# A Lexical Hierarchical Taxonomy of Gradient-based Changes for Cantilever-Snap Assembly Verification

----------------------------------------------------------------------------------------------------------------------------------------------------------------

## Juan Rojas

----------------------------------------------------------------------------------------------------------------------------------------------------------------

## February 2012

----------------------------------------------------------------------------------------------------------------------------------------------------------------  
  
**Updated July 2012**

----------------------------------------------------------------------------------------------------------------------------------------------------------------

**Hierarchical Taxonomy**

1. Primitive Classification.
2. Composites Classification.
3. Low-level Behavior Classification.
4. High-level Behavior Classification.
5. Success Verification Layer

BNeg

BPos

**1. Primitive Classification**

Primitives are classified based on degrees of change (gradients). The values are established empirically. They give a rough idea of degree, but their compositions, shown later will be more critical. The primitive classification is represented by the following strings and gradient values (see GradientClassification.m):

pimp =1000.0; nimp = -1\*pimp;   
bpos = 100.0; bneg = -1\*bpos;  
mpos = 10.0; mneg = -1\*mpos;  
spos = 1.0; sneg = -1\*spos;  
zero = 0.0;

* ‘pimp: m >= 1000
* ‘bpos’: 100.0 =< m < 1000.0
* ‘mpos’: 10.0 =< m < 100.0
* ‘spos’: 1.0 =< m < 10.0
* ‘nimp: m<=-1000
* ‘bneg’: -1000.0 < m =< -100.0
* ‘mneg’: -100.0 < m =< -10.0
* ‘sneg’: -10 < m =< -1.0
* ‘const’: 0.0 =< m < |0.2|

**Gradient Classification Optimization (July 2012)**

A new optimization method or calibration method was implemented. This method consists in using the Pivot Approach to implement the snap assembly for the first time between a given robot and part. If the assembly is successful we can partition the task into four automata states: Approach, Rotation, Snap, and Mating.

The process the consists in assigning **CONST** and **PIMP** automatically for each of the six force axes. The method for computing PIMP values is described first and currently applies for the 4-snap cantilever snap. For the forces in the x, y, and z directions, the Pivot Approach is really concerned with the forces in the X and Z directions. For this reason we only wish to compute the PIMP gradients for these two axes. It has been determined to simply copy the value of the PIMP gradient from the z-axes into the y-axes.

*PIMP/NIMP*

To compute the PIMP we look at the signals within the Snap state. Therein, we look for the maximum value gradient. We take this as our rule for measuring PIMPs. To insure some degree of flexibility, we use a value 10% smaller than the one registered in the signal. Hence, PIMP = 0.9\*PIMP. NIMP = -PIMP.

*CONST*

For CONST we separately compute this gradient for all six axes. To compute it we look at the signals in the Rotation state and compute the average value of the signals there.

**Statistical Data Structure**

The information of each primitive is saved in the statData structure of type cell array, which holds 7 different elements of data for each primitive (see StatisticalData.m):

statData = {

* Average value of primitivfe
* Maximum value of primitive
* Minimum value of primitive
* Start time of primitive
* End time of primitive
* Gradient value of primitive
* Gradient label for primitive

}

**statData – cell to vector - and Gradient Labels - Strings to Int’s**

For easing conversion between Matlab and C++, statData was converted from a CELL array to a vector array by converting gradient string labels into integer classifications. The following labels were converted:   
[ bpos, mpos, spos, bneg, mneg, sneg, pimp, nimp, none] => [1 2 3 4 5 6 7 8 9 10].

**A. Primitives Clean Up (Filtering for Segments**

**In cases where, with two neighboring segments, if one is more than 5 times longer than the other, and it’s amplitude is not 5 times bigger than the other, then merge them.**

**2. Composites Classification**

The composites are classified based on the sequence of two primitives. A window that examines:

* Every three primitives for positive and negative gradients are used to see if the sequence exists and gives a slight margin for error.
* Every two primitives for constant and impulse gradients are used.

The classification examines the gradients of the pair of primitives (positive, negative, constant, or impulse gradients), and assigns one of following six types of actions (defined in primMatchEval.m):

Action class {

‘a’, adjustment. A positive gradient followed by a negative gradient, or vice versa.

‘i', increase. A positive gradient followed by a positive gradient, or constant.

‘d’, decrease. A negative gradient followed by a negative gradient or constant.

‘k’, constant. A single or repeated occurrence.

‘pc’, pos contact. ‘i'||’d’||’c’ + pimp

‘nc’, neg contact ‘i'||’d’||’c’ + nimp

‘c' contact (impulse). A single or repeated occurrence.

