $0 = ((((f_5 - 2x_1f_4) - x_3f_1) - 4x_2f_3) + f_3)$. So $G = G$.
4. ... the remaining S-polynomials also reduce to 0. This calculation are left for the reader.

So we get the Gröbner basis

$$G = \{x_1^2x_3 + 4x_2x_3 - x_3, x_1x_3^2 + 2x_2, 4x_2x_3^2 - 2x_1x_2 - x_3^2, x_1^2x_2 + 4x_2^2 - x_2\}.$$

Lets have a look at some additional characterizations of Gröbner bases.

Theorem 4.14

Let I be an ideal generated by $F \subseteq K[X]$. Then the following are equivalent.

1. *F is a Gröbner basis for I.*
2. *$f \longrightarrow_F^* 0$ for every $f \in I$.*
3. *$f \longrightarrow_F$ for every $f \in I \setminus \{0\}$.*
4. *For all $g \in I, \underline{h} \in K[X]$: if $g \longrightarrow_F^* \underline{h}$ then $\underline{h} = 0$.*
5. *For all $g, \underline{h_1}, \underline{h_2} \in K[X]$: if $g \longrightarrow_F^* \underline{h_1}$ and $g \longrightarrow_F^* \underline{h_2}$ then $\underline{h_1} = \underline{h_2}$.*
6. *$\langle lm(F)\rangle = \langle lm(I)\rangle$.*

Proof: For a proof of this theorem see [Win96, Thm. 8.3.4.]. ■

Sometimes Gröbner bases are contain more polynomials than needed. The next theorem tells us which polynomials can be removed from the Gröbner basis.

Theorem 4.15

Let F be a Gröbner basis for an ideal I in $K[X]$. Let $f, g \in F$, $f \neq g$.

1. *If $lpp(f) | lpp(g)$ then $F' = F \setminus \{g\}$ is also a Gröbner basis of I.*
2. *If $g \longrightarrow_f g'$ then $F' = (F \setminus \{g\}) \cup \{g'\}$ is also a Gröbner basis of I.*

The reductions of this theorem just can be done if F is a Gröbner basis. Thus we can not do this reduction already in the Buchberger algorithm.

The previous theorem leads to some definitions of basic properties of Gröbner basis.

Definition 4.16

Let F be a Gröbner basis in $K[X]$.

- *F is called **minimal** if and only if $lpp(f)$ is not a factor of $lpp(g)$ for all $f, g \in F$ so that $f \neq g$.*
- *F is called **reduced** if and only if it is not possible to reduce f by g for all $f, g \in F$ and $f \neq g$.*
- *F is called **normed** if and only if all leading coefficiets are 1, $lc(f) = 1$ for all $f \in F$.*

Although there are infinitely many Gröbner bases for an ideal I there is an unique normed and reduced Gröbner basis for the ideal I w.r.t.given a term ordering. So if we compute a normed and reduced Gröbner basis of an ideal we always get the same unique Gröbner basis.

Theorem 4.17

Every ideal in $K[X]$ has an unique, finite, normed and reduced Gröbner basis w.r.t. a term ordering $<$.

Proof: You find a proof for the theorem in [Win96, Thm. 8.3.6.]. ■

Common computer algebra systems, like maple, automatically calculate a reduced Gröbner basis for an ideal w.r.t. an ordering.

4.3. Solving systems of algebraic equations by Gröbner bases

The essential application of Gröbner bases for this thesis is to solve algebraic equation systems.

Assume $p_1, \ldots, p_m \in K[X]$. Then we call the system

$$P(X) = 0 \quad \Leftrightarrow \quad \begin{array}{c} p_1(x_1, \ldots, x_n) = 0 \\ \vdots \\ p_m(x_1, \ldots, x_n) = 0 \end{array}, \tag{5}$$

a **system of polynomial** (or **algebraic**) **equations** in $K[X]$.

We get a polynomial ideal of the polynomial system by $I = \langle p_1, \ldots, p_m \rangle$. The corresponding algebraic set are the solutions of the polynomial equation system. In other words $\mathbf{V}(p_1, \ldots, p_m) = \mathbf{V}(\langle p_1, \ldots, p_m \rangle)$.

A polynomial equation system may do not have common solutions. If one have a Gröbner bases of ideal of the polynomial system over an algebraically closed field, then there is an easy way to check if it has solutions or not.

Theorem 4.18

Let K be algebraically closed and G be a normed, reduced Gröbner basis of $\langle p_1, \ldots, p_m \rangle$. $P(X) = 0$ is unsolvable in K^n if and only if $G = \{1\}$.

Proof: Suppose $1 \in G$, so $1 \in \langle G \rangle = I$. Thus every solution of $P(X) = 0$ is a solution of $1 = 0$. So this system is unsolvable.

On the other hand, assume $P(X) = 0$ is unsolvable, or rather there are no common roots.

So by Hilbert's Nullstellensatz all polynomials who do not have a root must be in $\sqrt{I}$. Thus 1 has to be in $\sqrt{I}$, what implies $1 \in I$. Because 1 must be reducible by G, $1 \in G$ and because we can reduce every other polynomial by 1, $G = \{1\}$. ■

Therefore we can decide whether $P(X) = 0$ has solutions over an algebraically closed field. It would be also interesting to know how many solutions are there. Is there a finite or infinite number of solutions? If finitely many, how many solutions are there? The next theorem gives an answer to that questions.

Theorem 4.19

Let G be a Gröbner basis of I. Then $P(X) = 0$ has finitely many solutions if and only if for every $i \in \{1; \ldots, n\}$ there is a polynomial $g_i \in G$ such that $lpp(g_i)$ is a pure power of x_i. Moreover, if I is 0-dimensional then the number of zeros of I is equal to the vector space dimension of $K[X]_{/I}$ over K.

Proof: For a proof of the theorem see [Win96, Thm. 8.4.4.]. ■

In the second part of this thesis we are interested in the case of infinitely many solutions. We will call them flexible solutions. By the previous theorem it is only possible to have flexible solutions if there is a variable x_i so that there is not any polynomial $g \in G$ with $lpp(g)$ a pure term of x_i ($lpp(g) = x_i^k$).

If we use a lexicographical ordering, Gröbner bases are similar to Gaussian elimination of linear polynomial systems. Both of them do a triangulation of the system or in other words, they do an elimination process. Therefore we can connect Gröbner bases to Elimination Theory, more precisely Gröbner bases are an easy way to do the elimination step. In the following theorem the

elimination property of Gröbner bases is described.

Theorem 4.20 (*Elimination property of Gröbner bases*)

Let G be a Gröbner basis of I w.r.t. the lexicographic ordering $x_1 > \cdots > x_n$. Then

$$I_j = I \cap K[x_{j+1}, \ldots, x_n] = \langle G \cap K[x_{j+1}, \ldots, x_n] \rangle$$

where the ideal on the right-hand side is generated over the ring $K[x_{j+1}, \ldots, x_n]$.

Proof: That the right side is included in the left one is trivial.

For the opposite direction, let f be a polynomial of the elimination ideal I_j. f reduces to 0 w.r.t. to G, $f \longrightarrow_G^* 0$. Because of the lexicographic ordering the polynomials of G which are used in this reduction depend only on the variables $x_{j+1}, \ldots, x_n$. Thus f can be represented by a linear combination $f = \sum h_i g_i$, where $g_i \in G \cap K[x_{j+1}, \ldots, x_n]$ and $h_i \in K[x_{j+1}, \ldots, x_n]$. ■

So the extension step will be done by solving the smallest polynomial in the Gröbner basis and substituting the solutions in the remaining polynomials. Then do the same step to the smallest element of the remaining polynomials and so on.

If all polynomials can be solved, you get the solutions of the polynomial system.

EXAMPLE:

To understand the applicability of the theory in this thesis we view a simple example in robotics. Lets have a look at the following robot hand which moves in a plane.

The robot hand is from the book "Ideals, varieties and algorithms" [CLO96, Page 169]. The modelling of the robot hand, which uses the unit circle approach, can be read in the mentioned book. Of course we study the modelling of robot arms in the second part of the thesis.

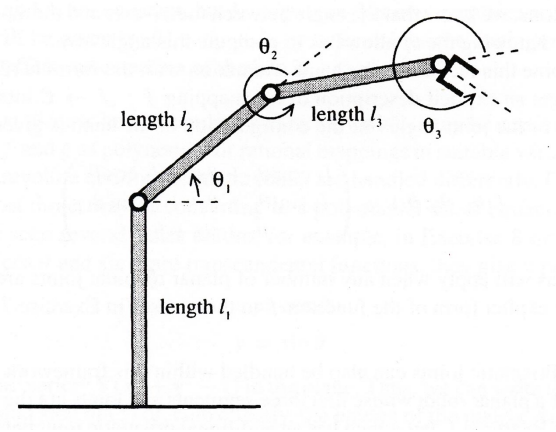


Figure 2: Example robot

As we can read in [CLO96, Page 175 (1)], we get the four polynomial equations:

$$\begin{aligned} x &= l_3(c_1c_2 - s_1s_2) + l_2c_1, \\ y &= l_3(c_1s_2 + c_2s_1) + l_2s_1, \\ 0 &= c_1^2 + s_1^2 - 1, \\ 0 &= c_2^2 + s_2^2 - 1 \end{aligned}$$

We fix the lengths of the segments to $l_1 = 0, l_2 = l_3 = 1$. So we get polynomials

$$\begin{aligned} x &= c_1c_2 - s_1s_2 + c_1, \\ y &= c_1s_2 + c_2s_1 + s_1, \\ 0 &= c_1^2 + s_1^2 - 1, \\ 0 &= c_2^2 + s_2^2 - 1 \end{aligned}$$

in the variables s_1, c_1, s_2, c_2.

We choose the graded lexicographic ordering with $c_1 > s_1 > c_2 > s_2$.

For instance if we want to reach the point $(1, 1)$ we substitute $x = 1, y = 1$ and

therefore use the basis:

$$F = [c_1c_2 - s_1s_2 + c_1 - 1, c_1s_2 + c_2s_1 + s_1 - 1, c_1^2 + s_1^2 - 1, c_2^2 + s_2^2 - 1].$$

To solve this polynomial system in an easy way we develop a Gröbner basis of F. Therefore we apply the Buchberger Algorithm to F.

So we need to calculate the reduced S-polynomials. For example, we calculate the S-Polynomial of the first critical pair or the first two polynomials in F.

$$Spol(c_1c_2 - s_1s_2 + c_1 - 1, c_1s_2 + c_2s_1 + s_1 - 1) = -c_2^2s_1 - s_1s_2^2 + c_1s_2 - c_2s_1 + c_2 - s_2$$

We can reduce this S-Polynomial with the second and fourth polynomial of F to

$$-2c_2s_1 + c_2 - 2s_1 - s_2 + 1.$$

This is not 0 therefore we have to add this polynomial to the basis and add the additional critical pairs.

If we apply the algorithm GRÖBNERBASIS to the rest of the critical pairs, we get the Gröbner basis:

$$\{c_1c_2 - s_1s_2 + c_1 - 1, c_1s_2 + c_2s_1 + s_1 - 1,$$
$$c_1^2 + s_1^2 - 1, c_2^2 + s_2^2 - 1, -2c_2s_1 + c_2 - 2s_1 - s_2 + 1,$$
$$-c_1 + c_2 - s_1 + 1, 2s_1s_2 - c_2 - s_2 + 1,$$
$$-c_2 + 2s_1 + s_2 - 1, s_2^2 + c_2 - 1, -c_2\}$$

If we reduce this basis to a reduced Gröbner basis we get:

$$\{2c_1 - s_2 - 1,\ 2s_1 + s_2 - 1,\ s_2^2 - 1,\ c_2\}. \tag{6}$$

Now it is easily to see that we have to two solutions for $\langle F \rangle$.

$$\{c_1 = 1, s_1 = 0,\ c_2 = 0, s_2 = 1\};$$
$$\{c_1 = 0, s_1 = 1, c_2 = 0, s_2 = -1\}.$$

So for the first solution $\theta_1 = 0, \theta_2 = \pi/2$ and for the second solution $\theta_1 = \pi/2, \theta_2 = -\pi/2$.

With a computer algebra system like Maple it is easy to compute a more general Gröbner basis with unknown x, y, l_2, l_3.

$$\{ (4l_2^2 l_3^2)s_2^2 + (l_2^4 - 2l_2^2 l_3^2 - 2l_2^2 x^2 - 2l_2^2 y^2 + l_3^4 - 2l_3^2 x^2 - 2l_3^2 y^2 + x^4 + 2x^2 y^2 + y^4),$$
$$(2l_2 l_3)c_2 + (l_2^2 + l_3^2 - x^2 - y^2),$$
$$(2l_2 x^2 + 2l_2 y^2)s_1 + (2l_3 l_2 x)s_2 - (l_2^2 y + l_3^2 y - x^2 y - y^3),$$
$$(2l_2 x^2 + 2l_2 y^2)c_1 - (2l_2 l_3 y)s_2 - (l_2^2 x + l_3^2 x - x^3 - y^2 x)\}.$$

If we substitute the values $x = 1, y = 1, l_2 = 1, l_3 = 1$ in this more general Gröbner basis and cancel the coefficients we get the Gröbner basis (6).

5. Resultants

In this chapter we have a look at the second main method to solve polynomial equation systems by elimination theory. Resultants are older than the method of Gröbner bases. The advantage of resultants is that they are faster than Gröbner bases under some conditions. The disadvantage is that resultants may just deliver candidates of solutions. So you have to check if the result is truly a solution of the system.

The main ideas of this chapter are from the books [CLO04, Ch. 3] and [CLO96, Ch. 3 (§5,6)]

Later in this chapter we will also see that resultants are so called "projection operators".

So the algebraic operation of taking resultants of the basis of an ideal is closely related to a projection map on the corresponding variety.

5.1. Resultants of two polynomials

How to construct a resultant of two polynomials is already well known and also implemented in all common computer algebra systems.

Definition 5.1

Assume you have two univariate polynomials $f, g \in R[x]$, R an integral domain,

$$f = a_0 x^s + \ldots a_s, \quad a_0 \neq 0,\ s > 0$$
$$g = b_0 x^t + \ldots b_t, \quad b_0 \neq 0,\ t > 0$$

*Then the **Silvester resultant** of f and g, called $Res(f, g)$, is the determinant*

$$Res(f,g) = det \begin{pmatrix} a_0 & & & & b_0 & & & \\ a_1 & a_0 & & & b_1 & b_0 & & \\ a2 & a_1 & \ddots & & b_2 & b_1 & \ddots & \\ \vdots & a_2 & \ddots & a_0 & \vdots & b_2 & \ddots & b_0 \\ a_s & \vdots & \ddots & a_1 & b_t & \vdots & \ddots & b_1 \\ & a_s & & a_2 & & b_t & & b_1 \\ & & \ddots & \vdots & & & \ddots & \vdots \\ & & & a_s & & & & b_t \end{pmatrix}$$

where we have t columns with the a_i and s columns with the b_i. (The blank spaces are filled with zeros.)

EXAMPLE:

$$Res(2x^2 - x + 1, 3x^3 + 5) = det \begin{pmatrix} 2 & 0 & 0 & 3 & 0 \\ -1 & 2 & 0 & 0 & 3 \\ 1 & -1 & 2 & 0 & 0 \\ 0 & 1 & -1 & 5 & 0 \\ 0 & 0 & 1 & 0 & 5 \end{pmatrix} = 134$$

This definition of the silvester resultant has its seeds in the lemma:

Lemma 5.2

Let $f, g \in K[x]$ be polynomials as above. Then f and g have a non-constant common factor if and only if there are non-zero polynomials $A, B \in K[x]$ with $deg(A) < deg(g), deg(B) < deg(f)$ so that $Af - Bg = 0$.

Therefore we get some properties of the resultants

- $Res(f, g) = (-1)^{st} Res(g, f)$
- $Res(f, g) = Af - Bg$ for some A, B in $R[x]$.
- $Res(f, g) = 0$ if and only if f and g have a non-trivial common factor in $K[x]$
- $Res(f, g), A, B$ are integer polynomials in the coefficients of f and g.
- $Res(f, g) \in \langle f, g \rangle \cap R$.

The proofs of these properties for polynomials over a field, you can find in [CLO96, Ch. 3]. The results of these proofs can be extended to polynomials over an integral domain.

Now we set up a connection between resultants and elimination theory. Therefore we assume that we have two polynomials f, g in $K[x, y]$ with positive degree in x. We can calculate the the resultant according to x by $Res(f, g)_x$. Because of the properties of resultants, the result will be a polynomial in $K[y]$.

EXAMPLE

Let $f = x^2y + xy - 2y, g = 2xy + 2$ be polynomials in $K[x, y]$. We can rewrite them as Polynomials in $K[y][x]$, in other words we have univariate polynomials in x with coefficients in $K[y]$.

$$f = (y)x^2 + (y)x + (-2y)$$
$$g = (2y)x + (2)$$

So we get the resultant,

$$Res(f,g)_x = det \begin{pmatrix} y & 2y & 0 \\ y & 2 & 2y \\ -2y & 0 & 2 \end{pmatrix} = 4y(-2y^2 - y + 1)$$

As you see we eliminated x out of f, g.

Because of the fact that $Af + Bg = Res(f,g)$, $(A, B \in K[y][x])$, $Res(f,g)_x = 4y(-2y^2 - y + 1)$ lies in the elimination ideal $\langle f,g\rangle \cap K[y]$. So $Res(f,g)_x$ vanishes at any common solution of $\langle f,g\rangle$. In other words you get the y-coordinates of the common solutions of $\langle f,g\rangle$ by solving $Res(f,g)_x = 4y(-2y^2 - y + 1) = 0$.

For $4y(-2y^2 - y + 1) = 0$ we get the solutions $y = 0, \frac{1}{2}, -1$.

If we substitute $y = 0$ in $f = g = 0$ we get a contradiction in g with $0 = 2$. In this case the leading coefficients of f and g become 0 and therefore the elimination step is not permitted by the Extension Theorem.

For the case $y = \frac{1}{2}$ we get $x = -2$ and for $y = -1$ we get $x = 1$.

So the common solutions are $(\frac{1}{2}, -2)$ and $(-1, 1)$ for the ideal $\langle f,g\rangle$.

5.2. Multi-polynomial resultants

In this section we study the resultant of arbitrary many polynomials $f_0, \ldots, f_n$ in $K[x_1, \ldots, x_n]$. For this purpose we homogenize the polynomials.

Let $d_i = deg(f_i)$ be the total degree of f_i

$$f_i(x_1, \ldots, x_n) = \sum_{|\alpha| \leq d_i} a_\alpha x^\alpha.$$

with $\alpha \in \mathbb{Z}^n$ are multivariate exponents.
Then the homogenization of f_i is

$$F_i(x_0, x_1, \ldots, x_n) = \sum_{j=0}^{d_i} \sum_{|\alpha|=j} a_\alpha x^\alpha x_0^{d_i - j}.$$

So we get the homogeneous polynomial equation system

$$\begin{aligned} F_0(x_0, \ldots, x_n) &= 0 \\ &\vdots \\ F_n(x_0, \ldots, x_n) &= 0 \end{aligned} \tag{7}$$

with the total degrees $d_0, \ldots, d_n$.

This equation system has always the trivial solution $(0, \ldots, 0)$. So just the non-trivial solutions are interesting for us.

Generally, the coefficients of the homogeneous polynomials F_i give us information about the existence of such non-trivial solutions. Mostly there are no non-trivial solutions but for some special values, there exist non-trivial solutions.

For instance in the case of a linear equation system we know that there is a non-trivial solution if and only if the determinant of the coefficient matrix is 0. In general this determinant is a polynomial in the coefficients.

We will consider the case where we have $n+1$ homogeneous polynomials $F_i \in \mathbb{C}[x_0, \ldots, x_n]$ with the Basic Question: *What conditions must the coefficients of $F_0, \ldots, F_n$ fulfil so that $F_0 = \cdots = F_n = 0$ have non-trivial solutions?*

The homogenous polynomials $F_0, \ldots, F_n$ can also be written like this:

$$F_i = \sum_{|\alpha| = d_i} a_{i,\alpha} x^\alpha, \qquad \alpha \in \mathbb{Z}^{n+1}.$$

If we replace each coefficient $a_{i,\alpha}$ by a variable $u_{i,\alpha}$ we get the **universal polynomials** on $\mathbb{C}^M \times \mathbb{C}^n$ where M is the number of coefficients.

