Geometric Routines of PTC

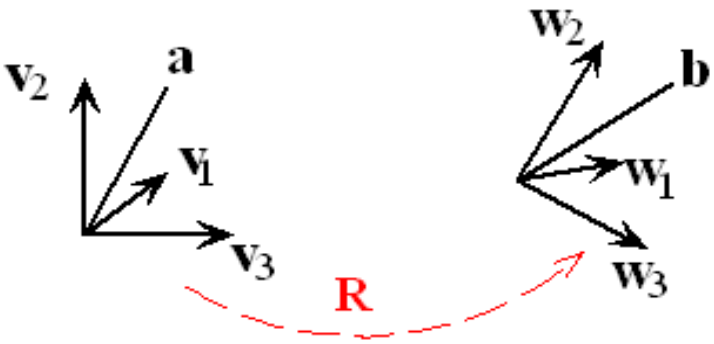
The geometric routines of PTC are divided into 3 categories.

1) Routines acting on a pure geometrical affine basis: A(3) and V(3,3) where A represents the coordinate of a point in the global frame and V(3,3) are the coordinates of a triad of vectors.
2) Routines acting on the affine bases of magnets, siamese, girders, integration nodes, fibres. These objects contain a myriad of affine bases to help us locate the trajectories in 3-d.
    a) Displacements which correspond to the design positioning and thus requiring patching
    b) Misalignments representing errors which do not require patching. They displace the magnet away from a fibre which contains it. They also displace girders away from their original position.
3) Finally, the geometrical rotations and translations of PTC which act on the tracked object itself. PTC has an "exact" dynamical group and an "approximate" dynamical group. It is remarkable that the approximate dynamical rotations and translations also form a group- but it is not isomorphic to the affine Euclidean group.

Item 1: Affine Routines

Theory:

Consider a point **a** and vector basis ($\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$) attached to a solid (magnet for example)



The vector **a** can be expressed as follows:

$\mathbf{a} = \Sigma\, a_i\, \mathbf{v}_i$

The basis vectors $\mathbf{v}_i$ can be expressed in terms of a global basis, i.e., the global frame of PTC:

$\mathbf{v}_i = \Sigma\, V_{ij}\, \mathbf{e}_i$

Suppose we rotate the solid by a rotation R which is defined by its action on the basis ($\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$) :

$\mathbf{w}_i = R\, \mathbf{v}_i = \Sigma\, R_{ij}\, \mathbf{v}_j$

Then we ask the following question: what are the components if $\mathbf{w}_i$ in the global basis ($\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$) in term of the component array V_{ij}.

Nota Bene: the components of **b** in the frame ($\mathbf{w}_1$, $\mathbf{w}_2$, $\mathbf{w}_3$), which is the image of **a** upon rotation of the magnet, are also given a_i because this point is fixed in the magnet.

Solution:

$\mathbf{w}_i = R\, \mathbf{v}_i = \Sigma\, R_{ij}\, \mathbf{v}_j = \Sigma\, R_{ij}\, V_{jk}\, \mathbf{e}_k = \Sigma\, W_{ik}\, \mathbf{e}_k$

Thus we have:

$$W = RV$$

The components of the vector **b** in the global frame are then:

$$b_k = \Sigma\, a_i\, R_{ij}\, V_{jk}\, \mathbf{e}_k \;\Rightarrow\; b=(RV)^{\dagger}\, a$$

We now address a slightly harder problem which is essential in PTC. The rotation R, instead of being defined in the frame ($\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$), might be defined on a totally different frame ($\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$):

$u_i = \Sigma\, U_{ik}\, \mathbf{e}_k$

Therefore, prior to rotating the basis ($\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$), we must expressed it in the frame ($\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$):

$\mathbf{a} = \Sigma\, a_i\, V_{ik}\, \mathbf{e}_k = \Sigma\, a_i\, V_{ik}\, U_{kn}^{-1}\, \mathbf{u}_n = \Sigma\, a_i\, V_{ik}\, U_{nk}\, \mathbf{u}_n$

We can now apply the rotation R defined on the basis ($\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$):

$\mathbf{b} = \Sigma\, a_i\, V_{ik}\, U_{nk}\, R_{nm}\mathbf{u}_m = \Sigma\, a_i\, V_{ik}\, U_{nk}\, R_{nm}U_{mk}\, \mathbf{e}_k$

Thus the final result of W is:

$$W = VU^{\dagger}RU$$

The point b in global coordinate is:

$$b_i = \Sigma\, a_i\, W_{ik} \;\Rightarrow\; b=(\, VU^{\dagger}RU)^{\dagger}\, a$$

Notice that if V=U, we regain the previous result.

The rotation R is actually factored as follows:

$$R= R_z R_y R_x$$

N.B. The need to go back and forth between frames originates from the

use of local variables in PTC as well as in other tracking codes. The magnets can only be placed if some knowledge of the global frame is present. However, the tracking is in local variables. So we need routines which are able to connect geometrically both points of view.

Actual routines of PTC

GEO_ROT(V(3, 3),W(3, 3),A(3),B(3),ANG(3),*BASIS(3,3)*)

This routine exactly reproduces the theory expounded in the previous section. *BASIS* is an optional variable which is set equal to the U of the previous section.

The real array ANG(3) is used to define R:

$$R= R_z(ang(3))R_y(ang(2))R_x(ang(1))$$

GEO_ROT(V(3, 3),W(3, 3),ANG(3),*BASIS(3,3)*)

Same as above except that A and B are not needed.

GEO_ROT(V(3, 3),A(3),ANG(3),I, *BASIS(3,3)*)

Here the final W and B are copied back into V and A. The integer I can be ±1 to produce the inverse rotation: $R^{\pm 1}$.

GEO_ROT(V(3, 3),ANG(3),I, *BASIS(3,3)*)
Same as above without the A vector.