‘u’ unstable. Impulse gradient followed by growing or constant gradient, unstable.

‘n’ noisy. Used for low-level behaviors when no match is possible.

‘z’ none. Should not be called.

}

*actionLbl Cell to Int Vector*

July 2012: The above cell array was converted to an int vector with values [1,2,3,4,5,6,7,8] corresponding to {a,i,d,k,pc,nc,c,u}

1. Positive
   1. Negative: adjustment, ‘a’.
   2. Positive: increase, ‘i'.
   3. Constant: increase, ‘i'
   4. Pimp: pos contact, ‘pc’
   5. Nimp: neg contact, ‘nc’
2. Negative
   1. Positive: adjustment, ‘a’.
   2. Negative: decrease, ‘d’.
   3. Constant: decrease, ‘d’.
   4. Pimp: pos contact, ‘pc’
   5. Nimp: neg contact, ‘nc’
3. Constant
   1. Positive: increase, ‘i'.
   2. Negative: decrease, ‘d’.
   3. Constant: constant, ‘k’.
   4. Pimp: pos contact, ‘pc’
   5. Nimp: neg contact, ‘nc’
4. Pimp
   1. Positive: pos contact, ‘pc’.
   2. Negative: pos contact, ‘pc’.
   3. Constant: pos contact, ‘pc.
   4. Pimp: unstable, ‘u’
   5. Nimp: contact, ‘c’.
5. Nimp
   1. Positive: neg contact, ‘nc’.
   2. Negative: neg contact, ‘nc’.
   3. Constant: neg contact, ‘nc’.
   4. Pimp: contact, ‘c’.
   5. Nimp: unstable, ‘u’

Each motion composition has a struct of type CELL ARRAY with 11 elements of information relevant to it (defined in primMatchEval.m):

motComps = {

actionClass: kind of action  
avgVal: average value of primitive pair  
rmsVal: root mean square value of primitive pair  
amplitude: max – min value of primitive pair  
gLabel1: gradient label for first primitive  
gLabel2: gradient label for second primitive  
t1Start: time at which first primitive occurs  
t1End: time at which first primitive ends  
t2Start: time at which second primitive starts  
t2End: time at which second primitive ends  
t2Index: time at which second primitive occurs  
tAvgIndex: avg. time or middle time of primitive

A. Motion Composition Clean Up (Filtering at 2nd Level)

The clean-up phase consists of three steps that filter less significant signals. To do so, compositions are analyzed under a couple of contexts: (1) composition’s time duration, (2) composition’s amplitude magnitude, and (3) composition repetition patterns.

The number of iterations that the clean-up stage can be run may vary according to the environment. Currently, we run it 3 times on the Pivot Approach with the PA-10 as well as HIRO.

Time Duration Context

1. If there is a composition (except any type of contact) where one of the two primitives that compose it is 7x smaller than the other (length), and the second composition is less than 5 times the amplitude, then merge by changing the action composition label to correspond to the longer primitive:
   * bpos/mpos/spos becomes ‘i'.
   * bneg/mneg/sneg becomes ‘d’.
   * const becomes ‘k’.

Repeated Compositions

1. For all alignments found in states 3 and 4 merge them as alignments.
2. For all compositions, that are not alignments, if they are repeated, then merge them.

Amplitude Value Context

The analysis of amplitude value context pertains to the formation of adjustment signals.

1. If there are contiguous primitives of types **PC/NC** or **NC/PC**, AND if:
   * Their amplitude is 10x smaller than the largest amplitude registered in the assembly, then treat them as an amplitude.
2. If there is an **adjustment** followed by either an **increase** or a **decrease** with similar amplitude (50%) and an average value within (100%) merge them into an adjustment.
3. If there is a **ik/ki** or **dk/kd** with similar amplitude (150%) and an average value within 100% of each other, merge as constant, k.

**3. Low-Level Behaviors**

This level combines motion composition actions and looks at their duration to produce lower-level behaviors in states in which there is contact between parts. This level compares degrees of change across two motion compositions. There are six low-level behaviors which are defined as follows:

* Fixed
  + Def: Occurs when the part is no longer moving and remains fixed in place.
  + Conditions: No change and a total minimum of time 0.10 seconds.
  + Sequence of mot. Comps: { k, kk}
* Contact
  + Def: Occurs when the male part hits the back wall of the female part
  + Conditions: any action, except for an unstable one, followed by a contact, followed by any action except an unstable action.
  + Sequence of mot. Comps: { \*c\* or \*cc\* or pc/nc, nc/pc}
* Pushing
  + Def: Occurs when force is applied and force rises.
  + Conditions: action is produced by one or more decrement actions and a total minimum of time 0.10 seconds.
  + Sequence of mot. Comps: {i,ii}. Changed meaning from {d,dd} to {i,ii} in 2013Aug.
* Pulling
  + Def: Occurs when force is withdrawn and less force is sensed.
  + Conditions: action is produced by one or more increment actions and a total minimum of time 0.25 seconds.
  + Sequence of mot. Comps: {d,dd}. Changed meaning {i,ii} from to {d,dd} in 2013Aug.
* Aligning
  + Def: Occurs when contiguous adjustments have **smaller** amplitudes. Can use a window of sz 1, if element of short duration (noisy) is present in between.
  + Conditions: action is produced by increasingly smaller adjustments.
  + Sequence of mot. Comps: {aa}.
* Shift
  + Def: Occurs when contiguous adjustments **increase** in value indicated small alignments over time.
  + Conditions: action is produced by increasingly smaller adjustments and a total minimum of time 0.25 seconds.
  + Sequence of mot. Comps: {s,ss}.
* Noise
  + If any of the previous ones cannot be recognized.

Update - July 2012

The LL behaviors were converted from string labels to integer representations to facilitate the conversion between matlab and c++. Hence we have the following rep:

llbehLbl = {'FX' 'CT' 'PS' 'PL' 'AL' 'SH' 'U' 'N'}; % {'fix' 'cont' 'push' 'pull' 'align' 'shift' 'unstable' 'noise'};  
 llbehLbl = [ 1, 2, 3, 4, 5, 6, 7, 8];

*Refinement*

The refinement phase consists of three steps that filter less significant signals. To do so, compositions are analyzed under three contexts: (1) composition’s time duration, (2) a composition’s amplitude magnitude, and (3) repetition patterns.

The number of iterations that the Refinement stage can be run may vary according to the environment. Currently, we run it 4 times on the Pivot Approach with the PA-10 as well as HIRO.

Time Value Context

1. If there is a composition (except any type of contact) where one of the two primitives that compose it is 5x smaller than the other, then merge by changing the action composition label to correspond to the longer primitive:

Amplitude Value Context

The analysis of amplitude value context pertains to the formation of alignment signals.

1. If there is **push-pull** or **pull-push** of similar amplitude (150%) and similar average value (100%), then turn into an ALIGNMENT.
2. If there is a SHIFT followed by an ALIGNMENT or an alignment followed by a shift where the second behavior has a smaller amplitude, then merge into an alignment.
3. If there is a pattern ALIGN/PS, ALIGN/PL or in reverse-order which have about similar amplitude (50%) and similar average value (100%), merge as ALIGN.
   1. Same as above with SHIFT.

Repeated Patterns

1. If there are any repeated signals merge.

**4. High-Level Behaviors**

The fourth layer of the taxonomy looks one state-at-a-time (context specific) across all six force-moment low-level behaviors to produce high-level behaviors. The high-level behaviors represent human-apropos behaviors associated with each stage of the pivot assembly process. Here is a list across states:

* State 1: Approach Y/N
* State 2: Rotation Y/N
* State 3: Alignment Y/N
* State 4: Snap Y/N
* State 5: Mating Y/N

Each of the high-level behaviors requires a specific combination of low-level behaviors across the different force-elements but not necessarily all of them. The key is that if certain key low-level behaviors are present, the presence of the high-level behavior can be ascertained.

**Note**: *for State 1 .*

*State 1: given that in State 1 the mating parts do not contact each other, we will not try to interpret this information to determine if the approach proceeds successfully. If, however, a rotation can be ascertained in state 2, then we can safely state that the approach has taken place.*

Here is the list of necessary state-sensitive low-level behavior requirements. In other words, if the low-level behaviors are present (or a sequence of them are present) then, we have a higher-level behavior. If not, we have the negative form of the high-level behavior.

(Other possible considerations to determine hlBeh: Total number or llbeh’s per state? Duration? Final stage? Single presence of a behavior?)

* Rotation:
  + Fz-> FX (with value not equal to zero)
  + Fy -> PL
  + Mx -> ALIGN
* Alignment
  + AL to show up in all axes (in our present case Fxyz, Mxyz). However, the moment axis corresponding to the direction of motion in which the insertion is taking place (Mz) could have just a FX reference or ALIGN->FX instead.
* Snap
  + Fz – CT+AL
  + FxFyMxMyMz = ALIGN+FX || FX
* Mating

This layer has a struc that lists the low-level behaviors contained in each state for each force axis

hlbehStruc = {

stateLbl2{ Fx{} … Mz{} }   
stateLbl3{ Fx{} … Mz{} }   
stateLbl4{ Fx{} … Mz{} }

}