We can answer our basic question with the following theorem.

Theorem 5.3

Let $d_0, \ldots, d_n$ be fixed positive degrees, then there is a unique polynomial $Res \in \mathbb{Z}[u_{i,\alpha}]$ which has the following properties:

1. *If $F_0, \ldots, F_n \in \mathbb{C}[x_0, \ldots, x_n]$ are homogeneous of degrees $d_0, \ldots, d_n$, then the equations (7) have a non-trivial solution over $\mathbb{C}$ if and only if $Res(F_0, \ldots, F_n) = 0$.*
2. *$Res(x_0^{d_0}, \ldots, x_n^{d_n}) = 1$*
3. *Res is irreducible, even as a polynomial in $\mathbb{C}[u_{i,\alpha}]$.*

Proof: A proof of this theorem can be found in Van der Warden's book [vdW67, §78]. ■

Definition 5.4

*The unique polynomial $Res(F_0, \ldots, F_n)$, from Theorem 5.3, is called the **(multivariate) resultant** of $F_0, \ldots, F_n$.*

To outline the degrees of the F_i's we can write $Res_{d_0,\ldots,d_n}(F_0, \ldots, F_n)$.

So we know there exists a resultant which gives us information about the solution of the system (7). There are many different ways to calculate such a resultant. One of the most general ways, the Macaulay resultant, we will study in Section 5.3.

One interesting application of resultants is to find a solution for the *implicitization problem*. Suppose, we are given a surface by parametric polynomial equations

$$\begin{aligned} x &= f(s,t) \\ y &= g(s,t) \\ z &= h(s,t) \end{aligned} \tag{8}$$

and want to find the equation $p(x, y, z) = 0$ which describes this surface. We denote the total degree of $f(s,t), g(s,t), h(s,t)$ as d_f, d_g, d_h.

To solve this problem with resultants we will homogenize the three polynomials by adding the variable u. For instance, we homogenize $f(s,t)$ by

$$F(s,t,u) = f_{d_f}(s,t) + f_{d_f-1}(s,t)u + \dots f_0(s,t)u^{d_f},$$

where f_d is the sum of the terms of f with degree j in s,t.

Equivalently we homogenize g,h, to get the homogeneous system

$$\begin{aligned} F(s,t,u) - xu^{d_f} &= 0 \\ G(s,t,u) - yu^{d_g} &= 0 \\ H(s,t,u) - zu^{d_h} &= 0 \end{aligned} \tag{9}$$

Through the following proposition we get an idea how to solve the implicitization problem:

Proposition 5.5 (*[CLO04, §2 Prop 2.6]*)

With the above notation, assume that the system of homogeneous equations

$$f_{d_f}(s,t) = g_{d_g}(s,t) = h_{d_h}(s,t) = 0$$

has only the trivial solution. Then, for a given triple $(x,y,z) \in \mathbb{C}^3$*, the equations (8) have a solution* $(s,t) \in \mathbb{C}^2$ *if and only if*

$$Res_{d_f,d_g,d_h}(F - xu^{d_f}, G - yu^{d_g}, H - zu^{d_h}) = 0.$$

So we get that,

$$p(x,y,z) = Res_{d_f,d_g,d_h}(F - xu^{d_f}, G - yu^{d_g}, H - zu^{d_h})$$

is the desired polynomial in x,y,z, because the resultant is a polynomial in the coefficients of (9) and as we can see from Proposition 5.5, this polynomial vanishes exactly on the points of the parametrization (8), if $f_{d_f}(s,t) = g_{d_g}(s,t) = h_{d_h}(s,t) = 0$ has only the trivial solution.

Degree of resultants

The main problem with multi-polynomial resultants is that they get very huge. For instance, if you take 3 quadratic homogeneous polynomials in 3 variables you have 18 coefficients and therefore 18 variables $u_{i,\alpha}$ in in the resultant. The total degree will be 12. So by calculations of Bernd Sturmfels, $Res_{2,2,2}$ has already 21894 terms.

Fortunately there are some more compact ways to compute resultants. One of these methods is the Macaulay resultant, which we will consider later.

The next theorem will give us information about the degree of the resultants.

Theorem 5.6

Let $F_0, \ldots, F_n$ be homogeneous polynomials with total degrees $d_0 \ldots d_n$. Then for $j \in [0, .., n]$, Res is homogeneous in the variables $u_{j,\alpha}, |\alpha| = d_j$, of degree $d_0 \cdots d_{j-1} d_{j+1} \cdots d_n$. Hence we conclude,

$$Res(F_0, \ldots, \lambda F_j, \ldots, F_n) = \lambda^{d_0 \cdots d_{j-1} d_{j+1} \cdots d_n} Res(F_0, \ldots, F_n).$$

Moreover, the total degree of Res is

$$deg(Res(F_0, \ldots, F_n)) = \sum_{j=0}^{n} d_0 \cdots d_{j-1} d_{j+1} \cdots d_n.$$

Proof: One can find a proof of this theorem in [GKZ94, Ch. 13]. ■

So we see that the degree of the resultant is generally very high. With every additional polynomial the degree of the resultant multiplies by the degree of the additional polynomial and adds the product of the degrees of the other polynomials. Therefore in general resultants are not very feasible for a larger number of multivariate polynomials.

5.3. Macaulay resultant

In this part we study a way to compute resultants, more precisely we determine how we calculate the Macaulay resultant. Therefore we consider some definitions of [CLO04, Ch.3 §4].

First we suppose that we have homogeneous polynomials $F_0, \ldots, F_n \in \mathbb{C}[x_0, \ldots, x_n]$ of total degrees $d_0, \ldots, d_n$. Then we define

$$d = 1 + \sum_{i=0}^{n}(d_i - 1) = \sum_{i=0}^{n} d_i \ - n.$$

For example if $(d_0, d_1, d_2) = (2, 1, 1)$ the we get $d = 2$.

If we have monomials $x^\alpha = x_0^{\alpha_0} \cdots x_n^{\alpha_n}$ of degree d, we can divide them at least by one $x_i^{d_i}$ with $i \in [0..n]$.

Hence we can partition the set of monomials with degree d into the following disjoint sets

$$\begin{aligned}
S_0 &= \{x^\alpha : \ |\alpha| = d \text{ and } x_0^{d_0} \text{ divides } x^\alpha\} \\
S_1 &= \{x^\alpha : \ |\alpha| = d \text{ and } x_1^{d_1} \text{ divides } x^\alpha\} \setminus S_0 \\
&\vdots \\
S_n &= \{x^\alpha : \ |\alpha| = d \text{ and } x_n^{d_n} \text{ divides } x^\alpha\} \setminus \bigcup_{i \in [0..n-1]} S_i
\end{aligned}$$

Remark: if $x^\alpha \in S_i$, we can write $x^\alpha = x_i^{d_i} \cdot \frac{x^\alpha}{x_i^{d_i}}$, where $\frac{x^\alpha}{x_i^{d_i}}$ is a monomial of degree $d - d_i$.

For our example $(d_0, d_1, d_2) = (2, 1, 1)$ we have

$$\begin{aligned}
S_0 &= \{x_0^2\} \\
S_1 &= \{x_0x_1, x_1^2, x_1x_2\} \\
S_2 &= \{x_0x_2, x_2^2\}
\end{aligned}$$

Lets have a look at the equation system:

$$\begin{aligned} \frac{x^\alpha}{x_0^{d_0}} \cdot F_0 = 0 \quad & \text{for all } x^\alpha \in S_0 \\ \vdots & \\ \frac{x^\alpha}{x_n^{d_n}} \cdot F_n = 0 \quad & \text{for all } x^\alpha \in S_n \end{aligned} \tag{10}$$

All equations have are total degree d, because F_i has the total degree d_i and $\frac{x^\alpha}{x_i^{d_i}}$ has the total degree $d - d_i$. So the polynomials on the left-hand side are homogeneous polynomials of degree d. It is easy to see that there are

$$N = \binom{d+n}{n}$$

such different monomials in $\bigcup_{i\in[0..n]} S_i$ and therefore there are N polynomials in (10). So if we see in each monomial of (10) one "unknown variable" we have a linear system of N linear equations in N unknowns.

Definition 5.7

The determinant of the coefficient matrix of the $N \times N$ system of equations given by (10) is denoted by D_n.

So in our example we have the polynomials

$$\begin{aligned} F_0 &= a_1x_0^2 + a_2x_1^2 + a_3x_2^2 + a_4x_0x_1 + a_5x_0x_2 + a_6x_1x_2 \\ F_1 &= b_1x_0 + b_2x_1 + b_3x_2 \\ F_2 &= c_1x_0 + c_2x_1 + c_3x_2 \end{aligned}$$

which will become

$$\begin{array}{lllllll} 1 \cdot F_0 & = a_1x_0^2 & +a_2x_1^2 & +a_3x_2^2 & +a_4x_0x_1 & +a_5x_0x_2 & +a_6x_1x_2 & = 0 \\ x_0 \cdot F_1 & = b_1x_0^2 & +0 & +0 & +b_2x_0x_1 & +b_3x_0x_2 & +0 & = 0 \\ x_1 \cdot F_1 & = 0 & +b_2x_1^2 & +0 & +b_1x_0x_1 & +0 & +b_3x_1x_2 & = 0 \\ x_2 \cdot F_1 & = 0 & +0 & +b_3x_2^2 & +0 & +b_1x_0x_2 & +b_2x_1x_2 & = 0 \\ x_0 \cdot F_2 & = c_1x_0^2 & +0 & +0 & +c_2x_0x_1 & +c_3x_0x_2 & +0 & = 0 \\ x_2 \cdot F_2 & = 0 & +0 & +c_3x_2^2 & +0 & +c_1x_0x_2 & +c_2x_1x_2 & = 0 \end{array}$$

and therefore

$$D_2 = det \begin{pmatrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ b_1 & 0 & 0 & b_2 & b_3 & 0 \\ 0 & b_2 & 0 & b_1 & 0 & b_3 \\ 0 & 0 & b_3 & 0 & b_1 & b_2 \\ c_1 & 0 & 0 & c_2 & c_3 & 0 \\ 0 & 0 & c_3 & 0 & c_1 & c_2 \end{pmatrix}$$

Let us consider some properties of D_n

- D_n is a polynomial in the coefficients of F_i.
- For a fixed $i \in [0..n]$, D_n is homogeneous in the coefficients of F_i of degree equal to the number of elements in S_i.
- D_n vanishes whenever $F_0 = \cdots = F_n = 0$ has a non-trivial solution
- D_n is divisible by the resultant $Res(F_0, \ldots, F_n)$.

Proofs for this properties can be found in [CLO04, Ch. 3 §4].

These properties imply that

$$D_n = Res(F_0, \ldots, F_n) \cdot \text{extraneous factor.} \tag{11}$$

So we almost have a resultant. The next proposition will bring us one step closer to the resultant.

Proposition 5.8

The extraneous factor in (11) is an integer polynomial in the coefficients of $\bar{F}_0, \ldots, \bar{F}_{n-1}$, *where* $\bar{F}_i = F_i(x_0, \ldots, x_{n-1}, 0)$.

Note that this extraneous factor does not contain any coefficients of F_n.

Unfortunately D_n can have a much larger degree than the actual resultant. Proposition 5.8 provides a method to compute the resultant of D_n, namely to factor D_n in irreducible factors. Then only the irreducible ones in which all variables appear are parts of the resultant.

The next proposition gives a formula for the appliance of this algorithm.

Proposition 5.9

The resultant is the greatest common divisor of the polynomials $D_0, \ldots, D_n$ *in the ring* $\mathbb{Z}[u_{i,\alpha}]$,

$$Res = \pm GCD(D_0, \ldots, D_n).$$

But the factorization of large multivariate polynomials is very time-consuming, so this algorithm is impractical.

Macaulay showed that the extraneous factor is a minor (i.e. the determinant of a sub-matrix) of the $N \times N$ coefficient matrix of (10).

Definition 5.10

Assume $d_0, \ldots, d_n$ *and* d *are degrees as above.*

1. *A monomial* x^α *of total degree* d *is* ***reduced*** *if* $x_i^{d_i}$ *divides* x^α *for exactly one* i.
2. *If we eliminate all rows and columns corresponding to reduced monomial* x^α *of the coefficient matrix of (10), we get a sub-matrix of the coefficients matrix. The determinant of this sub-matrix we denote as* D'_n.

The following theorem by Macaulay shows that the extraneous factor is precisely D'_n up to a sign.

Theorem 5.11

When $F_0, \ldots, F_n$ *are universal polynomials, the resultant is given by*

$$Res(F_0, \ldots, F_n) = \pm \frac{D_n}{D'_n}.$$

Further, if K *is any field and* $F_0, \ldots, F_n \in K[x_0, \ldots, x_n]$, *then the above formula for resultants holds whenever* $D'_n \neq 0$.

Proof: This theorem was first proven by Macaulay in [Mac02]. ■

If we calculate the multivariate resultant in this way we speak of the **Macaulay resultant**.

One big disadvantage of Theorem 5.11 is that you have to divide two very large polynomials in the universal case, which can be very costly in terms of time. Another disadvantage is that it is possible that both D_n and D'_n vanish in the numerical case.

So it would be great if we could get the resultant in a single determinant. If this is possible in general is not enlightened until now, but there are some special cases where it is possible. One of this special cases is considered in [CLO04, page 109 f.], where every polynomial in (7) has the same degree d.

Dr. Manfred Minimair implemented a Maple package for the Macauley resultant of Theorem 5.11, which we will use in the second main part about modelling robot arms. This package can be found at *http://minimair.org/MR.mpl*.

5.4. Projection operator

As mentioned in the beginning, resultants are projection operators. Now we will define the term Projection operator and show this property for some resultants.

Definition 5.12 (*Projection operator*)

Let $f_1, \ldots, f_n \in \mathbb{C}[a_1, \ldots, a_k][x_1, \ldots, x_{n-1}]$.

$$S : (\mathbb{C}[a_1, \ldots, a_k][x_1, \ldots, x_{n-1}])^n \to \mathbb{C}[a_1, \ldots, a_k]$$

is a ***projection operator*** *if and only if*

$$\forall (b_1, \ldots, b_k) \in \mathbb{C}^k : f_1|_{a_i=b_i}, \ldots, f_n|_{a_i=b_i} \text{ having a common root over } \mathbb{C}$$
$$\Rightarrow S(f_1, \ldots, f_m)|_{a_i=b_i} = 0.$$

So a projection operator reduces the number of variables in an ideal and thereby the dimensions of the corresponding algebraic sets. Thus the algebraic set of solutions of polynomials $f_1, \ldots, f_n$ is reduced to partial solutions, i.e. solutions of $S(f_1, \ldots, f_m)$. These partial solutions can be extended to full solutions in a following extension step.

Corollary 5.13

Let $f_1, \ldots, f_n \in K = \mathbb{C}[x_1, \ldots, x_{n-1}, x_n]$, and $S : K^n \to \mathbb{C}[x_n]$ a projection operator.

If $f_1, \ldots, f_n$ have a common root $(a_1, \ldots, a_n)$then

$$S(f_1, \ldots, f_n)(a_n) = 0$$

We can consider the projection operator as the geometrical projection.

Proposition 5.14

The Sylvester, Macaulay and Dixon resultant are projection operators.

Proof: For the Macaulay resultant the theorem follows from Theorem 5.3. The Sylvester resultant is a special case of Macaulay, so it is also true for it.

For the Dixon resultants Theorem 3.1.2 out of [Sax97, p.44] deliver the result. ■

Actually there are ways to show that other resultants are projection operators, but we do not consider them in this thesis.

6. Modelling of a robot arm

Nowadays robots appear in many different fields. There are various industrial robots, bigger ones which build cars and smaller ones which are used to produce microprocessors. Other mentionable robots are exploring robots, like the mars rovers or the PackBot. In big logistic centres robots support human packers. In the future we will develop nano robots which will be used in medicine or robotic limbs which replace amputated human limbs. We see that robots are very important for the technological progress. Therefore we have to develop them further and further.

In this chapter we learn how to model the kinematics of a robot arm. We especially consider robot arms which are fixed at a platform. As we see in Figure 3 a simple robot arm consists of joints and segments. The first segment, which is fixed at the platform, is called anchor and the last one is called robot hand. The anchor is fixed in position. The hand holds the tool which is used by the robot.

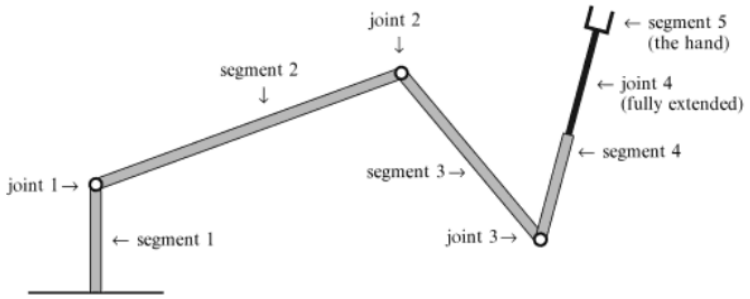


Figure 3:

Segments are the rigid body parts of the robot arm. In the model we have to include the dimensions of this rigid body, especially the length of a segment. Joints are the movable links between the segments, which make the robot flexible. So at a joint we have to consider the rotations and translations in this

joint.

To control the robot arm one has to set up a mathematical model of it. This thesis just considers possible movements of the robot arm. The forces which are used to move or affect an actual robot are not interesting for the studies in this thesis.

So the purpose is to find a mathematical model of the kinematics of a robot arm. Especially the position and orientation of the robot hand will be an issue. All movements, or so called transformations, are composed of rotations and translations in the joints.

6.1. Modelling of a 2-D robot

For an introduction in modelling robots we take a look at two dimensional robots which operate in a plane. The robot in Figure 3 is such a robot. We first study 2-dimensional robots because it is easier to understand their modelling. Later we have a look at 3-dimensional robots which are more common in the real world.

We find a good description of modelling 2-dimensional robots in the book *Ideals,Varieties and Algorithms* [CLO96, Ch. 6.]. This subsection is based on this chapter. The figures are also copied from that book.

Robots in 2-dimensional space use 2 different joints, revolute joints and prismatic joints, whose motions just happens in a 2-dimensional plane.

A revolute joint connects two segments and varies the angle of this connection (see Figure 4). Therefore it is mathematically described by an angle θ. This θ is the counterclockwise angle between line along the actual segment and the next segment. Depending on the freedom of the revolute joint it is parametrizied by a subset of the real interval $[0..2\pi]$. Of course the rotation axis is perpendicular to the plane.

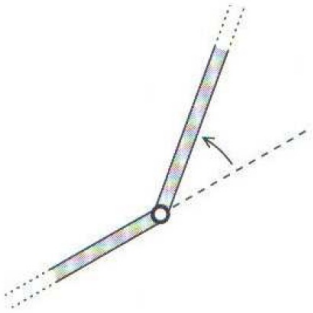


Figure 4: revolute joint

With a prismatic joint we describe an adjustable length of a segment (Figure 5). So we can extend a robot segment by a length d. So we do a translation along a segment. We parametrize a prismatic joint by a real interval $[d_{min}..d_{max}]$, where d_{min} is the length when the prismatic joint is retracted and d_{max} is the length when the joint is fully extended.

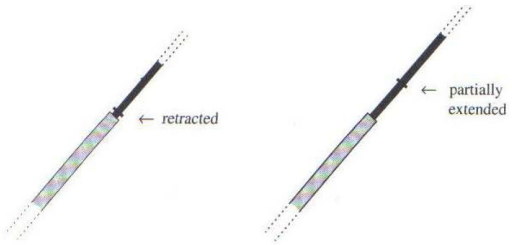


Figure 5: prismatic joint

The robot in Figure 3 has three revolute joints and one prismatic joint.