GEO_TRA(A(3),V(3, 3) (3),D,I)

A is translated by ±D expressed in the basis V; I=±1.

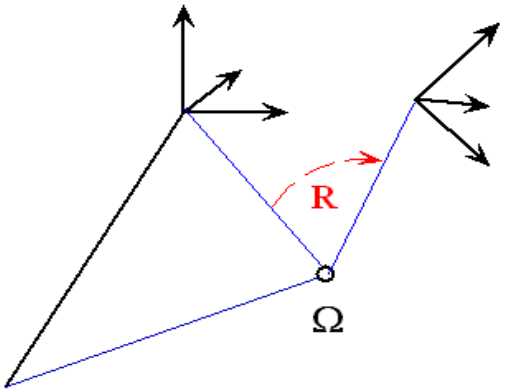
$$\text{result}=\mathbf{\Sigma}\ (a_j\mathbf{e}_j\pm d_i\ V_{ij}\ \mathbf{e}_j)$$

The result is put back in A, i.e., A<=A± $V^{\dagger}D$

Rotating and translating the frames of magnet PTC

ROTATE_FRAME(F,OMEGA,ANG,ORDER,*BASIS(3,3)*)

F is of type(magnet_frame) and contains 3 affine frames tied to the magnet.



F={(F%A(3), F%ENT(3)) , (F%O(3), F%MID(3)) , (F%B(3),F%EXI(3))}

The entire content of F is rotated as described on the following figure.

TRANSLATE_FRAME(F,D,ORDER,*BASIS(3,3)*)

```
CALL CHANGE_BASIS(D,BASIS,DD,GLOBAL_FRAME)

P%A=P%A+ORDER*DD
P%B=P%B+ORDER*DD
P%O=P%O+ORDER*DD
```

The frame is simply translated by D. The translation D is expressed in BASIS if present.

CHANGE_BASIS(A(3),V(3,3),B(3),W(3,3))

The component vector A, expressed in the basis V, is re-expressed as B using basis W. B is obviously the output of this subroutine.

$$\Sigma\, a_i\, V_{ik}\, \mathbf{e}_k = \Sigma\, b_i\, W_{ik}\, \mathbf{e}_k \Rightarrow b = W V^{\dagger} a$$

COMPUTE_ENTRANCE_ANGLE(V(3,3),W(3,3),ANG(3))

This is a crucial routine of PTC. Given two frames V and W which most likely represent a magnet or a beam line, it computes a rotation R in the standard PTC order, i.e.,

$$R = R_z(\mathrm{ang}(3))R_y(\mathrm{ang}(2))R_x(\mathrm{ang}(1)),$$

such that V is transformed into W. It is the reverse routine from GEO_ROT.

FIND_PATCH_B(A(3),V(3,3),B(3),W(3,3),D(3),ANG(3))

This connect the affine frame (A,V) to the affine frame (B,W).

```
 SUBROUTINE FIND_PATCH_B(A,V,B,W,D,ANG)
! FINDS PATCH BETWEEN V AND W : INTERFACED LATER FOR FIBRES
   IMPLICIT NONE
   REAL(DP), INTENT(INOUT):: V(3,3),W(3,3)
   REAL(DP), INTENT(INOUT):: A(3),B(3),D(3),ANG(3)
   CALL COMPUTE_ENTRANCE_ANGLE(V,W,ANG)
   D=B-A;   CALL CHANGE_BASIS(D,GLOBAL_FRAME,D,W);
 END SUBROUTINE FIND_PATCH_B
```

The above code is a simple example of the usage of the geometric routines of PTC. This geometric routine is very close to the dynamical set up of PTC. In PTC, patches are written as an “x” rotation, a “y” rotation, a “z” rotation followed by a translation (transverse + drift). The reader will notice that the translation D=B-A between is expressed in the final frame W. This is normal if dynamical rotations precede the translations.

INVERSE_FIND_PATCH(A(3),V(3,3),D(3),ANG(3),B(3),W(3,3))

This routine is the precise inverse of the previous routine. The affine frame (B,W) is the output.

```
 SUBROUTINE INVERSE_FIND_PATCH(A,V,D,ANG,B,W)
! USED IN MISALIGNMENTS OF SIAMESE
  IMPLICIT NONE
  REAL(DP), INTENT(INOUT):: V(3,3),W(3,3)
  REAL(DP), INTENT(INOUT):: A(3),B(3),D(3),ANG(3)
  REAL(DP) DD(3)
  W=V
  CALL GEO_ROT(W,ANG,1,BASIS=V)
  CALL CHANGE_BASIS(D,W,DD,GLOBAL_FRAME)
  B=A+DD
 END SUBROUTINE INVERSE_FIND_PATCH
```

Item 2-a:

Affine Routines on the "fibrous" Structures of PTC

We have discussed in the previous section the geometrical tools of PTC on affine frame. This is useful in giving a pictorial representation of a ring in 3-d. One can imagine linking PTC with a CAD program which would be equipped with a magnet widget containing at least one affine frame, say the cord frame (O(3),MID(3,3)) of a PTC magnet.

But, of course, PTC is dynamically a more complex structure than just magnets: fibres, integration nodes, layouts, etc… all these objects have affine frames attached to them and we must be able to move them.

We describe first the routines which displace PTC structures away from a standard "MAD8" configuration. These are used in the generation of "non-standard" systems. I purposely used quotation marks since there is really nothing standard about the so-called standard implicit geometry of MAD8. It is worth pointing out that the code MAD of CERN and the code SAD of KEK have different implicit geometry once vertical magnets are invoked: nothing is standard.

Patching routines

The central power of PTC is its ability to place magnets in some arbitrary positions. To do so, the concept of a fibre with a patch is necessary.

