CSci 8980
Project - Stick Solo
Report

Yashasvi Sriram Patkuri - patku001@umn.edu

September 28, 2022

Abstract

The problem of motion planning for wall-climbing two arm agent is considered in this work. It is modeled in two different ways; using a 4R chain agent and a $2 \times 2R$ tree of chains agent. Along with planning to reach the finish hold there is also an effort to make the motion as natural as possible using simple heuristics. Using constraints on q and Δq and center of mass control achieves significantly better natural motion. Local minima poses are avoided by using random-sample based globally-optimal inverse-kinematics solves. These coupled with gradient descent make the agent reach snap to holds reliably. By switching pivots and matching hands the agent can easily navigate a given climbing route. The neck goal for $2 \times 2R$ agent is predicted by a cross-entropy optimized fully connected network. The full policy is also visualized nicely. Using these methods both models achieve reliable and natural motion to reach finish holds. Moreover these methods can be applied generally to any tree of chains agent, including a full two arm, two leg human-like agent.

1 introduction

The problem considered in this work is the task of wall climbing by stick-figure agents. The goals are

1. To model human-like motion using simple heuristics.
2. To generate a controller for a given agent so that it reaches the finish hold.

All motion, environment and agents are in 2D.

1.1 NR chain agent

Define NR chain agent as serially connected N links using revolute joints. One end of the chain is a pivot, while the other end is free. Such an agent has N links, N + 1 vertices and N joint angles (q_is). The controls are the angles at each joint. Figure 1 shows examples of such agents. These are used to model individual limbs. A q assignment of a 2R agent is show in figure 2

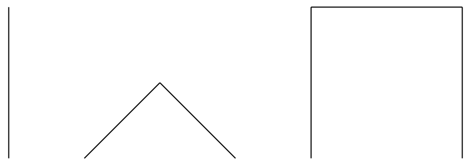

Figure 1: 1R, 2R, 3R agents.

1.2 Tree of NR chains agent

Multiple NR chain agents can be joined end-to-end to form a tree. The ends where joining occurs cannot be used as a pivot. At any point of time exactly one pivot. For a (parent, child) NR chain pair, we assume that the point of joining is controlled by the parent agent. The child agent uses the point of joining as it's pivot. Figure 3 shows examples of such

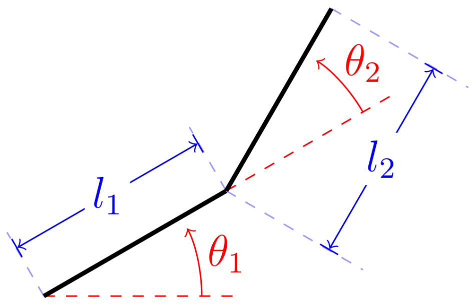

Figure 2: q or θ assignment for a 2R agent.

agents. Such a $2 \times 2R$ agent is used to model two arms or two legs. The difference between a $4R$ agent and a $2 \times 2R$ agent is that the latter can be controlled hierarchically. Thus multiple elements can be composed without much agent specific math. The methods discussed in this work are implemented for $2 \times 2R$ agent modeling two arms, but are mostly applicable for any Tree of NR chains agent.

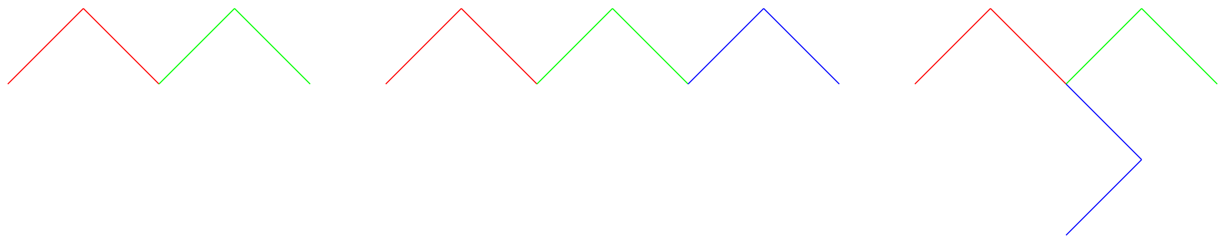

Figure 3: $2 \times 2R$, $3 \times 2R$ agents.

1.3 Environment

A 2D wall with holds is the environment. A free end can reach a hold and can use it as a pivot. In that case the existing pivot becomes a free end to maintain the invariant of exactly one pivot. An example is shown is figure 4.

1.4 Motivation and broader use

Firstly it is quite fun to watch robots climb walls. But also by solving this problem one can find solutions for new and potentially dangerous bouldering problems. If implemented in real robots, they can be used to climb difficult and challenging terrain for reconnaissance or rescue missions. In terms of animation, most games use fixed climb cycles. This method can be used to improve climbing character animation.

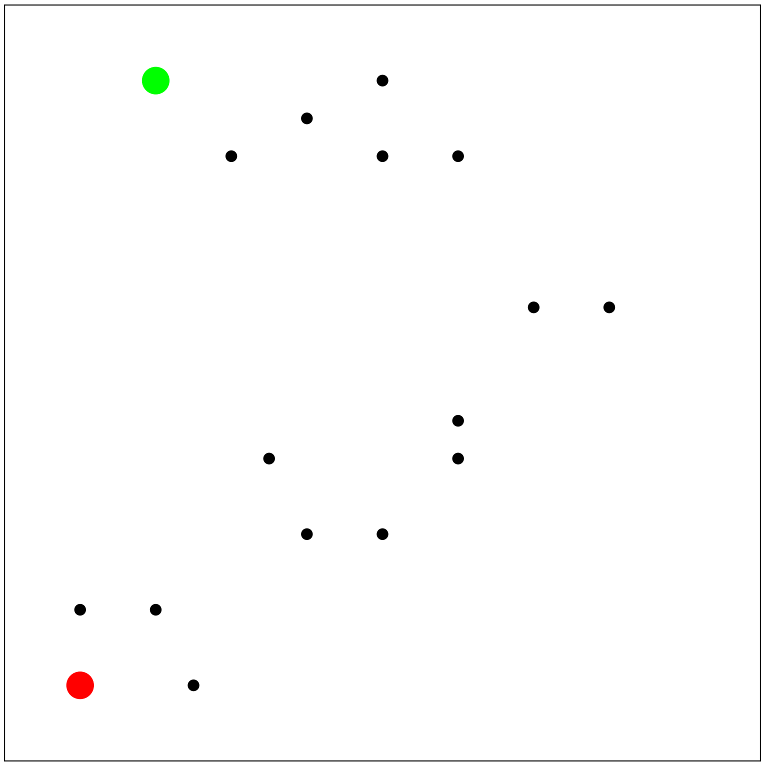

Figure 4: Wall environment. Dots are holds. Red dot is start hold. Green dot is finish hold.