But how does the mathematical model of a 2-dimensional robot arm look like?

Primarily we are interested in the position and orientation of the robot hand.

First we put a global coordinate system at the first joint (Figure 6), so that the y-axis is put along the anchor segment and the x-axis is perpendicular to it.

We denote (x, y) as the global coordinates of the hand position and let a vector

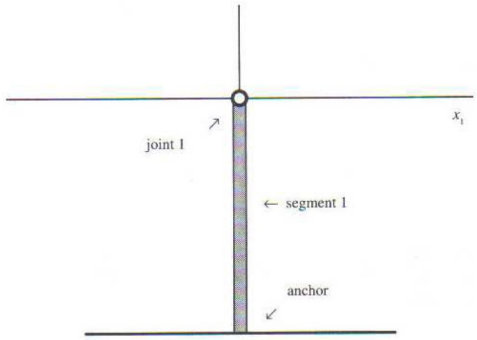


Figure 6: global coordinate system

$o = (o_1, o_2)$ be the orientation of the hand.

To get the global coordinates we have to visit each joint i. There we set up a local coordinate system (Figure 7), calculate the local coordinates (x_i, y_i) of the hand in the joint i and transform them to the previous local coordinate system in joint $i - 1$.

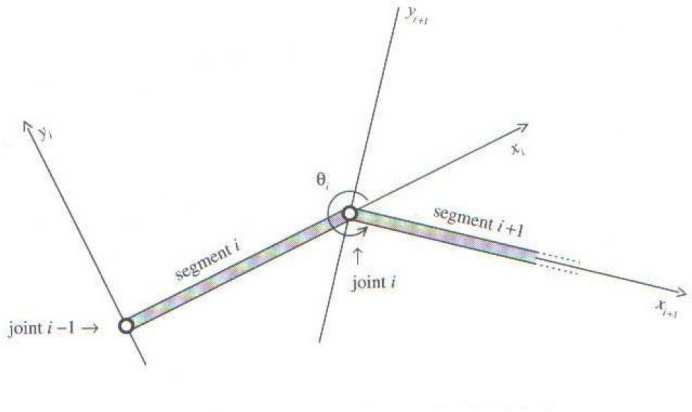


Figure 7: local coordinate system

To get the local coordinates of the hand position in the last joint m should be easy. Normally it is $(l_m, 0)$, where l_m is the length of the hand segment. Then we calculate the hand position w.r.t. the previous local coordinate system in joint $m - 1$ by a simple transformation.

In a planar robot such a transformation is described by:

$$\begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{pmatrix} \cdot \begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} + \begin{pmatrix} l_i \\ 0 \end{pmatrix}, \tag{12}$$

where θ_i is the rotation angle around the joint i and l_i is the length of the segment i.

We see that this is an affine transformation.

So two different matrix operations have to be done for a transformation from one local coordinate system to another. It would be nice if it could be done with one matrix and the transformation matrices of the different joints were similar.

If we work with projective vectors, we write the transformation (12) in just one 3×3-matrix:

$$\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta_i & -\sin\theta_i & l_i \\ \sin\theta_i & \cos\theta_i & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_{i+1} \\ y_{i+1} \\ 1 \end{pmatrix}.$$

We denote the transformation matrix in the i-th joint by

$$A_i = \begin{pmatrix} \cos\theta_i & -\sin\theta_i & l_i \\ \sin\theta_i & \cos\theta_i & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

If we apply this to the robot arm in Figure 3 we get:

$$\begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = A_1 \cdot A_2 \cdot A_3 \cdot A_4 \cdot \begin{pmatrix} x_4 \\ y_4 \\ 1 \end{pmatrix}.$$

Form A_1, A_2, A_3 we get the three rotation angels $\theta_1, \theta_2, \theta_3$ as variables. A_4 creates the variable l_4. Which leads us to the formula:

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = f(\theta_1, \theta_2, \theta_3, l_4)$$
$$= \begin{pmatrix} l_4 \cos(\theta_1 + \theta_2 + \theta_3) + l_3 \cos(\theta_1 + \theta_2) + l_2 \cos(\theta_1) \\ l_4 \sin(\theta_1 + \theta_2 + \theta_3) + l_3 \sin(\theta_1 + \theta_2) + l_2 \sin(\theta_1) \end{pmatrix},$$

where l_2, l_3 are fixed parameters.

Now we have an algorithm to find a mathematical model for robots in a plane. This model contains trigonometric functions. Later we find a way to get rid of them. But first we have a look at the 3-dimensional case.

6.2. Modelling of a 3-D robot

Modelling a robot arm in 3-dimensional space is more complicate than in the 2-dimensional space, but it works the same way. The difficulty in the 3-dimensional space is due to the much more complex joints. To explicate the theory of 3-dimensional joints we use the ideas of the book *Robot manipulators* by Richard Paul [Pau86].

As in the plane we consider the transformation matrices in projective space by adding a 4-th row and column to the rotation matrices. So we are able to put rotations and translations into one transformation matrix. Therefore we will work with projective vectors $(x, y, z, 1)$.

In 3-dimensional space we can rotate around each axis and translate by a vector (x, y, z). Thus in the 3-dimensional space we have the following 4 base transformations.

$$\mathrm{rot_x}(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13}$$

$$\mathrm{rot_y}(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

$$\mathrm{rot_z}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15}$$

$$\mathrm{trans}(x, y, z) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{16}$$

It is possible to describe all joints in 3-dimensional space by a combination of these 4 transformations.

For the sake of completeness, we also quote the rotation around an arbitrary line through the origin with the unit direction vector $v = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$.

$$rot_v(\theta) = \begin{bmatrix} v_x^2\ \mathrm{vers}\,\theta\cos\theta & v_x v_y\ \mathrm{vers}\,\theta - v_z\sin\theta & v_x v_z\ \mathrm{vers}\,\theta + v_y\sin\theta & 0 \\ v_x v_y\ \mathrm{vers}\,\theta + v_z\sin\theta & v_y^2\ \mathrm{vers}\,\theta + \cos\theta & v_y v_z\ \mathrm{vers}\,\theta - v_x\sin\theta & 0 \\ v_x v_z\ \mathrm{vers}\,\theta - v_y\sin\theta & v_y v_z\ \mathrm{vers}\,\theta + v_x\sin\theta & v_z^2\ \mathrm{vers}\,\theta + \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $\mathrm{vers}\,\theta = 1 - \cos\theta$.

The derivation of this formula you can find in [Pau86, Page 26 ff.].

Of course it is possible to gain this rotation by a combination of the base rotations defined above.

Before we use these transformations on joints in a 3-dimensional robot arm, we have to define how the coordinate frames are placed on the different joints. Especially the orientation of the axes have to be specified. We denote the x-axis along the segment which holds the joint. So the x-axis is the approach direction of the joint.

The axis are rotated by rotation matrices in every joint between the anchor and the actual joint, where the local coordinate system is placed. For instance for the directions of the y-axis in the new local coordinate axis the following has to apply:

$$\begin{pmatrix} y_{x_i} \\ y_{y_i} \\ y_{z_i} \\ 0 \end{pmatrix} = R_1 \dots R_{i-1}. \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix},$$

where R_j are the rotation matrices of the joint between the anchor and the actual local coordinate system. $\begin{pmatrix} y_{x_i} \\ y_{y_i} \\ y_{z_i} \end{pmatrix}$ is the direction of the y-axis in the new actual coordinate system.

Then we get the direction of each axis by applying the rotation matrices to the unit vectors $(1,0,0,0),(0,1,0,0),(0,0,1,0)$.

3D joints

Of course there are also revolute and prismatic joints in the 3-dimensional space. The transformation matrices for revolute joints are built up by a translation $\mathrm{trans}(l,0,0)$ for the fixed segment length l on which the joint is placed,

followed by one of the above defined rotations $\mathrm{rot_x}(\theta), \mathrm{rot_y}(\theta), \mathrm{rot_z}(\theta)$.

$$rjx(\theta) = \mathrm{trans}(l,0,0)\,\mathrm{rot_x}(\theta),$$
$$rjy(\theta) = \mathrm{trans}(l,0,0)\,\mathrm{rot_y}(\theta),$$
$$rjz(\theta) = \mathrm{trans}(l,0,0)\,\mathrm{rot_z}(\theta).$$

The prismatic joint is described by a translation

$$prj(d) = \mathrm{trans}(d,0,0),$$

where d is restricted to the interval $[d_{min}..d_{max}]$.

If we just want to put a special segment in our robot arm, we do that by

$$seg(x,y,z) = \mathrm{trans}(x,y,z),$$

where x, y, z is the endpoint of the segment w.r.t. the local coordinate system of the joint on which you place the segment.

Cylindrical joint

First we consider a cylindrical joint. A cylindrical joint has two variables, an angle θ and a length d and a fixed length r.

$$Cyl(d,\theta) = \mathrm{trans}(d,0,0)\,\mathrm{rot_x}(\theta)\,\mathrm{trans}(0,0,r) \tag{17}$$

So a cylindrical joint can be extended by the length d along the x-axis and rotated around it by an angle θ. Therefore we can also interpret a cylindrical joint my a prismatic joint, a revolute joint around the x-axis and a segment of length r.

Spherical/ball joint

In a spherical joint, or ball joint, you have two variable angles α, β.

$$Sph(\alpha,\beta) = \mathrm{trans}(l,0,0)\,\mathrm{rot_x}(\alpha)\,\mathrm{rot_y}(\beta) \tag{18}$$

We first rotate the whole system behind the the joint around the x-axis by α, so that the point we want to reach lies on the plane xy. Then we rotate the point by β in this plane.

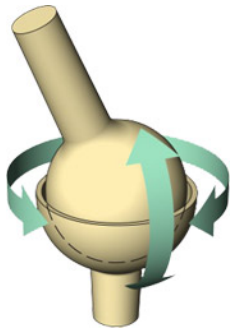


Figure 8: Ball joint

Euler joint

An euler joint is similar to the spherical joint. Additionally we have a third angle γ. As in the spherical joint we first rotate the y-axis in the right direction, then bend the joint around this y-axis by β. Additionally we screw along the following segment, the x-axis of the following local coordinate system.

$$Eul(\alpha, \beta) = \mathrm{trans}(l, 0, 0)\,\mathrm{rot_x}(\alpha)\,\mathrm{rot_y}(\beta)\,\mathrm{rot_x}(\gamma) \tag{19}$$

Roll, pitch, yaw joint

A roll, pitch, yaw joint is comparable to an aeroplane flying along the x-axis. With the vertical stabilizer you can control if the aeroplane rotates left or right, the so called yaw. This is the same as the rotation around the z-axis. The horizontal stabilizer controls if the plane goes up or down, which is also called pitch. This is nothing else than a rotation around the y-axis. The roll around the x-axis is comparable to bring the aeroplane upside down by a screw (without changing the flight direction).

For this joint we first roll the joint by α, then pitch it by β and finally yaw it by γ.

$$RPY(\alpha, \beta) = \mathrm{trans}(l, 0, 0)\,\mathrm{rot_x}(\alpha)\,\mathrm{rot_y}(\beta)\,\mathrm{rot_z}(\gamma) \tag{20}$$

EXAMPLE:

Now we try to model a 3-dimensional robot arm with this theory. We choose the well-known Stanford manipulator (Figure 9). This robot arm was developed by V.D. Scheinman in the book [Sch69].

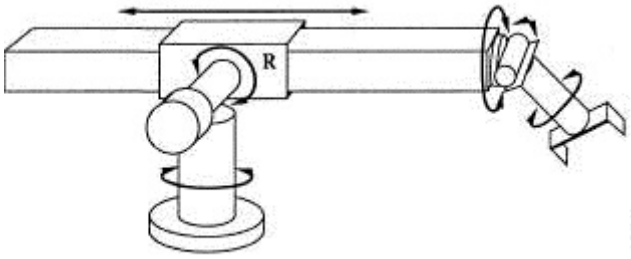


Figure 9: Stanford manipulator

A stanford manipulator is built up by two revolute joints, a prismatic joint and an euler joint. The first joint is a revolute joint around the x-axis , the second one is a rotation around the y axis, the third one is the prismatic joint and the last one is the euler joint.

Therefore we get the following transformation matrices:

$$A_1 = \begin{bmatrix} 1 & 0 & 0 & l_1 \\ 0 & \cos\theta_1 & -\sin\theta_1 & 0 \\ 0 & \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} \cos\theta_2 & 0 & -\sin\theta_2 & l_2 \\ 0 & 1 & 0 & 0 \\ \sin\theta_2 & 0 & \cos\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 1 & 0 & 0 & d_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} \cos\beta_4 & -\sin\beta_4\sin\gamma_4 & -\sin\beta_4\cos\gamma_4 & l_4 \\ -\sin\alpha_4\sin\beta_4 & \cos\alpha_4\cos\gamma_4 - \sin\alpha_4\cos\beta_4\sin\gamma_4 & -\cos\alpha_4\sin\gamma_4 - \sin\alpha_4\cos\beta_4\cos\gamma_4 & 0 \\ \cos\alpha_4\sin\beta_4 & \sin\alpha_4\cos\gamma_4 + \cos\alpha_4\cos\beta_4\sin\gamma_4 & -\sin\alpha_4\sin\gamma_4 - \cos\alpha_4\cos\beta_4\cos\gamma_4 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Let us choose some values for the constants l_1, l_2, l_4 and a segment with the vector v for the hand.

$$l_1 = 0$$

$$l_2 = 1$$

$$l_4 = 0$$

$$v = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

For d_3 we need some restrictions. Let $d_3 \in [2..4]$. This means if the prismatic joint is fully extended then $d_3 = 4$, if it is fully retracted than $d_3 = 2$.

By multiplying the transformation matrices and the hand vector we get the following formulas for the hand position:

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = A_1.A_2.A_3.A_4.v =$$

$$\begin{pmatrix} \cos\theta_2 \cos\beta_4 - \sin\theta_2 \cos\alpha_4 \sin\beta_4 + d_3 \cos\theta_2 + 1 \\ -\sin\theta_1 \sin\theta_2 \cos\beta_4 - \cos\theta_1 \sin\alpha_4 \sin\beta_4 - \sin\theta_1 \cos\theta_2 \cos\alpha_4 \sin\beta_4 - d_3 \sin\theta_1 \sin\theta_2 \\ \cos\theta_1 \sin\theta_2 \cos\beta_4 - \sin\theta_1 \sin\alpha_4 \sin\beta_4 + \cos\theta_1 \cos\theta_2 \cos\alpha_4 \sin\beta_4 + d_3 \cos\theta_1 \sin\theta_2 \\ 1 \end{pmatrix} \tag{21}$$

Note that the last angle γ_4 does not occur in the formulas. That is because the last rotation does not effect the hand position.

As before these formulas contain trigonometric functions. If we want to calculate algebraic solutions we have to get rid of these functions. The next section will provide 2 ways to do this.

6.3. Orientation of the robot hand

A model for the orientation of the robot hand is simply made. It is the dot product the rotation matrices. Rotation matrices look almost the same as the transformation matrices above. If we set the lengths of the transformation matrix to 0 we get the corresponding rotation matrix.

Thus for the example of the Stanford manipulator we have the rotation matrices

$$O_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_1 & -\sin\theta_1 & 0 \\ 0 & \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$O_2 = \begin{bmatrix} \cos\theta_2 & 0 & -\sin\theta_2 & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta_2 & 0 & \cos\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$O_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$O_4 = \begin{bmatrix} \cos\beta_4 & -\sin\beta_4\sin\gamma_4 & -\sin\beta_4\cos\gamma_4 & 0 \\ -\sin\alpha_4\sin\beta_4 & \cos\alpha_4\cos\gamma_4 - \sin\alpha_4\cos\beta_4\sin\gamma_4 & -\cos\alpha_4\sin\gamma_4 - \sin\alpha_4\cos\beta_4\cos\gamma_4 & 0 \\ \cos\alpha_4\sin\beta_4 & \sin\alpha_4\cos\gamma_4 + \cos\alpha_4\cos\beta_4\sin\gamma_4 & -\sin\alpha_4\sin\gamma_4 - \cos\alpha_4\cos\beta_4\cos\gamma_4 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Then the orientation matrix is

$$O = R_1.R_2.R_3.R_4$$

The first column of the orientation matrix is the direction of the local x-axis in the robot hand and the second and third column are the corresponding local y and z axes.

7. Polynomial formulation of robot equations

The mathematical model of a robot arm contains the trigonometric functions $\sin$ and $\cos$. To solve equations with trigonometric functions is much harder than to solve pure polynomial equations. Therefore it would be nice if we could transform trigonometric equations to polynomial ones.

We describe two methods to reach polynomial equations: the more common *unit circle approach* and the *tangent half-angle substitution*.

7.1. Unit circle approach

The unit circle approach is the simpler method to get rid of $\sin$ and $\cos$ in equation systems. It uses the fact that $\begin{pmatrix}\cos\theta\\ \sin\theta\end{pmatrix}$ is a point on the unit circle. The other way round we can parametrize the whole unit circle by $\begin{pmatrix}\cos\theta\\ \sin\theta\end{pmatrix}, \theta \in [0..2\pi]$.

Therefore we can choose

$$c_{\theta_i} = \cos(\theta_i)$$
$$s_{\theta_i} = \sin(\theta_i).$$

We substitute c_{θ_i} for $\cos(\theta_i)$ and s_{θ_i} for $\sin(\theta_i)$ in the formulas of the hand position to get a polynomial equation system. Obviously we get two variables of one angle. Thus we need an additional equation for each substitution. The above mentioned fact causes this additional equation, a restriction to c_{θ_i} and s_{θ_i}, the unit circle equation:

$$c_{\theta_i}^2 + s_{\theta_i}^2 - 1 = 0$$

EXAMPLE:
We apply this theory to the result of the Stanford manipulator example (21).

In the trigonometric formula for the hand position occur 4 angles $\theta_1, \theta_2, \alpha_4, \beta_4$. Thus we choose:

$$\begin{aligned} c_{\theta_1} &= \cos(\theta_1) & s_{\theta_1} &= \sin(\theta_1) \\ c_{\theta_2} &= \cos(\theta_2) & s_{\theta_2} &= \sin(\theta_2) \\ c_{\alpha_4} &= \cos(\alpha_4) & s_{\alpha_4} &= \sin(\alpha_4) \\ c_{\beta_4} &= \cos(\beta_4) & s_{\beta_4} &= \sin(\beta_4) \end{aligned}$$

We substitute these variables in (21) and we get:

$$\begin{aligned} x &= c_{\theta_2}c_{\beta_4} - s_{\theta_2}c_{\alpha_4}s_{\beta_4} + d_3c_{\theta_2} + 1 \\ y &= -s_{\theta_1}s_{\theta_2}c_{\beta_4} - c_{\theta_1}s_{\alpha_4}s_{\beta_4} - s_{\theta_1}c_{\theta_2}c_{\alpha_4}s_{\beta_4} - d_3s_{\theta_1}s_{\theta_2} \\ z &= c_{\theta_1}s_{\theta_2}c_{\beta_4} - s_{\theta_1}s_{\alpha_4}s_{\beta_4} + c_{\theta_1}c_{\theta_2}c_{\alpha_4}s_{\beta_4} + d_3c_{\theta_1}s_{\theta_2} \end{aligned} \tag{22}$$

Additionally we have to add the 4 restrictions

$$\begin{aligned} c_{\theta_1}^2 + s_{\theta_1}^2 - 1 = 0 \\ c_{\theta_2}^2 + s_{\theta_2}^2 - 1 = 0 \\ c_{\alpha_4}^2 + s_{\alpha_4}^2 - 1 = 0 \\ c_{\beta_4}^2 + s_{\beta_4}^2 - 1 = 0 \end{aligned}$$

Summarizing we have 7 equations in 9 variables $d_3, c_{\theta_1}, s_{\theta_1}, c_{\theta_2}, s_{\theta_2}, s_{\alpha_4}, c_{\alpha_4}, c_{\beta_4}, s_{\beta_4}$.