FIND_PATCH(EL1,*EL2_NEXT,NEXT,ENERGY_PATCH,PREC*)

EL1 and *EL2_NEXT*

PREC is a small real(dp) number. If the norm of the geometric patch is smaller than PREC, then the patch is ignored. If energy_patch is true, then it compares the design momenta of the magnets in both fibres. If the magnitude of difference is greater than PREC, then an energy patch is put on.

PTC has also a routine to check if a patch is needed without actually applying it to the layout:

CHECK_NEED_PATCH(EL1,*EL2_NEXT*,PREC,PATCH_NEEDED)

This routine returns the integer PATCH_NEEDED. If zero, it is not needed.

Fibre content:

Before going any further, we should remind the reader of the frames contained within a fibre:

1) the frames f the fibre itself located in type(chart),i.e., fibre %chart%f
2) The frames of the magnet fibre%mag%p%f and its polymorphic twin.
3) The frames of the integration nodes associated to this fibre/magnet if they are present
4) The frame of a girder which might be tagged on this magnet

Subroutines invoking the magnet and relying on the DNA

When constructing complex structures in PTC, it is preferable to follow a strict discipline. First various layouts with no "cloning" of magnets are created in a "mad universe" of type MAD_UNIVERSE. The actual code MAD-X calls this universe M_U.

These no-clone layouts contain the actual magnet database of the accelerator complex. Therefore we refer to these layouts as the DNA of the complex.

Then trackable structures are appended after the DNA. Since the magnets of these structures must be in the DNA, they are created with the append_point routine of PTC rather than the standard append_empty or append_fibre used for the DNA production.

When using the append_point, a magnet will automatically retain memory of its various fibre appearances though a type called "fibre_appearance"

```
TYPE FIBRE_APPEARANCE
  TYPE(FIBRE), POINTER :: PARENT_FIBRE
  TYPE(FIBRE_APPEARANCE), POINTER :: NEXT
END TYPE FIBRE_APPEARANCE
```

Each magnet of the DNA contains a pointer called

```
TYPE(FIBRE_APPEARANCE), POINTER :: DOKO
```

which constitutes a link list storing all the appearances of this magnet in the pointer parent_fibre. The linked list is terminated, or grounded, at the last fibre appearance. If more trackable structures are created, this linked list is extended for each magnet.

There are all sorts of reasons for which we may have multiple appearances of a DNA magnet: recirculation, common section of colliders or even

multiple trackable structures of the same physical object.

It is through the “doko” construct that the following routines know where all the magnets are located.

Translation Routines with no automatic patching

TRANSLATE_FIBRE(R,D(3),*ORDER,BASIS(3,3),DOGIRDER*)

Here R is a fibre to be translated by D. Dogirder=true forces the translation of the girder frame if present.

This routine will use the doko construct, if associated, to locate all the appearance of a magnet and rotate the frames on all integration nodes if present.

TRANSLATE_LAYOUT(R,D,*I1,I2,ORDER,BASIS*)

This routine scans the layout R from position i1 to i2 inclusive if present. I1 and i2 are defaulted to 1 and R%N respectively.

This routines calls TRANSLATE_FIBRE with dogirder=true on each fibre.

This routine will use the doko construct, if associated, to locate all the appearance of a magnet and rotate the frames on all integration nodes if present.

Rotation Routines with *no* automatic patching

ROTATE_FIBRE(R,OMEGA,ANG,*ORDER,BASIS,DOGIRDER*)

The same comments that applied to TRANSLATE_FIBRE apply to ROTATE_FIBRE.

ROTATE_LAYOUT(R,OMEGA,ANG,*I1,I2,ORDER,BASIS*)

Same comments as TRANSLATE_LAYOUT apply.

DNA designed Rotation Routines with *automatic patching*

ROTATE_MAGNET(R,ANG,OMEGA,*ORDER,BASIS,PATCH,PREC*)

R is a magnet. The parent_fibre is rotated using the above routine; which actually rotates all the frames of the magnet.

If Patch=true, then all the appearances of this magnets stored in doko are patched provided the norm of the patches is greater than PREC. It is advisable to set PREC to a not too small number (say 10^{-10}) to avoid useless patches.

If there is no DNA, i.e., PTC is ran in pure compatibility mode with standard codes, then patches are on the parent fibre of the actual magnet.

TRANSLATE_MAGNET (R,D,*ORDER,BASIS,PATCH,PREC*)

This routine operates like rotate_magnet described above.

Operation on siamese

Siamese are magnets which we tie together. They can be in the same layout or in totally different layout. In fact they are often magnets which are parallel, as in a collider, or magnets in different arcs of a recirculator sharing some common cryogenics for example.

In PTC, siamese are tied together by the pointer “siamese” which sits on fibre%mag%siamese.

To create a siamese structure, one simply makes a circular linked list of

magnets. For example, suppose 3 DNA fibres f1, f2 and f3 must be tied together. This can be done as follows:

f1%mag%siamese=>f2%mag
f2%mag%siamese=>f3%mag
f3%mag%siamese=>f1%mag

Siamese Frame

It is possible and advisable to set up a so-called affine_frame on the siamese. In the above example, one picks up any magnet of the siamese, for example f1%mag, and calls the routine:

```
CALL ALLOC_AF(F1%MAG%SIAMESE_FRAME)
CALL FIND_PATCH(F1%CHART%F%A,F1%CHART%F%ENT,A, ENT, &
F1%MAG%SIAMESE_FRAME%D,F1%MAG%SIAMESE_FRAME%ANGLE)
```

The siamese frame is located in relative coordinate from the entrance of the fibre which contains the magnet on which siamese_frame is attached (green objects above). Generally we may know the desired frame in absolute coordinates given by the red A(3) and ENT(3,3) above. The relative translation and rotation can be computed and the result is stored in the blue variables.

