Formal Foundations for Rigid Body Transformation

Reynald Affeldt Cyril Cohen

We are interested in the formal specification of safety properties of robot manipulators. To this end, we have been developing a formalization of rigid body transformations in the Coq proof-assistant using the Mathematical Components library. This paper provides an overview of the current status of our formalization. We start by providing theories for angles and for three-dimensional Euclidean geometry, including the cross-product. We use these theories to formalize the basic geometry required for robotics. First, we formalize a comprehensive theory of three-dimensional rotations, including exponentials of skew-symmetric matrices and quaternions. Then, we provide a formalization of rigid body transformations in terms of isometries and of their homogeneous representation. Finally, we formalize the basics of screw motions, including Chasles' theorem. Using these ingredients, it should now be possible to tackle the formalization of open chains for serial robots.

1 Towards a Formal Theory of Robotics

Our ultimate goal is the formal verification of safety properties of robots. Robots have already made their way in manufacturing plants in the form of robot manipulators on assembly lines. Even though industrial robots operate in a controlled environment, their safety is already subject to several standards (e.g., ISO 10218). Safety concerns will increase furthermore now that robots are considered for life-critical missions such as rescue and health care. It is therefore questionable whether testing can achieve a satisfactory level of safety. For this reason, formal methods are now being considered to improve the rigor of the safety analysis of robots. Most experiments carried so far have been using model-checking, see Sect. 10 for experiments using proof-assistants.

In this paper, we focus on the formal verification of the theoretical foundations of robot manipulators. Robot manipulators are an interesting target because they are already pervasive in the industry and because advanced robots such as humanoid robots can be understood as made of several robot manipulators. The theoretical foundations of robot manipulators essentially amount to three-dimensional geometry. This suggests the use of a proof-assistant to perform formal verification. Indeed, proof-assistants excel at the symbolic manipulation of formal algebra, in which geometry can be represented. We use the Coq proof-assistant [3] and the Mathematical Components library. The latter is a library for formal algebra that was developed to formalize the odd order theorem, an important theorem in group theory [4]. This library features in particular a formalization of linear algebra that we use as a starting point.

Let us give a concrete idea of the kind of three-dimensional geometry robotics is dealing with. A robot manipulator consists of (rigid) *links* connected by *joints*. It is modeled by the relative positioning of frames attached to its links. Figure 1 is a simple example of rigid body transformation. It consists of two rigid bodies connected

剛体変換のための形式基礎

Reynald Affeldt, 独立行政法人産業技術総合研究所 情報技術研究部門, Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology.

Cyril Cohen, フランス国立情報学自動制御研究所 MARELLE プロジェクト・チーム, MARELLE project-team, INRIA Sophia Antipolis.

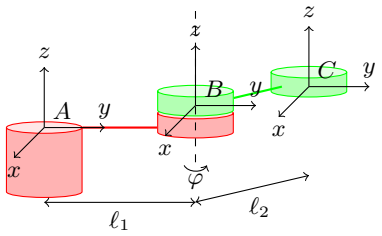

Fig. 1: A simple rigid body transformation [6] (Examples 2.1 and 2.6)

by a joint that can only rotate around a vertical axis (it is a *revolute joint*). If we denote by φ the angle of the rotation, then the orientation of frame B w.r.t. A can be expressed by a rotation matrix ${}^{A}R_{B} = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$. The coordinate of the origin of the frame B w.r.t. A is ${}^{A}P_{B} = [0; \ell_1; 0]$. It is customary to represent rigid body transformations using the so-called homogeneous representation, which is $\begin{bmatrix} {}^{A}R_{B} & 0 \\ {}^{A}P_{B} & 1 \end{bmatrix}$ (row-vector convention) in the case of Fig. 1.

The main contribution of this paper is to provide the various formal definitions and lemmas that serve as the theoretical foundations of the theory of robot manipulators. Concretely, we have formalized most aspects of rigid body transformations. Basic geometric elements (points, vectors, etc.) can be defined right away using the linear algebra from the Mathematical Components library. Yet, we lack basic support for reasoning about rotations (most notably, angles), in particular in three dimensions (in particular, we need the cross-product). In fact, there are many theories that are put at work to deal with rotations in robotics: rotation matrices, exponential coordinates, quaternions, etc. Rigid body transformations form a dedicated theory with various alternative representations such as isometries and screw motions. Using all the theories above, it becomes possible to model formally robot manipulators and more generally open chains for serial robots.

This paper is organized as follows. We start with background material: angles and trigonometric functions in Sect. 2 and basic elements of three-dimensional geometry in Sect. 3. Then, we provide an exhaustive formalization of rotations: formal definitions in Sect. 4, exponential coordinates in Sect. 5, and quaternions in Sect. 6. Last, we formalize rigid body transformations: equivalence with direct isometries in Sect. 7, homogeneous representations in Sect. 8, and screw motions in Sect. 9. We comment on related work in Sect. 10 and conclude in Sect. 11.

2 Angles and Trigonometric Functions

This section provides an overview of a formalization of angles and trigonometric functions that we have been developing in particular to formalize rotations (see Sect. 4 for example).

The basic idea to formalize angles and trigonometric functions is to use the complex numbers of the Mathematical Components library.