As we see the polynomial equations are similar to the trigonometric equations. They have the same degree. Of course, the solutions of the polynomial equations can be easily transformed to solutions of the trigonometric equations. The disadvantage of the unit circle approach is that the number of variables increases and therefore also the number of equations has to increase.

7.2. Tangent half-angle substitution

Another method to get rid of trigonometric functions is the tangent half-angle substitution. It is also known under the name Weierstrass substitution.

Here we use trigonometric identities to gain a polynomial form for the trigonometric functions. More precisely we use the double-angle formulae of $\sin$ and $\cos$ or rather the half-angle formula of $\tan$.

For this purpose let

$$u_i = \tan(\frac{\theta_i}{2}).$$

Then we derive by the double angle formulae:

$$\cos(\theta_i) = \frac{1 - u_i^2}{1 + u_i^2}$$
$$\sin(\theta_i) = \frac{2u_i}{1 + u_i^2}$$

Here we do not need an additional restriction because the number of variables does not increase. But we see that we substitute fractions for $\sin$ and $\cos$, therefore we properly will not get a polynomial equation system in the first place. To get one we have to multiply the rational formulas with their least common divisor.

EXAMPLE:

Similar to above we apply the tangent half-angle substitution to the result of the example about 3-dimensional joints (21).

In the trigonometric formula for the hand position 4 angles $\theta_1, \theta_2, \alpha_4, \beta_4$ occur.

Thus we choose:

$$\begin{aligned}
\frac{1-u_{\theta_1}^2}{1+u_{\theta_1}^2} &= cos(\theta_1) & \frac{2u_{\theta_1}}{1+u_{\theta_1}^2} &= sin(\theta_1)\\
\frac{1-u_{\theta_2}^2}{1+u_{\theta_2}^2} &= cos(\theta_2) & \frac{2u_{\theta_2}}{1+u_{\theta_2}^2} &= sin(\theta_2)\\
\frac{1-u_{\alpha_4}^2}{1+u_{\alpha_4}^2} &= cos(\alpha_4) & \frac{2u_{\alpha_4}}{1+u_{\alpha_4}^2} &= sin(\alpha_4)\\
\frac{1-u_{\beta_4}^2}{1+u_{\beta_4}^2} &= cos(\beta_4) & \frac{2u_{\beta_4}}{1+u_{\beta_4}^2} &= sin(\beta_4)
\end{aligned}$$

We substitute these variables in (21) and we get:

$$\begin{aligned}
x &= \frac{1-u_{\theta_2}^2}{1+u_{\theta_2}^2}\frac{1-u_{\beta_4}^2}{1+u_{\beta_4}^2} - \frac{2u_{\theta_2}}{1+u_{\theta_2}^2}\frac{1-u_{\alpha_4}^2}{1+u_{\alpha_4}^2}\frac{2u_{\beta_4}}{1+u_{\beta_4}^2} + \frac{1-u_{\theta_2}^2}{1+u_{\theta_2}^2}d_3 + 1\\
y &= -\frac{2u_{\theta_1}}{1+u_{\theta_1}^2}\frac{2u_{\theta_2}}{1+u_{\theta_2}^2}\frac{1-u_{\beta_4}^2}{1+u_{\beta_4}^2} - \frac{1-u_{\theta_1}^2}{1+u_{\theta_1}^2}\frac{2u_{\alpha_4}}{1+u_{\alpha_4}^2}\frac{2u_{\beta_4}}{1+u_{\beta_4}^2} - \frac{2u_{\theta_1}}{1+u_{\theta_1}^2}\frac{1-u_{\theta_2}^2}{1+u_{\theta_2}^2}\frac{1-u_{\alpha_4}^2}{1+u_{\alpha_4}^2}\frac{2u_{\beta_4}}{1+u_{\beta_4}^2} - \frac{2u_{\theta_1}}{1+u_{\theta_1}^2}\frac{2u_{\theta_2}}{1+u_{\theta_2}^2}d_3\\
z &= \frac{1-u_{\theta_1}^2}{1+u_{\theta_1}^2}\frac{2u_{\theta_2}}{1+u_{\theta_2}^2}\frac{1-u_{\beta_4}^2}{1+u_{\beta_4}^2} - \frac{2u_{\theta_1}}{1+u_{\theta_1}^2}\frac{2u_{\alpha_4}}{1+u_{\alpha_4}^2}\frac{2u_{\beta_4}}{1+u_{\beta_4}^2} + \frac{1-u_{\theta_1}^2}{1+u_{\theta_1}^2}\frac{1-u_{\theta_2}^2}{1+u_{\theta_2}^2}\frac{1-u_{\alpha_4}^2}{1+u_{\alpha_4}^2}\frac{2u_{\beta_4}}{1+u_{\beta_4}^2} + \frac{1-u_{\theta_1}^2}{1+u_{\theta_1}^2}\frac{2u_{\theta_2}}{1+u_{\theta_2}^2}d_3
\end{aligned} \tag{23}$$

This is not a polynomial equation system jet. We have to multiply these equations by there least common divisor, e.q. $(1+u_{\theta_1}^2)(1+u_{\theta_2}^2)(1+u_{\alpha_4}^2)(1+u_{\beta_4}^2)$. If we do that, the first equation in (23) is transformed into the polynomial equation:

$$\begin{aligned}
&(x+d_3-2)u_{\alpha_4}^2u_{\beta_4}^2u_{\theta_2}^2 + (x-d_3)u_{\alpha_4}^2u_{\beta_4}^2 + (x+d_3)u_{\alpha_4}^2u_{\theta_2}^2 + (x+d_3-2)u_{\beta_4}^2u_{\theta_2}^2 - 4u_{\alpha_4}^2u_{\beta_4}u_{\theta_2}\\
&\quad +(x-d_3-2)u_{\alpha_4}^2 + (x-d_3)u_{\beta_4}^2 + (x-d_3)u_{\theta_2}^2 + 4u_{\beta_4}u_{\theta_2} + x - d_3 - 2 = 0.
\end{aligned}$$

We see that the tangent half-angle substitution delivers polynomials of high degree. To solve them by Gröbner bases will take some time or overload the computer. But the number of variables and equations stays the same after the substitution.

Note that if $\theta_i = (2n+1)\pi$ then we have a singularity in u_i, but there are no singularities in $\frac{1-u_i^2}{1+u_i^2}$ and $\frac{2u_i}{1+u_i^2}$.

Comparison

Unit circle approach	**Tangent half-angle substitution**
• polynomial form	• rational form
• additional variables	• same number of variables
• additional equations	• same amount of equations
• same degree	• higher degree

8. Properties of the mathematical model

If we calculate a mathematical model of a robot arm we can use it in two different ways. We can consider the so-called Forward Kinematic Problem or the so called Inverse Kinematic Problem.

We use the Forward Kinematic Problem if the angles of the joints and the lengths of the segments and prismatic joints are given. So if there is a fix configuration for all joints in the robot arm we can calculate the hand position to this configuration.

If we look at the model from above we see that the Forward Kinematic Problem is easy to solve. We simply substitute the angles and lengths and get the hand position.

The Inverse Kinematic Problem works in the opposite direction. There is a point in space (sometimes also an orientation) and the goal is to calculate a configuration to reach this point. So we are interested in the configuration of the robot arm, the angles and lengths of the joints.

If we substitute a point which describes a hand position in the model we get a system of polynomials. Therefore we have to solve this system to get the configurations which are valid for the given hand position.

As we already know, a polynomial equation system can have no solutions, finitely many solutions or infinitely many solutions. This thesis mainly considers the last options. More precisely we consider if there is a continues curve of solutions, on which we can move the robot arm.

If the system of polynomial equations of a robot arm has no solutions, or especially no real solution, then there is no chance to reach the given point by the robot arm. When there are finitely many solutions for the mathematical model then we know that there are finitely many isolated configurations to reach the given point. And therefore we cannot move continuously between solutions.

But if there are infinitely many solutions then properly most of them are on a continuous curve or surface in the affine space over a field K.

The title of the thesis contains the phrase "flexible solutions". In the next definition we finally clarify what this means.

Definition 8.1

*A system of polynomial equations $P(X) = 0$ has **flexible solutions** over K if and only if it has two different solutions X_0, X_1, such that there exists a continuous function:*

$$\gamma : [0,1] \to K^n$$

with

$$X_0 = \gamma(0) \text{ and } X_1 = \gamma(1)$$

and

$$P(\gamma(t)) = 0 \text{ for } t \in [0,1].$$

So flexible solutions mean that there is at least a curve of solutions in K^n of the system of polynomial equations. On this curve we can move continuously between two solutions.

Note that in addition to flexible solutions there also can occur isolated inflexible solutions.

From the definition of flexible solutions we observe the following.

Lemma 8.2

If a system of polynomial equations has flexible solutions over $\mathbb{R}$ it has infinitely many real solutions.

Lemma 8.3

If a system of polynomial equations has flexible solutions over $\mathbb{R}$ or $\mathbb{C}$ it has infinitely many solutions over the complex numbers.

For the application in robotics this means that we can move the robot arm

without changing the position of the robot hand if we have flexible solutions over $\mathbb{R}$.

It would be nice if we could decide if a system of polynomial equations has flexible solutions or not. For algebraically closed fields K Theorem 4.19 gives us a way to decide if there are flexible solutions or not.

As we know the algebraic set of the ideal of the system of polynomial equations is equal to the solutions of the system of polynomial equations. The consequence is that if there exists a x_i which do not have a polynomial with a pure leading power product of x_i in the Gröbnerbasis of $P(X) = 0$, $P(X) = 0$ has to have flexible solutions over $\mathbb{C}$.

Another way to decide if there are flexible solutions of a system of polynomial equations over a field K is via resultants. Professor Lewis stated the idea for the following theorem in his paper "Algorithmic Search for Flexibility Using Resultants of Polynomial Systems" [LC07].

The paper only considers non-degenerate situations where the flexible solutions are not parallel to the projection axis. To deal with such degenerate cases we added the perturbation matrix M in the following theorem statement. A system of polynomial equations is degenerated if the system of polynomial equations and therefore the solutions are independent of the variable which is eliminated in the resultants algorithm.

Theorem 8.4

Let $(p_1, \ldots, p_n)$ polynomials in $\mathbb{C}[x_1, \ldots, x_n]$ with flexible solutions over $\mathbb{C}^n$ and $M : \mathbb{C}^n \to \mathbb{C}^n$ an invertible affine transformation in $\mathbb{C}^n$.

Then for almost all M,

$$Res_{x_1,\ldots,x_{n-1}}(p_1 \circ M, \ldots, p_n \circ M) = 0$$

Proof: In the following we write $R = Res_{x_1,\ldots,x_{n-1}}(p_1 \circ M, \ldots, p_m \circ M)$. We will show the

Theorem by contradiction.

The resultant R is a polynomial in $\mathbb{C}[x_n]$. Assume $R \neq 0$. We distinguish two cases:

Case 1: R is a constant different from 0. If that is the case there is no solution for $P(X) \circ M = 0$ per definition of resultants, so there is no flexible solution.

Case 2: R is a non-constant polynomial in $\mathbb{C}[x_n]$. This polynomial has finitely many solutions. We know that the resultant is a Projection Operator. Thus we look at resultants in a geometrical way.

In a non-degenerate situation, infinitely many points in $\mathbb{C}^n$ were projected to infinitely many points in $\mathbb{C}^{n-1}$. Consequentially, finitely many solutions of R imply finitely many solutions of $P(X) = 0$. Thus $P(X) = 0$ has no flexible solutions if there is not an infinite-to-finite projection.

Infinite to finite projection just occurs if the variety of the solutions is parallel to the projection axis. So for the special case of degenerate projection we transform the polynomials and thereby their common solution by M. A hyperplane in $\mathbb{C}^n$ has 2 rotation transformations around an axis which are parallel to this axis, but infinitely many which are not parallel. Thus for almost all transformations there is no degenerate projection. ■

Remark: This theorem works for all invertible transformations but the essential transformations are rotations to get rid of the degenerate situation. It is easy to choose a suitable affine transformation to get rid of a degenerate situation.

Example: We consider the 3 polynomials in $\mathbb{C}[x, y, z]$:

$$f = 2x + 3y + 5$$
$$g = x - 7y + 2$$
$$h = 3x - 4y + 7$$

their common solution is $\begin{pmatrix} -\frac{41}{17} \\ -\frac{1}{17} \\ z \end{pmatrix}$ with $z \in \mathbb{C}$.

Because they are constant in z it is a degenerative system, so the resultant respective z is 1.

If we transform the polynomials with the Matrix M:

$$M = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

we get:

$$f_M = 2x + 3y + 5z + 5$$
$$g_M = x - 7y - 6z + 2$$
$$h_M = 3x - 4y - z + 7$$

These polynomials are not constant in z any more. If we calculate the resultant of these new polynomials w.r.t. z and another variable we will get 0. This works for almost all invertible transformations. □

Nevertheless we do not know a way to decide if a system of polynomial equations over $\mathbb{R}$ has flexible solutions or not.

So for the applications in robotics we have to calculate the algebraic set of a system of polynomial equations in the algebraic closure $\mathbb{C}$ to see if flexible solutions over $\mathbb{C}$ occur. Afterwards we filter the interesting real solutions of the complex ones.

9. Solution methods

To get a solution for our mathematical model we have to solve a system of polynomial equations. In theory we learned 2 ways to do this, Gröbner basis and resultants.

9.1. Solving by Gröbner bases

One way to solve a system of polynomial equations are Gröbner bases. All common computer algebra systems provide functions to calculate Gröbner bases. In Maple these functions are contained in the package `groebner`. To calculate a Gröbner basis of a polynomial ideal over $K[X]$ in Maple, we need an arbitrary basis F of the ideal and an ordering $<$ on the terms $[X]$. Then the command `Basis( F, <)` returns the corresponding reduced Gröbner basis.

We will see that if we apply the Gröbner basis algorithm to the polynomials we get from the unit circle approach, we mostly get a nice Gröbner basis, depending on the term ordering we choose. But if we apply the algorithm to the polynomials we get from the tangent half-angle substitution the calculation may take some time, especially if the degree of the polynomials in the model is high. This reflects the fact that the complexity of Gröbner basis computation depends heavily on the degree of the initial basis polynomials.

The term ordering on the polynomials is very important at the calculation of a Gröbner basis. If we chose the wrong ordering the calculation runs a very long time or the computer algebra system crashes. Some experiments showed that many term orderings bring problems in the computation of Gröbner bases of the robot arm models.

Of course we have to choose a lexicographical ordering to do a proper elimination step. Because the length variables d_i just occur in the polynomials of the hand position, the length variables should be eliminated first, and therefore

they should be the highest variables. The angle variables s_i, c_i then should be ordered ascending or decreasing, according to their occurrence in the robot arm. Some tests showed that in general a decreasing order delivers a nicer Gröbner basis.

Therefore we order the polynomials so that the length variables are greater than the angle variables. The length variables just as angle variables are order decreasing, according to their occurrence in the robot arm. For example for the Stanford manipulator we choose the lexicographical ordering:

$$d_3 > c_{\theta_1} > s_{\theta_1} > c_{\theta_2} > s_{\theta_2} > c_{\alpha_4} > s_{\alpha_4} > c_{\beta_4} > s_{\beta_4}. \tag{24}$$

Next we discuss the application of Gröbner bases to the model of the Stanford manipulator we derived above. We apply the Buchberger algorithm to the above equations (22) w.r.t. the lexicographical ordering (24). We get a Gröbner basis with 10 polynomials of total degrees less or equal to 7 in the variables.

$$\texttt{Basis(}(22), plex(d_3, c_{\theta_1}, s_{\theta_1}, c_{\theta_2}, s_{\theta_2}, c_{\alpha_4}, s_{\alpha_4}, c_{\beta_4}, s_{\beta_4})) = \tag{25}$$

1: $c_{\beta_4}^2 + s_{\beta_4}^2 - 1,$

2: $c_{\alpha_4}^2 + s_{\alpha_4}^2 - 1,$

3: $(-2x+4)s_{\theta_2}c_{\alpha_4}s_{\beta_4} + s_{\alpha_4}^2 s_{\beta_4}^2 s_{\theta_2}^2 - s_{\beta_4}^2 + (-x^2 - y^2 - z^2 + 4x - 4)s_{\theta_2}^2 + y^2 + z^2,$

4: $c_{\theta_2}^2 + s_{\theta_2}^2 - 1,$

5: $yc_{\alpha_4}c_{\theta_2}s_{\alpha_4}^2 s_{\beta_4}^3 + (x^2y - y^3 - yz^2 - 4xy + 4y)c_{\theta_2}c_{\alpha_4}s_{\beta_4} + (-xy + 2y)c_{\theta_2}s_{\alpha_4}^2 s_{\beta_4}^2 s_{\theta_2} +$
$(x^3y + xy^3 + xyz^2 - 6x^2y - 2y^3 - 2yz^2 + 12xy - 8y)s_{\theta_2}c_{\theta_2} + zs_{\alpha_4}^3 s_{\beta_4}^3 +$
$(y^2 + z^2)s_{\alpha_4}^2 s_{\beta_4}^2 s_{\theta_1} - zs_{\alpha_4}s_{\beta_4}^3 + (x^2z - 4xz + 4z)s_{\alpha_4}s_{\beta_4} + (-y^2 - z^2)s_{\theta_1}s_{\beta_4}^2 +$
$(x^2y^2 + x^2z^2 - 4xy^2 - 4xz^2 + 4y^2 + 4z^2)s_{\theta_1},$

6: $zc_{\alpha_4}s_{\alpha_4}s_{\beta_4}^2 + (y^2 + z^2)c_{\alpha_4}s_{\beta_4}s_{\theta_1} - yc_{\theta_2}s_{\alpha_4}^2 s_{\beta_4}^2 + (y^3 + yz^2)c_{\theta_2} + (xz - 2z)s_{\alpha_4}s_{\beta_4}s_{\theta_2} +$
$(xy^2 + xz^2 - 2y^2 - 2z^2)s_{\theta_1}s_{\theta_2},$

7: $yc_{\alpha_4}s_{\beta_4} + zc_{\theta_2}s_{\alpha_4}s_{\beta_4} + (y^2 + z^2)s_{\theta_1}c_{\theta_2} + (xy - 2y)s_{\theta_2},$

8: $s_{\alpha_4}^2 s_{\beta_4}^2 + 2z s_{\alpha_4} s_{\beta_4} s_{\theta_1} + (y^2 + z^2) s_{\theta_1}^2 - y^2,$

9: $y c_{\theta_1} + z s_{\theta_1} + s_{\alpha_4} s_{\beta_4},$

10: $-z c_{\alpha_4} s_{\alpha_4} s_{\beta_4}^2 + (-y^2 - z^2) c_{\alpha_4} s_{\beta_4} s_{\theta_1} + (xy - 2y) c_{\beta_4} + y c_{\theta_2} s_{\alpha_4}^2 s_{\beta_4}^2 +$
$(-x^2 y - y^3 - y z^2 + 4xy - 4y) c_{\theta_2} + (xy - 2y) d_3 + xy - 2y.$

If we apply the same algorithm to a half-angle approach (23), we get a Gröbner basis with 17 polynomials of total degrees greater or equal to 10 in the variables.

`Basis(`$(23), plex(d_3, u_{\theta_1}, u_{\theta_2}, u_{\alpha_4}, u_{\beta_4})$).

Also the calculation of the Gröbner basis in this example from the tangent half-angle substitution takes more than 50 times longer than the unit circle approach. Other calculations also show that the unit circle approach is in general much faster than the tangent half-angle substitution for Gröbner basis.

Thus in the following we assume that we model the robot with the unit circle approach for Gröbner basis calculations.

After we have calculated a Gröbner basis the solutions of the equation system can be found easily. Later we discuss the extension step of Grobner bases in detail. Before that we have a look at another method to solve systems of polynomial equations.

9.2. Solving by resultants

Alternatively to Gröbner bases we can calculate multivariate resultants of the polynomial model. For the calculation of multivariate Macaulay resultants we use the package `mr.mpl` by Minimayr. In the following we do the example of the Stanford manipulator to study the application of resultants for the problem in this thesis.

