3451 Project 3 2016

Tango Dancer

Objectives: Gain expertise in the derivation of mathematical formulation and in the implementation of a computational model of a responsive behavior of an articulated human character in 3D. The particular behavior chosen for this project is one of a tango dancer, because of its strict rules and yet expressive/creative freedom. The project will provide opportunities for learning, understanding, and using 3D geometric constructions (involving dot, cross, and mixed products), inverse kinematics (in the form of 3 and 4 bar linkages), path planning (for dancer's feet), motion smoothing, and 3D graphic interaction and rendering.

Two parts: Project 3 has two parts (A and B), each with a different deadline. Part A is to be performed, implemented, written, and submitted **individually** by each student without any collaboration. It is highly scripted with very precise rubrics and guidelines. Part B is to be performed, implemented, and written up either individually or in teams of two (with additional requirements for teams of two). The maximum grade points will be equally divided between the two parts (not counting extra credit).

Modules: To guide implementation, simplify grading, and increase fairness and transparency, Part A will be broken into several modules, each proposing a specific problem that the student must solve. In the submitted solution: each module must be

(1) clearly identified in the source code,

(2) clearly documented in a well identified section of the report

(3) illustrated with pictures from the students' program plus diagrams as needed,

(4) demonstrated in a clearly labelled section of the video.

Deliverables: For each part, the student(s) must submit:

(1) The ***source code*** of their sketch with a clear indication of each modules that they have implemented,

(2) A ***PDF*** file with clearly marked section that discuss each module, and

(3) a ***video*** showing the program's functionality and clearly illustrating each module.

The length of the video is limited to 2 mns. The file size is limited to 20 MBs.

Overall grade: The grade weights will be allocated as follows: Implementation (50%), report (30%), video (20%).

Implementation: The implementation will be graded in terms of functionality, correctness, and generality. To obtain full grade, the code must fully support the specified functionality and produce correct results for all non-singular cases. The implementation must be a Processing sketch (unless negotiated a priori with the instructor and TAs). Students are free to implement their own code from scratch. But they may also use the base code provided by the instructor either as a starting point for their project, or as inspiration for their own implementation of the desired functionality. Students may edit the base code provided (formatting, variable naming convention, comments) to make it better fits their taste or favorite coding practices.

When coding each module, except for this base code provided, students are **not allowed to use any other code** (from the Internet, papers, colleagues, TAs). Students must author themselves all code not provided by the instructor or TAs. For example, it is not acceptable to use inverse kinematic or dynamic animation libraries or collision tools. Students are however allowed to use Processing or Java libraries for items that are not related to the geometry, animation, interaction that are the core focus of the project. These include music/audio, for file I/O, for OpenGL graphics, and for menus.

Rubrics for the report: The student/team should submit a single report for each part in PDF format. The report heading must contain a **title**, **date**, **course name** and **number**, **project number**, **phase letter**, and the **name**(s) of the student(s). The report must have clear section titles for each module. Up to 20% of points will be taken off for a disorganized report that does not have these details.

The write up for each module must contain:

(1) a complete and precise description of the **problem** addressed, including the nature and representation of the **input** (10% of the grade for that module),

(2) a brief high-level **outline** of the approach used (10% of the grade),

(3) **detailed** math, geometry, algorithm or a technical explanation of the **solution** (50% of the grade),

(4) a convincing **justification** of its correctness (20% of the grade),

(5) impressive images of **results** illustrating that the solution works in general (10% of the grade).

Each rubric of the write-up of a module will be evaluated based on the "6 C" criteria discussed in class: **Clear**, **Complete**, **Correct**, **Concise**, **Convincing**, **Clever**. (For example, each of the 5 portions must be sufficiently Clear, Correct, and Complete.) Failing dramatically on one of the criteria may result in a zero grade for that portion. (For example, if the portion is incomprehensible, or missing major parts, or wrong.) Additional (extra credit) points (up to 30%) will be given for exceptional Conciseness, Convincing explanations, and Clever solutions or presentations, but up to 20% may be taken away for failing on these criteria (unnecessarily verbose text, illogical explanations/justifications, unconvincing results, badly chosen on insufficiently general images that do not show that the solution works, the use of vague terms lacking a precise definition.