2 related work

In general the problem of motion planning for wall-climbing is not as explored as other tasks like running, jumping, dribbling and even swimming. [Bull et al., 1995] propose wall climbing robot gait control using genetic algorithms and Q-learning. [Grieco et al., 1998] and [Nagakubo and Hirose, 1994] demonstrate real life wall-climbing robots. The dynamics of their robot are similar to agents considered in this work. [Kalisiak and Van de Panne, 2001] is one of the first papers to work on animating 2D parkour agents involving swinging, climbing and crawling. They propose a kinematic motion planning algorithm based on stochastic search procedures guided by geometric constraints. This work also uses random sampling to solve for globally optimal inverse kinematics. [Naderi et al., 2017] addresses the problem of offline path and movement planning for wall climbing humanoid agents. [Peng et al., 2017] uses the idea of hierarchical controller generation for the tasks of walking, running, jogging, dribbling etc. This idea of hierarchical control is used in this work. A neural network is used to plan goals for a lower level inverse kinematics solver.

2.1 Baseline

Baseline is chosen as my previous work on this problem. The source code, demo and report can be found at `https://github.com/buggedbit/stick-solo`. The baseline considers similar NR chain agents but it

1. Has no constraints on q_i and Δq_i.

2. Has no center of mass considerations.
3. Has no multi-limb coordination.
4. Mainly uses gradient descent on loss function. $\Delta q \equiv -\frac{\delta(goal-end)^2}{\delta q}$

One of the main problems of the baseline, more specifically the gradient descent, is the possibility of local minima. Combined with no constraints on q_i and Δq_i, modeling limbs using these agents this produces weird poses and transformations between poses. These issues are addressed in this work.

2.2 Oracle

There are two components of evaluation in this problem.

1. Reaching the goal.
2. Moving in a human-like way.

While the first one can be evaluated automatically, the latter one is best evaluated using user-studies.

2.3 Contribution

The main contribution of this work over baseline is to

1. Add constraints on q_i and Δq_i.
2. Add center of mass terms to the loss function to model natural human motion.
3. Develop arbitrarily close globally optimal solves for NR chain agents.
4. Formulating and implementing the control of $2 \times NR$ as an RL problem.

3 approach

In this section the approach is incrementally presented.

3.1 Constraints on q_i and Δq_i

For each join of the agent two constraints; one on its value and one on its rate of change is implemented. Any rate of change of magnitude more than certain threshold is clamped. Similarly if a change in joint angle makes its magnitude more than a certain threshold then the angle is clamped.

3.2 Gradient descent on center of mass

Baseline uses gradient descent on the following loss function.

$$L \equiv (goal - end)^2$$

So an update on q_is would be

$$\Delta q \equiv -\frac{\delta L}{\delta q}$$

But this does not explicitly model any nature of motion, and therefore there are no preferred joint angles. To make the motion more natural, the most basic thing to simulate is gravity. Since simulating forces and torques on each link and interaction forces b/w all pairs of touching links would be cumbersome, we simulate its effect on the center of mass of the whole agent. We try to bring the center of mass of the whole body down and towards the midpoint of current pivot and current goal. Therefore the loss function becomes

$$L \equiv \alpha * (goal - end)^2 + \beta * (com_x - (goal_x - pivot_x)/2)^2 + \gamma com_y$$

where α, β, γ are tunable constants. Higher values of α models stronger agents, while lower values model weaker agents. Now the update becomes

$$\Delta q \equiv -\frac{\delta L}{\delta q} \equiv -\alpha \frac{\delta (goal - end)^2}{\delta q} - \beta \frac{\delta (com_x - (goal_x - pivot_x)/2)^2}{\delta q} - \gamma \frac{\delta com_y}{\delta q}$$

The first term is already calculated in the baseline. The other terms can be easily calculated once

$$\frac{\delta com_x}{\delta q}, \frac{\delta com_y}{\delta q}$$

are known.

Consider

$$\frac{\delta com_x}{\delta q}$$

Assuming same mass for all links and given x-coordinates of vertices of NR chain

$$x_0, x_1, ...x_n$$

The x-coordinate center of mass of each link will be at its center

$$c_i \equiv (x_i + x_{i+1})/2; i = 0, 1, ...n - 1$$

Therefore x-coordinate of center of mass of entire agent is

$$com_x \equiv x_0/2 + x_1 + x_2 + ... + x_{n-1} + x_n/2$$

But from the dynamics of NR chain we know that

$$x_i \equiv \sum_{i=1}^{n} l_i cos(\sum_{j=1}^{i} q_j)$$

$$y_i \equiv \sum_{i=1}^{n} l_i sin(\sum_{j=1}^{i} q_j)$$

Thus we have formulated com_x in terms of q_is, which makes calculating

$$\frac{\delta com_x}{\delta q}$$

feasible. The calculation of

$$\frac{\delta com_y}{\delta q}$$

follows the same arguments. The effect of this term is illustrated in figure 5.

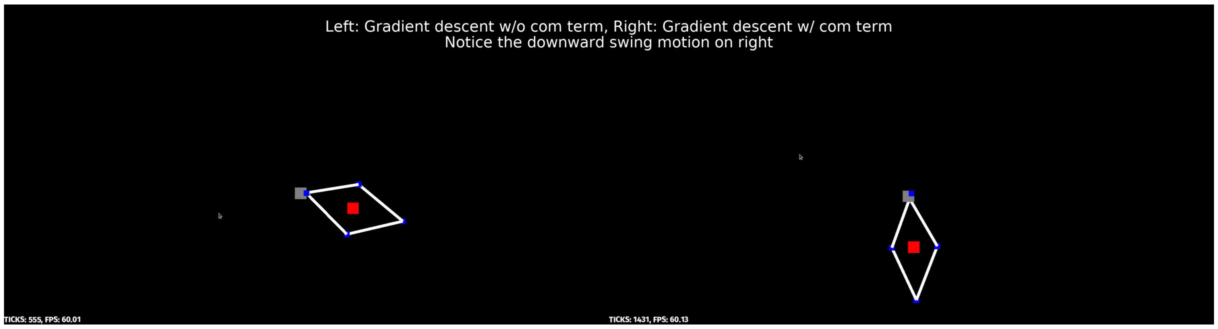
Left: Gradient descent w/o com term, Right: Gradient descent w/ com term
Notice the downward swing motion on right
TICKS: 555, FPS: 60.01
TICKS: 1431, FPS: 60.13

Figure 5: Left agent has no com control, therefore stays in a tilted position. Right agent has com control, therefore swings downward.

