IN480 Larstruct Module

Dèsirèe Adiutori Alessia Giulia Cossu

May 21, 2018

Contents

Introduction

This module of LAR-CC library is about a hierarchical structures with LAR. Hierarchical models of complex assemblies are generated by an aggregation of subassemblies, each one defined in a local coordinate system, and relocated by affine transformations of coordinates.

In this module there are:

- The **Affine transformations** are based on elementary matrices for affine transformation of vectors in any dimensional vector space, including translation, scaling and rotation;
- The **Struct iterable class**, starting from an array representable geometric object, generates a new object representing the initial one in an alternative way, by means of specific fields attribution, e.g. body, box, etc..
- The **structure to LAR conversion**
- The **hierarchical complexes**

1 Implementation

In questa sezione viene mostrata la traduzione, di alcune delle funzioni del modulo piu importanti, dal linguaggio python a julia.

1.1 checkStruct

1.1.1 Conversion

Python

```
def checkStruct(lst):
   obj = lst[0]
   if(isinstance(obj,tuple) or isinstance(obj,list)):
      dim = len(obj[0][0])
   elif isinstance(obj,Model):
      dim = obj.n
   elif isinstance(obj,Mat):
      dim = obj.shape[0]-1
   elif isinstance(obj,Struct):
      dim = len(obj.box[0])
   return dim
```

Julia

```
function checkStruct(lst)
   obj = lst[1]
   if isa(obj,Matrix)
      dim=size(obj)[1]-1
   elseif(isa(obj,Tuple) || isa(obj,Array))
      dim=length(obj[1][1])
   elseif isa(obj,Struct)
      dim=length(obj.box[1])
   end
   return dim
end
```

1.1.2 Parallelization

1.1.3 Unit-Test

```
@testset "checkStruct" begin
  list=([[0.575,-0.175],[0.575,0.175],[0.925,-0.175],[0.925,0.175]],[[0,1,2,3]])
  @test checkStruct(list)==length(list[1][1][1])
  @test typeof(checkStruct(list))==Int
end
```

1.2 Traversal

1.2.1 Conversion

Python

```
def traversal(CTM, stack, obj, scene=[]):
   for i in range(len(obj)):
      if isinstance(obj[i],Model):
         scene += [larApply(CTM)(obj[i])]
      elif(isinstance(obj[i],tuple) or isinstance(obj[i],list)) and
         (len(obj[i]==2 or len(obj[i])==3):
         scene += [larApply(CTM)(obj[i])]
      elif isinstance(obj[i],Mat):
         CTM = scipy.dot(CTM, obj[i])
      elif isinstance(obj[i],Struct):
         stack.append(CTM)
         traversal(CTM, stack, obj[i], scene)
         CTM = stack.pop()
   return scene
```

Julia

```
function traversal(CTM,stack,obj,scene=[])
   for i in range(1,len(obj))
      if isa(obj.body[i],Matrix)
         CTM=CTM*obj.body[i]
      elseif (isa(obj.body[i],Tuple) || isa(obj.body[i],Array)) && (length(obj.b
         l=larApply(CTM)(obj.body[i])
         push!(scene,l)
      elseif isa(obj.body[i],Struct)
         push!(stack,CTM)
         traversal(CTM,stack,obj.body[i],scene)
         CTM=pop!(stack)
      end
   end
   return scene
end
```

1.2.2 Parallelization

1.2.3 Unit-Test

```
@testset "traversal" begin
   square=([[0, 0], [0, 1], [1, 0], [1, 1]], [[0, 1, 2, 3]])
   @everywhere structure=Struct([square])
   @everywhere dim=checkStruct(structure.body)
   @test length(traversal(eye(dim+1),[],structure,[]))==length(structure.body)
   @test typeof(traversal(eye(dim+1),[],structure,[]))==Array{Any,1}
end
```

1.3 larApply

1.3.1 Conversion

Python

```
def larApply(affineMatrix):
    def larApply0(model):
        if isinstance(model,Model):
            V = scipy.dot(array([v+[1.0] for v in model.verts]), affineMatrix.T)
            V = [v[:-1] for v in V]
            CV = copy.copy(model.cells)
            return Model((V,CV))
        elif isinstance(model,tuple) or isinstance(model,list):
            if len(model)==2: V,CV = model
            elif len(model)==3: V,CV,FV = model
            V = scipy.dot([list(v)+[1.0] for v in V], affineMatrix.T).tolist()
            if len(model)==2: return [v[:-1] for v in V],CV
            elif len(model)==3: return [v[:-1] for v in V],CV,FV
    return larApply0
```

Julia

```
function larApply(affineMatrix)
   function larApply0(model)
```

```
      if length(model)==2
         V,CV=model
      elseif length(model)==3
         V,CV,FV = model
      end
      V1=Array{Float64}[]
      for (k,v) in enumerate(V)
         append!(v,[1.0])
         push!(V1,vec((v')*transpose(affineMatrix)))
         pop!(V[k])
         pop!(V1[k])
   end
      if length(model)==2
         return V1,CV
      elseif length(model)==3
         return V1,CV,FV
      end
   end
   return larApply0
end
```

1.3.2 Parallelization

1.3.3 Unit-Test

```
square=([[0, 0], [0, 1], [1, 0], [1, 1]], [[0, 1, 2, 3]])
cubes=([[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1, 0, 1], [1, 1,

@testset "larApply Tests" begin
   @testset "2D" begin
      @testset "larApply Translation 2D" begin
      @test typeof(larApply(t(-0.5,-0.5))(square))==Tuple{Array{Array{Float64,N}
      @test larApply(t(-0.5,-0.5))(square)==([[-0.5, -0.5], [-0.5, 0.5], [0.5, -
   end
      @testset "larApply Scaling 2D" begin
         @test typeof(larApply(s(-0.5,-0.5))(square))==Tuple{Array{Array{Float64
         @test larApply(s(-0.5,-0.5))(square)==([[0.0,0.0],[0.0,-0.5],[-0.5,0.0]
      end
      @testset "larApply Rotation 2D" begin
```

```
         @test typeof(larApply(r(0))(square))==Tuple{Array{Array{Float64,N} wher
         @test larApply(r(0))(square)==square
      end
   end

   @testset "3D" begin
      @testset "larApply Translation 3D" begin
         @test typeof(larApply(t(-0.5,-0.5,-0.5))(cubes))==Tuple{Array{Array{Flo
         @test larApply(t(-0.5,-0.5,-0.5))(cubes)==([[-0.5, -0.5, -0.5], [-0.5,
      end

      @testset "larApply Scaling 3D" begin
         @test typeof(larApply(s(-0.5,-0.5,-0.5))(cubes))==Tuple{Array{Array{Flo
         @test larApply(s(-0.5,-0.5,-0.5))(cubes)==([[0.0, 0.0, 0.0], [0.0, 0.0,
      end

      @testset "larApply Rotation 3D" begin
         @test typeof(larApply(r(pi,0,0))(cubes))==Tuple{Array{Array{Float64,N} 
         @test larApply(r(pi,0,0))(cubes)[1]?[[0.0, 0.0, 0.0], [0.0, -1.22465e-1
      end
   end
end
```

1.4

1.4.1 Conversion

Python

julia

1.4.2 Parallelization

1.4.3 Unit-Test

2 Conclusion