Appendix A

Representation, Performance metric and Mutation Operation

This appendix describes in detail the representation of an individual solution used in the evolutionary algorithm. It also gives an intuitive explanation of various terms used in the performance metric and the reasons and motivation for choosing them. Further, the mutation operation which is used to manipulate the individual solution is also documented in detail.

A.1 Representation of Solution

In evolutionary programming, the representation of individual solution plays a very crucial role in the overall performance of the algorithm. In our method, an individual solution represents a controller. We have chosen a Pose control graph to represent a controller. Pose control graphs have been described earlier in section 2.6.2.

The arcs of the pose control graph specify the fixed time interval upon which the transition between the states takes place. The desired pose associated with the state is kept fixed for the duration of the time interval.

A pose control graph can be represented using a set of parameters called solution vector. All these parameters have direct influence over the motion produced
by the controller, and so have to be chosen carefully for a particular motion. The aim of the evolutionary algorithm is to synthesize these values for a required motion. We have used the following parameters in our representation of pose control graphs.

- Poses represent the internal configuration of the body. Internal configuration is nothing but a set of joint angles which can be varied over time. In the case of a $n$ link tree structured body there can be maximum $n - 1$ joints. Each pose is defined as some combination of these joint angles which are to be achieved after a specified time interval. If there are $m$ poses defining the pose control graph, there will be $m$ sets of $n - 1$ joint angles. So the solution vector representing the controller will have the combination of $m(n - 1)$ joint angles.

- Spring and damping constants. The torque applied at a joint is given by the function

$$\tau = k_p(\theta_d - \theta) - k_d\dot{\theta}$$

It is proportional to the difference between the desired joint angle and the current angle. Also spring and damping constants $k_p$ and $k_d$ can significantly affect the type of motion. Hence we have also included some of these constants in our solution vector.

- Time interval between poses. This parameter plays an important role. We want the creature to achieve the desired pose but before it achieves the desired pose the time interval will get over and the state will change. Once the state changes the current configuration will be compared to the parameters associated with that state.

A three link Luxo creature with two joints and two pose control graph is shown in the figure A.1. The solution vector for such a representation is

$$\begin{bmatrix} t_1 \theta_d^{11} \theta_d^{12} k_p^{11} k_p^{12} \\ t_2 \theta_d^{21} \theta_d^{22} k_p^{21} k_p^{22} \end{bmatrix}$$

1 Experimentally a value of damping constant as one tenth of spring constant has been found to be suitable.
Given an initial state a transition is made to pose1 \((\theta_d^{11}, \theta_d^{12})\) in time interval \(t_1\), with values of \(k_p^{11}\) and \(k_p^{12}\), then a transition from pose1 to pose2 \((\theta_d^{21}, \theta_d^{22})\) in time interval \(t_2\), with values of \(k_p^{21}\) and \(k_p^{22}\). After that the cycle repeats. Every time pose change it cause certain torques to be generated which form the input to the simulator.

An ideal automated synthesis system would be able to design a controller given only the mechanical structure of the creature, and no other \textit{a priori} information. However, by specifying a small but useful amount of additional information it becomes possible to greatly reduce the search time and improve the performance of the algorithm. The additional information we are providing is as follows:

1. the ranges of time interval between the poses
2. the number of poses required for the motion, and
3. the expected ranges of spring constants

These numbers are not very difficult to estimate. An estimation of the time interval is done based upon the size and shape of the creature. Values of spring constants are estimated with the help of mass description of the body. If the links between which the spring has been simulated are having higher weight, higher valued ranges of spring constants are required in order to protect the springs from a possible collapse(spring failure). Also values should not be so high that they would cause high torques at the joint causing unexpected motions.
Estimating the number of poses for a creature is comparatively more difficult task and usually needs in depth knowledge of expected motion of the creature. Experience also helps in deciding this number.

Given the above mentioned information, the synthesis technique must find the controller that will perform well with respect to a given performance metric which in turn leads to a desired motion.

