EECS 287 Project 2 - Kinematics

In this project you will work on a topic related to kinematics and/or motion data processing. You may use as starting point one of the example projects *exmovegraph* or *exmotionplayer*.

Requirements

1. Your project will need to consider a **hierarchy of many joints** (like more than ten). You may use one of the available skeletons in the example projects, or import or build your own.

2. Your project has to address an interesting kinematics-related problem and you have to **provide your own implementation solution for one key part of the problem**. This requirement is here to avoid that you develop a project that is merely using available functions in SIG. Include at least one non-trivial solution/algorithm that you implemented yourself.

3. You will also need to **evaluate your main algorithm/solution**, and you will describe your evaluation in a 1-page project report. Your report is not a summary of everything your project does, think of it as a poster highlighting the most interesting thing your project can do. (You will write in a reame.txt file how I can see what your program can do.)

The Support Code

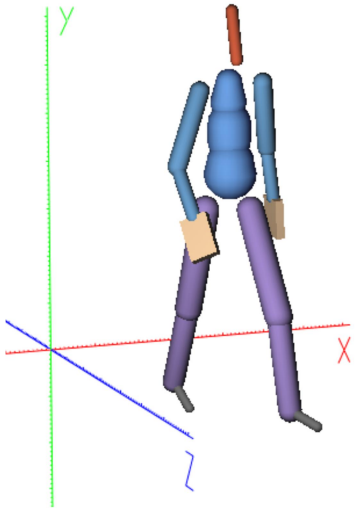
First of all study the support code: *exmotionplayer* is a very simple project showing how to load a skeleton and a motion, and showing how to play the motion between two given key times.

Project *exmovegraph* shows additional operations for concatenating multiple motions, including how to declare postures and interpolate them for achieving a simple concatenation blending between motions. The project implements a move graph structure and you should study it per parts: in the main.cpp file the variable mainloop can be set to 1, 2, 3, or 4 in order to specify which mainloop to use. Each one adds more functionality than the previous one, so study one at a time to see where/how the new functionality was added.

Project *exmotionplayer* is mostly implemented inside methods of the MyViewer class, which is the recommended way to organize your code. Project *exmovegraph* has most of the operations in main.cpp so that it is easier to see what is happening in a single function. However, its interaction with the viewer (to get keys and mouse events) is less intuitive. Use the best approach for your development when you start working on your project.

Some concrete topics are given below but note that you are free to build your own skeleton, capture your own motions using the motion capture suits in our lab, or also import skeletons/motions from the web or from mixamo.com. (SIG also has an importer for fbx files, in ascii format, but this feature has not been tested much.)

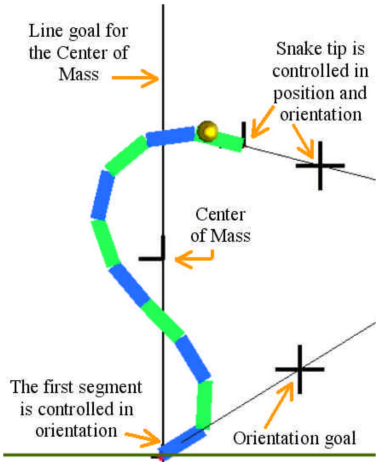
Recommended Topics



1. Move-Graph with Keyboard Control: complement the move graph example in *exmovegraph* to include all of the motions in file data/fullmovegraphdata.7z, and then add keyboard controls to precisely steer the character in the following way:
- when the Up key is pressed the character walks,
- when Shift+Up is pressed the character runs,
- when the Down key is pressed the character turns 180 degrees, and
- when the Left/Right arrows are pressed you can increment rotations to the global root rotation in order to slightly deform the straight walk animation to the left or to the right, in a smooth way.

As you will see, achieving perfect results is not trivial. Make a scene with many boxes and see if your keyboard control is precise enough to navigate around the boxes while still achieving smooth animations. You will have several details to think about related to how to control your main loop and your blending operations. You may also add more keys to show the other animations available in fullmovegraphdata.7z.

