Articulated Rigid Body Locomotion

William Braga

Design of Quadruped

My quadruped is made up of a box torso and four equal legs. Each leg is comprised of two segmented capsules modelling the proximal and distal links. There are joints linking the proximal legs to the torso and the distal legs to the proximal legs (8 total joints). Listed below are the specifications for the quadruped shape.

Torso: *length – 0.3; width = 0.2; height = 0.1; mass = 0.3*

Proximal Leg: *length – 0.05; radius = 0.02; mass = 0.2*

Distal Leg: *length – 0.02; radius = 0.02; mass = 0.2*

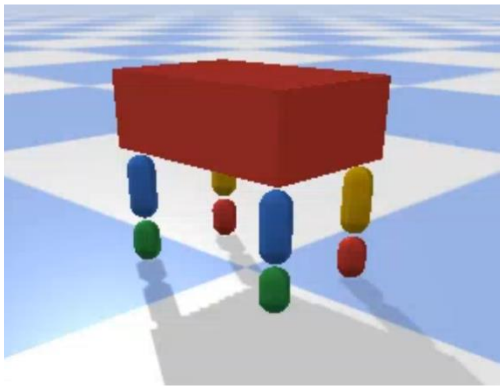


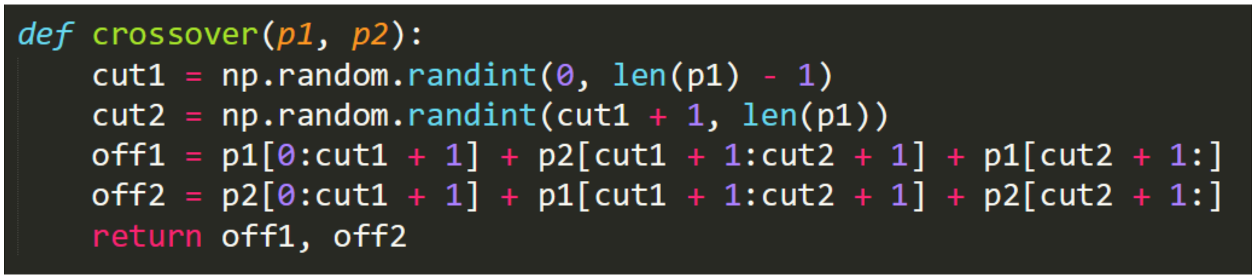
Figure 1: Quadruped Design

Parameters

My simulation can be run using a list of 26 parameters. 25 of these parameters are optimized during evolution. The first parameter is the frequency of the motion system. At first, I was optimizing this value but found that it would always be maximized to meet my distance reward metric. Instead, I manually set this value when looking for certain gaits – like walking or galloping. The second parameter is the force by which the joints move. The final 24 parameters control the sinusoid motion of the joints. Each joint rotates based on a triplet of values: the maximum angle it can reach, its offset, and its phase relative to the other joints. The order of the joint parameters are as follows (named by the limb it connects *to*): left hind leg proximal, left hind leg distal, left front leg proximal, left front leg distal, right hind leg proximal, right hind leg distal, right front leg proximal, right front leg distal. At one point, I was also optimizing for the shape of the quadruped, but it was taking too long for the computer to converge on a gait, so I settled for a static design. The parameters for the gaits are listed in the order described above. [frequency, force, $\text{max\_angle}_{\text{left\_hind\_proximal}}$, $\text{offset}_{\text{left\_hind\_proximal}}$, $\text{period}_{\text{left\_hind\_proximal}}$, …, $\text{period}_{\text{right\_front\_distal}}$]

Optimization

The 25 parameters are trained using a genetic algorithm. At each epoch, enough random creatures are created to make sure the population size is 30. The bounds on the parameters are 5 for force, 25 for maximum angle, 25 for offset angle, and 360 for phase. Too high a bound for maximum angle or offset angle, and the limbs of the quadruped could potentially unhinge and go around the creature. The 30 individuals are then simulated independently. I use the multiprocessing library to try to parallelize these tests. I worked with two different evaluation (reward) metrics: Euclidean distance and x distance – y distance (staying in a straight line). Once the simulations are done, the creatures are sorted and the top two continue unchanged to the next epoch. The rest are randomly crossed over and mutated with a probability proportional to their ranking. Crossing over is done by making two cuts in each of the two individuals' parameters and switching the data between the cuts.



```python
def crossover(p1, p2):
    cut1 = np.random.randint(0, len(p1) - 1)
    cut2 = np.random.randint(cut1 + 1, len(p1))
    off1 = p1[0:cut1 + 1] + p2[cut1 + 1:cut2 + 1] + p1[cut2 + 1:]
    off2 = p2[0:cut1 + 1] + p1[cut1 + 1:cut2 + 1] + p2[cut2 + 1:]
    return off1, off2
```

Figure 2: Crossing Over

There is a 1% chance that a point mutation occurs where one parameter is reset to a valid random number. 60% of the creatures in the next epoch are crossovers from the previous epoch. The rest are either the number one and two creatures from the previous epoch or new random individuals.