Notes In practice we found it better to have more discounted com control for links farther from pivot since it is natural for them to be more responsible about reaching the goal rather than maintaining com position. Also observe that the goal for x-coordinate for center of mass can be changed with very little change in math. For both terms the gradient becomes zero at both local minima and local maxima. Ideally the local maxima case should be handled explicitly. But since it is an unstable equilibrium and there is almost always a perturbation from some other control (like from the first term). Therefore it is very rare for the agent to be stuck at local maxima and hence handling it explicitly is not needed in practice.

Time complexity Computing these terms has $O(n^3)$ time complexity when done in a brute force way. But due to the repeated summations it is actually possible prune some computations by maintaining total sums and subtracting from them. This approach has $O(n^2)$ time complexity, which is the one implemented in the code. But since for the purposes of this work we are considering single NR agents with small N, it does not make much difference. However if work-like agent (NR chain with large N) or large crowds of climbing agents are simulated then this can make a significant difference.

3.3 Globally optimal inverse kinematics solves

Gradient descent with constraints can cause agents getting stuck at local minima poses. This is illustrated in figure 6. Even though this can be mitigated a bit with parameter tuning it is still an intrinsic and major problem. To solve this we need a globally optimal solve for inverse kinematics under constraints. But there is no generalized inverse kinematics solve for NR chain agents for $N > 2$ yet. To solve this we can soften the requirement to solve for arbitrarily close globally optimal solves.

3.4 No prior random sample solve

From the spirit of RANSAC, this is a simple method based on randomization. Given a goal, randomly sample qs (in their ranges) and keep the track of q^* which achieves closest approach. This can be stopped after a fixed number of samples or if closest approach is less than a threshold. This at limit should not be stuck at local minima and therefore is a

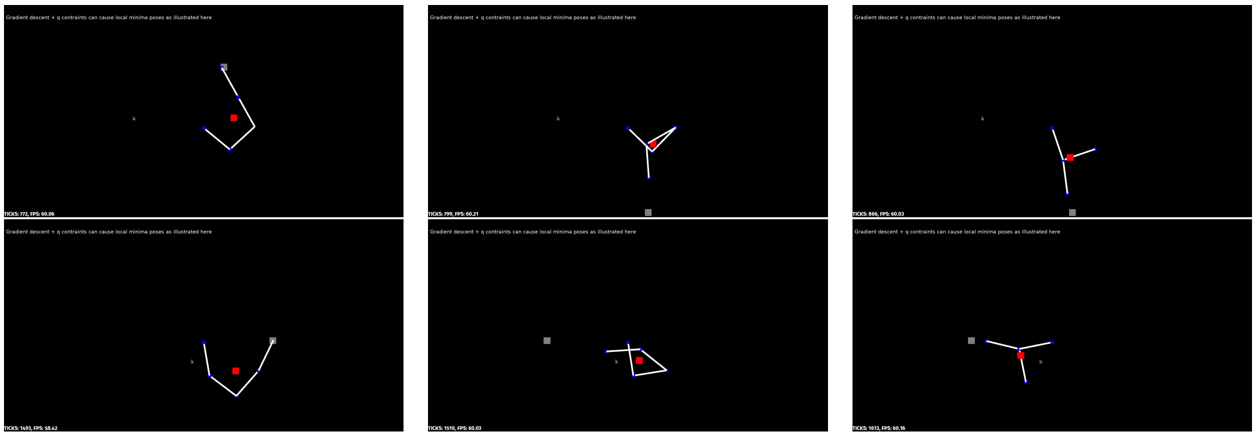
Gradient descent + q contraints can cause local minima poses as illustrated here
TICKS: 772, FPS: 60.06
Gradient descent + q contraints can cause local minima poses as illustrated here
TICKS: 799, FPS: 60.21
Gradient descent + q contraints can cause local minima poses as illustrated here
TICKS: 866, FPS: 60.03
Gradient descent + q contraints can cause local minima poses as illustrated here
TICKS: 1493, FPS: 58.42
Gradient descent + q contraints can cause local minima poses as illustrated here
TICKS: 1510, FPS: 60.03
Gradient descent + q contraints can cause local minima poses as illustrated here
TICKS: 1613, FPS: 60.16

Figure 6: Top 3 one motion, Bottom 3 one motion. Agent crosses arms to get stuck in a local minima pose.

arbitrarily near optimal solve. Given q^* just interpolate from current q to q^* to reach the goal. Since the distance calculation part is independent for each randomly sampled q, that part of the algorithm is parallelizeable. Since the actual function which ranks sampled qs can be arbitrary, we can actually use this to optimize more compound loss functions. For example in the implementation

$$L \equiv \alpha * (goal - end)^2 + \beta * (com_x - (goal_x - pivot_x)/2)^2 + \gamma com_y$$

is globally optimized.

3.5 Current state random sample solve

From the spirit of genetic algorithms, this is a simple method based on randomization, similar to no prior random sample solve. Instead of sampling qs randomly in whole range, we sample in small region around current q, set the best q^* as the new q and repeat. This method is more prone to local minima than previous method but given enough big sampling vicinity local minima can be avoided. Therefore this is asymptotically equivalent to no prior random sample solve. On the other hand this method is faster than the former, especially when number of dimensions of q space is large, since it only samples in a small vicinity around current q. Therefore this method scales better with q dimension and/or number of agents.

3.6 Integration with gradient descent

Both of the above solves are implemented and presented in the demo. But even with the globally optimal solve, most of the times, we do not exactly reach global minima but shall be in a close but unknown vicinity of it, depending on number of samples and dimension of q space. Since for continuing on a wall we need to actually get to a specified distance from the goal, we cannot just rely on these globally optimal solves. Therefore we first use globally optimal control to get within a close vicinity of global minima and then switch to gradient descent for ensuring reaching the goal. In practice we found that

$$\alpha * c_{global} + \beta * c_{gradient\_descent}; \alpha = \frac{1}{(1+ticks)^{0.5}}, \beta = 1 - \alpha$$

works well for our demos; where $ticks$ is the number of ticks completed from initial pose where globally optimal solve q was calculated. Using these methods the agents were able to avoid getting stuck in local minima poses.