2.1 Angles

Angles are defined using complex numbers of unit norm. In Mathematical Components, the type of complex numbers is `R[i]`, where `R` is a real closed field (`rcfType`). An angle is defined as a complex number with norm 1:

```
Variable R : rcfType.
Record angle := Angle {
  expi : R[i];
  _ : `| expi | == 1 }.
```

In particular, the argument of a complex number defines an angle:

```
Definition arg (x : R[i]) : angle :=
  insubd angle0 (x / `| x |).
```

(`angle0` is the angle corresponding to the complex number 0; `insubd` is a Mathematical Components construct that projects a type into a subtype with a default proof.)

We need to equip above definitions to deal with scaling of angles (e.g., double-angles in Sect. 6.2) and half-angles (e.g., to prove `etwist_is_onto_SE` in Sect. 9.2). Specific angles are defined as the arguments of specific complex numbers. For example, we define π as the argument of -1:

```
Definition pi := arg (-1).
```

2.2 Trigonometric Functions

Table 1 displays the most important trigonometric functions of our formalization. The functions cos and sin are defined using the real and imaginary parts of the complex numbers that define angles. Other trigonometric functions are formalized using their definition in terms of complex numbers.

Table 1: Main trigonometric functions used in this paper

function	formalization in Coq	pencil-and-paper definition
$\cos(a)$	`Re (expi a)`	
$\sin(a)$	`Im (expi a)`	
$\arcsin(x)$	`arg (Num.sqrt (1 -x^2) +i* x)`	$\arg(\sqrt{1-x^2}+xi)$
$\arccos(x)$	`arg (x +i* Num.sqrt (1 -x^2))`	$\arg(x+i\sqrt{1-x^2})$
$\arctan(x)$	`if x == 0 then 0 else arg ((x^-1 +i* 1) *~ sgz (x))`	$\arg(sgn(x)(\frac{1}{x}+i))$ (0 if $x=0$)

For example, we use arccos in Sect. 5.3 to define the angle-axis representation of a rotation matrix, arctan in Sect. 6.2 to define rotations using quaternions, and cotangent in Sect. 9.2 to prove the surjectivity of the exponential of twists.

We equip this formalization of trigonometric functions with various lemmas, in particular the cancellation laws between trigonometric functions and their inverse, and many trigonometric identities (Pythagorean identity, double-angle and half-angle formulas, etc.).

3 Basics of Three-dimensional Geometry

In this section, we introduce the basic elements of our formalization of three-dimensional geometry.

3.1 Vectors

Vectors are provided by the Mathematical Components library in the form of a type denoted by `'rV[R]_n` whose elements are row-vectors of length `n` with components of type `R`.

We denote by `'e_0`, `'e_1`, and `'e_2` the vectors of the canonical basis. Concretely, `'e_k` is formalized by the row-vector $[\delta_{0k};\delta_{1k};\delta_{2k}]$, where δ is Kronecker's.

We denote by `u` $*_d$ `v` the dot-product of the vectors `u` and `v`. It is formally defined by the only component of the 1×1 matrix `u` $*_m$ `v^T`, where $*_m$ is matrix multiplication and `^T` is matrix transpose.

The norm of a vector $\mathbf{u}$ is defined using the square root of the dot-product $\langle \mathbf{u}\cdot\mathbf{u}\rangle$:

```
Definition norm u := Num.sqrt (u *d u).
```

3.2 The Cross-Product

The cross-product of two vectors $\mathbf{u}$ and $\mathbf{v}$ can be defined using determinants:

$$\mathbf{u}\times\mathbf{v} = \begin{vmatrix} 1 & 0 & 0 \\ u_0 & u_1 & u_2 \\ v_0 & v_1 & v_2 \end{vmatrix} \text{'e\_0} + \begin{vmatrix} 0 & 1 & 0 \\ u_0 & u_1 & u_2 \\ v_0 & v_1 & v_2 \end{vmatrix} \text{'e\_1} + \begin{vmatrix} 0 & 0 & 1 \\ u_0 & u_1 & u_2 \\ v_0 & v_1 & v_2 \end{vmatrix} \text{'e\_2}$$

We use of the determinant `\det` of the Mathematical Components library to formalize the above formula:

```
Definition crossmul u v :=
  \row_(k < 3) \det (col_mx3 'e_k u v).
```

`'e_i` are the vectors of the canonical basis introduced in the previous section (Sect. 3.1). `col_mx3` is the concatenation of three row-vectors into a 3×3 matrix. Hereafter, we denote `crossmul u v` by `u` $*_v$ `v`.

We equip this formalization of the cross-product with the expected standard lemmas. For example, the double cross-product is a technical but useful lemma about the cross-product and the dot-product (used for example in the proof of `rodriguesP` in Sect. 5.2 and to deal with quaternions in Sect. 6):

```
Lemma double_crossmul u v w :
  u *v (v *v w) = (u *d w) *: v -(u *d v) *: w.
```

3.3 Frames

As highlighted in the introduction, frames are important to model robot manipulators.

We define a non-oriented frame by three unit vectors `i`, `j`, and `k` that are pairwise orthogonal:

```
Record t := mk {
  i : 'rV[R]_3 ; j : 'rV[R]_3 ; k : 'rV[R]_3 ;
  _ : norm i = 1 ; _ : norm j = 1 ; _ : norm k = 1 ;
  _ : i *d j = 0 ; _ : j *d k = 0 ; _ : i *d k = 0 }.
```

The sign of a non-oriented frame is defined as `i` $*_d$ (`j` $*_v$ `k`) and can only be 1 or -1. When it is 1, we talk about a positive (or right-handed) frame.