Rubrics for the Code: Because each student may write code differently, it is often difficult to decipher parts of code or to check for implementation of specific modules in the code. Therefore, we recommend that students annotate their code with **detailed comments**, at the very least each module should have a comment explaining what that code does. For example, <//Rendering human body via caplets> or <//Producing animation in upper body> and so forth. If the code is not well readable and the grader is not able to understand if your code implements a module in the correct way, up to 5% of your grade may be deducted for each such incident in your project.

Part A (individual): Due in 2 weeks (Oct 17 before class)

The objective for Part A is to design and implement a simple articulated human figure **in 2D** and to animate the motion of the foot for simple dance steps. The modules address: (1) the creation of the body parts as **caplets** from disk pairs, (2) the **inverse kinematics** for computing the joint points (knee, ankle, heel) from the control points (hip and ball of the foot) and from body measures, (3) the **animation** of a 4 phases of a **tango step** (Reach, Transfer, Collect, Rotate), and (4) the computation of the **path** for the free foot.

In Part A, the first 3 modules will be solved in 2D using a side view and the 4th module will be solved in 2D using a top view.

In Part B, the student or team will convert these 2D solutions to work on a 3D dancer and will use/customize a 3D GUI to let the "choreographer" design/edit sequences of steps easily, save them to file, load them and show their animation from a static camera or from a camera that follows the dancer.

The articulated body of the tango dancer will be made of caplets. In 3D, a caplet is defined as the convex hull of two balls of possibly different radii. In 2D, a caplet is defined as the convex hull of two disks.

You are given code that uses points (disk centers), associated radii, and distance measures between pairs of points for the hip, knee, ankle…

```
pt _H=P(), _K=P(), _A=P(), _E=P(), _B=P(), _T=P(); // centers of Hip, Knee, Ankle, hEel, Ball, Toe
float _rH=100, _rK=50, _rA=20, _rE=25, _rB=15, _rT=5; // radii of Hip, Knee, Ankle, hEel, Ball, Toe
// leg measures (to update press '/' and copy print here):
float _hk=319.979, _ka=266.46463, _ae=28.718777, _eb=117.23831, _ab=113.9619, _bt=44.75581;
```

You have two copies of these center points. For example, the global variable _B is the center of the ball of the dancer's foot. The corresponding local variable B defined in draw() may be controlled by the user.

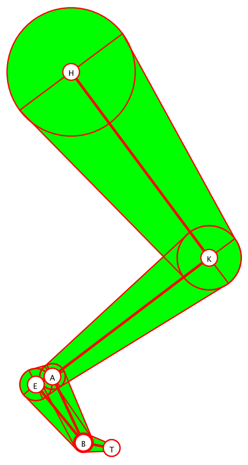
The following call in draw() computes the global variables _K, _A, _E, _T form H and B using the global measures, such as _hk and from the angle, _hipAngle, between HB (hip to ball) and HK (hip to knee):

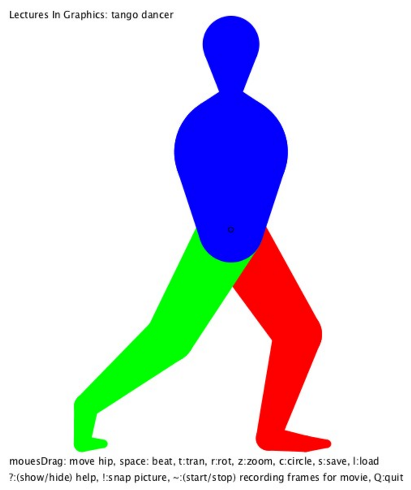
```
  student_computeDancerPoints( H, B,_hipAngle ); // computes _H,_K,_A,_E,_B,_T from measures and _hipAngle
```

You are also given the code for rendering the dancer:

```
void student_displayDancer(pt H, pt K, pt A, pt E, pt B, pt T) // displays dancer using dimensions
 {
 caplet(H,_rH,K,_rK);
 caplet(K,_rK,A,_rA);
 caplet(A,_rA,E,_rE);
 caplet(E,_rE,B,_rB);
 caplet(B,_rB,T,_rT);
 caplet(A,_rA,B,_rB);
 }
```

It only renders the leg (see left below). You will need to improve it to render the rest of the body and the other leg (see right below).