3.7 Switching pivot

When the goal is reached by a free end, then the agent has to switch pivot. Switching pivot is essentially changing the pivot position, order of lengths of links, q_is and $q_i clamps$. When pivot is switched

1. Pivot position is the previous end position.
2. Order of lengths of links are reversed.
3. From the dynamics $q_{new} = [\sum_{i=1}^{n} q_i - \pi, -q_{n-1}, ..., -q_1]$.
4. From the dynamics $q_{new} clamps = [...(-q_i max, -q_i min)]; i = 1, 2, ..., n$.

We can see from points 3 and 4 that the first q is in general not bounded. Therefore the $q_1 clamps$ have to be $(-\infty, \infty)$. This means that we cannot enforce constraints on first joint angle if we want to switch pivot generally. This becomes a problem when one goal is high up the pivot and the next goal is low below the pivot, in which case agent tends to make unnatural cartwheeling motion. This is illustrated in figure 7. This can be solved by only

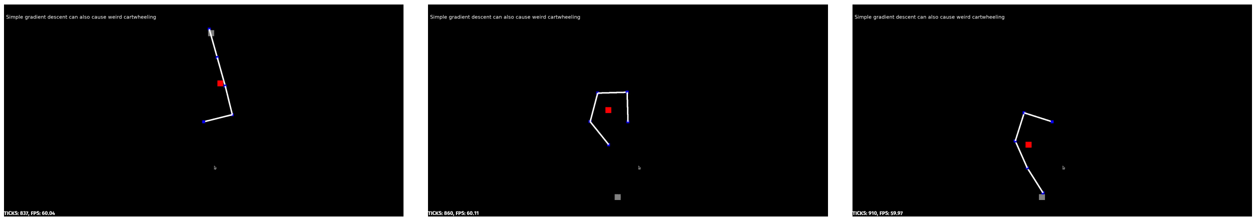
Simple gradient descent can also cause weird cartwheeling
TICKS: 837, FPS: 60.04
Simple gradient descent can also cause weird cartwheeling
TICKS: 860, FPS: 60.11
Simple gradient descent can also cause weird cartwheeling
TICKS: 910, FPS: 59.97

Figure 7: Left hand pivoting agent makes a cartwheel in attempt to reach the goal, because constraints cannot be enforced on q_1.

sampling certain range of q_1 values while performing globally optimal solves.

3.8 Matching hands

This alternating hand + switching pivot method does not always work as illustrated in figure 8. This can be resolved by matching hands.

1. If agent is reaching with left hand but goal on on right of pivot, set goal as pivot.
2. If agent is reaching with right hand but goal on on left of pivot, set goal as pivot.

This is illustrated in figure 9

3.9 Formulating $2 \times 2R$ limb coordination as an RL problem

Consider $2 \times 2R$ agent. We model a pair of arms using this agent. Since (global + gradient descent) inverse kinematics planners for individual arm (modeled using 2R chain agent) are

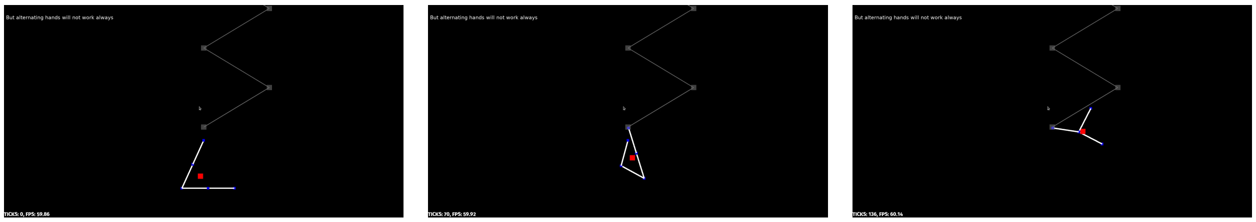
But alternating hands will not work always
TICKS: 0, FPS: 59.86
But alternating hands will not work always
TICKS: 70, FPS: 59.92
But alternating hands will not work always
TICKS: 136, FPS: 60.14

Figure 8: Switching pivots for alternating hands results in local minima pose.

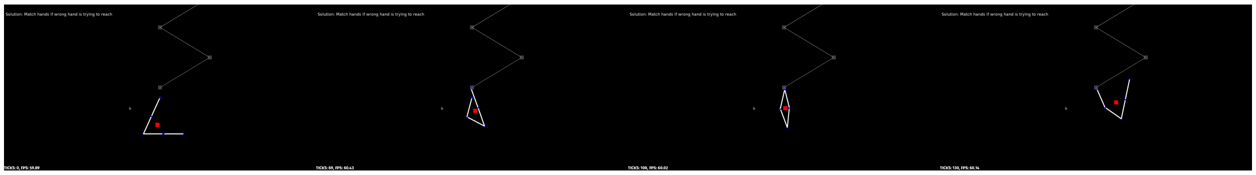
Solution: Match hands if wrong hand is trying to reach
Solution: Match hands if wrong hand is trying to reach
Solution: Match hands if wrong hand is trying to reach
Solution: Match hands if wrong hand is trying to reach

Figure 9: Switching pivots + matching hands avoids local minima pose.

already in place, we now just plan for the goals of ends of each 2R chain agent. Again since given a climbing route the goal of non-holding NR chain is determined, the only real variable here is the holding arm goal (i.e. where the neck should go). Since it is not very trivial as to where the neck should be placed, given a hand goal it is better to use a good function approximator like neural network for this task. We model this policy as a neural network optimized using cross-entropy method.

Input/Output design Since the lower level IK planners take care of current q state, q and $qclamps$ need not be part of input. It should only depend on current pivot, lengths of links and current goal. Due to this the policy becomes a discrete control (instead of continuous control) sampled at every pivot switch. This hugely decreases the burden on the network and really only outsources the most proper work to the network viz. holding arm goal prediction. This is reflected in the training where good results are achieved in relatively small training times. Further more, now in order to test the network we just need to check the predictions for goal positions, instead of pair of start and goal positions. Finally to make the system pivot shift and scale invariant we choose input as a concatenated vector of lengths of holding arm and non-holding arm and goal relative to pivot all of them scaled by sum of all lengths of links. The output is a two dimensional vector specifying the neck goal. After a prediction is made we scale back and shift the output before further use.