A.2 Performance metric

In a broad sense, the synthesis process searches through a space of controllers and selects the best one satisfying the motion features specified by the animator.

How good the match is, is determined by the performance metric. The performance metric typically evaluates the controller by plugging it into the simulator and generating the motion. We have considered following performance metric:

\[
f = w_1 \sum_{s=1}^{\text{samples}} \sum_{j=1}^{\text{joints}} (1 - (\theta^s_{(s,j)} - \theta_{(s,j)})) + w_2 \left(1 - \frac{(E_0 - E)}{E_{\text{max}}}ight) + w_3 \left(1 - \frac{(D_0 - D)}{D_{\text{max}}}ight)
\]

where,

\(w_1, w_2, w_3\) are weights, assigned to the features depending on their relative importance, the value ranging between 0 and 1.

\(\theta_{(s,j)}\) is the angle at joint \(j\) for posture,

\(E\) is the external energy

\(D\) is the horizontal distance travelled

\(E_{\text{max}}, D_{\text{max}}\) are the maximum expected external energy and horizontal distance, respectively. These values are used for normalization of the two quantities.

\(\theta^s_{(s,j)}, E_0, D_0\) are the feature values specified by the animator.
It is clear from the above function that, closer the match between the specified and synthesized feature values, higher would be the value of the performance metric. The specified and achieved poses are compared during each cycle and the difference is summed up. The external energy and the horizontal distance are summed during each cycle.

Poses provide the actual snapshot of the object at an instance, like keyframes. In order to get similar kind of motion, these values have to match as close as possible with the corresponding controllers values. Thus, higher weight \( w_1 \) is given to this comparison. External energy and horizontal distance traveled help in optimization process in case of ties in other feature values. For example, object can achieve the same configuration without even moving from its place in the environment. Generation of such a motion has been largely averted by using external energy and horizontal distance features.

A.3 Mutation Operation

Mutation operation is the backbone of SPHC algorithm. In every iteration all solutions go through this operation. Each solution is represented by a set of parameters as discussed above. These parameters are time interval, desired joint angles and spring constant parameters.

Change in one of the parameters can change the motion drastically. So a careful mutation of these parameters is required with only a small change in original value. We have selected to mutate only one parameter in the whole state once the mutation operator is applied. The selection of this parameter is a randomized process, with equal probability is given to all the parameters. This is found suitable through experiments as it helps the algorithm to fully explore the region near an existing solution. If we try to mutate more than one parameters at a time, the solution may jump from one region to another without exploring the current one. As the function is multimodel, it may be the case that optimal solution is in the vicinity of solution being mutated.

Each time mutation is called the selected state goes through a creep operation. In creep operation a randomly selected parameter is modified with a very small
factor or all its parameters are randomized from scratch. As there are three
types of parameters to be modified, there are three possible creep operations
which are defined as follows:

1. The original time interval is multiplied by a randomly chosen factor close
to unity (0.8 – 1.2).

2. One of the joint angles is selected randomly and changed by a randomly
chosen amount between $-10^\circ$ and $10^\circ$, and

3. One of the joint angles is selected randomly and multiplied by a randomly
chosen factor close to unity (0.8 – 1.2).
Appendix B

Animation System Reference Manual

This appendix list the syntax of all the commands supported by different modules constituting the experimental animation system used for synthesizing the animations in this thesis.

B.1 Symbolic equation generator

Command summary:

< <filename>
  Reads the script in the given file as input.
cat <filename>
  Prints out the given file
compile
  Compiles a procedure for solving the equations of motion.
echo <args>
  Prints arguments
fix <link_num> <fix_x> <fix_y>
  Fixes the location fix_x, fix_y on the given link.
link <link_num> <parent_num> <attach_x> <attach_y> <mass> <inertia> <mass_x> <mass_y>
  Creates link number link_num, attached to the given parent at the given point, having the given mass, inertia
(with respect to centre of mass) and centre of mass.