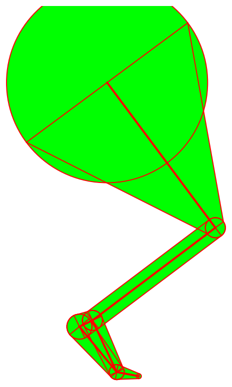



Lectures In Graphics: tango dancer

mouesDrag: move hip, space: beat, t:tran, r:rot, z:zoom, c:circle, s:save, l:load
?:(show/hide) help, !:snap picture, ~:(start/stop) recording frames for movie, Q:quit

Module A.1: Caplet (convex hull of two disks)

The code provided does not display the proper caplet as shown in the image below, where I exaggerated the hip radius to clearly show that straight lines of the caplet do not meet the hip disk tangentially. You should use such an exaggerated radius when attempting to produce convincing examples of your solution at work.



It shows correctly the two disks, but shows the wrong cone (I use here the 3D "cone" for a 2D counterpart that is an "Isosceles Trapezoid" in 2D).

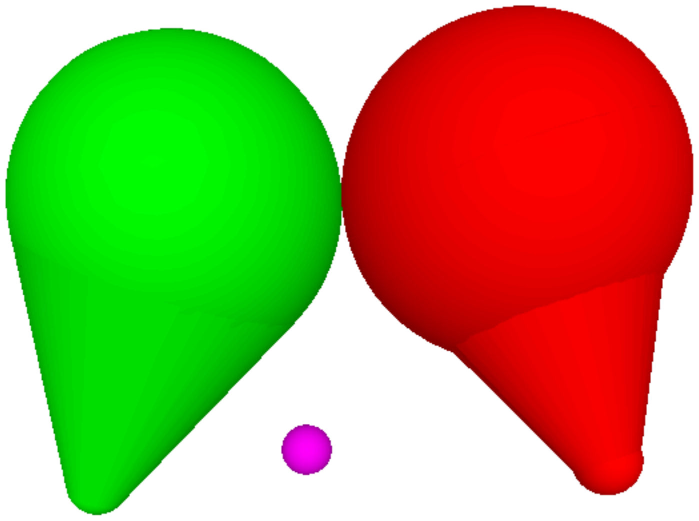
```
void caplet(pt A, float rA, pt B, float rB) // displays Isosceles Trapezoid of axis(A,B) and half lengths rA and rB
 {
 show(A,rA);
 show(B,rB);
 // replace the line blow by your code to draw the proper caplet (cone) that the function displays th ecnvex hull of the two disks
 cone(A,rA,B,rB);
 }

void cone(pt A, float rA, pt B, float rB) // displays Isosceles Trapezoid of axis(A,B) and half lengths rA and rB
 {
 vec T = U(A,B);
 vec N = R(T);
 pt LA = P(A,-rA,N);
 pt LB = P(B,-rB,N);
 pt RA = P(A,rA,N);
 pt RB = P(B,rB,N);
 beginShape(); v(LB); v(RB); v(RA); v(LA); endShape(CLOSE);
 }
```

You should replace the code for caplet by one that computes the correct points and radii parameters for the cone call so that the cone's edges meet the two disks tangentially.

If you have no idea how to solve this, start by solving a simpler problem where one of the radii is zero.

The 3D version of the incorrect caplet is shown below (red), along with the correct one (green)/.