ROTATE_SIAMESE(S2,ANG,*OMEGA,ORDER,BASIS,PATCH,PREC*)

This routine rotates the siamese by a set of angles ang(1:3) in the usual PTC order. S2 is any fibre which contains an element of the siamese string. The intricate usage of the optional variables *OMEGA,ORDER,BASIS* are best explained by displaying the actual code:

```
CALL FIND_AFFINE_SIAMESE(S2,CN,FOUND)
IF(FOUND) CALL FIND_FRAME_SIAMESE(CN,B,EXI,ADD=MY_FALSE)

IF(PRESENT(BASIS)) THEN
  BASIST=BASIS
ELSE
```

```
  IF(FOUND) THEN
    BASIST=EXI
  ELSE
    BASIST=GLOBAL_FRAME
  ENDIF
ENDIF
IF(PRESENT(OMEGA)) THEN
  OMEGAT=OMEGA
ELSE
  IF(FOUND) THEN
    OMEGAT=B
  ELSE
    OMEGAT=GLOBAL_ORIGIN
  ENDIF
ENDIF
```

If OMEGA is present, then it is used. If BASIS is present it is also used. Normally one would expect both to be present or both to be absent. PTC does not check for this.

If they are not present, PTC looks for a siamese frame which is then used if it exists. Otherwise the global frame of PTC is used.

This routine calls the equivalent "magnet" routines over the entire string of siamese and this permits automatic patching.

TRANSLATE_SIAMESE(S2,D,*ORDER,BASIS,PATCH,PREC*)

This routines functions like the above rotate_siamese with the same priorities concerning the optional variable BASIS.

Operation on girders

Girders are also magnets which we tie together. But additionally they are tied to a plate and thus the "plate" must have an affine frame of its own.

To create a girder structure, one simply makes a circular linked list of magnets. Let us create a girder structure with the 3 above siamese f1,f2 and f3 and additionally let us put a single magnet f0 on the girder.

f0%mag%girder=> f1%mag
f1%mag%girder=> f2%mag
f2%mag% girder =>f3%mag
f3%mag% girder =>f0%mag

Four magnets are put on the same girder; 3 happen to be in a single siamese as well.

Girder Frame

The girder is given an absolute magnet frame unlike the siamese. For example we may elect to put the frame on f0:

CALL ALLOC_AF(F0%MAG%GIRDER_FRAME,GIRDER=.TRUE.)

Then we set the two following affine frames of the girder to the same value:

F0%MAG%GIRDER_FRAME%ENT= ENT
F0%MAG%GIRDER_FRAME%A= A
F0%MAG%GIRDER_FRAME%EXI= ENT
F0%MAG%GIRDER_FRAME%B= A

The affine frame (A,ENT) can be any convenient frame chosen by the people who actually align the girder on the floor of the machine.

If a girder is misaligned, then the affine frame (B,EXI) will contain the new position of the girder. If the misalignments are removed, then (B,EXI) coincides with (A,ENT). This allows the girder to have an existence independent of the fibres themselves. One can move fibres within a girder during its creation while keeping this frame fixed.

ROTATE_GIRDER(S2,ANG,OMEGA,ORDER,BASIS,PATCH,PREC)

TRANSLATE_GIRDER(S2,D,ORDER,BASIS,PATCH,PREC)

The routines operate exactly as the siamese routines do and therefore do not require any special description. The routines describe "design" displacements of the girder and therefore the frame (A,ENT) and (B,EXI) are moved together.

Item 2-b: Misalignments in PTC

PTC has three fundamental misalignment routines related to a single magnet, a siamese and a girder.

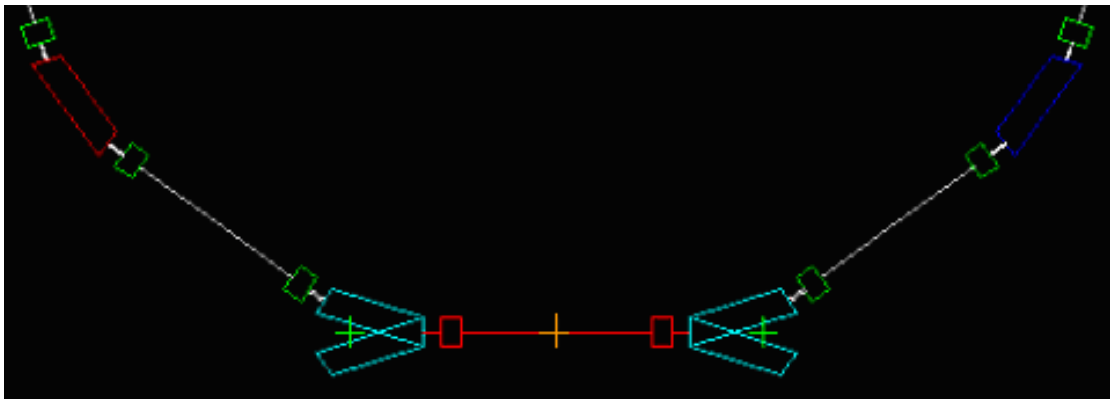
The single magnet and the siamese are defined with respect to their fibre position and are thus inherently similar. The girder has a special frame independent of the fibre. This was explained above.

If one is not careful, a single magnet or a siamese misalignment may break the girder. So we will start with the girder misalignment.

MISALIGN_GIRDER(S2,S1,*OMEGA,BASIS,ADD*)

In this discussion we will assume that a girder frame has been defined; otherwise the girder becomes simply a giant siamese.