\[ \text{monitor } \langle \text{link\_num} \rangle \langle \text{mon\_num} \rangle \langle \text{x} \rangle \langle \text{y} \rangle \]

Allows for external forces to applied at the given x,y on the given link.

\[ \text{quit} \]

Exits the program shell

\[ \text{set } \langle \text{var} \rangle \langle \text{value} \rangle \]

Sets the variable to the given value.
The values are obtainable by executing ‘set’ without any arguments

\[ \text{procname STRING} \]

name of desired procedure

### B.2 Simulator

Command summary:

\[ \langle \text{filename} \rangle \]

Reads the script in the given file as input.

\[ \text{cat } \langle \text{filename} \rangle \]

Prints out the given file

\[ \text{dyn [proc]} \]

Chooses the given dynamics procedure to be used.
With no arguments, it lists the current dynamics procedures available.

\[ \text{echo } \langle \text{args} \rangle \]

Prints arguments

\[ \text{quit} \]

Exits the program shell

\[ \text{set } \langle \text{var} \rangle \langle \text{value} \rangle \]

Sets the variable to the given value.
The values are obtainable by executing ‘set’ without any arguments

\[ \text{debug bool} \]

turns debugging info on or off

\[ \text{dispfile string} \]

name of file for output of display information

\[ \text{dtdisp float} \]

display time step

\[ \text{dtsim float} \]

simulation time step

\[ \text{kdamp float} \]

damping constant

\[ \text{state\_size int} \]

the size of the state vector

\[ \text{sim } \langle \text{time} \rangle \]
Simulates the system for the given amount of time, in seconds. The simulation time step is given by 'dtsim'. The display time step is given by 'dtdisp' (see the 'set' command)

```
state <x,vx,y,vy,th,dth,th1,dth1,th2,dth2...>
```

Sets the current system state to the given value.

### B.3 Anix server

#### summary of commands:

< <file_name>

Read input from file.

```
aniset <var1> <valuel> <var2> <value2> ...
```

Sets environment values to desired values

Aniset without any arguments returns the current value of all the variables. A brief description of the use of all the accessible variables is as follows:

- **debug** flag to output excess information for debugging
- **degmode** flag to specify rotation transformations in degrees
- **device** "ps" for postscript, "display" for X-11 window
- **display** name of host to use as an X-11 server
- **erase** erase display between frames? (X-11 only)
- **eyedist** distance of eye from screen for perspective projection
- **helpfile** file containing documentation
- **newsfile** file containing list of recent modifications
- **psfile** name of file to send postscript output to
- **showtime** echo current time
- **sleep** time to pause between displaying frames
- **viewdist** distance of eye from viewpoint for perspective proj.
- **viewto** the viewpoint, placed in the centre of the screen
- **viewfrom** specifies line of sight
- **viewup1** which direction to consider 'up' for display purposes
- **viewup2** a second choice for an 'up' vector in case viewup1 is very close to being parallel to the line of sight
- **winx,winy** size of X-window
- **xwin,ywin** window placement

attach <objname> to <objname>
Attaches one object instance to another object instance.

`clear`
Clears the x-display

close
Closes animation display.

`comments`
All text following a '#' character on a line is ignored

detach <objname>
Detaches the object from its parent

`help [topic]`
Lists information in help file on the specified topic.
Type 'help sum' for a summary of available commands.

`init`
Calculate world to screen transformation, prepare for display.

`instance`
Creates an instance of an object

`news`
Print the most recent updates.

`objectname <objectname>`
Begin a new object

`path`
Begin a new path.

`pts <x> <y> <z> ...`
Adds points to the current path.

`quit`
Exit program.

`show <val1> <val2> ... <valn>`
Supplies a series of missing transformation values and then displays using the new transformations.

`sleep <time>`
Causes a pause for the specified number of seconds.