Gaits

Parameters rounded to two decimals, full parameters can be found in quadruped.py

1. Boogie:
   The boogie gait causes the quadruped to sway back and forth. It has symmetric motion across all four legs. This was found manually while testing designs. Params: [0.04, 4, 20, 0, 0, 20, 0, 0, 20, 0, 0, 20, 0, 0, 20, 0, 0, 20, 0, 0, 20, 0, 0, 20, 0, 0]

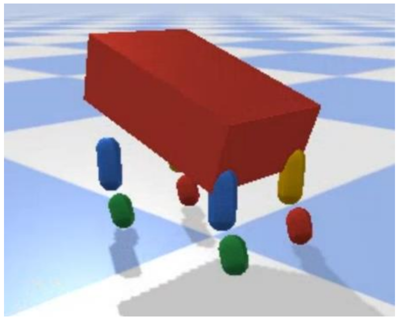


Figure 3: Boogie Gait

2. Pacing:
   The pacing gait was found by optimizing on a frequency value of 0.08. The gait has more front-hind similarity than right-left, i.e. the front and hind legs move at about the same time. Params: [0.08, 2.56, 0.53, 3.24, 214.05, 15.93, 6.99, 220.55, 4.17, 11.39, 62.79, 22.34, 12.06, 350.13, 21.40, 4.70, 65.40, 19.95, 17.76, 146.89, 11.93, 11.39, 91.66, 23.33, 18.28, 122.09]

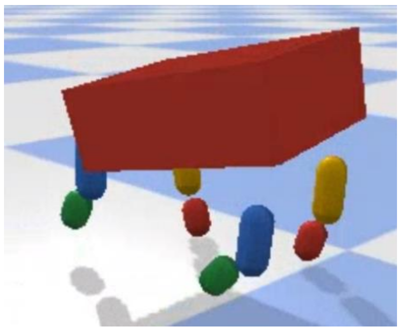


Figure 4: Pacing Gait

3. Trot:
   The trot gait was optimized on a frequency of 0.12. The hind legs move forward while the front legs are moving back, like in a horse trot or gallop. Params: [0.12, 3.11, 13.94, 17.07, 230.67, 25.00, 4.59, 235.96, 2.14, 18.85, 24.19, 24.87, 23.76, 101.01, 23.46, 24.03, 103.76, 17.14, 0.90, 83.38, 4.16, 4.03, 342.99, 24.67, 21.6, 224.86]

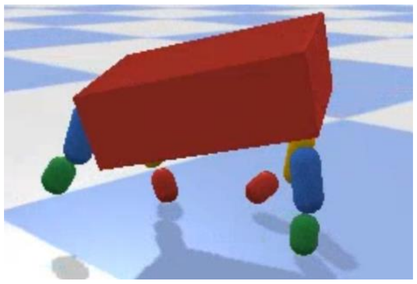


Figure 5: Trot Gait

4. Extra (Speed Boat):
   This is an extra gait I found when testing my evolution code. It is also my best performing model – at least in terms of distance travelled… Params: [0.76, 4.43, 6.65, 125.64, 137.17, 28.39, 3.88, 166.08, 9.16, 49.95, 297.88, 42.93, 29.55, 297.04,   2.68, 65.42, 155.97, 0.62, 34.65, 343.36, 16.41, 44.75, 148.88, 41.75, 37.06, 122.6]

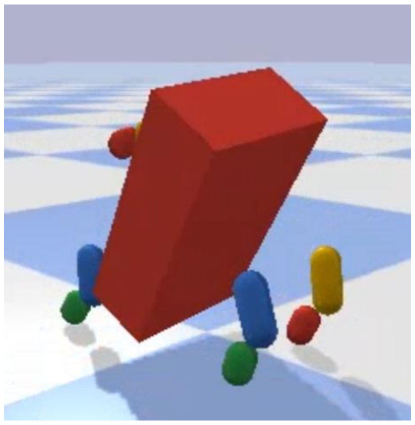


Figure 6: Extra (Speed Boat) Gait

<u>References</u>

[1] Dr. Turk's sample code
https://gatech.instructure.com/courses/179608/files/21505089?wrap=1

[2] Horse Gaits https://www.youtube.com/watch?v=skhwe4LQ2pY

[3] Multiprocessing https://stackoverflow.com/questions/9786102/how-do-i-parallelize-a-simple-python-loop