Example 0

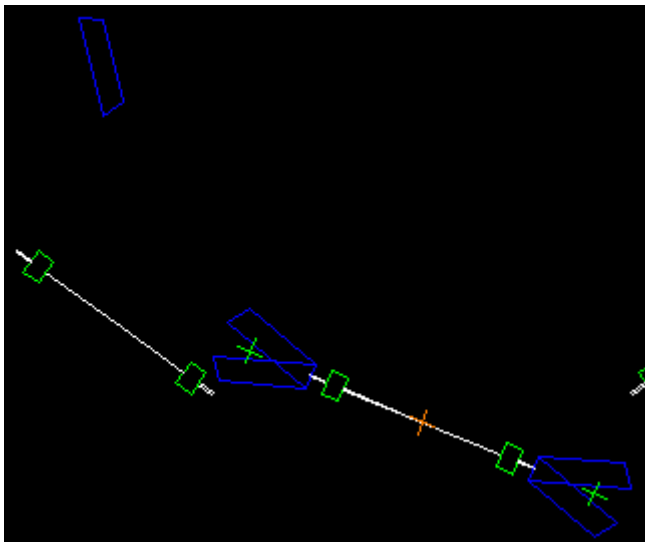


On the above image, the red and the cyan magnets are part of a single

girder. The origin of the girder frame is located in the middle of the red drift and displayed with an orange cross.

The cyan magnets form 2 siamese: one on the left and one on the right.
The array MIS(1:6) contains the actually misalignments: the translations in MIS(1:3) and the rotations in MIS(4:6). They are applied in the standard PTC order: rotation around the x, then y, and the z-axis, followed by the translation.

Now we rotate the girder by 22.5 degrees:



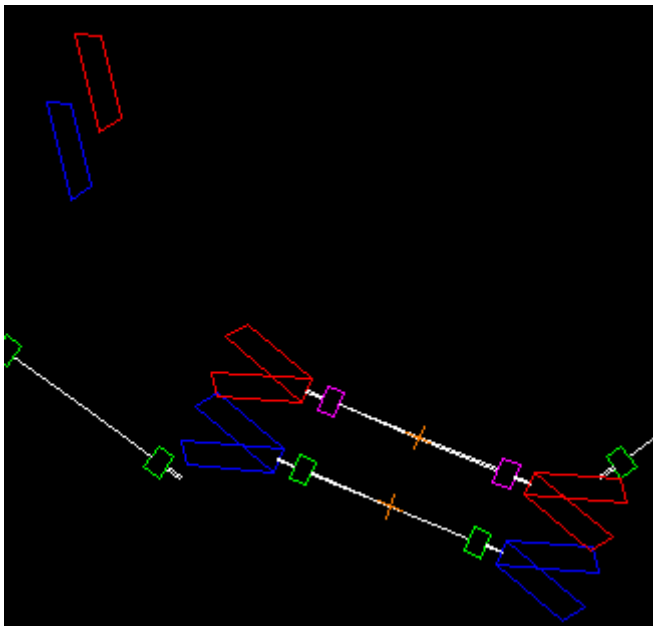
This was done with the command:

Example 1

```
MIS=0.d0
MIS(5)=PI/8.d0
CALL MISALIGN_GIRDER(B,MIS,ADD=.FALSE.)
```

If we follow this command by:

```
MIS=0.d0
MIS(1)=2.d0
CALL MISALIGN_GIRDER(B,MIS,ADD=.TRUE.)
```



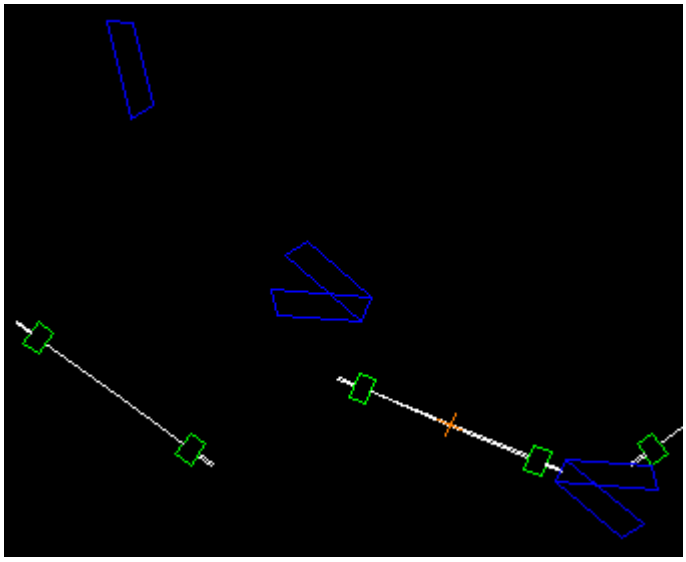
The ADD=.TRUE. indicates that the second misalignment of the girder is additive. One sees that the misalignment is in the direction of the rotated girder.

MISALIGN_SIAMESE (S2,S1,*OMEGA,BASIS,ADD,PRESERVE_GIRDER*)

Let us consider the following sequence of calls where the fibre pointer B is actually pointing to a member of the left siamese:

Example 2

```
MIS=0.d0
MIS(1)=2.d0
CALL MISALIGN_SIAMESE(B,MIS,ADD=.FALSE.)
MIS=0.d0
MIS(1)=2.d0
MIS(5)=PI/8.d0
CALL MISALIGN_GIRDER(B,MIS,ADD=.TRUE.)
```