We will need to build positive frames given one non-zero vector (for example to specify rotations in Sect. 4.1). For this purpose, we provide a module `Base` that features in particular two functions `j` and `k` such that, given a unit vector `i`, the vectors `i`, `j i`, and `k i` form a positive frame. Concretely, the

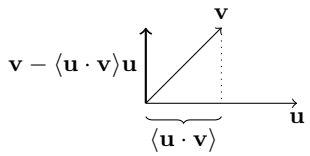

Fig. 2: The normal component of **v** w.r.t. **u**

function `j` is defined as follows[†1]:

```
Definition j i :=
  if colinear i 'e_0 then 'e_1
  else normalize (normalcomp 'e_0 i).
```

If `i` is colinear with `'e_0` then the desired frame can be built using the vectors of the canonical basis. Otherwise, we compute the normal component of `'e_0` w.r.t. `i` (see Fig. 2) and we normalize to a unit vector. As for the function `k i`, it is simply defined by the cross-product of `i` and `j i`: this guarantees that the frame is positive.

4 Rotations and Rotation Matrices

In this section, we formally define three-dimensional rotations and show the equivalence with rotation matrices.

4.1 Definition of Rotations

A rotation of angle a around a vector $\mathbf{u}$ is a linear function f such that $f(\mathbf{u}) = \mathbf{u}$ (in other words, the axis is invariant by rotation), $f(\mathbf{j}) = \cos(a)\mathbf{j} + \sin(a)\mathbf{k}$, and $f(\mathbf{k}) = -\sin(a)\mathbf{j} + \cos(a)\mathbf{k}$, where $\frac{\mathbf{u}}{||\mathbf{u}||}$, $\mathbf{j}$, and $\mathbf{k}$ form a positive frame.

To build the positive frame from vector $\mathbf{u}$ we use the functions `Base.j` and `Base.k` explained in Sect. 3.3. The formal specification of a rotation takes the following form:

```
CoInductive is_around_axis u a
  (f : {linear 'rV_3 →'rV_3}) : Prop :=
  mkIsAroundAxis of
  f u = u &
  let: j := Base.j u in let: k := Base.k u in
  f j = cos a *: j + sin a *: k &
  let: j := Base.j u in let: k := Base.k u in
  f k = -sin a *: j + cos a *: k.
```

†1 The functions `colinear`, `normalize`, and `normalcomp` are utility functions from our formalization. The naming should be self-explanatory.

4.2 Rotations Matrices

A matrix M such that $MM^T = 1$ is *orthogonal*. It is a *rotation matrix* when moreover $\det(M) = 1$. Let us denote by `'O[R]_n` the set of orthogonal matrices and by `'SO[R]_n` the set of rotation matrices.

We show that any rotation in the sense of Sect. 4.1 can indeed by represented by a rotation matrix:

```
(* assume u != 0 *)
Lemma is_around_axis_SO a f :
  is_around_axis u a f →lin1_mx f \is 'SO[R]_3.
```

(`lin1_mx f` is the matrix corresponding to the linear application `f`.) The idea of the proof is to express `lin1_mx f` as a product $P^{-1}R_x(a)P$ where $R_x(a)$ is a rotation around the canonical vector `'e_0` and P represents the vectors of the frame `normalize u`, `j u`, `k u` (see Sect. 3.3) in the canonical basis.

Conversely, given a rotation matrix, one can find an axis and an angle such that the corresponding linear application is a rotation:

```
Lemma SO_is_around_axis M : M \is 'SO[R]_3 →
  exists u a, norm u = 1 /\
    is_around_axis u a (mx_lin1 M).
```

(`mx_lin1` turns a matrix into the corresponding linear function.) The proof uses Euler's theorem:

```
Lemma euler (M : 'M[R]_3) : M \is 'SO[R]_3 →
  {x : 'rV[R]_3 | (x != 0) && (x *_m M == x)}.
```

We use Euler's theorem to obtain an eigenvector of the rotation matrix. We use this vector to build a positive frame (see Sect. 3.3) and inspect the result of the action of the matrix `M` on the frame vectors. It happens that the coordinates of the resulting vectors have properties that are fulfilled by the trigonometric functions `cos` and `sin` and that satisfy the definition of a rotation.

5 Exponential Coordinates for Rotations

Rotation matrices can be represented using exponentials of skew-symmetric matrices. In general, the exponential of a matrix is a power series. For skew-symmetric matrices, there is a closed expression: $e^{\varphi S(w)} = 1 + (\sin\varphi)S(w) + (1 - \cos\varphi)S(w)^2$, where $S(w)$ is the skew-symmetric matrix corresponding to vector w. In the following, we show that rotation matrices can be represented by this

formula.

First, we define the set of skew-symmetric matrices 'so[R]_n, i.e., the matrices M such that $M = -M^T$. To a vector $[w_x; w_y; w_z]$ corresponds the (row-vector convention) skew-symmetric matrix $\begin{bmatrix} 0 & w_z & -w_y \\ -w_z & 0 & w_x \\ w_y & -w_x & 0 \end{bmatrix}$. Hereafter, we denote the skew-symmetric matrix corresponding to the vector w by \S(w).