As we already know, calculation of the resultant of many polynomials needs a vast amount of time and delivers very huge polynomials. Thus it is impractical to apply resultants to the model of the unit circle approach. We also see the infeasibility of resultants on the unit circle approach when we try to calculate a Macaulay resultant of the simple model of the Stanford manipulator (22). The computer algebra system Maple requires enormous amount of time to do the calculations. Most of the time the system crashes.

Another issue we have to deal with when we work with resultants on the unit circle approach is that we may have a degenerate projection. For practical reasons one could just calculate a resultant of the three hand position polynomials and manually choose values which fulfil the unit circle equations. But this can be complicated and is hard to program.

So we use the resultant method if we use the tangent half-angle substitution. But we will also see that the Gröbner basis algorithm on the unit circle approach is more useful than resultants for the needs of robotic simulation.

First we use the Macaulay resultant by Minimayr on the tangent half-angle substitution model of the Stanford manipulator (23). This delivers a huge polynomial.

```
R:= MR:-MResultant(F, [d[3], u[theta[1]]])
```

$$= -16(y^2+z^2)yu_{\theta_2}^{22}u_{\alpha_4}^{22}u_{\beta_4}^{20} + \ldots 5638 \textit{ terms} \ldots - 16(y^2+z^2)y$$

The gained resultant has 6 indeterminates. The 3 indeterminates x, y, z are the coordinates of the hand position and therefore the parameters to be chosen. For example we choose the hand position $(3, 2, 2)$.

The rest are variables and the total degree of the resultant w.r.t. these variables u_i is 64. The degrees w.r.t. to the single variables u_i is 22. To solve the resultant we freely choose 2 of the 3 u_i in the allowed interval and substitute the values. For the example we choose $u_{\beta_4} = \tan(\frac{1.0177}{2}), u_{\alpha_4} = \tan(\frac{2.0354}{2})$.

After some calculations we get the simplified resultant:

$(u_{\theta_2}+2.2869)(u_{\theta_2}+1)^2(u_{\theta_2}+0.4373)(u_{\theta_2}-0.5796)(u_{\theta_2}-1)^2(u_{\theta_2}-1.7253)(u_{\theta_2}^2+0.0164u_{\theta_2}+1.0009)(u_{\theta_2}^2+0.0059u_{\theta_2}+0.9848)(u_{\theta_2}^2+0.0043u_{\theta_2}+1.016)(u_{\theta_2}^2-0.0127u_{\theta_2}+0.9897)(u_{\theta_2}^2-0.0139u_{\theta_2}+1.009)(u_{\theta_2}^2-0.2001u_{\theta_2}+0.04988)(u_{\theta_2}^2-4.0125u_{\theta_2}+20.0484)$

Afterwards we solve this resultant of degree 22.

The calculations so far take only little time.

Now we have to filter the real solutions out of all solutions of the resultant and choose one.

$\{-2.2869,-1,-1,-0.4373,0.5796,1,1,1.7253\}$

These are now real solution candidates for u_{θ_2}. If we choose the wrong candidate we would not get a valid solution. So we probably have to calculate a whole solution which each candidate. $u_{\theta_2}=-0.4373$ will deliver a valid solutions so we choose this value.

To get solution values for the eliminated variables, in the above example d_3 and u_{θ_1}, we need further elimination steps. First we have to substitute the solution into the original basis. Then one way, is to calculate other resultants which eliminate all remaining variables but one. In the example we eliminate d_3. We have 3 polynomials but just one variable to eliminate. So we have to take 2 polynomials of the three and calculate the resultant. So we get two resultants.

$$(u_{\theta_1}-1.58437)(u_{\theta_1}-3.812)$$
$$(u_{\theta_1}+1.58437)(u_{\theta_1}-3.812)(u_{\theta_1}^2+2.2\cdot 10^{-10}u_{\theta_1}+1)$$

Again we calculate the roots of these resultants. Now we have to choose a common root and substitute it again in the original basis. We see that $u_{\theta_1}=3.812$ is the only common root of the resultants.

If we substitute the solutions we get so far in the original basis we get the 3 polynomials:

$$-10.8854 + 3.8428d_3, \quad -89.7393 + 31.6805d_3, \quad -159.2735 + 56.2276d_3$$

The common root of these 3 polynomials is $d_3 = 2.8326$.

Therefore we get the solution,

$$\{u_{\alpha_4} = 1.61978, u_{\beta_4} = 0.55785, u_{\theta_2} = -.437282, u_{\theta_1} = 3.812, d_3 = 2.83265\}.$$

Summarizing, by the resultant method we just get a polynomial for determining a partial solution which we then need to extend to a full solution of the whole system. The calculations of the resultant on the tangent half-angle approach are quick but the extension step is much more complicated than the extension step of Gröbner Bases. First we have to filter the real solutions out of all solutions. Then we have to choose the right candidate or more precisely calculate solutions for all candidates to choose the right candidate. Additionally we have to do further elimination steps in the extension step to get the whole solution.

In calculation of the simulation of a robot arm the elimination step normally has to be done just once. The extension step has to be done several times, actually each time the robot arm is moved. Thus the efficiency of the extension step is much more important than the efficiency of the elimination step.

For these reasons we will write a computer program, which uses Gröbner bases on the unit circle model to simulate the robot arm.

9.3. Parametric Gröbner bases

We calculate a general Gröbner basis for all possible hand positions in the space of motion of the robot arm. Therefore the Gröbner basis contains also the parameters x, y, z. Thus, we have computed a so called parametric Gröbner

basis. In a few particular cases this specialisation of this basis can lead to problems.

In the program to simulate the robot arm we calculate a Gröbner basis over the function field $\mathbb{C}(x, y, z)$. Thus, we have polynomials in $\mathbb{C}(x, y, z)[\mathcal{J}]$, where $\mathcal{J}$ are the set of the variables of the robot arm. When we substitute the hand position in this Gröbner basis, we do a **specialization** of the Gröbner basis. But when we do this specialization we have to consider the case that the leading coefficients or their denominators become 0. In that case the polynomials we get if we specialize a Gröbner basis, do not constitute a Gröbner basis.

EXAMPLE:

Let $f, g \in \mathbb{C}(t)[x, y]$

$$f = x^2 - ty$$
$$g = 2tx + y^2$$

If we calculate the Gröbner basis wrt. lexicographical ordering $x > y$ we get

$$G = \{-4t^3y + y^4, 2tx + y^2\}$$

During the calculation we divide the polynomials by $2t$.

$$spol(f, g) = x^2 - ty - \frac{x}{2t}(2tx + y^2) = \frac{xy^2}{2t} - ty$$

These calculations are wrong if $t = 0$. Therefore, if $t = 0$, G will not be a Gröbner basis any more.

We also see that if we substitute $t = 0$ into f, g and calculate the Gröbner basis we get:

$$G' = Basis(f_0, g_0, x > y) = \{x^2, y^2\}.$$

If we substitute $t = 0$ into G we get $G'' = \{y^2\}$. So the x disappears. Therefore G'' is not a Gröbner basis for $\langle f_0, g_0 \rangle$. □

There are different ways to handle the disappearing coefficients. The simplest way is to substitute the parameters in the original basis and to calculate a new non-parametric Gröbner basis afterwards.

Another way are comprehensive Gröbner bases. This kind of parametric Gröbner basis remains a Gröbner basis after specialisation. For the existence and construction of a comprehensive Gröbner basis for a polynomial ideal we refer to Weispfenning in [Wei92].

We also can build a Gröbner system by separating the field of parameters into partitions. These partitioning is defined by the leading coefficients of the Gröbner basis. So one distinguishes the areas where the leading coefficients are 0. More details you can find in the paper [MM09].

In the case of the robot arm model the parameters are the coordinates x, y, z of the hand position.

9.4. Extraction of solutions from a Gröbner basis

To get a Gröbner basis for a specific hand position we have to substitute this coordinates into the parametric Gröbner basis of the robot arm. If the specialisation delivers a Gröbner basis it can be solved by doing some simple steps.

First take the smallest non-zero polynomial w.r.t. the term order. If this polynomial contains more than one variable, choose values for all but one variable and remember them. The variables with values are now free variables. Substitute the values for the free variables into the polynomial. Solve the polynomial after the remaining variable and add the solution to the system solution. Substitute the gained solutions into the Gröbner basis. The chosen smallest polynomial should be the zero polynomial now. Repeat this step until the Basis just contain zero polynomials. (See Algorithm 2.)

The resulting list of solution contains each variable maximal once. If every

Algorithm 2 FINDSOLUTION $(Input : G, <\ Output : sol)$

▷ where G is a Gröbner basis w.r.t. to the term ordering $<$ and sol is a common root of the polynomials in G, e.q. a point in the corresponding algebraic set.

```
sort G ascending w.r.t. <;
freevars = sol = [];
for p in G do
    vars := indeterminates of p;
    n := numelems(vars);
    if n < 1 then
        if p = 0 then
            next;
        else
            return no solution;
        end if
    else if n > 1 then
        add vars[2..n] to freevars;
        chose values for the all freevars and add them to sol;
        substitute sol into p;
    end if
    solve p w.r.t. first variable;
    add the solution to sol;
    substitute sol into G;
end for
return sol
```

variable in the Gröbner Basis has a value and all polynomials become 0 if we substitute the solution we have a solution of polynomial system.

If there is a variable which has no value there is a problem with the specialisation of the Gröbner basis. Then we have to do specialisation before Gröbner basis computation and repeat the steps.

At the first look Algorithm 2 may look a bit long or complicated but it is simple and easy to understand. It will also compute rather fast. The only time consuming step is the calculation of the roots of the polynomials. If there are no free variables then the algorithm has to solve, at most the number of variables in the Gröbner basis, univariate polynomials. For an actual computer algebra system this is no problem.

If we do some tests we can observe that mostly the free variables come from the restrictions we get from the unit circle approach. If we have a free angle θ_i then the corresponding unit circle polynomial is often the smallest polynomial which has to be 0 for $c_{\theta_i}, s_{\theta_i}$. So the number of free variables is normally less than the number of unit circle polynomials in the Gröbner basis.

In the simulation it is possible to choose values for the free variables and calculate a solution to them. Another option will be to run through the interval of valid values for one specific free angle where the other free angles are fixed.

Certainty we have to check if the gained solution is valid for the mathematical model of the robot arm. Especially prismatic joints mostly have maximal and minimal limits on their lengths. So after we calculate a solution we have to verify its validity for the model.

In this thesis we will check 4 conditions to validate a solution for a robot arm.

- Is the solution over the real numbers?
- Are the restrictions from the unit circle approach fulfilled?
- Are the limit restrictions of the modelling fulfilled? (E.q. prismatic joints

limits)

- Does the mathematical model actually reach the hand position if the solution is substituted?

If we can answer these 4 questions with "yes" we calculated a valid solution for the robot arm. If one of the answers is "no" we have to calculate a different solution and check if it is valid.

Of course it is possible that we can not find a valid solution because there is none. So we should stop the program after a certain amount of checked solutions. Some tests showed that in 50 randomly calculated solutions there is most likely a valid solution, if there are any valid solutions.

When we have a valid solution we can calculate the coordinates of the different joints and the hand position by substituting the solution in the corresponding mathematical models. Thus we can draw a semantic view of the robot arm.

10. Usage of `RS.mpl` by the example of the Stanford manipulator

We have modelled and solved the Stanford manipulator in the previous sections. Now we want to do this with the Maple program, Robot Simulation `RS.mpl`. The code of `RS.mpl` you can find in Appendix A.

10.1. User guide

In the following steps the code examples are the commands we have to put into a Maple worksheet to run the simulation. To use the Robot simulation program we have to read the file with

```
read("...\RS.mpl")
```

First we have to put the Stanford manipulator in an input format for the program. The input format for the program is a list of equations. Each equation represents a joint in the robot arm. On the left hand side of the equation there has to be a four character string of the form `"ijjj"`, where `i` is the position of the joint and `jjj` is the type of the joint. On the right hand side of the equation are the parameters for the joint. The list has to be sorted increasingly by the position of the joint.

Table 1 is a list of the types of joints in `RS.mpl`, d_{i-1} is the length of the segment along the x axis, on which the joint is placed on.

So for the Stanford manipulator with two revolute joints, a prismatic joint and an euler joint we get the list:

```
j:=["1rjx"= 1, "2rjy"= 1, "3prj"= [1, 3], "4ecj"= 1, "5seg"= 1].
```

The first segment has length 1, then there is a revolute joint around the x axis, followed by another segment of length 1 and a revolute joint around the y axis. a prismatic joint with minimal length 1 and maximal length 3 is placed on the

Type	**Name**	**Parameters**
`rjx`	revolute joint around x	d_{i-1}
`rjy`	revolute joint around y	d_{i-1}
`rjz`	revolute joint around z	d_{i-1}
`prj`	prismatic joint	max_{length} or $[min, max]$
`baj`	Balljoint	d_{i-1}
`ecj`	Euler joint	d_{i-1}
`rpy`	Roll-pitch-yaw	d_{i-1}
`cyj`	Cylindrical joint	$[rad, min, max]$
`seq`	Segment	x or $[x, y, x]$

Table 1: Types of Joints

last revolute joint. Afterwards we have a segment of length 1 with a euler joint. Finally there is a segment of length 1 which represents the robot hand.

After we have defined the robot arm we model the robot arm with the command:

```
B := RS:-modelrobot(j).
```

As result we get the Gröbner basis for the robot arm.

During the computation interesting information about the robot arm is printed to the Maple worksheet. The output for the Standford manipulator you can find in Figure 10.

Let us explain some features of this output. First you get the trigonometric formulas for the hand position. Then the occurring angles are listed. The formuals get transformed into a polynomial form. Together with the restrictions they form the input basis for the Gröbner basis calculations. Therefore we also need a polynomial order. Then the Gröbner basis is printed out.

"MODELLING of the robot arm"

["1rjx" = 1, "2rjy" = 1, "3prj" = [1, 3], "4ecj" = 1, "5seg" = 1]

"Equations for a point (x,y,z):", [[$\cos(\gamma_2)\cos(\beta_4) - \sin(\gamma_2)\cos(\alpha_4)\sin(\beta_4) + 2 + \cos(\gamma_2) + \cos(\gamma_2)\, l_3 - x$],
[$-\sin(\gamma_1)\sin(\gamma_2)\cos(\beta_4) - \cos(\gamma_1)\sin(\alpha_4)\sin(\beta_4) - \sin(\gamma_1)\cos(\gamma_2)\cos(\alpha_4)\sin(\beta_4) - \sin(\gamma_1)\sin(\gamma_2) - \sin(\gamma_1)\sin(\gamma_2)\, l_3 - y$],
[$\cos(\gamma_1)\sin(\gamma_2)\cos(\beta_4) - \sin(\gamma_1)\sin(\alpha_4)\sin(\beta_4) + \cos(\gamma_1)\cos(\gamma_2)\cos(\alpha_4)\sin(\beta_4) + \cos(\gamma_1)\sin(\gamma_2) + \cos(\gamma_1)\sin(\gamma_2)\, l_3 - z$],
[1]]

"CPU time:", 0.015

$[\gamma_1, \gamma_2, \alpha_4, \beta_4, \gamma_4], [\gamma_1, \gamma_2, \alpha_4, \beta_4]$

"Polynomial equations:",
$$\begin{bmatrix} -c_{\alpha_4} s_{\beta_4} s_{\gamma_2} + c_{\beta_4} c_{\gamma_2} + c_{\gamma_2} l_3 - x + c_{\gamma_2} + 2 \\ -c_{\alpha_4} c_{\gamma_2} s_{\beta_4} s_{\gamma_1} - c_{\beta_4} s_{\gamma_1} s_{\gamma_2} - c_{\gamma_1} s_{\alpha_4} s_{\beta_4} - l_3 s_{\gamma_1} s_{\gamma_2} - s_{\gamma_1} s_{\gamma_2} - y \\ c_{\alpha_4} c_{\gamma_1} c_{\gamma_2} s_{\beta_4} + c_{\beta_4} c_{\gamma_1} s_{\gamma_2} + c_{\gamma_1} l_3 s_{\gamma_2} - s_{\alpha_4} s_{\beta_4} s_{\gamma_1} + c_{\gamma_1} s_{\gamma_2} - z \\ 1 \end{bmatrix}$$

"Variables:", 9

"Restictions:", $\left[c_{\gamma_1}^2 + s_{\gamma_1}^2 - 1, c_{\gamma_2}^2 + s_{\gamma_2}^2 - 1, c_{\alpha_4}^2 + s_{\alpha_4}^2 - 1, c_{\beta_4}^2 + s_{\beta_4}^2 - 1\right], [1 \le l_3, l_3 \le 3]$

"CPU time:", 0.

"Basis:", $\left[c_{\gamma_1}^2 + s_{\gamma_1}^2 - 1, c_{\gamma_2}^2 + s_{\gamma_2}^2 - 1, c_{\alpha_4}^2 + s_{\alpha_4}^2 - 1, c_{\beta_4}^2 + s_{\beta_4}^2 - 1, -c_{\alpha_4} s_{\beta_4} s_{\gamma_2} + c_{\beta_4} c_{\gamma_2} + c_{\gamma_2} l_3 - x + c_{\gamma_2} + 2, -c_{\alpha_4} c_{\gamma_2} s_{\beta_4} s_{\gamma_1} - c_{\beta_4} s_{\gamma_1} s_{\gamma_2} - c_{\gamma_1} s_{\alpha_4} s_{\beta_4} - l_3 s_{\gamma_1} s_{\gamma_2} - s_{\gamma_1} s_{\gamma_2} - y, c_{\alpha_4} c_{\gamma_1} c_{\gamma_2} s_{\beta_4} + c_{\beta_4} c_{\gamma_1} s_{\gamma_2} + c_{\gamma_1} l_3 s_{\gamma_2} - s_{\alpha_4} s_{\beta_4} s_{\gamma_1} + c_{\gamma_1} s_{\gamma_2} - z\right]$, " Dim:", 7

"order: ", $plex\left(l_3, c_{\gamma_1}, s_{\gamma_1}, c_{\gamma_2}, s_{\gamma_2}, c_{\alpha_4}, s_{\alpha_4}, c_{\beta_4}, s_{\beta_4}\right)$

"Groebnerbasis:", $\left[c_{\beta_4}^2 + s_{\beta_4}^2 - 1, c_{\alpha_4}^2 + s_{\alpha_4}^2 - 1, (-2x + 4) s_{\gamma_2} c_{\alpha_4} s_{\beta_4} + s_{\alpha_4}^2 s_{\beta_4}^2 s_{\gamma_2}^2 - s_{\beta_4}^2 + (-x^2 - y^2 - z^2 + 4x - 4) s_{\gamma_2}^2 + y^2 + z^2, c_{\gamma_2}^2 + s_{\gamma_2}^2 - 1, y c_{\alpha_4} c_{\gamma_2} s_{\alpha_4}^2 s_{\beta_4}^3 + (x^2 y - y^3 - y z^2 - 4xy + 4y) c_{\gamma_2} c_{\alpha_4} s_{\beta_4} + (-xy + 2y) c_{\gamma_2} s_{\alpha_4}^2 s_{\beta_4}^2 s_{\gamma_2} + (x^3 y + x y^3 + x y z^2 - 6x^2 y - 2y^3 - 2yz^2 + 12xy - 8y) c_{\gamma_2} s_{\gamma_2} + z s_{\alpha_4}^3 s_{\beta_4}^3 + (y^2 + z^2) s_{\alpha_4}^2 s_{\beta_4}^2 s_{\gamma_1} - z s_{\alpha_4} s_{\beta_4}^3 + (x^2 z - 4xz + 4z) s_{\alpha_4} s_{\beta_4} + (-y^2 - z^2) s_{\gamma_1} s_{\beta_4}^2 + (x^2 y^2 + x^2 z^2 - 4xy^2 - 4xz^2 + 4y^2 + 4z^2) s_{\gamma_1}, z c_{\alpha_4} s_{\alpha_4} s_{\beta_4}^2 + (y^2 + z^2) c_{\alpha_4} s_{\beta_4} s_{\gamma_1} - y c_{\gamma_2} s_{\alpha_4}^2 s_{\beta_4}^2 + (y^3 + y z^2) c_{\gamma_2} + (xz - 2z) s_{\alpha_4} s_{\beta_4} s_{\gamma_2} + (xy^2 + xz^2 - 2y^2 - 2z^2) s_{\gamma_1} s_{\gamma_2}, y c_{\alpha_4} s_{\beta_4} + z c_{\gamma_2} s_{\alpha_4} s_{\beta_4} + (y^2 + z^2) s_{\gamma_1} c_{\gamma_2} + (xy - 2y) s_{\gamma_2}, s_{\alpha_4}^2 s_{\beta_4}^2 + 2z s_{\alpha_4} s_{\beta_4} s_{\gamma_1} + (y^2 + z^2) s_{\gamma_1}^2 - y^2, y c_{\gamma_1} + z s_{\gamma_1} + s_{\alpha_4} s_{\beta_4}, -z c_{\alpha_4} s_{\alpha_4} s_{\beta_4}^2 + (-y^2 - z^2) c_{\alpha_4} s_{\beta_4} s_{\gamma_1} + (xy - 2y) c_{\beta_4} + y c_{\gamma_2} s_{\alpha_4}^2 s_{\beta_4}^2 + (-x^2 y - y^3 - y z^2 + 4xy - 4y) c_{\gamma_2} + (xy - 2y) l_3 + xy - 2y\right]$, " Dim:", 10

"Leading terms:", $\left[c_{\beta_4}^2, c_{\alpha_4}^2, s_{\alpha_4}^2 s_{\beta_4}^2 s_{\gamma_2}^2, c_{\gamma_2}^2, s_{\alpha_4}^2 s_{\beta_4}^2 s_{\gamma_1}, s_{\gamma_1} s_{\gamma_2}, s_{\gamma_1} c_{\gamma_2}, s_{\gamma_1}^2, c_{\gamma_1}, l_3\right]$

"flexible solutions:", *true*

"CPU time:", 0.702

Figure 10: Maple output of `modelrobot(j)`

The leading terms of the Gröbner basis show if the system has flexible solutions. We measure the CPU time which is needed for the three main steps: setup of a model, polynomial form of the model, Gröbner basis calculation.