Reward function design The reward function has multiple components. During episode we have

1. Distance of holding arm end to its goal.
2. Distance of non-holding arm end to its goal.
3. Value of com_y.
4. Distance of com_x from its goal.

Since the input is scaled and shifted the rewards can be directly applied without any more pre-processing. The weights of distance from goal are generally more than those for com. Since we are using raw value of com_y as a reward, we need to make sure that we sample

key	value
generations	500
batch_size	50
num_episodes	20
num_episode_ticks	200
elite_frac	0.25
initial_std	1.0
noise_factor	1.0

Table 1: Cross entropy optimizer parameters.

many goals in the training area for each model, or else some good model can be thrown out for being unfairly tested just on high lying goals. After the episode we have

1. High reward for distance of holding arm end to its goal being small.
2. High reward for distance of non-holding arm end to its goal being small.

These rewards ensure the arm reaching the hold.

Network design The network used is simple fully connected network with two hidden layers of ReLU activation each with 16 nodes. Since the problem given to the network is quite simple a small network generally suffices. We have also tried Sigmoid activations but that did not work as well.

CEO parameters Table 1 shows cross entropy optimizer parameters used. It is parallelized on CPU cores and achieves a speedup of 36% on 8 cores for the parameters given in the table as illustrated in 10.

Left and right holding networks Since the cases where the agent holds the pivot using left hand and right hand can be treated as separate problems, we train two networks, one of each case. These are called Left and right holding networks. This further reduces the training time and achieves differing dexterous behavior.

Policy visualization Our design of input/output is such that for a given agent (the lengths are fixed) the policy is a mapping from $R^2 \to R^2$ (non-holding goal to holding goal). Because of this low dimensionality the policy can be visualized well. The figures 11, 12, 13 visualize left holding network. The figures 14, 15, 16 visualize right holding network. These visualizations helped a lot in the design and debugging of reward function, optimizer and the network.

4 experiments and results

We first model two arms using a 4R agent controlled by

1. Vanilla gradient descent.
2. Gradient descent + relaxation on every hold.

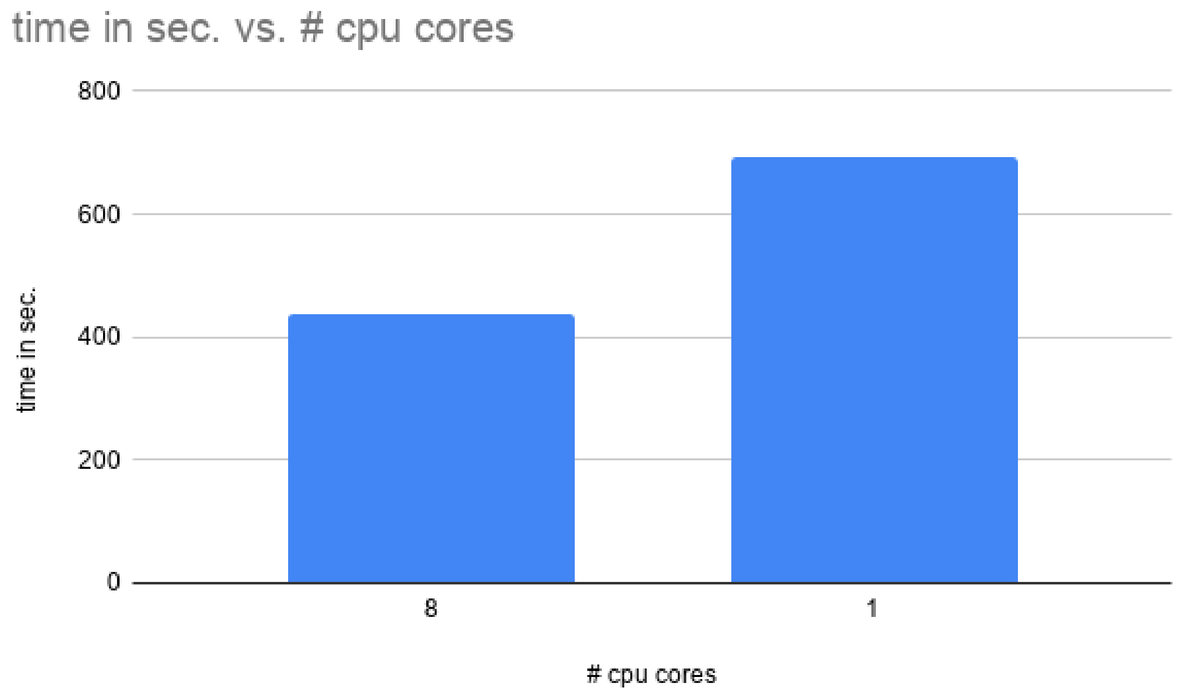

Figure 10: Speedup of cross entropy optimization using CPU parallelization.

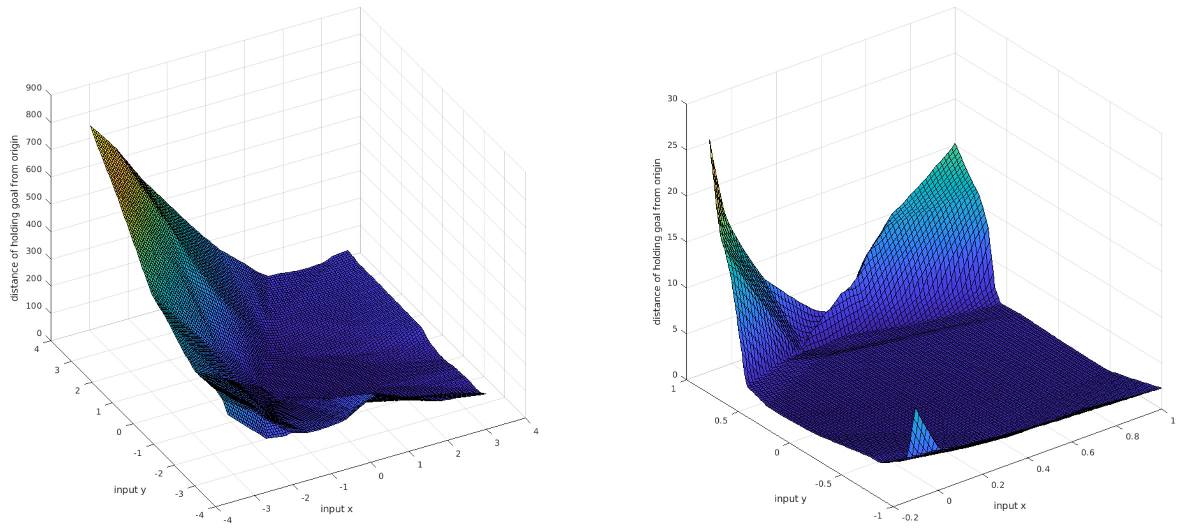

Figure 11: For the left holding network, distance of neck goal w.r.t hand goal. Left side shows zoomed out graph, right side shows zoomed in graph. Notice the valley of good predictions among the hills of random noise.