Now, let us define the following function (where a is an angle and M is a 3×3 matrix):

```
Definition emx3 a M :=
  1 + sin a *: M + (1 -cos a) *: M ^+2.
```

We are interested in the case where the matrix M is \S(w) for some w. We denote such a matrix by `e^(a, w) and call a and w the exponential coordinates.

5.1 Exponential Coordinates of Rotation Matrices

Any rotation matrix can be represented by its exponential coordinates.

First, we observe that for a unit vector w, the matrix `e^(a, w) is a rotation of angle a around w:

```
Lemma is_around_axis_eskew a w : norm w = 1 →
  is_around_axis w a (mx_lin1 `e^(a, w)).
```

Conversely, any rotation matrix can be represented using exponential coordinates:

```
Lemma eskew_is_onto_SO M : M \is 'SO[R]_3 →
  exists a w, norm w = 1 /\ M = `e^(a, w).
```

The proof is as follows. We use the lemma SO_is_around_axis of Sect. 4.2 to derive an angle a and an axis w corresponding to M. We want to show that M = `e^(a, w). The lemma is_around_axis_eskew above says that `e^(a, w) is a rotation of angle a around w. This rotation and the rotation corresponding to M are the same, so are the matrices.

5.2 Rodrigues' Formula

We have shown in the previous section (Sect. 5.1) that multiplication by a matrix `e^(a, w) provides rotation. Rodrigues' formula is a classical result that provides an alternative expression for such a rotation.

Rodrigues' formula transforms a vector v according to angle a and vector w by means of the dot-product and of the cross-product:

```
Definition rodrigues v a w := cos a *: v
  + (1 -cos a) * (v *_d w) *: w + sin a *: (w *_v v).
```

It provably performs the same transformation as rotation using `e^(a, w):

```
Lemma rodriguesP u a w : norm w = 1 →
  rodrigues u a w = u *_m `e^(a, w).
```

The proof is by appealing to the properties of skew-symmetric matrices and of the cross-product (in particular the double cross-product—see Sect. 3.2).

5.3 Angle-Axis Representation

The angle-axis representation of a rotation matrix M is a pair of an angle and a vector computed directly from M.

The angle of a matrix M is defined by:

$$\arccos\left((\mathrm{tr}(M) - 1)/2\right).$$

Put formally:

```
Definition angle_of_rot M := acos ((\tr M -1) / 2%:R).
```

The (row-vector convention) axial vector of a matrix M is defined by:

$$\mathbf{w} = [M_{1,2} - M_{2,1}; M_{2,0} - M_{0,2}; M_{0,1} - M_{1,0}].$$

The rotation axis of a matrix M is defined by $\frac{1}{2\sin(a)}\mathbf{w}$, where a is the angle of the matrix M as defined above and $\mathbf{w}$ is the axial vector. Put formally:

```
Definition axis_of_rot M := let a := angle_of_rot M in
  1 / ((sin a) *+ 2) *: axial_vec M.
```

We can show that M is a rotation of angle angle_of_rot M and axis axis_of_rot M (as long as none of them is zero):

```
Lemma angle_axis_eskew M : M \is 'SO[R]_3 →
  axis_of_rot M != 0 →
  sin (angle_of_rot M) != 0 →
  let a := aangle (angle_axis_of_rot M) in
  let w := aaxis (angle_axis_of_rot M) in
  M = `e^(a, w).
```

6 Formalization of Quaternions

Quaternions provide an alternative way to represent rotations. They are concise (4 elements as opposed to the 9 elements of a matrix) and their composition is computationally more efficient. Quaternions are interesting in their own right as the first non-commutative algebra to be studied and they are used pervasively in robotics (including com-

puter vision).

6.1 Formal Definition of Quaternions

A quaternion has a scalar part and a vector part:

```
Record quat := mkQuat {quatl : R ; quatr : 'rV[R]_3 }.
```

Let us denote the scalar (resp. vector) part of a quaternion a by a`0 (resp. a`1). We denote by a^*q the conjugate of a; it is defined as follows:

```
Definition conjq a := mkQuat (a`0) (- a`1).
```

The norm of a quaternion is defined as follows:

```
Definition sqrq a := a`0 ^+2 + norm (a`1) ^+2.
Definition normq a := Num.sqrt (sqrq a).
```

In particular, *unit quaternions* (type uquat) are quaternions with norm 1.

Addition of quaternions is defined componentwise. Multiplication is defined using the dot-product and the cross-product (Sections 3.1 and 3.2):

```
Definition mulq a b := mkQuat
  (a`0 * b`0 -a`1 *_d b`1)
  (a`0 *: b`1 + b`0 *: a`1 + a`1 *_v b`1).
```

Inverse is defined using the norm and the conjugate:

```
Definition invq a := (1 / sqrq a) *: (a ^*q).
```

Using above operations, we showed that quaternions form a ring (ringType) with scaling (lmodType) and units (unitRingType).

6.2 Rotation with Quaternions

Let a be a quaternion. It turns out that the application $v \mapsto ava^*$ provides us with rotation of (three-dimensional) vectors v:

```
Definition quat_rot (a : quat) (v : vector) : quat :=
  (a : quat) * v%:v * a^*q.
```

We now make precise the nature of this rotation.