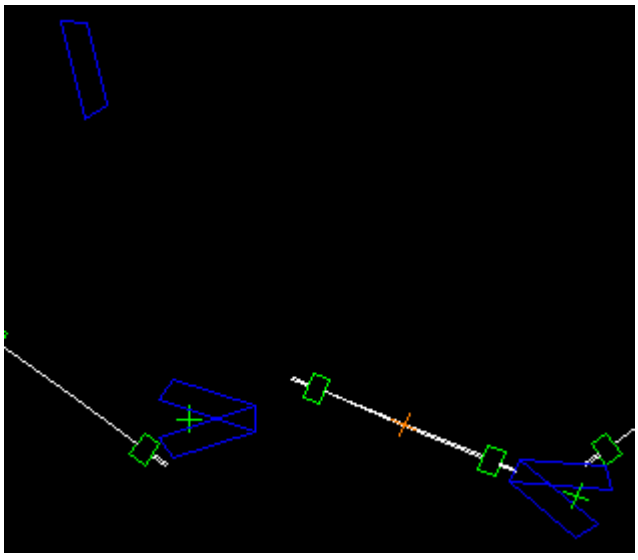


Now let us switch the order

Example 3

```
MIS=0.d0
MIS(1)=2.d0
MIS(5)=PI/8.d0
CALL MISALIGN_GIRDER(B,MIS,ADD=.FALSE.)
MIS=0.d0
MIS(1)=2.d0
CALL MISALIGN_SIAMESE(B,MIS,ADD=.FALSE.)
```

Here I purposely made ADD=.FALSE. on the siamese call since naively one expects the position to be relative to the girder on which it is attached. The results should be the same, but it is not as we see below.

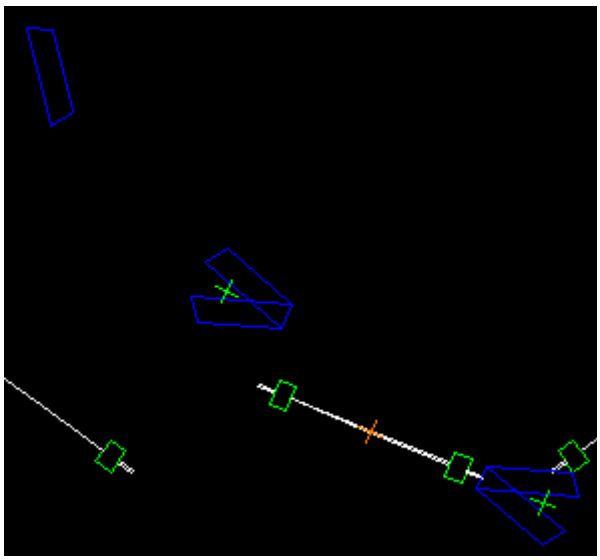


All the magnets store their effective misalignments in one array. Therefore the siamese has no knowledge of being on a girder. ADD=.FALSE. sends it back to its original fibre.
This can be avoided by the command:

Example 4

MIS=0.d0
MIS(1)=2.d0
CALL MISALIGN_SIAMESE(B,MIS,ADD=.FALSE.,PRESERVE_GIRDER=.TRUE.)

Here is the result



Obviously the same command with MIS=0.d0 in the siamese, always returns the siamese to its girder position, not fibre position.
MISALIGN_FIBRE(S2,S1,*OMEGA,BASIS,ADD,PRESERVE_GIRDER*)

This routine behaves like the siamese routine but acts on a single fibre.

N.B. If the misalign_fibre command is used on a siamese, it will mildly break the siamese. One can imagine very tiny errors internal to a siamese and bigger errors on the siamese and yet bigger error on a girder containing siamese and individual elements.

Therefore the following sequence of commands is acceptable if B is a element part of a siamese and part of a girder:

CALL MISALIGN_FIBRE(B,MIS1)
CALL MISALIGN_SIAMESE(B,MIS2,ADD=.TRUE.)
CALL MISALIGN_GIRDER(B,MIS3,ADD=.TRUE.)

Here we would imagine MIS1<MIS2<MIS3. Notice that ADD=.TRUE. on a siamese does not break a girder.

CALL MISALIGN_GIRDER(B,MIS3)
CALL MISALIGN_SIAMESE(B,MIS2,ADD=.FALSE. ,PRESERVE_GIRDER=.TRUE.)
CALL MISALIGN_FIBRE(B,MIS1,ADD=.TRUE.)

Item 3: Dynamical Routines

The ultimate goal of PTC or any tracking code is to propagate particles and maps thanks to Taylor Polymorphism. Placing geometrical objects in 3-dimension is worthless unless our rotations and translations can be translated into dynamical equivalents acting on rays (and on spin).

It is useful first to look at the Lie algebra acting on the t-based dynamics since it contains within it as a subgroup the affine part discussed above.

1) Exact Patching and Exact Misalignments: Dynamical Group

In PTC the patching and the misalignments are computed first geometrically. The connection between two affine frames is always expressed as follows through pure geometrical computations:

Connection = $T(d_x, d_y, d_z) \circ R_z \circ R_y \circ R_x$

This product of operators is in the usual matrix ordering. Therefore the rotation in the x-axis acts first, followed by the y-axis and so on. The rotations are computed using COMPUTE_ENTRANCE_ANGLE described in the first section of this note.

We factor the rotation in that manner because the operators R_x and R_y are

drifts in polar coordinates and are therefore *nonlinear*. For example, the rotation R_y is a pole face rotation, dubbed PROT by Dragt in the code MARYLIE. The formula is given by:

$$\begin{aligned}
\overline{x}\left(\alpha\right) &= \frac{x}{\cos\left(\alpha\right)\left(1-\frac{p_x}{p_z}\tan\left(\alpha\right)\right)} \text{ where } p_z = \sqrt{\left(1-2\frac{p_t}{\beta_0}+p_t^2\right)-p_x^2-p_y^2} \\
\overline{p_x}\left(\alpha\right) &= p_x\cos\left(\alpha\right)+\sin\left(\alpha\right)p_z \\
\overline{y}\left(\alpha\right) &= y+\frac{xp_y\tan\left(\alpha\right)}{p_z\left(1-\frac{p_x}{p_z}\tan\left(\alpha\right)\right)} \\
\overline{p}_y &= p_y \\
\overline{t} &= t+\frac{x\tan\left(\alpha\right)\left(\frac{1}{\beta_0}-p_t\right)}{p_z\left(1-\frac{p_x}{p_z}\tan\left(\alpha\right)\right)} \\
\overline{p}_t &= p_t
\end{aligned}$$

Here (t,p_t) form a canonical pair. In PTC ($-p_t$,t) form a canonical pair.