3. No prior random sampling solve + gradient descent.
4. Current state random sampling solve + gradient descent.

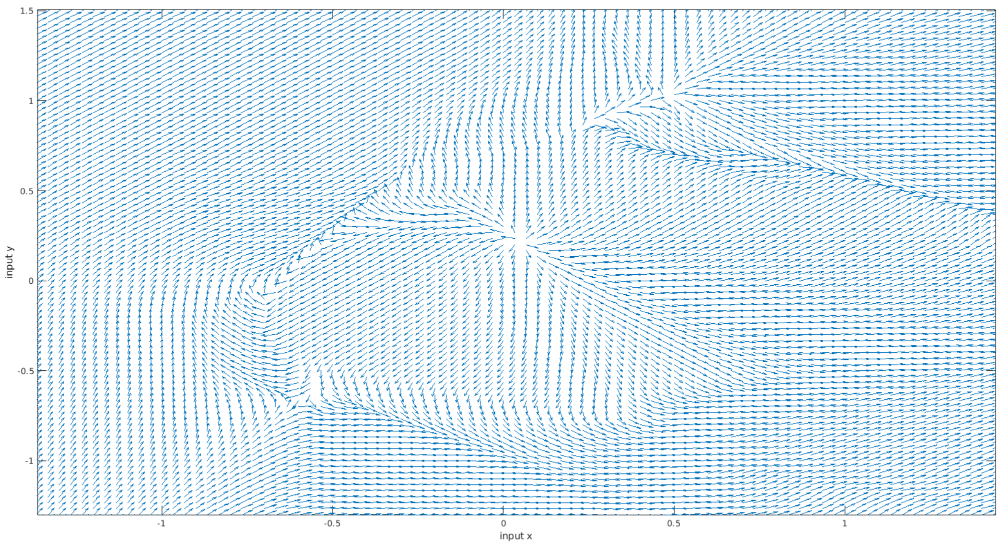

Figure 12: For the left holding network, direction of neck goal w.r.t. hand goal.

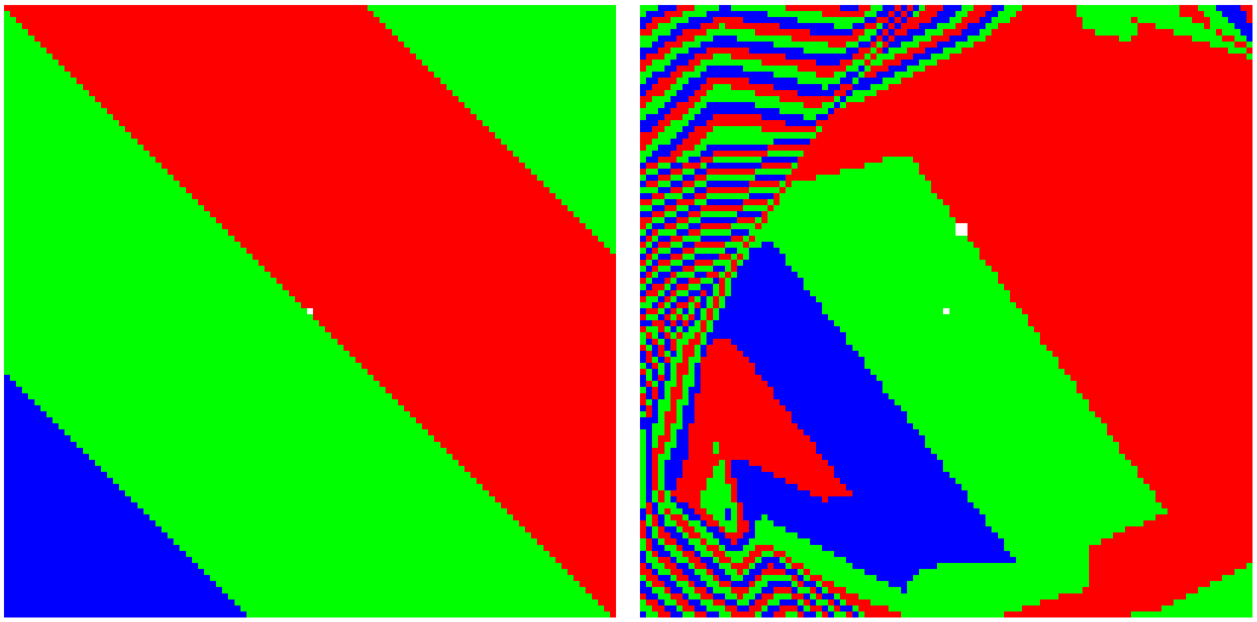

Figure 13: For the left holding network, texture distortion map. On the left side is the original texture, on the right is the mapped texture. Notice the smooth predictions in the trained region (right) and noisy predictions in the untrained region (left). Pivot is in the center.

All of these are tested on the same sophisticated climbing route involving left/right, up/down and diagonal moves. The standard route is shown in figure 17.

1. Vanilla gradient descent is simple but quickly gets stuck in local minima pose and cannot proceed.
2. Gradient descent + relaxation on every hold performs a little better but tuning the