First, we observe that quat_rot is a linear function:

```
Lemma quat_rot_is_linear q :
  linear (fun v => (quat_rot q v)`1).
```

Similarly to complex numbers, one can define the polar coordinates of a quaternion (see Sect. 2.2 for the definition of atan and Sect. 3 for normalize and norm):

```
Definition polar_of_quat a :=
  (normalize a`1, atan (norm a`1 / a`0)).
```

Let us consider a unit quaternion q with a non-zero scalar part (i.e., it is not "pure") and polar coordinates (u, a). We can show that the function $v \mapsto (u, a)v(u, a)^*$ is a rotation of angle $2a$ around $\mathbf{u}$:

```
Lemma quat_rot_is_Rot q :
  q \is uquat →¬pureq q →
  let: (u, a) := polar_of_quat q in
  u != 0 →
  is_around_axis u (a *+ 2)
    (Linear (quat_rot_is_linear q)).
```

7 Rigid Body Transformation

A rigid body transformation (hereafter, RBT) is a mapping $\mathbb{R}^3 \to \mathbb{R}^3$ that preserves lengths and orientation. The preservation of orientation by a RBT is defined as the preservation of the cross-product. In this section, we show that a RBT is actually a direct isometry. We define isometries in Sect. 7.1 and orientation in Sect. 7.2.

7.1 Isometries

An isometry is a mapping $\mathbb{R}^n \to \mathbb{R}^n$ that preserves lengths:

```
(* Module Iso *)
Record t := mk {
  f :> 'rV[R]_n →'rV[R]_n ;
  P : {mono f : a b / norm (a -b)} }.
```

We denote the type of isometries by 'Iso[R]_n.

There is a theorem that says that, from any isometry f, one can extract an orthogonal matrix (ortho_of_iso f, the "orthogonal part" of the isometry) and a translation (trans_of_iso f) such that f x = x *_m ortho_of_iso f + trans_of_iso f.

The isometry is direct when the determinant of the orthogonal matrix (its "sign") is 1. We denote the type of three-dimensional direct isometries by 'DIso_3[R]:

```
(* Module DIso *)
Record t := mk {
  f :> 'Iso[R]_3 ;
  P : iso_sgn f == 1 }.
```

7.2 Preservation of Orientation

The action of a RBT f on points induces an action f_* on vectors. Let v be the vector $b - a$ where b and a are points. f_* is defined by $f_*(v) = f(b) - f(a)$ [6] (Chapter 2, Sect. 1, p. 21). f_* is the derivative map of f [8] (Chapter III, Sect. 2). To formalize the derivative map, we define *tangent vectors*. We denote by `u `@ p` the vector `u` with point of application `p`, and by `p.-vec` the type of such vectors. $f_*$ transforms a tangent vector `p.-vec` into a tangent vector `(f p).-vec` according to the following definition:

```
Definition dmap (f : 'Iso[R]_3) p (v : p.-vec) :=
  let C := ortho_of_iso f in
  (v *_m C) `@ f p.
```

Hereafter, we denote the derivative map of `f` by `f`*`.

We use the derivative map to state what it means for an isometry to preserve orientation. An isometry f preserves orientation when $f_*(u \times v) = f_*(u) \times f_*(v)$ [6] (Definition 2.1). Let p by the point of application of $u \times v$. Then, the point of application of $f_*(u) \times f_*(v)$ is $f(p)$. Put formally:

```
Definition preserves_orientation f :=
  ∀p (u v : p.-vec),
  f`* ((u *_v v) `@ p) = ((f`* u) *_v (f`* v)) `@ f p.
```

Using above definitions, we prove formally that a direct isometry preserves orientation:

```
Lemma diso_preserves_orientation (f : 'DIso_3[R]) :
  preserves_orientation f.
```

Since it also preserves lengths by definition, this means that a direct isometry is a RBT.

Conversely, an isometry that preserves the cross-product of two non-colinear vectors is direct:

```
Lemma preserves_crossmul_is_diso (f : 'Iso[R]_3)
  p (u v : p.-vec) : ¬colinear u v →
  f`* ((u *_v v) `@ p) = ((f`* u) *_v (f`* v)) `@ f p →
  iso_sgn f = 1.
```

This means that a RBT can be represented by a direct isometry.

8 Rigid Body Transformation using Homogeneous Representation

In Sect. 7, we showed that a RBT can be represented by a direct isometry. In this section, we show that a direct isometry can be represented by an element of the special Euclidean group, i.e., a pair of a translation and a rotation:

```
(* Module SE *)
Record t (R : rcfType) : Type := mk {
  trans : 'rV[R]_3;
  rot : 'M[R]_3 ;
  rotP : rot \in 'SO[R]_3 }.
```

Before showing in Sect. 8.2 that an element of the special Euclidean group indeed defines a RBT, i.e., that it preserves lengths and orientation, we define in Sect. 8.1 the *homogeneous representation* of an element of the special Euclidean group.