The rotation R_x is gotten from R_y by interchanging x and y. Both rotations rotate the magnet towards the direction of propagation, i.e., the z-direction.

The rotation R_z is the usual affine rotation along the z-axis. It is linear and transforms (x,y) and (p_x,p_y) identically leaving (t,p_t) untouched. It rotates the x-axis of the magnet towards its y-axis.

Actually, the Lie operators for the 3 translations and rotations are given by:

$$\begin{aligned}
:T_x: &= :p_x: \\
:T_y: &= :p_y: \\
:T_z: &= :\sqrt{\left(1+\delta\right)^2-p_x^2-p_y^2}: \\
:L_x: &= :y\sqrt{\left(1+\delta\right)^2-p_x^2-p_y^2}: \\
:L_y: &= :-x\sqrt{\left(1+\delta\right)^2-p_x^2-p_y^2}: \\
:L_z: &= :xp_y-yp_x:
\end{aligned}$$

It is remarkable that these Lie operators have the same Lie algebra as the ordinary affine (or time) Lie operators namely:

$$
\begin{aligned}
[L_x, L_y] &= L_z \\
[L_y, L_z] &= L_x \\
[L_z, L_x] &= L_y \\
[L_x, p_z] &= -p_y \\
[L_x, p_y] &= p_z \\
[L_y, p_x] &= -p_z \\
[L_y, p_z] &= p_x \\
[L_z, p_x] &= p_y \\
[L_z, p_y] &= -p_x
\end{aligned}
$$

Therefore the Lie groups are locally isomorphic. Of course one notices that "prot" (R_x and R_y) has a divergence at α=90 degrees. It is not possible in the "lens" or "s" paradigm to rotate a magnet map by 90 degrees and get meaningful propagators. Therefore the dynamical group is locally isormorphic.

2) Inexact Patching and Inexact Misalignments

PTC provides for an emasculated pseudo-Euclidean group. The Lie operators are obtained by expanding the dynamical maps keeping the energy dependence exact. This is in tune with the exact_model=false option of PTC.

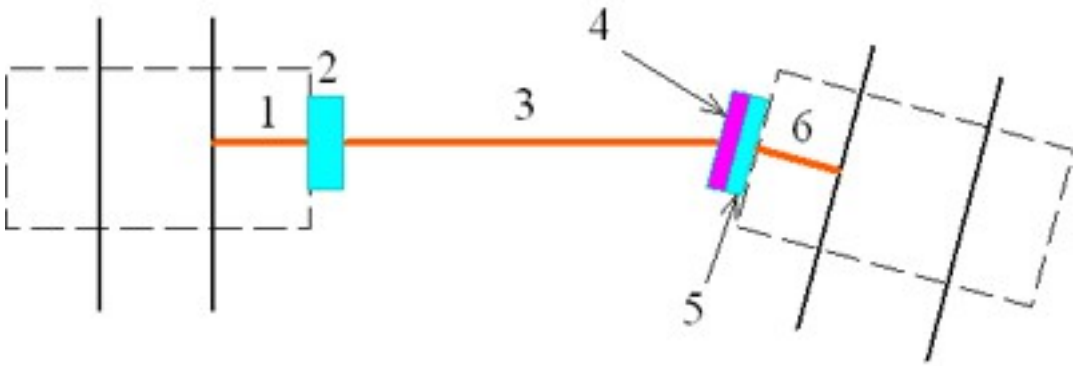
$$
\begin{aligned}
:T_x: &= :p_x: \\
:T_y: &= :p_y: \\
:T_z(r_1, r_2): &= :r_1 \underbrace{\left(-\frac{p_x^2 + p_y^2}{2\,(1+\delta)}\right)}_{D_z} + r_2\delta: \\
:L_x: &= :y\,(1+\delta): \\
:L_y: &= :-x\,(1+\delta): \\
:L_z: &= :xp_y - yp_x:
\end{aligned}
$$

This group has 7 generators for convenience. The Lie algebra differs from the original algebra of the Euclidean group as follows:

$$
\begin{aligned}
[L_x, L_y] &= 0 \\
[L_x, p_y] &= T_z(0,1) = \text{emasculated } p_z \\
[L_y, p_x] &= -T_z(0,1)
\end{aligned}
$$

Why do we care about the approximate Euclidean group?

The reason is speed: if a fast post processor to PTC is written. Let us assume that we have represented a (large) machine with exact_model=false and the drift-kick-drift option. Schematically, here are two magnets separated by a drift:



The drifts (1,3,6) are in orange. Some of the drifts (1,6) are part of the integration scheme. The misalignments are made of our 6 pseudo-Euclidean operators; they are the cyan (2,4). Finally a patch is needed (4) in purple.

The blue and cyan operators contain our 6 pseudo-Euclidian maps. Therefore to go from 1 to 6 in the figure may involve over 20 maps. By using the group properties, this can be reduced to 6 maps!

<u>BMAD example of Misaligned Rotated Magnet</u>

1) Pure Geometrical Method

The first approach uses pure geometrical routines of PTC: the misalignment is done once at the end.