Module A.2: Inverse Kinematics

In this module, you should write the code for

```
void student_computeDancerPoints
   (
   pt H,     // hip center
   pt B,     // ball center
   float a   // angle between HB and HK
   )
```

```
 {
  _H.setTo(H);
  _B.setTo(B);
  _K=P.G[1];
  _A=P.G[2];
  _E=P.G[3];
  _T=P.G[5];
  }
```
to compute the locations of the global points

_H (hip center),

_B (ball of foot center),

_K (knee center),

_A (ankle center),

_E (heel center), and

_T (toe center)

from the input parameters:

H (hip center)

B (ball of foot center)

_hk (distance |HK|)

_ka (distance |KA|)

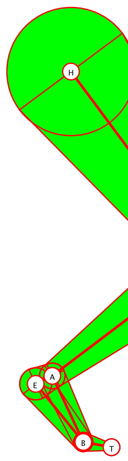
_ae (distance |AE|)

_eb (distance |EB|)

_bt (distance |BT|)

_ab (distance |AB|)

_hipAngle (HB^HK)



In the starting code provided, I simply copy the points of the object P, which can be edited by the user. You should replace that code with one that constructs the proper positions of the _H, _K, _A, _E, _B, _T points so as to meet all **angle and distance constraints**, plus the **additional constraints** discussed below:

1) _B should lie at a vertical distance of _rb above ground so that the whole disk is tangent to the ground
2) _T should lie on the ground
3) The disk of center _E and radius _rE should stay above ground

Note that the first two additional constraints accomplish two objectives:

1) The ball of the moving foot slides on the floor and never lifts up. This is a particular stylish version of the tango. Most dancers lift the ball slightly for effect or comfort during part of most steps. We adopt if here for simplicity and elegance.
2) Part of the toe will be hidden in the 3D version. This is adopted to produce a realistic rendering of a stylish tango shoe tip.

The _hipAngle parameter which defines the angle HB^HK is one particular choice of how to parameterize the one-parameter family of configurations defined by _H, and _B. Although others are possible, you should solve for the configuration using this parameter first, but can then suggest a better parameter.

Here is a suggested organization your code for

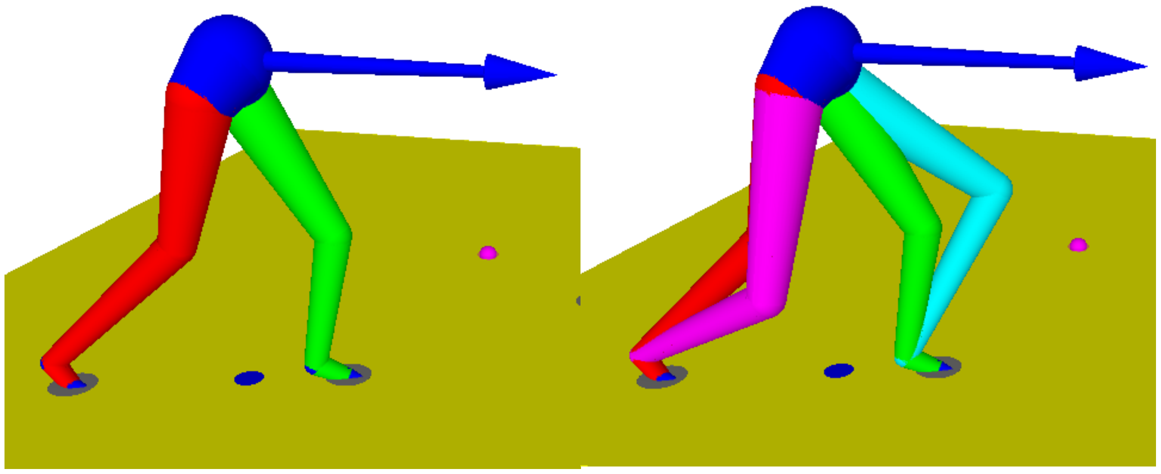
void student_computeDancerPoints

1) Set _H to H, (Later, in Module A.3, you will snap _H to have a given fixed height above the ground)
2) Compute _B so that the ball disk is tangent to the floor and that _B and B are aligned vertically
3) Compute _T to lie on the floor and at distance _bt from _B