8.1 Homogeneous Representation

The homogeneous representation of an element of the special Euclidean group is a 4×4 matrix $\left[\begin{smallmatrix} \mathtt{r} & 0 \\ \mathtt{t} & 1 \end{smallmatrix}\right]$ where `r` is a rotation and `t` is a vector (representing a translation). We define formally such matrices as follows:

```
Definition hom (r : 'M[R]_3) (t : 'rV[R]_3) : 'M[R]_4 :=
  block_mx r 0 t 1.
```

We denote by `SE3[R]` the set of matrices of this shape such that `r` is a rotation. We show that `SE3[R]` is indeed a group by using in particular the following definition of inverse:

$$\left[\begin{smallmatrix} \mathtt{r} & 0 \\ \mathtt{t} & 1 \end{smallmatrix}\right]^{-1} = \left[\begin{smallmatrix} \mathtt{r}^T & 0 \\ -\mathtt{t}*\mathtt{r}^T & 1 \end{smallmatrix}\right].$$

We denote by `'hP[R]` (resp. `'hV[R]`) the set of homogeneous points $[p_0; p_1; p_2; 1]$ (resp. vectors $[v_0; v_1; v_2; 0]$).

The application of an element of the special Euclidean group (object `T` of type `SE.t`) to a homogeneous point (resp. vector) is the matrix multiplication by its homogeneous representation:

```
Coercion mx (T : SE.t) := hom (rot T) (trans T).
Definition hom_ap (T : SE.t) x : 'rV[R]_4 := x *_m T.
```

When `x` is an homogeneous point, the application of `hom_ap` performs a rotation and a translation, but only a rotation when `x` is an homogeneous vector.

8.2 Elements of the Special Euclidean Group are Rigid Body Transformations

We can now define application of an element of the special Euclidean group to a (three-dimensional) point or vector. The homogeneous point (resp. vector) corresponding to the three-dimensional row-vector `x` is `to_hpoint x` (resp. `to_hvector x`). `from_h` is the projection that cancels

to_hpoint and to_hvector. We use these functions to embed points and vectors into their homogeneous representation and recover points and vectors after having applied hom_ap:

```
Definition ap_point (T : SE.t) p :=
  from_h (hom_ap T (to_hpoint p)).
Definition ap_vector (T : SE.t) v :=
  from_h (hom_ap T (to_hvector v)).
```

We can show that the application of an element of the special Euclidean group to a point using ap_point preserves length between points:

```
Lemma SE_preserves_length (T : SE.t R) :
  {mono (ap_point T) : a b / norm (a - b)}.
```

Since they preserve lengths, the elements of the special Euclidean group can be used to build isometries. We can show moreover that an isometry built in this way preserves orientation:

```
Lemma SE_preserves_orientation (T : SE.t R) :
preserves_orientation (Iso.mk (SE_preserves_length T)).
```

We have therefore established that the special Euclidean group coincides with RBT.

9 Screw Motion

In this section, we formalize the basics of screw theory, in particular the fundamental Chasles' theorem. In Sect. 9.1, we show that there exists a screw axis for any RBT. The existence of a screw axis suggests that any RBT can be represented as the composition of a rotation around an axis followed by a translation along the same axis. In Sect. 9.2, we show that such a *screw motion* can be represented using exponential coordinates.

9.1 Existence of a Screw Axis

Let us assume a RBT f whose orthogonal part is a rotation with angle a around the unit vector w:

```
Variable f : 'DIso_3[R].
Let Q : 'M[R]_3 := ortho_of_iso f.
Variable w : vector.
Hypothesis w1 : norm w = 1.
Variable a : angle R.
Hypothesis Qaxis : is_around_axis w a (mx_lin1 Q).
```

One can show that the components along w of the displacements of any two points are the same:

```
Lemma displacement_proj q p :
  displacement f q *_d w = displacement f p *_d w.
```

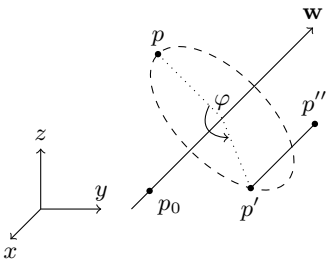

Fig. 3: Illustration of a screw motion

As a consequence, we have Chasles' theorem [2] (Theorem 3.2.2). Let us denote by d0 the component along w of the displacement of the origin. Then, any displacement of norm d0 is parallel to w:

```
Lemma MozziChasles p :
  norm (displacement f p) = d0 →
    colinear (displacement f p) w.
```

This theorem shows that there is a set of points that undergo just a translation: this is the *screw axis*.

9.2 Screw Motion

A screw motion is defined by a line (a point and a vector that play the role of the axis), an angle, and a pitch (put together in the type Screw.t). Figure 3 shows the effect of a screw motion (that is not a pure translation) on a point p (the axis is $(p_0, \mathbf{w})$, the angle is φ, the pitch is the ratio of translation to rotation). The effect of this screw motion can be formalized as follows:

```
Definition screw_motion s p :=
  let: (l, a, h) := (Screw.l s, Screw.a s, Screw.h s) in
  let (p0, w) := (line_point l, line_vector l) in
  p0 + (p - p0) *_m `e^(a, w) + (h * radian a) *: w.
```

It turns out that a screw motion can be represented by exponential coordinates, similarly to the exponential coordinates of rotation matrices in Sect. 5.1. This requires to generalize skew-symmetric matrices to *twists*. A twist \T(v, w) is a pair of two vectors arranged as a matrix $\begin{bmatrix} \mathsf{S}(w) & 0 \\ v & 0 \end{bmatrix}$ (row-vector convention). The exponential of a twist is defined as the following homogeneous matrix †2:

†2 One can find this definition in the literature [6] (Equation 2.36), [10] (Equation 1.27). The relation with the exponential function is explained in Appendix A.

$$\begin{bmatrix} I & 0 \\ \varphi v & 1 \end{bmatrix} \quad \text{if } w = 0$$
$$\begin{bmatrix} e^{\varphi S(w)} & 0 \\ \frac{(w \times v)(1-e^{\varphi S(w)}) + (\varphi v)(w^T w)}{||w||^2} & 1 \end{bmatrix} \quad \text{if } w \neq 0.$$

We denote the exponential of a twist by `e$(a, t) and formalize it as follows:

```
Definition hom_twist t a e :=
  let (v, w) := (\v( t ), \w( t )) in
  if w == 0 then hom 1 (a *: v)
  else hom e ((norm w)^-2 *:
    ((w *v v) *m (1 -e) + (a *: v) *m (w^T *m w))).
Definition etwist a t :=
  hom_twist t (radian a) (`e^(a, \w( t ))).
```

For illustration, the screw_motion function above (more precisely, its homogeneous representation hom_screw_motion) corresponds to the following twist [6] (p. 47):

```
Lemma hom_screw_motion_etwist s :
  let: (l, a, h) := (Screw.l s, Screw.a s, Screw.h s) in
  let (p0, w) := (line_point l, line_vector l) in
  let v := -w *v p0 + h *: w in
  hom_screw_motion s = `e$(a, \T(v, w)).
```

The parameters v, w, and a are called the exponential coordinates for the corresponding RBT. Indeed, any RBT can be represented by exponential coordinates:

```
Lemma etwist_is_onto_SE f : f \is 'SE3[R] →
  exists t a, f = `e$(angle_of_radian a, t).
```

The proof is as follows. Let us assume that f is of the form $\left[\begin{smallmatrix} r & 0 \\ p & 1 \end{smallmatrix}\right]$. If f is a pure translation, then it suffices to choose the twist \T(normalize p, 0) and the angle norm p. Otherwise, let a, w be the exponential coordinates of r (obtained through lemma eskew_is_onto_SO from Sect. 5.1). Then it suffices to choose the twist \T(v, w) and the angle a, where v is such that p = (norm w)^-2 *: (v *m G) with G = \S(w) *m (1 -r) + radian a *: (w^T *m w). The inverse of the latter matrix is $\frac{1}{a} - \frac{1}{2}S(w) + \left(\frac{1}{a} - \frac{1}{2}\cot\left(\frac{a}{2}\right)\right) S(w)^2$ [9].

9.3 Example of Twist Computation

We can use the (constructive) proof of the last lemma (etwist_is_onto_SE, Sect. 9.2) to build the twist \T(v, w) of the rigid body transformation of Fig. 1.

The coordinate of the frame C w.r.t. the frame A is $\left[\begin{smallmatrix} {}^A R_C & 0 \\ {}^A T_C & 1 \end{smallmatrix}\right]$ where ${}^A T_C$ is $[-\ell_2 \sin\varphi; \ell_1 + \ell_2 \cos\varphi; 0]$. The exponential coordinates of the rotation are the axis 'e_2 and the angle φ (see lemma is_around_axis_eskew of Sect. 5.1). We already know that w is 'e_2. v is given by ${}^A T_C G^{-1}$ (where G comes from the proof of etwist_is_onto_SE), that is, after calculation, $[(\ell_1 - \ell_2)/2; (\ell_1 + \ell_2)/(2(1 - \cos\varphi)); 0]$.

10 Related Work

In this section, we focus on related work using proof-assistants to perform formal verification of robotics (we leave aside the more numerous related work dealing with model-checking).

Walter et al. perform in Isabelle the verification of a collision avoidance algorithm for a vehicle moving in a plane [11]. More precisely, this algorithm computes safety zones as supersets of braking areas. It therefore involves mathematics that we have not been dealing with: velocity and computational geometry in a plane (as opposed to three-dimensional geometry for robot manipulators). An important aspect of their experiment is that it was used as part of a certification effort and received positive reviews in this context.

Farooq et al. propose a significant formalization of two-link planar manipulators in HOL-Light [5]. They apply their theories to the analysis of a (two-dimensional) biped walking robot (4 links, 3 joints). Our work can be seen as a tentative extension to three-dimensional geometry.

Anand et al. use Coq to write, verify, and execute programs to be used on actual robots [1]. For that purpose, they provide an event-based programming framework dealing with time. They tackle the problem of verified robot software in a pragmatic way. In comparison, we are dealing with the complementary task of formalizing formal foundations.

Ma et al. propose a formalization of conformal geometric algebra in HOL-Light [7]. They use their formalization to reason about rigid body transformations and apply their results to the formalization of a grasping algorithm. In comparison, we are focusing on standard techniques for robot specification. Though conformal geometric algebra is not mainstream, its formalization may turn out to be inspiring since it deals with multivectors whose product generalizes the three-dimensional

Table 2: Overview of our formalization of rigid body transformations

File name	Contents	l.o.c.
aux.v	Utility definitions and tactics	147
angle.v	Angles and trigonometric functions (Sect. 2)	826
euclidean3.v	Dot-product, cross-product, etc. (Sect. 3); rotation matrices (Sect. 4. 2)	1269
vec_angle.v	Utility definitions such as colinear, normalize, normalcomp (Fig. 2, footnote †1)	479
frame.v	Frames (Sect. 3. 3)	894
skew.v	Skew-symmetric matrices (used in Sections 5 and 9)	621
rot.v	Specification of rotations and exponential coordinates (Sections 4 and 5)	1235
quaternion.v	Rotation with quaternions (Sect. 6)	654
rigid.v	Rigid body transformations (Sections 7 and 8)	1103
screw.v	Screw motions (Sect. 9)	1193

cross-product.