`text x y z string`
Prints text at the given point

`tf <obj> <tf_name> <tf_args>`
Performs the specified transformation on the object.
The acceptable transformations (as specified by `tf_name` and `tf_args`) are:

`rot <x|y|z> <angle>`
Rotates the object about the given axis by the angle
in degrees.
prot <x|y|z> <angle>
    Rotates the object about the given axis in the parent’s coordinate system. The angle is in degrees.
trans <x> <y> <z>
    Translates (moves) the object as specified in the objects own coordinate system.
ptrans <x> <y> <z>
    Translates (moves) the object as specified in the parent’s coordinate system.
scale <x> <y> <z>
    Postmultiply the ctm by the given scaling factors. The object itself and all following transformations are affected by the scaling.
pscale <x> <y> <z>
    Premultiply the ctm by the given scaling factors. All transformations and the object itself are affected by the scaling.
tpipe <file1> <pipefile>
    Repeatedly pipes in <pipefile> while <file1> exits.
Appendix C

Documentation of Scripts

This appendix documents all the scripts for Luxo, Pogo and Walker in generating the animations. Scripts with extension .anix gives the geometric description of the creature. Script with extension .desc describe the physical structure of the creature and the script with extension .sim is used in producing the simulation and recording of the frames.

C.1 Scripts for Luxo

luxo.anix

aniset winx 600
aniset winy 600
objname baselink
path
pts -0.27 0.02 0
pts 0.27 0.02 0
pts 0.27 -0.02 0
pts -0.27 -0.02 0

objname middlelink
path
pts -0.02 0.02 0
pts 0.52 0.02 0
pts 0.52 -0.02 0
pts -0.02 -0.02 0
objname toplink
path
  pts -0.02 0.02 0
  pts 0.42 0.02 0
  pts 0.35 0.02 0
  pts 0.40 0.02 0
  pts 0.30 -0.15 0
  pts 0.47 -0.15 0
  pts 0.42 -0.02 0
  pts -0.02 -0.02 0
instance baselink link1
instance middlelink link2
instance toplink link3
attach link1 to world
attach link2 to link1
attach link3 to link2
tf trans link2 0.0 0.0 0
tf trans link3 0.5 0.0 0
< gnd.anix
init
tf trans link1 -3.0 0 0
  tf trans link1 _1 _2 0
  tf rot link1 Z _3
  tf rot link2 Z _4
  tf rot link3 Z _5
  tf trans world 15 4 0
  tf scale world 3 3 3
  aniset degmode f
  aniset erase t

luxo.desc

  link 1 0 0.0 0.0 0.15 0.003123 0 0
  link 2 1 0.0 0.0 0.10 0.002082 0.25 0
  link 3 2 1.0 0.0 0.30 0.006246 0.25 0
  monitor 1 1 -0.27 0
  monitor 1 2 0.27 0
  set proclname fall
  compile
  quit
luxo.sim

set kdamp 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
set dt sim 0.001
dyn fall
mon mon_fall 2
set state_size 10
state 0.0000,0.0008,-0.0010,0.0069,-0.0016,-0.0100,-4.6161,
     -0.5956,4.6127,0.0025
set dispfile luxo.out
sim 8.0
quit

C.2 Scripts for Pogo

pogo.anix

aniset winx 600
aniset winy 600
objname base
path
   pts -0.02 0.02 0
   pts 0.52 0.02 0
   pts 0.52 -0.02 0
   pts -0.02 -0.02 0
   pts 0.00 0.08 0
   pts -0.15 0.03 0
   pts -0.05 0.0 0
   pts -0.15 -0.03 0
   pts 0.00 -0.08 0
objname leg
path
   pts -0.02 0.02 0
   pts 0.27 0.02 0
   pts 0.27 -0.02 0
   pts -0.02 -0.02 0
instance base link1
instance leg link2
instance leg link3
instance leg link4
instance leg link5
attach link1 to world
attach link2 to link1
attach link3 to link2
attach link4 to link1
attach link5 to link4
tf trans link2 0.0 0.0 0
tf trans link3 0.25 0.0 0
tf trans link4 0.50 0.0 0
tf trans link5 0.25 0.0 0
< gnd.anix
init
    tf trans link1 3.5 0.0 0
    tf trans link1 _1 _2 0
    tf rot link1 Z _3
    tf rot link2 Z _4
    tf rot link3 Z _5
    tf rot link4 Z _6
    tf rot link5 Z _7
    tf trans world 14 6 0
    tf scale world 3 3 1
    aniset degmode f
    aniset erase t