One must follow the motion of an affine frame which exists in PTC as well

as the frame of BMAD.

```
!!!!!!!!!!!!!!!!   BMAD Misalignment   !!!!!!!!!!!!!!
!!! PTC Entrance frame from which the BMAD frame is computed
ent=p%chart%f%ent
! below is (o,mid) of PTC. We need to follow this frame
mid=p%mag%p%f%mid
o_chord=p%mag%p%f%o

 alphah=p%mag%p%ld*p%mag%p%b0/2   ! this is the half angle of the fibre

!exi    BMAD_frame = Rz(-tiltd)Ry(α/2)Rz(tiltd)ENT
exi=ent
ang=0
ang(3)=p%mag%p%tiltd
call GEO_ROT(exi,ang,1,exi)
ang=0
ang(2)=alphah
call GEO_ROT(exi,ang,1,exi)   ! Here exi is the frame p%mag%p%f%mid of PTC
ang=0
ang(3)=-p%mag%p%tiltd
call GEO_ROT(exi,ang,1,exi) ! Here BMAD derotates immediately (PTC at the end only)

!  PTC must know the new chord point
! it is geometrically computed in terms of BMAD's frame just computed

x=0; x(1)=-offset_x ; x(2)=-offset_y; x(3)=offset_z
do i=1,3
 o_chord=o_chord+x(i)*exi(i,1:3)   ! final displacement using BMAD frame
enddo

! rolling along the BMAD z-axis
x=0; x(6)=roll;
call GEO_ROT(mid,x(4:6),1,exi) !  roll PTC's mid frame
call GEO_ROT(exi,x(4:6),1,exi) !  roll BMAD's mid frame
```

```
!  next we roll around the arc which is not a PTC saved point
o_arc=o_chord+d_o*mid(1,1:3)    ! BMAD pitches around the arc, saving it

x=0; x(4)=-pitch_x;
call GEO_ROT(mid,x(4:6),1,exi) !  roll PTC's mid
call GEO_ROT(exi,x(4:6),1,exi) !  roll BMAD's mid
o_chord=-d_o*mid(1,1:3)   ! chord with respect to arc
o_chord=o_chord+o_arc   ! chord in global frame
!  the new chord must be computed for PTC

 x=0;x(5)=-pitch_y;
call GEO_ROT(mid,x(4:6),1,exi) !  roll PTC's mid in BMAD frame
call GEO_ROT(exi,x(4:6),1,exi) !  roll BMAD's mid in BMAD frame

o_chord=-d_o*mid(1,1:3)   ! chord with respect to arc
o_chord=o_chord+o_arc   ! chord in global frame

!  angles from original mid to final mid are computed and put in x(4:6)
call COMPUTE_ENTRANCE_ANGLE(p%mag%p%f%mid,mid,x(4:6))

!  The translation of the chord must be computed in BMAD's frame
! i.e., the final frame
x(1:3)=o_chord-p%mag%p%f%o
CALL CHANGE_BASIS(x(1:3),GLOBAL_FRAME,x(1:3),mid);

!  Now PTC computes the actual misalignment
CALL MISALIGN_FIBRE(p,x,p%mag%p%f%o,p%mag%p%f%mid)
```

2) Using Misalign command as much as possible

The second approach we incrementally moved the magnet in PTC itself, instead of finding a single “grand” misalignment.

The part in red below is identical to the method described in 1). It simply computes the mid-frame of BMAD.

```
ent=p%chart%f%ent
mid=p%mag%p%f%mid
o_chord=p%mag%p%f%o
```

```
 alphah=p%mag%p%ld*p%mag%p%b0/2

write(6,*) " alphah ",alphah,p%mag%p%tiltd

exi=ent
ang=0
ang(3)=p%mag%p%tiltd
call GEO_ROT(exi,ang,1,exi)
ang=0
ang(2)=alphah
call GEO_ROT(exi,ang,1,exi)
ang=0
ang(3)=-p%mag%p%tiltd
call GEO_ROT(exi,ang,1,exi)

!  translating the magnet
x=0; x(1)=-offset_x; x(2)=-offset_y; x(3)=offset_z;
CALL MISALIGN_FIBRE(p,x,p%mag%p%f%o,basis=exi)

! roll around the chord (automatic in PTC) in BMAD frame
x=0;x(6)=roll;
CALL MISALIGN_FIBRE(p,x,p%mag%p%f%o,basis=exi,ADD=ADDIN)
call GEO_ROT(exi,x(4:6),1,exi)

!  find the arc point (not in PTC's magnet frame )
d_o=0
if(p%mag%p%b0/=0.0_dp) then
 d_o=1.0_dp/p%mag%p%b0
if(d_o>0) then
 d_o= d_o-sqrt(d_o**2 -(p%mag%p%lc/2.0_dp)**2)
 else
 d_o= d_o+sqrt(d_o**2 -(p%mag%p%lc/2.0_dp)**2)
endif
endif
```

```
o_arc=p%mag%p%f%o+d_o*p%mag%p%f%mid(1,1:3)

!  x pitch around the arc
x=0; x(4)=-pitch_x;
CALL MISALIGN_FIBRE(p,x,o_arc,basis=exi,ADD=ADDIN)
!  Bmad frame new orientation is computed
call GEO_ROT(exi,x(4:6),1,exi)
!  y pitch arounf the arc
x=0; x(5)=-pitch_y;
CALL MISALIGN_FIBRE(p,x,o_arc,basis=exi,ADD=ADDIN)
```

This completes the procedure.

BMAD Floor Element

The purpose of the floor element is to host a Taylor map which connects to parts of a lattice potentially far from each other.

Thus a floor element has patch which is only active in the survey command : it has not dynamical effect.

PTC example:

```
floor_fibre=>als%end

CALL FIND_PATCH(floor_fibre%chart%f%b,floor_fibre%chart%f%exi, &
als%start%chart%f%a,als%start%chart%f%ent, &
floor_fibre%patch%b_d,floor_fibre%patch%b_ANG)
floor_fibre%patch%track=.false.
floor_fibre%patch%patch=2
```

In this example, the end of the lattice is a marker. It is connected to the first element. Notice that two elements anywhere in the lattice could be connected to each other.

floor_fibre%patch%track=.false. Removes the dynamical tracking.