11 Conclusion and Perspectives

In this paper, we gave an overview of an ongoing effort to formalize the foundations of robotics. Our formalization is about 8,300 l.o.c. (breakdown in Table 2). This formalization covers a substantial part of the basic material one can find in standard textbooks, e.g., [10] (Section 1.2: Position and Orientation Representation), [6] (Chapter 2: Rigid Body Motion). Currently, we are working on applying our work to the formalization of open chains for serial robots.

The formalization of the theoretical foundations of robotics is only one step towards the formal verification of robots. At some point, we will need to extend our work to deal with the dynamics of robots by introducing velocity. The correctness of numerical computations used in robot software is another aspect of formal verification of robots. This is not yet our concern but since Coq also excels at verification of numerical computations, it should be possible to build on top of our work a comprehensive formal verification framework for robot manipulators.

Acknowledgments The authors acknowledge partial support from a Grant-in-Aid for Scientific Research (B) (project number 15H02687).

References

[1] Anand, A. and Knepper, R. A.: ROSCoq: Robots Powered by Constructive Reals. *Proc. ITP 2015, LNCS*, Vol. 9236, pp. 34–50.

[2] Angeles, J.: Fundamentals of Robotic Mechanical Systems—Theory, Methods, and Algorithms. Springer, 2014. 4th Edition.

[3] The Coq Development Team: Reference Manual. INRIA, 1999–2016. Ver. 8.5pl2. Available at http://coq.inria.fr.

[4] Gonthier, G., Asperti, A., Avigad, J., Bertot, Y., Cohen, C., Garillot, F., Le Roux, S., Mahboubi, A., O'Connor, R., Ould Biha, S., Pasca, I., Rideau, L., Solovyev, A., Tassi, E., Théry, L.: A Machine-Checked Proof of the Odd Order Theorem. *Proc. ITP 2013, LNCS*, Vol. 7998, pp. 163–179.

[5] Farooq, B., Hasan, O., and Iqbal, S.: Formal Kinematic Analysis of the Two-Link Planar Manipulator. *Proc. ICFEM 2013, LNCS*, Vol. 8144, pp. 347–362.

[6] Murray, R. M., Li, Z., Sastry, S. S.: A Mathematical Introduction to Robotic Manipulation. CRC Press, 1994. First edition.

[7] Ma, S., Shi, Z., Shao, Z., Guan, Y., Li, L., Li, Y.: Higher-Order Logic Formalization of Conformal Geometric Algebra and its Application in Verifying a Robotic Manipulation Algorithm. *Advances in Applied Clifford Algebras*. March 2016. In press.

[8] O'Neill, B.: Elementary Differential Geometry. Academic Press, 1966.

[9] Park, F. C., Lynch, K.: Introduction to Robotics: Mechanics, Planning, and Control. Seoul National University, 2012. Course text.

[10] Siciliano, B., Khatib, O. (Eds.): Springer Handbook of Robotics. Springer, 2008.

[11] Walter, D., Täubig, H., Lüth, C.: Experiences in Applying Formal Verification in Robotics. *Proc. SAFECOMP 2010, LNCS*, Vol. 6351, pp. 347–360.

A The Exponential of a Twist using Taylor Expansion

In Sect. 9. 2, we define a screw motion in term of a twist by the expression etwist. In the literature, this expression is derived from definitions of exponential coordinates in terms of power series. We reproduce this reasoning here using Taylor expansions.

We first define the Taylor expansion of the exponential function:

```
Definition ecoef n (M : 'M[R]_n.+1) i :=
  i`!%:R^-1 *: M ^+i.
Definition emx n (M : 'M[R]_n.+1) k :=
  \sum_(i < k) ecoef M i.
```

We now define the exponential coordinates of a rotation as the exponential of a skew-symmetric matrix `emx (a *: \S( w )) k` (instead of the definition using `emx3` from Sect. 5).

Similarly, we also define the exponential coordinates of a RBT as the exponential of a twist `emx (a *: t) k`.

We now justify the definition of `etwist` in Sect. 9.2. Let us define a new function `emx_twist` similar to `etwist`: it uses the same definition `hom_twist`, with the difference that we replace `` `e^(a, \w( t )) `` by `emx \S( a *: \w( t ) ) k`:

```
Definition emx_twist a t k :=
  hom_twist t a (emx \S( a *: \w( t ) ) k).
```

We show that this definition is actually equivalent to the exponential of a twist [6] (Proposition 2.8):

```
Lemma emx_twistE (t : Twist.t R) a k :
  emx (a *: t) k.+2 = emx_twist a t k.+2.
```

In other words, instead of computing the exponential as a sum using `emx`, one may also use the more direct expression `expmx_twist`. This justifies the definition of `hom_twist` (and therefore `etwist`) in Sect. 9.2.