Finally we start the robot simulation for a fixed hand position (x, y, z) in the 3-dimensional space.

```
erg := RS:-simulateposition(3, 2, 2)
```

This command opens the simulation window which is a Maple maplet.

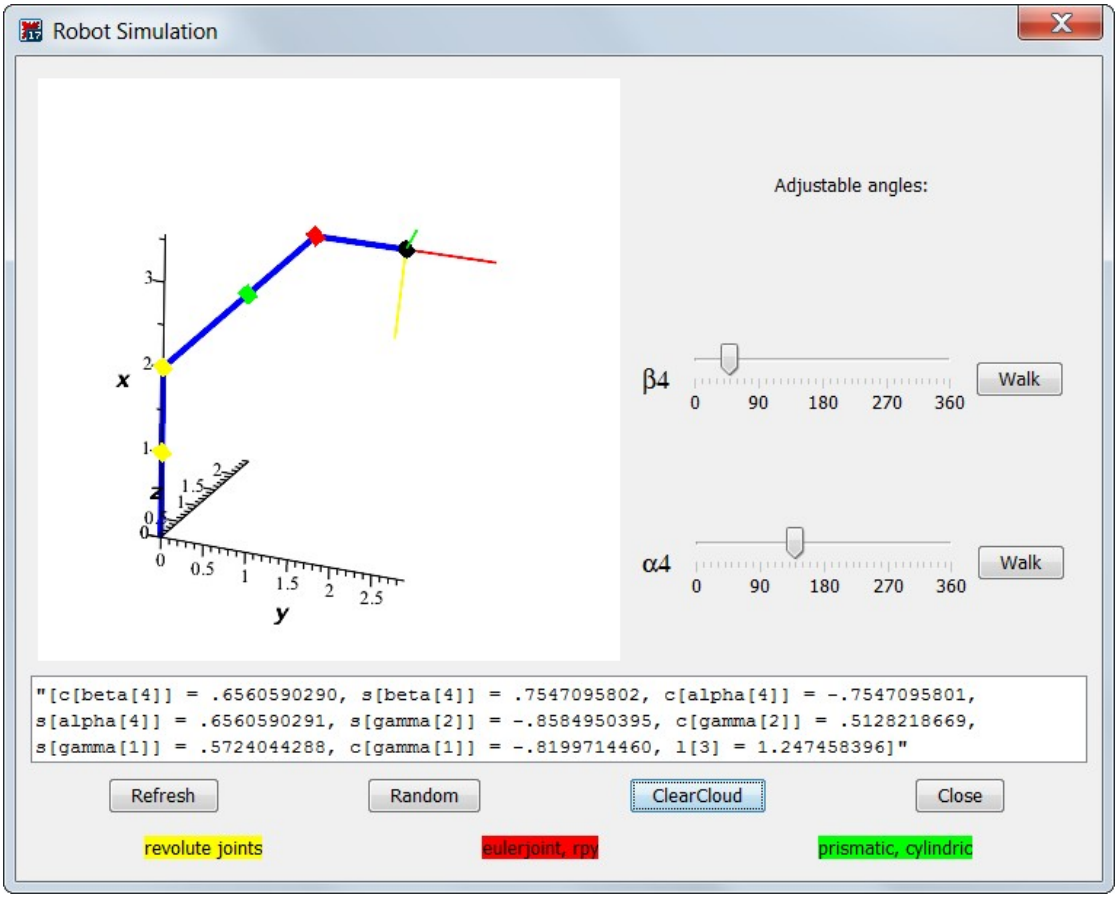


Figure 11: Simulation start

On the left side a 3-dimensional plot of the robot arm is placed. Segments are represented though blue lines. Joints are points of different color. The

color shows the kind of the joint. Yellow joints are revolute joints, red joints are euler, ball or roll-pitch-yaw joints, green joints are prismatic or cylindrical joints. The hand position is the black point at the end of the robot arm. In the robot hand the orientation system is located, the red line corresponds to the local x-axis, the green one to the y- axis and the yellow one to the z-axis.

On the right side you find the sliders for the free angles. With them you can move the robot arm. The button walk walks through the whole circle for the given angle, in doing so the other angles are fixed. Of course, as we can see in Figure 12 and Figure, just the valid solutions are drawn. 13.

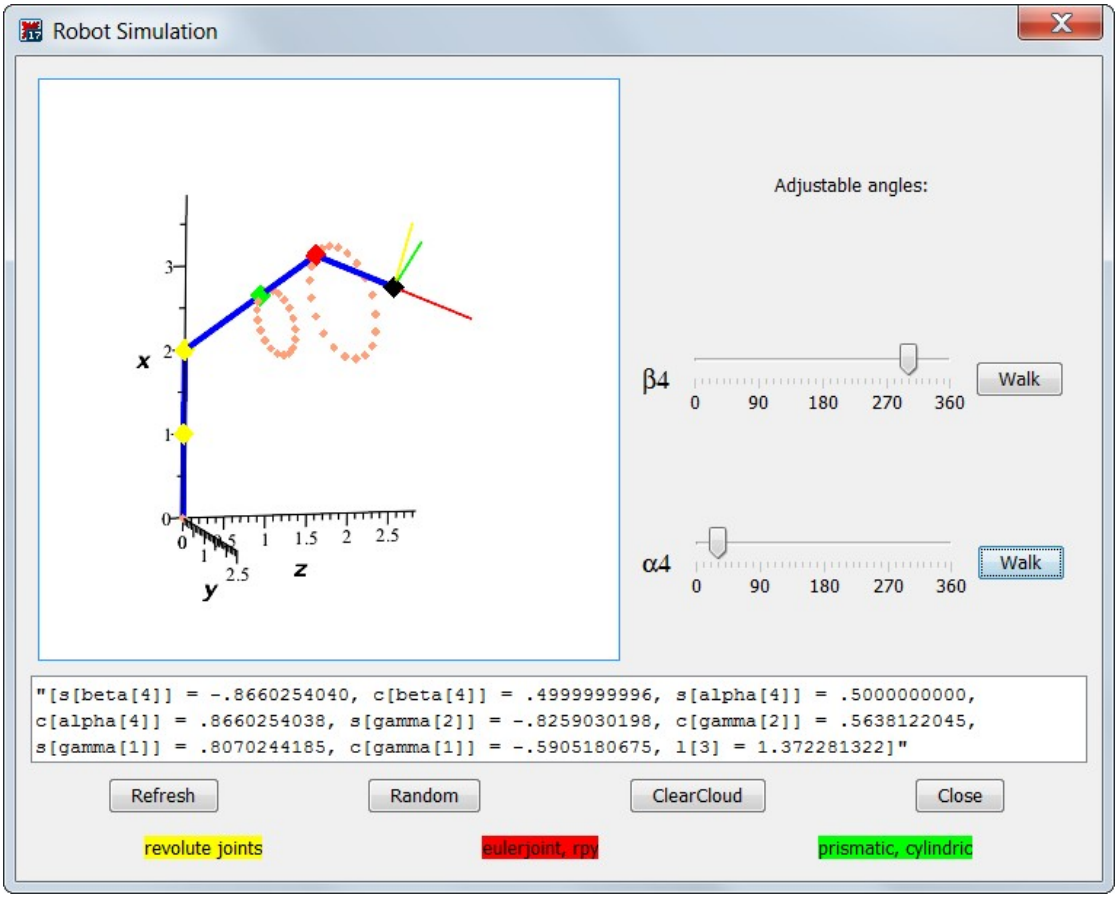


Figure 12: Walk α_4

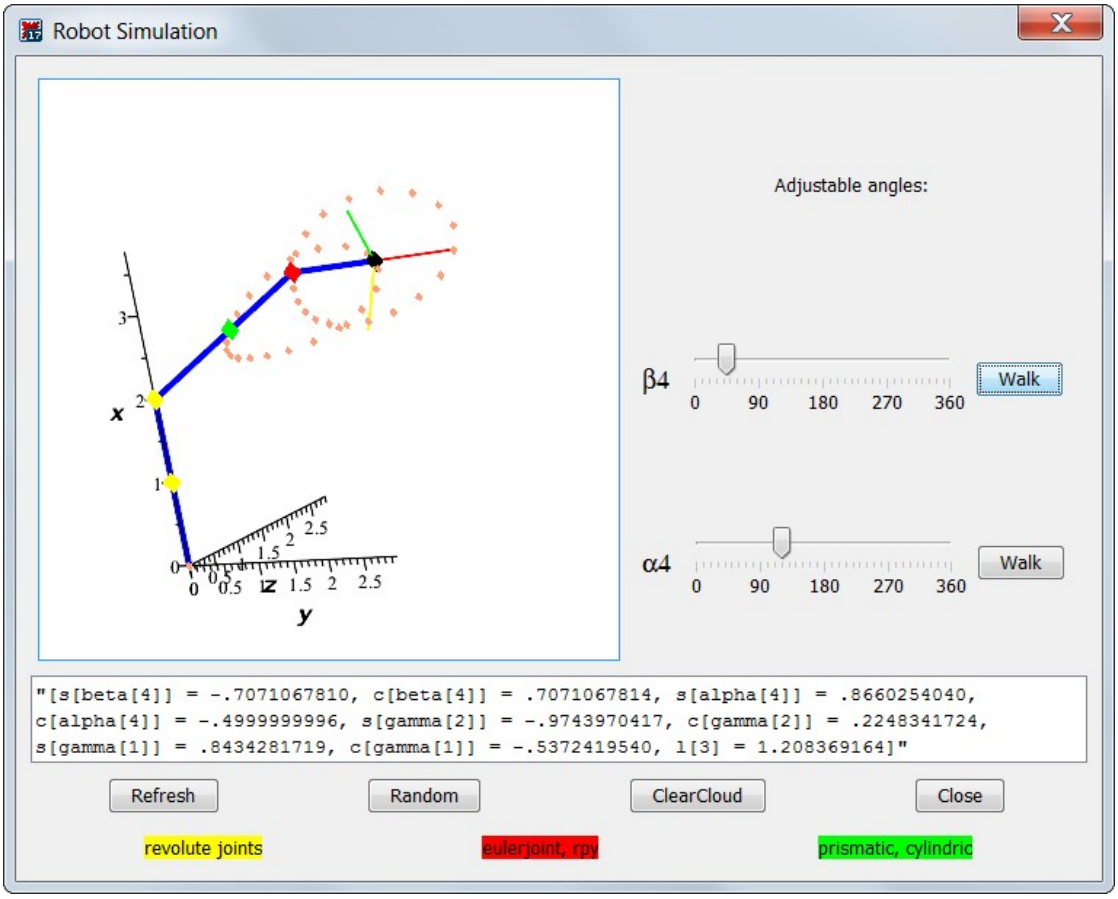


Figure 13: Walk β_4

On the bottom you first have a text field with the current solution in it. Then there are 4 Buttons. "Refresh" redraws the robot arm. "Random" calculates and draws a random configuration for the given robot arm. "ClearCloud" removes the point cloud of previous solutions. And "Close" of course closes the simulation. At the end there is a legend for the joint colors.

As mentioned there are some hand positions where the specialization of the Gröbner basis is not valid. One such point is $(0, 0, 0)$. Thus, for `erg := RS:-simulateposition(0, 0, 0)` we get a corresponding message in the maple worksheet (Figure 14).

"SIMULATION"

"FIND SOLUTION"

0, 0, 0

"Parametric gröbnerbasis failed! Solution incomplete:", $\left[s_{\beta_4} = 0.8660254040,\ c_{\beta_4} = -0.4999999996,\ s_{\alpha_4} = 0.2588190451,\ c_{\alpha_4} = -0.9659258263,\ s_{\gamma_2} = -0.4235783462 - 0.1023052420\ I,\ c_{\gamma_2} = -0.9128533651 + 0.04747124442\ I\right]$

"Basis:", $\left[c_{\gamma_1}^2 + s_{\gamma_1}^2 - 1,\ c_{\gamma_2}^2 + s_{\gamma_2}^2 - 1,\ c_{\alpha_4}^2 + s_{\alpha_4}^2 - 1,\ c_{\beta_4}^2 + s_{\beta_4}^2 - 1,\ -c_{\alpha_4} s_{\beta_4} s_{\gamma_2} + c_{\beta_4} c_{\gamma_2} + c_{\gamma_2} l_3 + c_{\gamma_2} + 2,\ -c_{\alpha_4} c_{\gamma_2} s_{\beta_4} s_{\gamma_1} - c_{\beta_4} s_{\gamma_1} s_{\gamma_2} - c_{\gamma_1} s_{\alpha_4} s_{\beta_4} - l_3 s_{\gamma_1} s_{\gamma_2} - s_{\gamma_1} s_{\gamma_2},\ c_{\alpha_4} c_{\gamma_1} c_{\gamma_2} s_{\beta_4} + c_{\beta_4} c_{\gamma_1} s_{\gamma_2} + c_{\gamma_1} l_3 s_{\gamma_2} - s_{\alpha_4} s_{\beta_4} s_{\gamma_1} + c_{\gamma_1} s_{\gamma_2}\right]$, " Dim:", 7

"order: ", $plex\left(l_3, c_{\gamma_1}, s_{\gamma_1}, c_{\gamma_2}, s_{\gamma_2}, c_{\alpha_4}, s_{\alpha_4}, c_{\beta_4}, s_{\beta_4}\right)$

"Groebnerbasis:", $\left[c_{\beta_4}^2 + s_{\beta_4}^2 - 1,\ s_{\alpha_4} s_{\beta_4},\ c_{\alpha_4}^2 + s_{\alpha_4}^2 - 1,\ -c_{\alpha_4} s_{\beta_4} + 2 s_{\gamma_2},\ 4 c_{\gamma_2}^2 + s_{\beta_4}^2 - 4,\ c_{\gamma_1}^2 + s_{\gamma_1}^2 - 1,\ l_3 + 1 + c_{\beta_4} + 2 c_{\gamma_2}\right]$, " Dim:", 7

"Leading terms:", $\left[c_{\beta_4}^2, s_{\alpha_4} s_{\beta_4}, c_{\alpha_4}^2, s_{\gamma_2}, c_{\gamma_2}^2, c_{\gamma_1}^2, l_3\right]$

"flexible solutions:", *true*

"free variables:", $\left[s_{\beta_4}, s_{\alpha_4}, s_{\gamma_1}\right]$

$\left[s_{\beta_4} = 0.,\ c_{\beta_4} = -1.,\ s_{\alpha_4} = 0.8660254040,\ c_{\alpha_4} = 0.4999999996,\ s_{\gamma_2} = 0,\ c_{\gamma_2} = -1.,\ s_{\gamma_1} = 0.8660254040,\ c_{\gamma_1} = -0.4999999996,\ l_3 = 2.\right]$

Figure 14: Maple Output $(0, 0, 0)$

The program notices that we do not have a complete solution, so the specialisation does not deliver a Gröbner basis. Thus the program substitutes the hand position $(0, 0, 0)$ into the original basis and calculates a valid Gröbner basis.

In the simulation we do not notice anything of that.

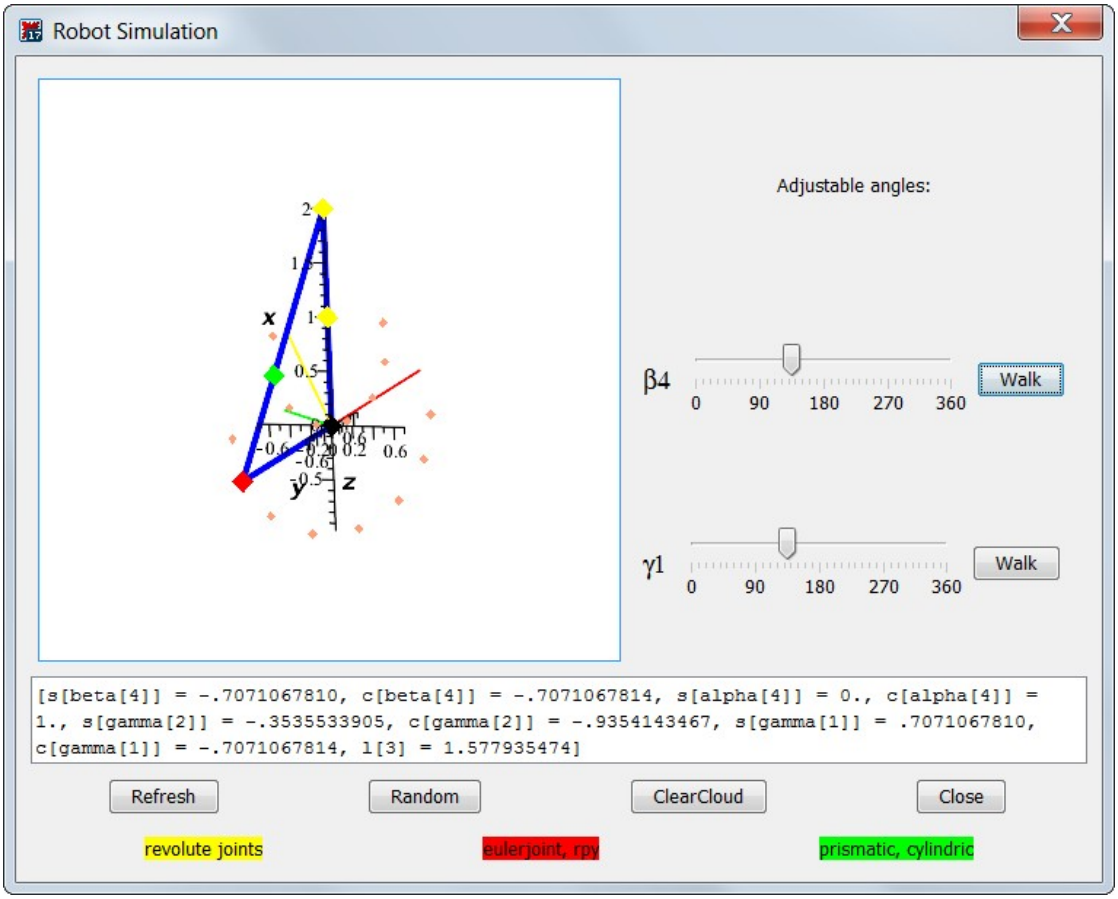


Figure 15: Hand position $(0, 0, 0)$

10.2. Further examples

So far we did all calculations and simulations on the Stanford manipulator, but of course, the program also works for other robot arms. In this subsection we give two additional examples.