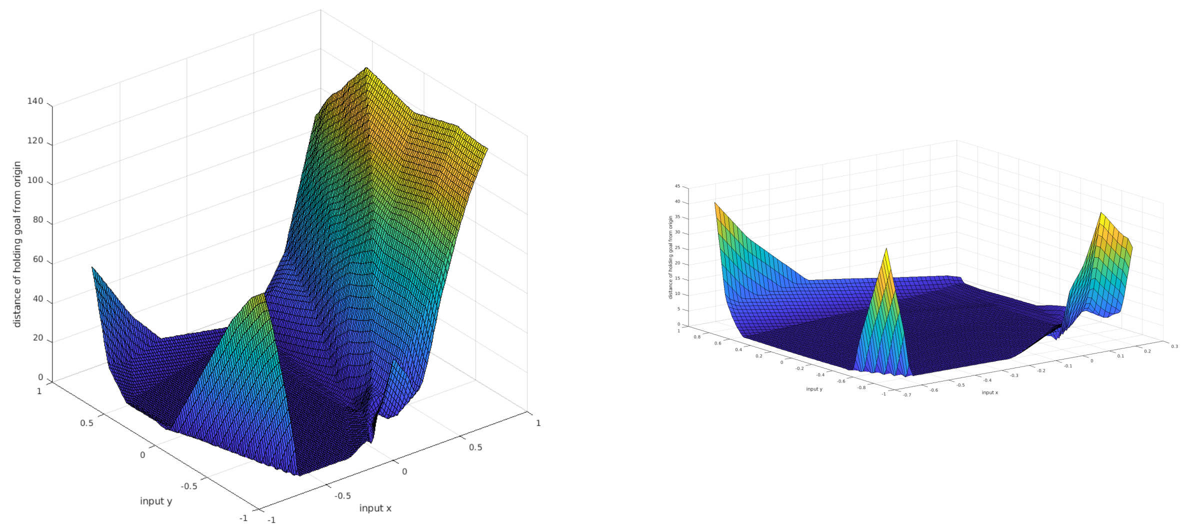

Figure 14: For the right holding network, distance of neck goal w.r.t hand goal. Left side shows zoomed out graph, right side shows zoomed in graph. Notice the valley of good predictions among the hills of random noise.

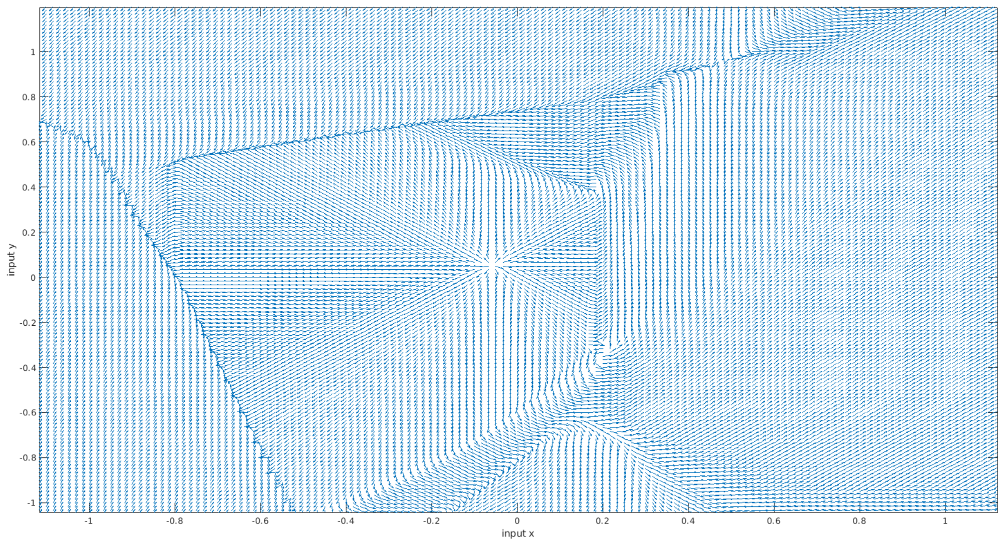

Figure 15: For the right holding network, direction of neck goal w.r.t. hand goal.

relaxation time is difficult and agent still generally gets stuck. Also using this method the agent reaches goal terribly slowly due to all the relaxation.

3. No prior random sampling solve + gradient descent does not get stuck in local minima, is decently fast and makes the agent reach the final hold pretty consistently.
4. Current state random sampling solve + gradient descent is also decently fast and makes

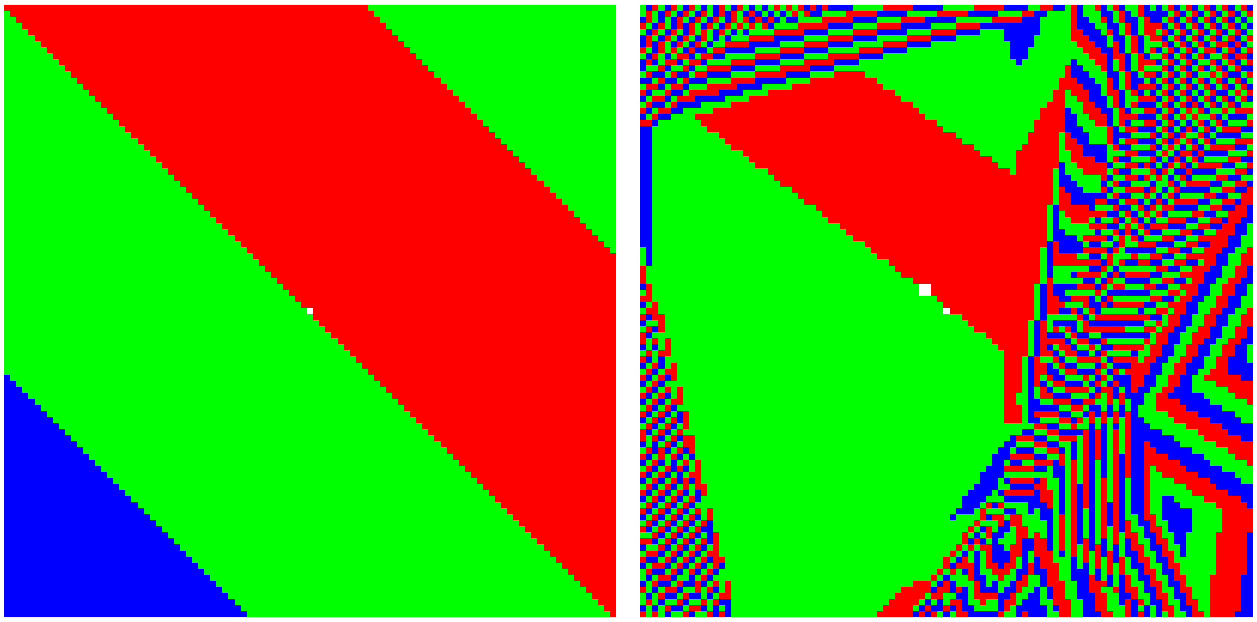

Figure 16: For the right holding network, texture distortion map. On the left side is the original texture, on the right is the mapped texture. Notice the smooth predictions in the trained region (left) and noisy predictions in the untrained region (right). Pivot is in the center.