4) Compute _K from _H and _B to satisfy the constraints : _hk == |HK| and _hipAngle == HB^HK
5) Compute _A from _K and _B to satisfy the constraints : _ka == |KA| and _ab == |AB|
6) Compute _E from _A and _B to satisfy the constraints : _ae == |AE| and _eb == |EB|

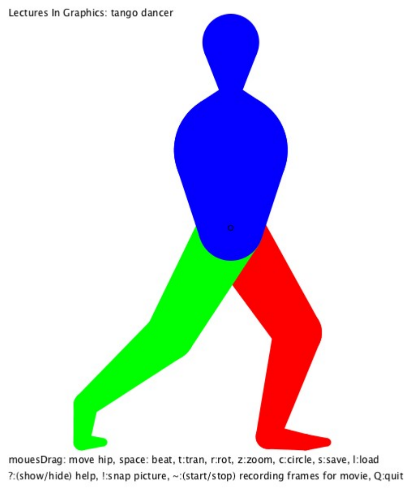
You can debug your solution by clicking and dragging _H and by changing _hipAngle by keeping '/' pressed while dragging the mouse (pressed) horizontally. A line segment form _H with the prescribed angle is shown in magenta. Your code should align the vector _H_K with that segment. When demonstrating the robustness of your solution, you should include several figures in your report and a video segment where the user changes the position of _H throught horizontal, vertical, diagonal, circular motions and changing the _hipAngle for several positions of _H.

A naïve 3D version of a configuration for both legs is shown below (left) and shown with a corrected version superimposed (right) in which the different leg parts have prescribed lengths.



Module A.3: Animation of the 4 phases of a tango step

For module 3, you should show the two legs (in different colors) as illustrated below. You should fill the Right leg in Red and the left leg in Lime color (the image below does not use this convention). The rest of the body should be blue.



Lectures In Graphics: tango dancer

mouesDrag: move hip, space: beat, t:tran, r:rot, z:zoom, c:circle, s:save, l:load
?:(show/hide) help, !:snap picture, ~:(start/stop) recording frames for movie, Q:quit

Your solution should play the animation of a single, forward moving, tango step, as precisely described below.

I will drop the "_" and typesetting in the variable names for simplicity.

We will assume that the hips project on the same point H (side view).

I will not use B, but **L** and **R** when referring to the **ball** of the **Left** or of the **Right** foot.

I will also use **S** and **F** to identify the **Support foot** and **Free foot** (so that I do not have

The following constraints should be enforced throughout the animation of the entire step:

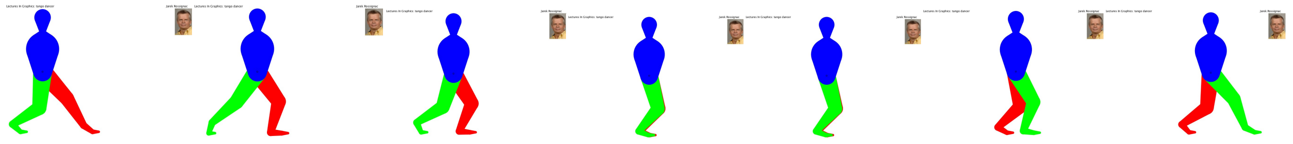
1) H remains at a constant height _h.
2) The upper part of the body remains rigid and vertical above H (no leaning, bending…)

The GUI code

```
if(key==' ') {_h=_H.y; animate=true;}
```

sets _h to the current height of H and starts animation. The animation should last 4 seconds (120 frames).

It is divided into 4 **phases**. These will ultimately (after module 4 and in the 3D version) have different durations. In this module, we allocate 40 frames to each one of Transfer, Collect, and Aim and no frames to Rotate.



| Transfer | Collect | (Rotate) | Aim |

In the figure: initially S=R and F=L (the weight is primarily on the Right foot, which is hence the support foot and the Left foot is Free).

We use C to denote the floor projection of H. Hence, H=C+h$\underline{U}$, where $\underline{U}$ is the unit up-vector.