For example the input

```
joints := ["1baj" = 1, "2baj" = 2, "3seg" = 2];
B := RS:-modelrobot(joints):
erg := RS:-simulateposition(3, 2, 2):
```

delivers a Gröbner basis with 7 polynomials in 8 variables of maximal degree 4. The robot arm has one free variable. The calculations of this robot arm take about 2 seconds.

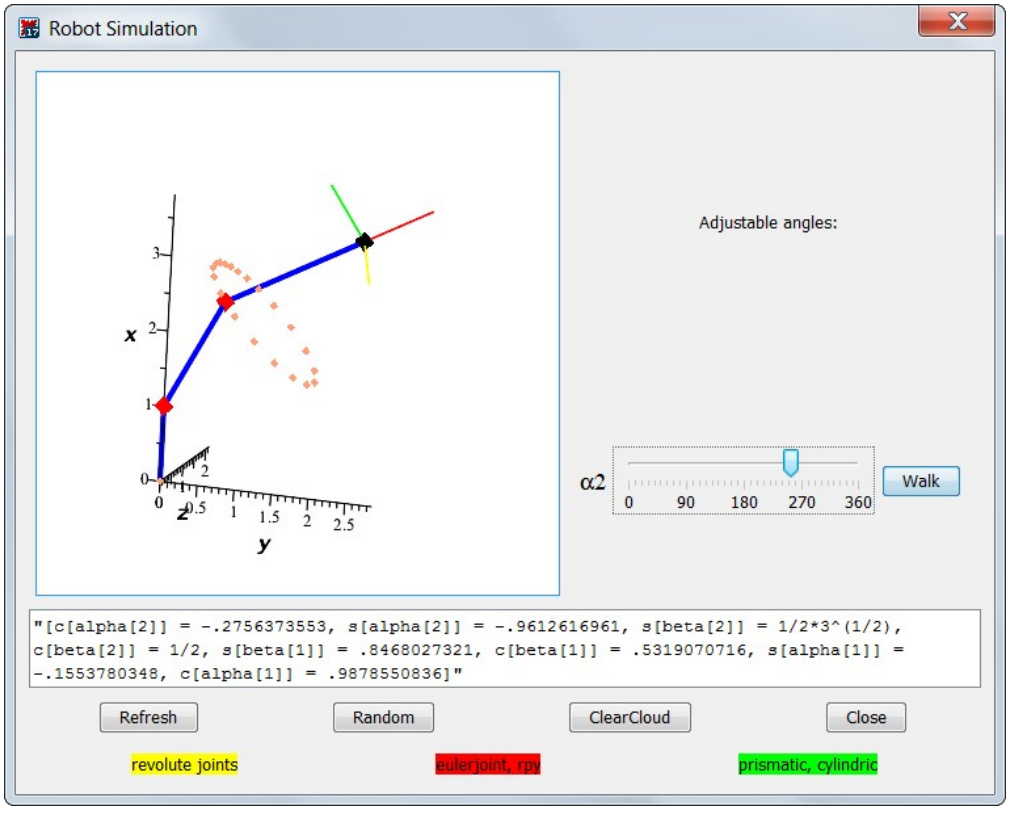


Figure 16: Hand position $(3, 2, 2)$

The input

```
joints := ["1rjx"=0, "2rjy"=1, "3rjy"=1, "4ecj"=1, "5seg"=1];
B := RS:-modelrobot(joints):
erg := RS:-simulateposition(1, 2, 1):
```

delivers a Gröbner basis with 16 polynomials in 10 variables of maximal degree 12. The system has two free variables. The calculations of this robot arm take about 80 seconds.

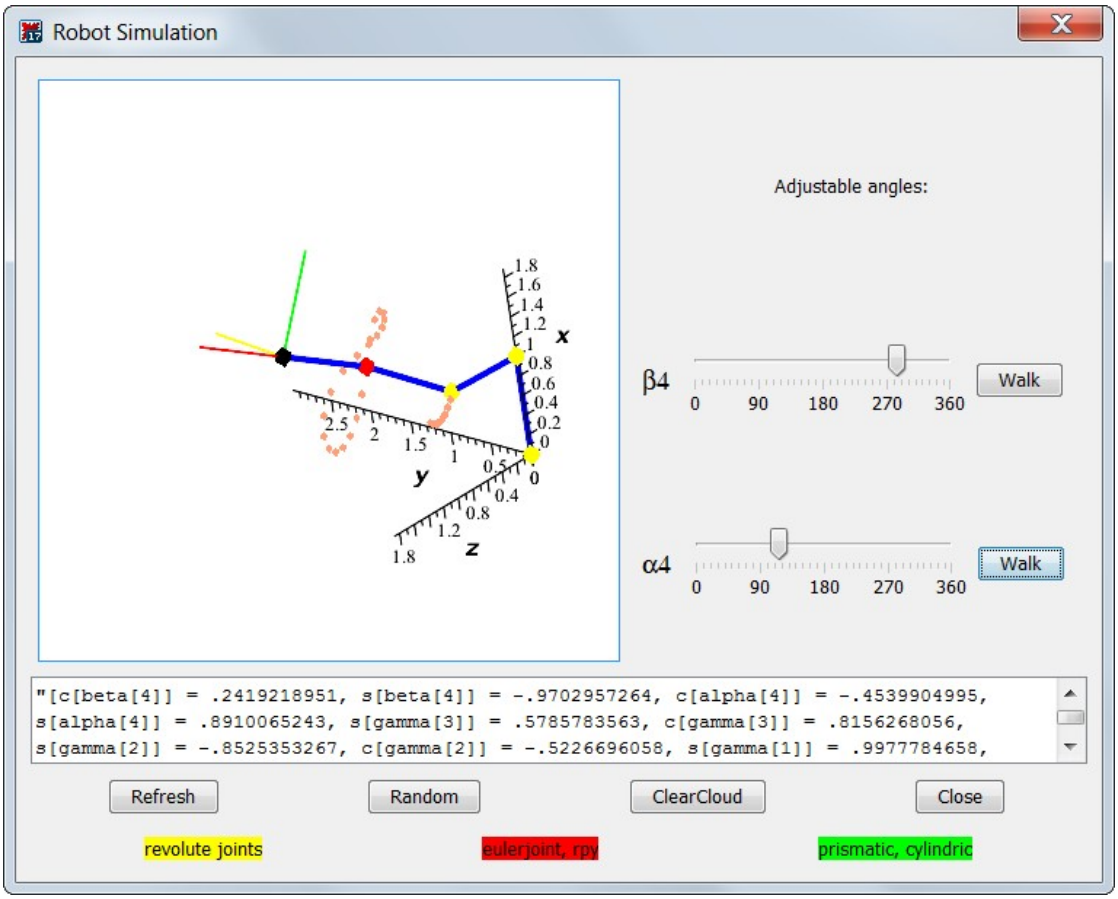


Figure 17: Hand position $(1, 2, 1)$

11. Conclusion and Outlook

The theory of part one is well-known and therefore nothing new. The two books of Cox, Little and O'Shea [CLO96] and [CLO04] give a good introduction to the theories we need for our applications.

The modelling of the kinematics of a robot arm in part two of the thesis is also known. Basics to modelling robot arms you can find in the book Robot manipulators by Paul [Pau86].

We see that every joint in the 3-dimensional space can be represented by a combination of the rotations around the axes and translations by an arbitrary vector. Thus, every joint can be described by a so called transformation matrix. By multiplying these transformation matrices in the correct order we get trigonometric equations for the hand position.

Because we want to solve the problem with algebraic methods we have to bring the equations into polynomial form. We do this in two different ways: the unit circle approach and the tangent half-angle substitution. Both methods work rather fine but have different advantages. We observe that the unit circle approach performs better with Gröbner bases and the tangent half-angle substitution is the better alternative for resultants.

For transforming systems of polynomial equations into a suitable form for solving, systems of polynomial equations, we consider two different approaches, Gröbner bases and resultants. We noticed that resultants are a bit faster in the elimination step but have a complicated extension step. Gröbner bases on the other hand have a rather simple extension step. Thus, for the purpose of simulating a robot arm, where the extension step has to be done very often, Gröbner bases fit better.

When we use Gröbner bases we first calculate a parametric Gröbner basis and specialize it in a later step. In this specialization we may lose the Gröbner

property of the basis. Thus, we have to check if we still have a Gröbner basis after specialization and take actions if not.

If we finally get a solution we still have to check if it is valid, otherwise we have to calculate another one.

Thus in this thesis we put together the different theories of elimination theory, robotic modelling, etc. to develop a simulation of the kinematics of a robot arm in a computer algebra system. We chose Maple to do the calculations and Maplets for the graphical user interface. The most difficult part of the thesis was the application of the elimination theory on the ideals of robotic arms. There are many different ways you can do that, to find a feasible way and spot the unusable ones took some time.

Outlook

In this thesis we have built a solid basis to model the kinematics of a robot arm with algebraic methods but one can go into more detail at some issues.

This thesis considers a semantic model of an robot arm. To simulate a more realistic robot arm one can extend the mathematical models. For instance, real robots are affected by physical forces, this forces can be included into the mathematical models. Some joints can not rotate a whole circle around their links, so one also can add limits to the joints.

In the real world there may exist barriers in the field of motion. One may can study how to handle such barriers.

One also can try to find a simpler way to use resultants, more precisely develop a practical method to do the extension step with resultants. Maybe there are methods to use resultants on the unit circle approach or other methods to get a polynomial systems.

Another issue, which this thesis just touched on the surface, are parametric Gröbner bases. Maybe the calculation of comprehensive Gröbner bases or

Gröbner systems leads to better performance in the simulation of the robot arm.