pogo.desc

link 1 0 0.0 0.0 0.15 0.003123 0.25 0
link 2 1 0.0 0.0 0.10 0.002082 0.125 0
link 3 2 0.25 0.0 0.10 0.002082 0.125 0
link 4 1 0.50 0.0 0.10 0.002082 0.125 0
link 5 4 0.25 0.0 0.10 0.002082 0.125 0
monitor 3 1 0.27 0
monitor 3 2 -0.02 0
monitor 5 3 0.27 0
monitor 5 4 -0.02 0
set procname pogo
compile
quit

176
pogo.sim

set kdamp 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
set dtsim 0.001
dyn pogo
mon mon_pogo 4
set state_size 14
state 0.0000,0.2487,0.3512,-0.4275,-0.1337,-1.7569,-1.7014,
     0.2469,5.2939,0.0589,-1.1427,-0.3886,4.4908,-0.5539
set dispfile pogo.out
sim 8.0

C.3 Scripts for Walker

walker.anix

aniset winx 600
aniset winy 600
objname base
path
  pts 0.02 0.02 0
  pts 0.52 0.02 0
  pts 0.52 -0.02 0
  pts -0.02 -0.02 0
objname leg
path
  pts 0.02 0.02 0
  pts 0.27 0.02 0
  pts 0.27 -0.02 0
  pts -0.02 -0.02 0
objname foot
path
  pts -0.082 0.02 0
  pts 0.082 0.02 0
  pts 0.082 -0.02 0
  pts -0.082 -0.02 0
instance base link1
instance leg link2
instance leg link3
instance foot link4
instance leg link5
instance leg link6
instance foot link7
attach link1 to world
attach link2 to link1
attach link3 to link2
attach link4 to link3
attach link5 to link1
attach link6 to link5
attach link7 to link6
tf trans link2 0.0 0.0 0
tf trans link3 0.25 0.0 0
tf trans link4 0.25 0.0 0
tf trans link5 0.0 0.0 0
tf trans link6 0.25 0.0 0
tf trans link7 0.25 0.0 0
< gnd.anix
init
tf trans link1 0.0 0.0 0
tf trans link1 _1 _2 0
tf rot link1 Z _3
tf rot link2 Z _4
tf rot link3 Z _5
tf rot link4 Z _6
tf rot link5 Z _7
tf rot link6 Z _8
tf rot link7 Z _9
tf trans world 10 6 0
tf scale world 4 4 1
aniset degmode f
aniset erase t
**walker.desc**

```plaintext
link 1 0 0.00 0.0 3.0 0.0625 0.250 0
link 2 1 0.00 0.0 5.0 0.0260 0.125 0
link 3 2 0.25 0.0 4.0 0.0208 0.125 0
link 4 3 0.25 0.0 5.0 0.0260 0.0 0
link 5 1 0.00 0.0 4.0 0.0208 0.125 0
link 6 5 0.25 0.0 1.0 0.0052 0.125 0
link 7 6 0.25 0.0 1.0 0.0052 0.0 0
monitor 4 1 -0.082 0.0
monitor 4 2  0.082 0.0
monitor 7 3 -0.082 0.0
monitor 7 4  0.082 0.0
set  procname  walker
compile
quit
```

**walker.sim**

```plaintext
set  kdamp  0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
set  dtsim  0.001
dyn  walker
mon  mon_walker  4
set  state_size  18
state  0.0,0.0,0.52,0.0,1.120,0.0,-2.793,0.0,-0.175,0.0,1.745,0.0,-2.382,0.0,-0.885,0.0,1.745
set  dispfile  walker.out
sim  8.0
quit
```