Transfer: The feet do not move. At the start of Transfer SF==3SC. The body, and hence H and C, move horizontally with constant speed so that C LERPs between S+SF/3 and S+2SF/3. During Transfer, the weight changes progressively between the two feet. Hence, at the end of transfer, the S and F roles of the feet have switched. In the figure above, the second image shows the state at the end of Transfer where S=L and F=R.

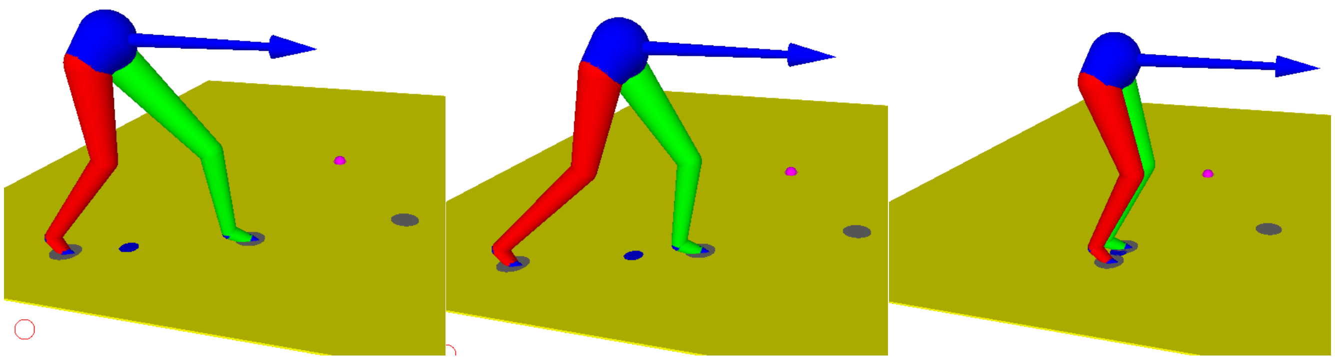
Collect: S does not move. SF==3SC is preserved. F moves from its initial position to its “collect” position, which in this 2D preliminary work is defined by F==S.

Rotate: In 3D, during Rotate, the two feet will be parallel and close together. The dancer will be rotating around S. In this 2D preliminary study, there is no such rotation. Hence, rotate has no effect. You may consider an option where the 120 frames are uniformly distributed to Transfer

Aim: S does not move. SF==3SC is preserved. F moves from its initial “collect” position (F==S) to its extended position (forward in this module). Module 4 will provide options for “aiming” in other directions.

Your implementation of the sequence of Transfer, Collect, and Aim should be executed in 4 seconds each time you press the space bar. Please verify that, when the collect and aim distances are equal, H travels with constant speed during this animation. Hence, if a series of such simple steps is executed for uniformly spaced footprints long a line, the upper body will travel with constant speed and no other movement.

The three dimensional version of the dancer’s lower body may look like this, showing the end of Aim, then after Transfer and after Collect.



Module A.4: Computation of the path for the free foot

In a separate version of a copy of the base code, develop a module showing (from the top) the keyframe locations of the free foot F during the Transfer, Collect, Rotate and Aim sequence.

The input is a series of points on the floor, defined by the consecutive points of P.

Odd points represent the left foot fixed positions and even points of P represent the right foot fixed positions. Your job is to compute the path of the free foot between two consecutive odd or two consecutive even points of P.

To help me define these paths, I will use vector $\underline{\mathbf{V}}$ to denote the **forward direction** of the dancer's hips and vector $\underline{\mathbf{U}}$ to denote $\underline{\mathbf{V}}^{\circ}$, the **sideway direction** of the dancer's hips (from left to right).

Let F_0 be the initial position of F during one of these steps. The consecutive phases of a step are defined as follows:

Collect: F moves from F_0 with constant velocity towards its new position $F_1 = S + s\underline{\mathbf{U}}$ when F==R or $F_1 = S - s\underline{\mathbf{U}}$ when F==L. In other words, it brings F so that F and S are in their natural parallel positions where LR= s$\underline{\mathbf{U}}$. The constant s is a small default spacing (the equivalent of about 6 inches for a real dancer).