2. Walking controller from single motion: in this topic you use *exmotionplayer* and you will start by taking note of initial and final times in the included motion for delimiting submotions that can be used for walking straight, turning left, and turning right. You will then use similar keys as described in the previous topic in order to control the character moving around. To have good results you will need to blend between the consecutive submotions you want to concatenate. For this you can work with KnPosture to capture postures and interpolate them before sending the result to the skeleton. Study how this can be done in *exmovegraph*. Here the advantage is that each submotion will have extra frames before and after the submotion, and these extra frames can be used for a proper transition blending. Review the lecture slides on this topic.
Once your implementation starts to work you can also make a scene with many boxes and see if your key control is precise enough to navigate around the cubes while still achieving a smooth animation.



3. Inverse kinematics: in this topic you can define a skeleton of your own by editing/creating a .s file and then you will develop your own Inverse Kinematics solver. The simplest one to implement is based on CCD iterations, but you may decide to implement and explore any method. In order to test it, use a SnManipulator to interactively move a goal sphere to be reached by your end-effector. At the end of this document an example of a simple skeleton file and code fragments illustrating how to access joints are given. Study also the several .s files available in the data folder inside *exmotionplayer* and *exmovegraph*.

4. Other topics: There are many possibilities, for example to compute the similarity image (Fig. 3) of the Motion Graph's paper for the motion in *exmotionplayer*, or to define your own topic based on the requirements listed in the first page - just please discuss your idea with me before committing to it.

Grading

75% - basic requirements and overall quality of your project,
15% - your report (see specific report instructions below),
5% - your report has 1 page only,
5% - reame.txt file explaining how I can see everything your program can do.

Submission

Submit by the deadline a 7zip/zip file with: 1) project code (**do a cleanall before!**) and 2) report.

Report

The report is a **1-page report** in the format described by the template below. The report will have a few paragraphs and at least one image. Note that you have to describe an evaluation of your main algorithm/solution. You have to make everything fit in 1 page, which means you have to put in the report only the most relevant/interesting results of your project. You may use any layout (like 2-columns) as long as it fits in 1 page.

Project Title:
Name and email:

1) Project Description and Goal
Explain here your project.

2) Evaluation and Results
Comment here your main algorithm(s)/solution(s) implemented to achieve the results of your project. For example: describe the key solutions you implemented for your blending and key control to work well, and evaluate in which conditions/situations it works well and when it does not work. The goal is to describe here the most interesting capabilities of your project, and to evaluate them. Use at least one image to illustrate results. Your evaluation may be based on showing images of cases working well and cases not working.

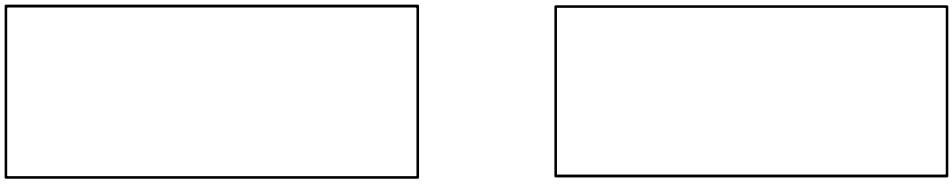


Image 1 – Illustrative image(s). You can put the image(s) anywhere and in any layout.

3) Conclusion
Write here the main conclusion(s) you have achieved as a result of your evaluation.

Simple example .s skeleton file:

```
KnSkeleton

name simplesk

skeleton
root pos
{
  visgeo primitive capsule 1 1 4  10 color 255 0 0 255;

  joint hand
   { offset 0 4 0
     channel XRot 0 free
     channel YRot 0 free
     channel ZRot 0 free
     visgeo primitive capsule 1 1 2  10 center 0 2 0 color 0 0 255 255;
   }

}
```

Example instructions to load and access the hand joint rotations in the skeleton:

```
   // Add one skeleton:
   KnSkeleton* sk = new KnSkeleton;
   sk->load ( "../data/simplesk.s" );
   KnScene* sc = new KnScene;
   sc->connect ( sk );
   root()->add ( sc );

   // Example main loop moving the hand:
   handx = 0;
   handy = 0;
   handz = 0;
   float incang = GS_TORAD( 5 ); // degrees->radians convertion
   while ( true )
    {
      sk->joint("hand")->euler()->value(0,handx);
      sk->joint("hand")->euler()->value(1,handy);
      sk->joint("hand")->euler()->value(2,handz);
      sc->update ();

      render();
      ws_check(); // process user interface events, draw

      handx+=incang;
      handy+=incang;
      handz+=incang;
   }
```