A. Source Code

Listing 1: Robot Simulation

```
# RS.mpl
# Robot Simulation
# (C) 2014 Michael Steglehner

# Usage:
# RS:-modelrobot(joints)
  # joints # list of equations of the form "ijjj" = parameters,
         # each equation represent a joint in the robot arm
     # where i is the number of the joint and jjj the type of the joint
  # Return a general GröbnerBasis for the given Robot arm

# RS:-simulateposition(x,y,z)
  # x,y,z # substitute (x,y,z) for the handposition in the GB from modelrobot()
         # and start a maplet which simulates the robot arm.
  # Returns the last solution of the simulation.

# RS:-getvarlist()/RS:-getorientation()/RS:-gettermordering()
  # Return the properties of the actual robot arm
#########################################################################

RS := module()
  # export functions
  export modelrobot, simulateposition, getvarlist, getorientation, gettermordering;
  # function which work with the maplet
  global findnewsolution, move,drawArm,myrandom,clearPlot,walk;
  #local functions
  local makeeqsystem, getpolynomialsystem, solvesystem,
       checkflexiblesolutions,findsolution, checksol,
       getpointlist, buildmaplet, calculateangles, rotx,roty,rotz,trans;
  # jointtypes/jointmatrices
  local baj, rjx, rjy, rjz, ecj, cyj, prj, rpy;
  # module variabels
  local anglelist, sigangle, E, F, B, gB, tord, tempB, operationradius,
       handpos, OrientationVector, varlist, freevars , joints, cursol:=[],
       restrictions, limitrestrictions, polymethod, checkcounter,
       oldpoints:=[], jointcolor;

# used maple packages
with(Groebner);
with(plottools);
with(plots);
with(Maplets[Elements]);
with(Maplets[Tools]);

# translation by an vector (x,y,z)
trans:=proc(x,y,z)
      Matrix([[1,0,0,x],[0,1,0,y],[0,0,1,z],[0,0,0,1]]);
end proc;
# rotation around x-axis by alpha
rotx:=proc( alpha)
     Matrix([[1,0,0,0],[0,cos(alpha),-sin(alpha),0],[0,sin(alpha),cos(alpha),0],[0,0,0,1]]);
end proc;
#rotation around y-axis by alpha
roty:=proc( alpha)
     Matrix([[cos(alpha),0,-sin(alpha),0],[0,1,0,0],[sin(alpha),0,cos(alpha),0],[0,0,0,1]]);
end proc;
```

```
# rotation around z-axis by alpha
rotz:=proc( alpha)
     Matrix([[cos(alpha),-sin(alpha),0,0],[sin(alpha),cos(alpha),0,0],[0,0,1,0],[0,0,0,1]]);
end proc;

baj := trans(l,0,0).rotx(alpha).roty(beta); #balljoint
rjx:=trans(l,0,0).rotx(gamma);            #revolute joint around x
rjy:=trans(l,0,0).roty(gamma);            #revolute joint around y
rjz:=trans(l,0,0).rotz(gamma);            #revolute joint around z
ecj:=trans(l,0,0).rotx(alpha).roty(beta).rotx(gamma); #euler corridante joint
prj:= trans(d,0,0);                       #prismatic joint
cyj := trans(d,0,0).rotx(alpha).trans(0,0,r); #cylindircal joint
rpy :=trans(l,0,0).rotx(alpha).roty(beta).rotz(gamma); #roll pitch yaw joint

#make a general equation system, transform it to a polynomial one and apply GB to it
modelrobot:=proc( jointlist::list,{polymeth::string :="unitcircle"})
  local st,a,indE;
  st := time():
  sigangle:=[]; cursol:=[];
  polymethod:= polymeth;
  joints:=jointlist;
  print("MODELLING of the robot arm");
  E:= makeeqsystem(joints);
  print("CPU time:", evalf(time()-st));
  st:=time();
  indE:= indets(E);
  for a in anglelist do
    if a in indE then
         sigangle:= [op(sigangle),a];
    end if;
  end do;
  print("All angles/Position significant angles",anglelist, sigangle);
  E:= getpolynomialsystem(E);
  F:= [op(restrictions), E[1],E[2],E[3]];
  print("CPU time:" , evalf(time()-st));
  st:=time();
  gB:= solvesystem(F);
  print("CPU time:" , evalf(time()-st));
  gB;
end proc;


getvarlist:=proc() varlist end proc;

gettermordering:=proc() tord end proc;

getorientation:= proc()
   if nops(cursol) = 0 then return OrientationVector; end if;
   subs(calculateangles(),OrientationVector)[[1..3],[1..3]];
end proc;

# make equations out of the joint list
makeeqsystem:=proc( joints::list )
    local A:= Matrix(4, shape = identity),joint, para, O,tempE, jointstr, jnr;
    anglelist:=[];
    varlist:=[];
    restrictions:=[];
    limitrestrictions:=[];
    operationradius:=0;
    O:= A;
    print(joints);
    for joint in joints do
```

```
119       jnr :=parse(lhs(joint)[1]);
120       jointstr := lhs(joint)[2..4];
121       para := rhs(joint);
122       if evalb( jointstr = "baj") then
123         A:=A.subs(alpha=alpha[jnr] ,beta= beta[jnr],l=para,baj);
124         O:=O.subs(alpha=alpha[jnr] ,beta= beta[jnr],l=0,baj);
125         anglelist:=[op(anglelist), alpha[jnr],beta[jnr]];
126         operationradius:= operationradius + para;
127       elif evalb( jointstr = "rjx") then
128          A:=A.subs(gamma=gamma[jnr] ,l=para,rjx);
129          O:=O.subs(gamma=gamma[jnr] ,l=0,rjx);
130          anglelist:=[op(anglelist), gamma[jnr]];
131          operationradius:= operationradius + para;
132       elif evalb( jointstr = "rjy") then
133          A:=A.subs(gamma=gamma[jnr] ,l=para,rjy);
134          O:=O.subs(gamma=gamma[jnr] ,l=0,rjy);
135          anglelist:=[op(anglelist), gamma[jnr]];
136          operationradius:= operationradius + para;
137       elif evalb( jointstr = "rjz") then
138          A:=A.subs(gamma=gamma[jnr] ,l=para,rjz);
139          O:=O.subs(gamma=gamma[jnr] ,l=0,rjz);
140          anglelist:=[op(anglelist), gamma[jnr]];
141          operationradius:= operationradius + para;
142       elif evalb( jointstr = "prj") then
143          A:=A.subs(d = l[jnr] ,prj);
144          varlist := [op(varlist), l[jnr]];
145          if nops(para) = 2 then
146            limitrestrictions:= [op(limitrestrictions), l[jnr] >=para[1], l[jnr]
                   <=para[2]];
147            operationradius:= operationradius + para[2];
148          elif nops(para) = 1 then
149            limitrestrictions:= [op(limitrestrictions),l[jnr] >=0, l[jnr] <=para];
150            operationradius:= operationradius + para;
151          else
152            limitrestrictions:= [op(limitrestrictions), l[jnr] >=0];
153          end if;
154       elif evalb( jointstr = "cyj") then
155          A:=A.subs(alpha=alpha[jnr] , d=l[jnr] ,r = para[1],cyj);
156          O:=O.subs(alpha=alpha[jnr] , d=0 ,r = 0,cyj);
157          anglelist:=[op(anglelist), gamma[jnr]];
158          varlist := [op(varlist), l[jnr]];
159          limitrestrictions:= [op(limitrestrictions), l[jnr] >=para[2], l[jnr]
                 <=para[3]];
160          operationradius:= operationradius + sqrt(para[3]^2+para[1]^2);
161       elif evalb( jointstr = "ecj") then
162          A:=A.subs(alpha=alpha[jnr],beta= beta[jnr], gamma=gamma[jnr] ,l=para,ecj);
163          O:=O.subs(alpha=alpha[jnr],beta= beta[jnr], gamma=gamma[jnr] ,l=0,ecj);
164          anglelist:=[op(anglelist),alpha[jnr], beta[jnr], gamma[jnr]];
165          operationradius:= operationradius + para;
166       elif evalb( jointstr = "rpy") then
167          A:=A.subs(alpha=alpha[jnr],beta= beta[jnr], gamma=gamma[jnr] ,l=para,rpy);
168          O:=O.subs(alpha=alpha[jnr],beta= beta[jnr], gamma=gamma[jnr] ,l=0,rpy);
169          anglelist:=[op(anglelist), alpha[jnr], beta[jnr], gamma[jnr]];
170          operationradius:= operationradius + para;
171       elif evalb( jointstr = "seg") then
172          if nops(para) = 3 then
173            A:=A.trans(para[1],para[2],para[3]);
174            operationradius:= operationradius + sqrt(para[1]^2+ para[2]^2+para[3]^2);
175          elif nops(para) = 1 then
176            A:=A.trans(para,0,0);
177            operationradius:= operationradius + para;
178          end if;
```

```
179       else
180         print("Incorrect jointtype:",jnr, jointstr);
181       end if;
182    end do;
183    tempE:=A.Vector([0,0,0,1])-Vector([x,y,z,0]);
184    OrientationVector:=O;
185    print("Equations for a point (x,y,z):" ,tempE);
186    tempE;
187 end proc;
188
189 # make a polynomial system out of the general equations system
190 getpolynomialsystem := proc(paraE)
191    local lcd:=1,a, tempE:= paraE,i;
192    if evalb(polymethod="unitcircle") then
193        for a in sigangle do
194          varlist :=[op(varlist), c[a], s[a]];
195          tempE:=subs( {cos(a)=c[a],sin(a)=s[a]},tempE);
196          restrictions:=[op(restrictions), c[a]^(2)+s[a]^(2)-1];
197        end do;
198    elif evalb(polymethod="halfangle") then
199        for a in sigangle do
200         varlist :=[op(varlist), u[a]];
201         tempE:=subs(
              {cos(a)=(1-u[a]^(2))/(1+u[a]^(2)),sin(a)=(2*u[a])/(1+u[a]^(2))},tempE);
202        end do;
203        print(tempE);
204        for i to nops(tempE) do
205          tempE[i] := simplify(evala(tempE[i]*denom(tempE[i])));
206        end do;
207    else
208       print("Incorrect polymethod", polymethod);
209    end if;
210    print("Polynomial equations:", tempE);
211    print("Variables:" ,nops(varlist));
212    print("Restictions:", restrictions , limitrestrictions);
213    tempE;
214 end proc;
215
216 # apply the buchbergers algorithm to the polynomial system
217 solvesystem:=proc(paraF)
218     local v,tempB:=paraF,i;
219     print("Basis:" , paraF, " Number of Polynomials:", nops(paraF));
220     tord := plex( op(varlist));
221     print("order: ",tord);
222     try
223       tempB:=timelimit(600, Groebner:-Basis(paraF,tord));
224     catch "time expired":
225       print("The calculation take to much time");
226       return[];
227     end try;
228     print("Groebnerbasis: " , tempB, " Number of Polynomials:", nops(tempB));
229     print("Leading terms: ", LeadingMonomial(tempB, tord));
230     print("Solutions: ", checkflexiblesolutions(tempB));
231     tempB
232 end proc;
233
234 checkflexiblesolutions:= proc(paraB)
235    local leadingterms, check,v,l,ind;
236    if paraB[1] = 1 then return "No solutions!"; end if;
237    leadingterms:=LeadingMonomial(paraB, tord);
238    for v in varlist do
239      check:= false;
```

```
240     for l in leadingterms do
241        ind:= indets(l);
242        if nops(ind)= 1 and v = ind[1] then
243           check:= true;
244           break;
245        end if;
246     end do;
247     if check= false then
248        return "Flexible solutions!";
249     end if;
250    end do;
251    return "Isolated solutions!";
252 end proc;
253
254 # find a random start configuration of the arm with the fixed hand at (x1,y1,z1)
255 simulateposition:= proc(x1::numeric,y1::numeric,z1::numeric)
256      local sol,i,st,mymaplet,s;
257      if(polymethod="halfangle") then
258        print("Simulation for halfangle substitution is in process!");
259        return [];
260      end if;
261      print("SIMULATION");
262      print("FIND SOLUTION");
263      print(x1,y1,z1 );
264      freevars:=[]; oldpoints:=[]; cursol:=[];
265      handpos:= [x=x1, y=y1, z=z1];
266      B:= subs(x=x1,y=y1, z=z1, gB);
267      if operationradius < evalf(sqrt(x1^2+ y1^2+z1^2)) then
268        print("Point:", [x1,y1,z1], "out of robot arm range!");
269        return [];
270      end if;
271      sol:=findsolution();
272      if sol = [] then
273         print("no solution found");
274         return [];
275      end if;
276      print("free variables:", freevars);
277      print(sol);
278      print(evalf(subs(op(sol),E)));
279      Maplets[Display](buildmaplet());
280      cursol;
281   end proc;
282
283 #find a solution for the commited GB
284 findsolution:=proc()
285     local sol, check:=false,idvar,idnr, v,i,val,erg;
286     checkcounter:=0;
287     while not check do
288         tempB:=B;
289         sol:=[]; freevars:=[];
290         if evalb(checkcounter >= 50) then
291           return sol;
292         end if;
293         for i from 1 to nops(tempB) do
294           idvar:= indets(tempB[i]);
295           idnr :=nops(idvar);
296           if idnr < 1 then next; end if;
297           if idnr > 1 then
298             for v in idvar[2..idnr] do
299               freevars:=[op(freevars),v];
300               if member(op(v), anglelist) and polymethod="unitcircle" then
301                 val:=evalf(cos((rand(24)()*(Pi))/(12)));
```

```
302                elif member(op(v), anglelist) and polymethod="halfangle" then
303                  val:=evalf(tan(((rand(24)()-12)*(Pi))/(24)));
304                else val:=rand(3)()+1;
305                end if;
306                tempB:= subs(v=val,tempB);
307                sol:= [op(sol), v=val];
308              end do;
309            end if;
310            erg:= {solve(tempB[i],idvar[1])};
311            val:= erg[rand(nops(erg))()+1];
312            tempB:= subs(idvar[1]=val,tempB);
313            sol:= [op(sol), idvar[1]=val];
314          end do;
315
316        if nops(sol) < nops(varlist) then
317          print("Parametric gröbnerbasis failed! Solution incomplete:", sol);
318          B:= solvesystem(subs(handpos,F));
319          if 1 in B then return []; end if;
320          tempB:= B;
321          next;
322        end if;
323        check := checksol(sol);
324      end do;
325      cursol:=sol;
326      sol;
327  end proc;
328
329  #evaluate if the committed solution is a real solution for our model
330  checksol:=proc(solution)
331      local sol:=solution,s,r,pE;
332      checkcounter:= checkcounter+1;
333      for s in sol do
334         if type(rhs(s),nonreal) then return false; end if;
335      end do;
336      r:=[op(subs(op(sol),restrictions)),op(subs(op(sol),limitrestrictions))];
337      for s in r do
338        if type(s,'`<=`') then
339           if not evalb(evalf(s)) then
340                return false;
341           end if;
342        elif type(s,numeric) then
343           if round(s*10^(6)) <>0 then
344                return false;
345           end if;
346        end if;
347      end do;
348      pE:=subs(handpos,subs(op(sol),E));
349      for s in pE[1..3] do
350        if evalb(evalf(abs(s)) > 10^(-6)) then
351                return false;
352        end if;
353      end do;
354      true;
355  end proc;
356
357  # calculate the angles to the corresponding variables in current solution
358  calculateangles:= proc()
359    local s,c,x, v, val,a,curangles:=[],u;
360    if evalb(polymethod="unitcircle") then
361     for a in anglelist do
362       c:=1;s:=0;
363       for x in cursol do
```

```
364         v:= lhs(x);
365         if op(v) = a then
366           if convert(v,string)[1] = "c" then c:= rhs(x); end if;
367           if convert(v,string)[1] = "s" then s:= rhs(x); end if;
368         end if;
369       end do;
370       val := arccos(c);
371       if evalf(arcsin(s)) < 0 then val := 2*Pi - val; end if;
372       curangles :=[op(curangles), a = val]
373     end do;
374   elif evalb(polymethod="halfangle") then
375     for a in anglelist do
376       c:=1;s:=0;
377       for x in cursol do
378         v:= lhs(x);
379         if op(v) = a then
380           u := rhs(x);
381         end if;
382       end do;
383       val := arctan(u)*2 + Pi;
384       curangles :=[op(curangles), a = val]
385     end do;
386   end if;
387   curangles;
388 end proc;
389
390 # find a solution for the commited GB with substituted free variables
391 findnewsolution:=proc(dsol::list)
392   local sol, check:=false ,idvar,idnr,i,val,erg,p,pval,distance,e;
393   checkcounter:=0;
394   tempB:=subs(dsol, B);
395
396   while not check do
397       sol:=dsol;
398       tempB:= subs(op(sol), B);
399
400       for i from 1 to nops(tempB) do
401           idvar:= indets(tempB[i]);
402           idnr := nops(idvar);
403           pval:= 0;
404           if idnr < 1 then next; end if;
405           if idnr > 1 then
406             print("Error: The free variables are not defined",tempB[i]);
407             return "Free variables not bounded!";
408           end if;
409           erg:={solve(tempB[i],idvar[1])};
410           val := erg[rand(nops(erg))()+1];
411           for p in cursol do
412               if idvar[1] = lhs(p) then pval:= rhs(p); break; end if;
413           end do;
414           distance:=abs(val-pval);
415           for e in erg do
416             if evalf(abs(e-pval)) < evalf(distance) then
417               val:=e;
418               distance:=abs(e-pval);
419             end if;
420           end do;
421           tempB:=subs(idvar[1]=val,tempB);
422           sol:=[op(sol), idvar[1]=val];
423       end do;
424       check := checksol(sol);
425       checkcounter:= checkcounter+1;
```

```
426         if checkcounter > 20 then
427           print("Error: The new values are not valid!",sol);
428           sol := cursol;
429           return "No solution for these values!";
430         end if
431     end do;
432     cursol:=sol;
433 end proc;
434
435 clearPlot:=proc( )
436     oldpoints:=[];
437     drawArm();
438 end proc;
439
440 drawArm:=proc( )
441     local plist,i,Lines,Points,p,n,colorlist,hp,Or;
442     plist:=[[0,0,0]];
443     colorlist:= [];
444     Lines:={}; Points:={};
445     n:= nops(joints);
446     for i from 1 to n do
447       plist:=[op(plist), getpointlist(joints[1..i])];
448       colorlist:= [op(colorlist), jointcolor];
449     end do;
450     for p in oldpoints do
451        Points:={op(Points) ,point(p,color="LightSalmon",symbolsize=15)};
452     end do;
453     if nops(oldpoints) > n*25 then oldpoints:= oldpoints[1..n*25]; end if;
454     oldpoints:= [op(plist),op(oldpoints)];
455
456     for i from 1 to nops(plist)-1 do
457       Lines:={op(Lines),line(plist[i],plist[i+1],color=blue,thickness=4)} :
458       Points:={op(Points) ,point(plist[i+1],color=colorlist[i],symbolsize=35)}:
459     end do;
460
461     hp:= [rhs(handpos[1]),rhs(handpos[2]),rhs(handpos[3])];
462     Or:=evalf(getorientation());
463     Lines:={op(Lines),line(hp, hp + convert(Or[1..3,1],list),color=red),line(hp, hp +
          convert(Or[1..3,2],list),color=green),line(hp, hp +
          convert(Or[1..3,3],list),color=yellow)} :
464
465     display( (Lines union Points),axes=normal,scaling=constrained, axis=[thickness =0],
466       labels=[x,y,z], labelfont=[Courier,BOLD,13],
467       'orientation'=[-10, 160, -25], ambientlight=[1, 1, 1] );
468 end proc;
469
470
471
472 # do the forward kinematic problem for robotics
473 getpointlist:=proc(joints::list )
474     local A:=Matrix(4, shape = identity),joint,j,para,P,a,pv;
475
476     for joint in joints do
477       j:= lhs(joint);
478       para :=rhs(joint);
479       if evalb( j[2..4] = "baj") then
480         A:=A.subs(alpha=alpha[parse(j[1])],beta= beta[parse(j[1])],l=para,baj);
481         jointcolor:= 'red';
482       elif evalb( j[2..4] = "rjx") then
483         A:=A.subs(gamma=gamma[parse(j[1])],l=para,rjx);
484         jointcolor:= 'yellow';
485       elif evalb( j[2..4] = "rjy") then
```

```
        A:=A.subs(gamma=gamma[parse(j[1])],l=para,rjy);
        jointcolor:= 'yellow';
      elif evalb( j[2..4] = "rjz") then
        A:=A.subs(gamma=gamma[parse(j[1])],l=para,rjz);
        jointcolor:= 'yellow';
      elif evalb( j[2..4] = "prj") then
        A:=A.subs(d = l[parse(j[1])] ,prj);
        jointcolor:= 'green';
      elif evalb( j[2..4] = "cyj") then
        A:=A.subs(alpha=alpha[parse(j[1])], d=l[parse(j[1])] ,r = para[1],cyj);
        jointcolor:= 'green';
      elif evalb( j[2..4] = "ecj") then
        A:=A.subs(alpha=alpha[parse(j[1])],beta=beta[parse(j[1])],
           gamma=gamma[parse(j[1])] ,l=para,ecj);
        jointcolor:= 'red';
      elif evalb( j[2..4] = "rpy") then
        A:=A.subs(alpha=alpha[parse(j[1])],beta= beta[parse(j[1])],
            gamma=gamma[parse(j[1])] ,l=para,rpy);
        jointcolor:= 'red';
      elif evalb( j[2..4] = "seg") then
        if nops(para) = 3 then
            A:=A.trans(para[1],para[2],para[3]);
        elif nops(para) = 1 then
            A:=A.trans(para,0,0);
        end if;
        jointcolor:= 'black';
      end if;
    end do;
    P:=A.Vector([0,0,0,1]);
    if evalb(polymethod="unitcircle") then
      for a in sigangle do
        P:=subs( {cos(a)=c[a],sin(a)=s[a]},P);
      end do;
    elif evalb(polymethod="halfangle") then
      for a in sigangle do
        P :=subs( {cos(a)=(1-u[a]^(2))/(1+u[a]^(2)),sin(a)=(2*u[a])/(1+u[a]^(2))},P);
      end do;
    end if;
    pv:=evalf(subs(op(cursol),P));
    [pv[1],pv[2],pv[3]];
end proc;

# move the robot arm in the simulation if the free variables get modified
move:=proc()
   local sinval,sol,i, cosval, uval,a;
   sol:=[];
   if evalb(polymethod="unitcircle") then
     for i from 1 to nops(freevars) do
       sinval :=evalf(sin(Get(cat("SL",i)::numeric)*Pi/180));
       cosval :=evalf(cos(Get(cat("SL",i)::numeric)*Pi/180));
       a:= op(freevars[i]);
       sol:=[op(sol), c[a] = cosval, s[a] = sinval];
     end do;
   elif evalb(polymethod="halfangle") then
     for i from 1 to nops(freevars) do
       uval :=evalf(tan( (Get(cat("SL",i)::numeric)*Pi/180 -Pi) /2 ));
       sol:=[op(sol), u[op(freevars[i])] = uval];
     end do;
   end if;
   convert(findnewsolution(sol),string);
end proc;
```

```
548 #calculate a random solution and draw it in the simulation
549 myrandom:=proc()
550    local val,sol,i,s ;
551    sol:=findsolution();
552    print(sol);
553    if sol=[] then return -1; end if;
554    for i from 1 to nops(freevars) do
555       val:=0;
556       for s in calculateangles() do
557            if lhs(s)=op(freevars[i]) then
558              val:= round(rhs(s)*180/Pi);
559              break;
560            end if;
561       end do;
562       Set(cat("SL",i) = val);
563    end do;
564    return convert(sol,string);
565 end proc;
566
567 # create a curve of solution for one free variable
568 walk:=proc(fnr)
569    local val,sol,i ,cura, steps, walkvar,newsol, cosval, sinval,j,uval,a,walkdist;
570    steps:=20;
571    walkvar:= freevars[fnr];
572    sol:=[];
573    clearPlot():
574
575    if evalb(polymethod="unitcircle") then
576      for i from 1 to nops(freevars) do
577       sinval :=evalf(sin(Get(cat("SL",i)::numeric)*Pi/180));
578       cosval :=evalf(cos(Get(cat("SL",i)::numeric)*Pi/180));
579       a:= op(freevars[i]);
580       if (a <> op(walkvar)) then
581         sol:=[op(sol), c[a] = cosval, s[a] = sinval];
582       else
583         cura:= arccos(cosval);
584       end if;
585      end do;
586
587    elif evalb(polymethod="halfangle") then
588      for i from 1 to nops(freevars) do
589       uval :=evalf(tan( (Get(cat("SL",i)::numeric)*Pi/180 -Pi) /2 ));
590       if (op(freevars[i]) <> op(walkvar)) then
591         sol:=[op(sol), u[op(freevars[i])] = uval];
592       else
593          cura:= arctan(uval);
594       end if;
595      end do;
596    end if;
597
598    walkdist:= evalf(2*Pi/steps);
599    for j from 1 to steps do
600     cura:= (cura + walkdist);
601     if evalf(cura) > evalf(2*Pi) then cura:= cura-evalf(2*Pi); end if;
602     newsol:=[op(sol), s[op(walkvar)]= sin(cura),c[op(walkvar)]=cos(cura)];
603     findnewsolution(newsol);
604     Set(cat("SL",fnr)= round(cura*180/Pi));
605     Set('Plotter1' = drawArm());
606    end do;
607 end proc;
608
609 # build the simulation popup
```

```
610 buildmaplet:=proc()
611    local maplet3d,pl,sliderlist,s,a,wa,vl,i, actlist:=[],l,val::integer,
           colorleg,ang:=[];
612    pl:= Plotter['Plotter1']('value'=drawArm());
613    l:= TextBox['l1'](convert(cursol,string),height = 3);
614    actlist:=[Action['ref'](
615         Evaluate('function'="drawArm",'target'='Plotter1')),
616       Action['rand'](Evaluate('function'="myrandom",'target'='l1'),
617         Evaluate('function'="drawArm",'target'='Plotter1'))
618      ];
619    sliderlist:= [ Label("Adjustable angles:")];
620    for i from 1 to nops(freevars) do
621       val:=0;
622       for s in calculateangles() do
623            if lhs(s)=op(freevars[i]) then
624              val:= round(rhs(s)*180/Pi);
625              break;
626            end if;
627       end do;
628       s:=Slider[cat("SL",i)](0 .. 360, 'value'= val, 'majorticks' = 90,
629              'minorticks' = 10, 'snapticks' = 'false', 'filled' = true,
630              'showticks' = true, 'onchange'=cat("AC",i));
631       ang:=[op(ang), freevars[i]];
632       vl:=Label(cat(convert(op(freevars[i]),string)[1],op(op(freevars[i]))),
              font=Font("Symbol",18));
633       a:=Action[cat("AC",i)](Evaluate(function = "move",'target'='l1'),
634          Evaluate('function'="drawArm",`option`="value",'target'='Plotter1'));
635       wa:=Action[cat("Walk",i)](Evaluate(function = cat("walk(",i,")" )));
636       sliderlist:=[op(sliderlist), [vl, s,Button("Walk",'onclick'=cat("Walk",i))]];
637       actlist:=[op(actlist),a,wa];
638    end do;
639    colorleg := [ Label("revolute joints", background = yellow),
640              Label("eulerjoint, rpy", background = red),
641              Label("prismatic, cylindric", background = green)];
642    Maplet('onstartup' = RunWindow('W1'),Window['W1'](
643         'title'="Robot Simulation",[[pl,sliderlist], l,[Button("Refresh", 'onclick'='ref'),
644         Button("Random", 'onclick'='rand'),
645         Button("ClearCloud",Evaluate('function'="clearPlot",`option`="value",'target'='Plotter1')),
646         Button("Close",Shutdown([]))],colorleg ]),actlist);
647 end proc;
648 end module;
649
650 save(RS, "robotsimulation") ;
```

References

[Buc65] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, University of Innsbruck, Austira, 1965.

[CLO96] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties and Algorithms*. Library of Congress Cataloging-in-Publication. Springer-Verlag New York, Inc, 1996.

[CLO04] D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry*. Library of Congress Cataloging-in-Publication. Springer-Verlag New York, Inc, 2004.

[CSJD04] E. A. Coutsias, C. Seok, M. P. Jacobson, and K. A. Dill. A kinematic view of loop closure. *Journal of Computational Chemistry*, 25:510–528, 2004.

[GKZ94] I. M. Gelfand, M. M. Kapranov, and A. V. Zelevinsky. *Discriminants, Resultants and Multidimensional Determinants*. Birkhäuser, Boston, 1994.

[LC07] R. H. Lewis and E. A. Coutsias. Algorithmic search for flexibility using resultants of polynomial systems. Lecture Notes in Computer Science 4869:68–79, 2007.

[Mac02] F. S. Macaulay. On some formulas in elimination. *Proc. London Math. Soc.*, 3:3–27, 1902.

[MM09] A. Monte and M. Manubens. Minimal canonical comprehensive gröbnersystems. *Journal of Symbolic Computation*, 44:463–478, 2009.

[Pau86] R. Paul. *Robot manipulators: Mathematics, Programming and control*.

Library of Congress Cataloging-in-Publication. The Massachusetts Institute of Technology, 1986.

[Sax97] T. Saxena. *Efficient Variable Elimination Using Resultants*. PhD thesis, State University of New York at Albany, 1997. UMI Order No. GAX97-19914.

[Sch69] V.D. Scheinman. *Design of a Computer Controlled Manipulator*. Department of Mechanical Engineering, Stanford University., 1969.

[vdW67] B. L. van der Waerden. *Algebra II*. Springer-Verlag Berlin Heidelberg New York, 1967.

[Wei92] V. Weispfenning. Comprehensive gröbner bases. *Journal of Symbolic Computation*, 14:1–29, 1992.

[Win96] F. Winkler. *Polynomial Algorithms in Computer Algebra*. Springer-Verlag Wien New York, 1996.