Rotate: In this phase, the whole dancer (including F, H, $\underline{\mathbf{U}}$ and $\underline{\mathbf{V}}$) rotate around S. Hence, F, with constant speed moves along a Circle(S,s) from F_1 to F_2, which is computed as explained below. The rotation direction is also explained below.

Aim: During the Aim phase, F moves from F_2 with constant velocity towards its new position F_3, which is the next rest position for F, as defined by P.

How to compute F_2?

Your program should assign a char label ('f','s','b') to each point in P. By default, assign 'f'. But let the user change it by pressing 'f', 's', or 'b' when them mouse is close to a selected point of P. The label of the point closest to the mouse should be changed accordingly. Hence, we have three different versions of Aim:

F-aim: When F_3 is at a point of P with label 'f', then vector F_2F_3 should be parallel to the direction of $\underline{\mathbf{V}}$ at the end of the Rotate phase and hence Line(F_3,F_2) should be tangent to Circle(S,s) at F_2. The dancer's Aim move is forward.

B-aim: When F_3 is at a point of P with label 'b', then Line(F_3,F_2) should be tangent to Circle(S,s) at F_2, but on the other side of that circle. In a sense, this is the time-reverse of Collect . The dancer's Aim move is backward.

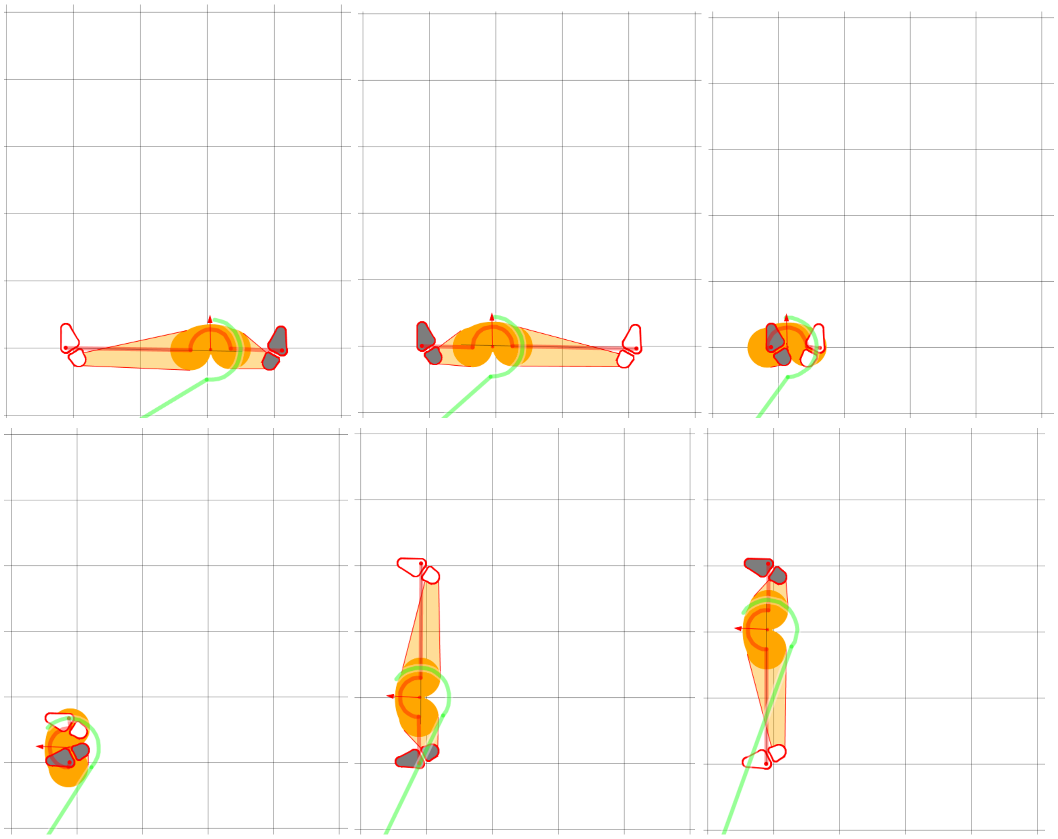
S-aim: When F_3 is at a point of P with label 's', F_2 should be the point on Circle(S,s) that is the closest to F_3. The dancer's Aim move is sideway, away from S.