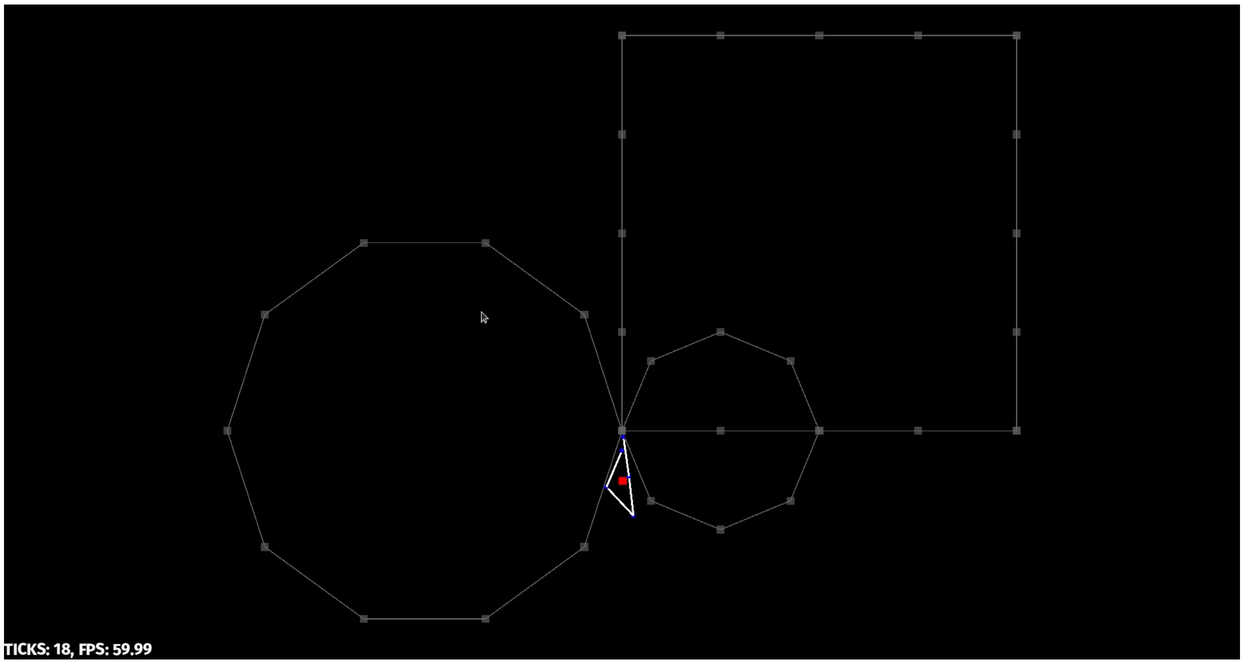

Figure 17: Standard climbing route: grey points represent holds, edges represent path. Route: left circle counter clockwise, right circle clockwise and then square in clockwise.

the agent reach the final hold pretty consistently. For this method the vicinity region parameter has to be tuned to avoid local minima poses.

We then model two arms using a $2R$ agent controlled by cross-entropy optimized network.

It is also tested on the same climbing route. This agent is also quite fast and finishes the route pretty consistently. This demonstrates the aspect of multi-limb coordination. Please refer to supplementary video for demonstration.

All the random global solves use randomness at their core and therefore produce different motion for same condition on each run.

Comparison to baseline

1. Baseline has no constraints on q_i and Δq_i. The current version has them.
2. Baseline has no explicit center of mass considerations. The current version has tunable center of mass control.
3. Baseline has no multi-limb coordination. The current version uses RL for that.
4. Baseline mainly uses gradient descent on loss function. The current version mainly uses globally optimal solves with gradient descent just for snapping on to a hold once close enough, thus avoiding local minina poses.

5 conclusion

We started out with the goal

1. To model human-like motion using simple heuristics.
2. To generate a controller for a given agent so that it reaches the finish hold.

Using the simple center of mass control we were able to make the motion significantly more natural looking. The near globally optimal random sample solves + gradient descent provides good planning stack for inverse kinematics level by avoiding local minima poses and preferring joint angles or enforcing joint angle constraints. The cross-entropy optimized neural network makes a good higher level planner which guides the lower level IK planner.

Biases There are no datasets involved in this work. No simulator was used for this work hence any heuristics used are entirely based on our judgment. There heuristics may fail completely (although unlikely) in a simulator or real world. Moreover energy of the agent is limited either as a robot or a human, but this is not modeled in this work.

Limitations Modeling human motion is done only using the center of mass control. Tuning for this is a cumbersome and difficult process and still does not guarantee avoiding weird poses. To resolve this a good physical simulator can be used on which these algorithms can be developed further. Even after globally optimal solves there is no guarantee that agent reaches the hold in a given amount of time since gradients vanish near local minima. This was the case with some runs of the best models. This is a problem for time critical missions. To resolve this problem one could scale gradients in a better task specific way. Since this work did not use a simulator, there might also be additional challenges in controlling an agent in a simulator or a real world, like torque limits, precision of motors, friction, air drag, noise etc.

Future work Although the methods in this work are shown for a $2 \times 2R$ agent, they can be easily extrapolated to any Tree of NR chains agent, which includes a full human-lik stick figure agent. The current work does not deal with multiple simultaneous pivots which are essential in actual climbing, where climbers generally uses two holds to hang. These problems would be a good follow-up work.

References

[Bull et al., 1995] Bull, L., Fogarty, T. C., Mikami, S., and Thomas, J. G. (1995). Adaptive gait acquisition using multi-agent learning for wall climbing robots. *Automation and Robotics in Construction XII*, pages 80–86.

[Grieco et al., 1998] Grieco, J. C., Prieto, M., Armada, M., and de Santos, P. G. (1998). A six-legged climbing robot for high payloads. *Proceedings of the 1998 IEEE International Conference on Control Applications (Cat. No.98CH36104)*, 1:446–450 vol.1.

[Kalisiak and Van de Panne, 2001] Kalisiak, M. and Van de Panne, M. (2001). A grasp-based motion planning algorithm for character animation. *The Journal of Visualization and Computer Animation*, 12(3):117–129.

[Naderi et al., 2017] Naderi, K., Rajamäki, J., and Hämäläinen, P. (2017). Discovering and synthesizing humanoid climbing movements. *ACM Trans. Graph.*, 36(4).

[Nagakubo and Hirose, 1994] Nagakubo, A. and Hirose, S. (1994). Walking and running of the quadruped wall-climbing robot. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 1005–1012 vol.2.

[Peng et al., 2017] Peng, X. B., Berseth, G., Yin, K., and van de Panne, M. (2017). Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)*, 36(4).