You should show P as a polygon and show the 'f', 's', or 'b' labels of its vertices using showId().

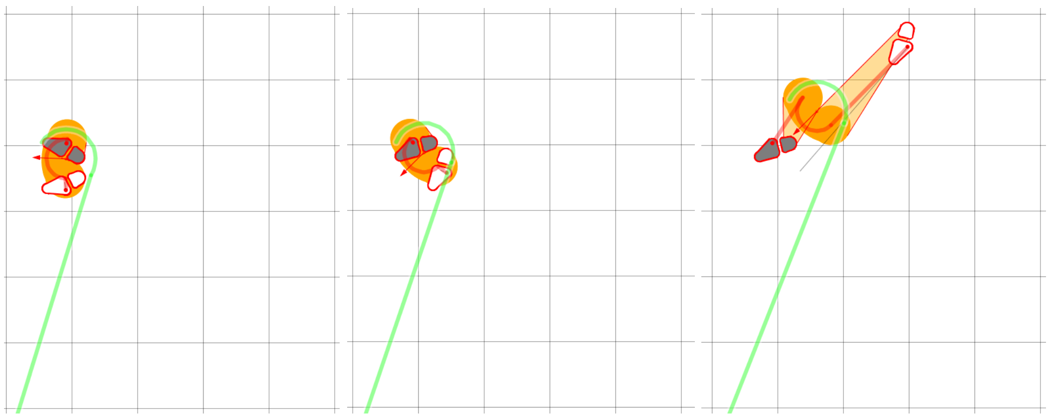
Then you should show the footprints of each foot using the corresponding Red or Lime colors using a small dot per frame of animation. The dots of the same color should lie on straight lines or on circular arcs. The straight lines should join a point of P to a circle around an adjacent point of P and meet that circle tangentially or orthogonally, depending on the corresponding label.

The spacing of the dots for one foot should be at **constant arclength intervals** along the corresponding path. A safe way to perform the sampling is to compute the total arc length of the curved traveled by F during Collect, Rotate, and Aim, divide it into 120, and generate samples with the resulting arc length spacing.

The free foot F travels at constant speed during a sequence of Collect, Rotate, Aim phases, then stops for Transfer, for the Collect, Rotate, Aim phases of the other foot, and during the next Transfer, at which point this foot becomes free again and embarks on another sequence. So, you may implement the computation of the 119 sample positions of F given three consecutive points P_{i-1}, P_i, P_{i+1} and the labels s_i and s_{i+1}. Below, we show a dancer (with initially the weight on R) and illustrate the poses for a sequence of: Transfer, Collect, Rotate, S-Aim, and Transfer.



The next images show the results after Collect, Rotate, B-Aim.



Part B (individual or team of 2): Due November 1 before class

Module B.1: 3D version of Part A

The first requirement for Part B is to implement the 3D counterparts of the four modules of Part A and adding a 3D model of the upper body. The student(s) must also provide a file save/read for the 3D points and their labels.

Students may use part or all of the base code provided in the zipped file "P3 b Dancer simplified.zip" and are welcome to alter its formatting or the names of variables or methods.

Module B.2: Two dancers (required only for teams of two)

As additional duty, Teams of two must provide support for a pair of dancers that stay at a approximately constant distance from one another and so that the leader is always facing the follower. The animation must show them dancing in sync.

Module B.3: Creative components

Students should explore extending these tools to other steps or to other dances.

Module B.4: Extra credit

Here are some ideas:

- Read, implement, present a paper on the proper inverse kinematic for a human walk
- Make sure that the GUI prevents the creation of invalid steps when the feet are too far apart
- Make sure that the GUI prevents the creation of invalid steps when the dancers self-intersect or collide with one another
- Add audio with a dance music and provide GUI for synching the dance with the music beat