PegScrat: Quadruped Foot Placement Hierarchical Planning

Neil Janwani, Josh Hejna, Shiva Sreeram, Krishna Pochana

1) Introduction

For a quadruped robot, path planning presents a unique challenge. For an autonomous car, one needs to consider constraints like the size of the robot for avoiding obstacles and wheel size for turning radius. Compared to these systems, a quadruped contains more choices for a single action due to the high number of DOFs in the system. For example, a quadruped requires one to plan foot placement when moving from one location to another, which must be repeated for each leg, as well as assigning specific locations to four legs within areas of possible placement. Thus, it is easy to see that the dimensionality of the problem increases greatly along with the complexity of a local planner, thus requiring unique planning methods. This project seeks to implement a path planning algorithm for a quadruped robot on a simplified terrain model consisting of discrete allowed foot locations ("pegs") with motion constrained by leg lengths and geometry. In this paper, we will discuss the implementation of a hierarchical planner for navigating this quadruped over terrain with both random and patterned valid foot placements.

2) Approach

2.1) Problem Setup

To simplify the problem of quadruped navigation, it was decided to generate a set of pegs with uniform height over a finite space. These pegs represent the space of valid leg placements in arbitrarily complex terrain, thus discretizing valid paths through the space. All possible quadruped foot placement states exist within a 4-dimensional discrete space, where each axis selects a peg.

Continuous regions of valid foot placements (floors, etc) can be modeled by sampling pegs within these spaces. Non-navigable areas, such as high elevation areas or ravines, are simply excluded from sampling to ensure the quadruped avoids them.

We limit the scope of our problem to planning the series of foot placements themselves, ignoring the dynamics between these states. There is taken to be a valid movement between two foot placements if and only if they differ in the location of exactly one foot. Taking each state to be balanced, there will always be a dynamic transition between two such states, though the robot may not remain balanced while carrying out this transition. Generating paths with particular properties in the state transitions was deemed out-of-scope.

In sum, the 4-dimensional configuration can be viewed as a subset of $\{1, 2, ..., n\}^4$ with connections between any two points Manhattan distance 1 away from each other. Our overall task was to find robust methods for generating paths through this highly connected 4-dimensional space.

2.2) Approach Introduction and Rationalization

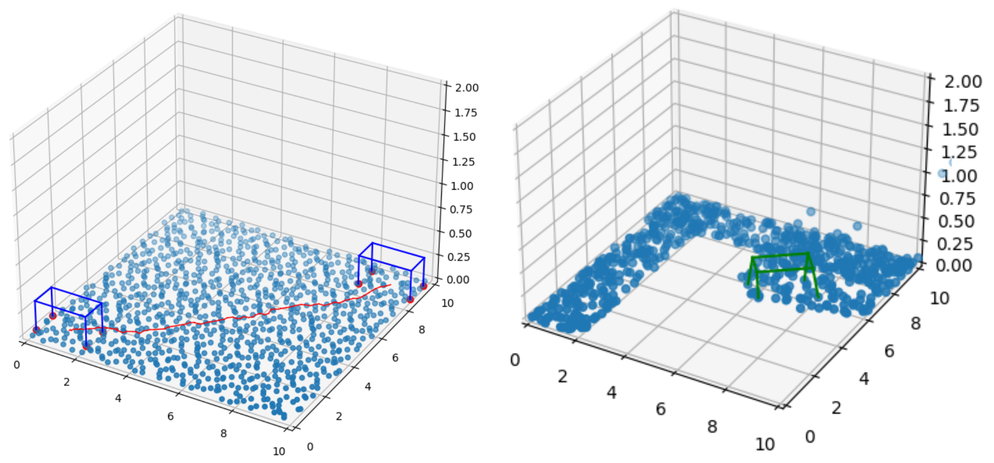
Naive graph search over this space is quickly shown intractable, as a reasonable model of terrain will easily contain at least one thousand pegs, resulting in a search graph with over one trillion nodes. Clearly, any search method hoping to find the actual best path, including A* with admissible or consistent heuristics, will require intractable computation.

Presented with the task of path planning over such a space, the two main methods covered in class of PRM and RRT based planning would both be functional at a first glance. Briefly, in the connected map case, the generated set of pegs would be sampled to make states of 4 pegs for the quadruped to stand on, with valid states being maintained. Then, the standard PRM process could be followed, with valid transitions being defined as those in which only one or two leg placements are moved. In the RRT case, the quadruped would begin in a valid start state (set of four pegs with each leg assigned to be on a peg). Then, at random a leg could be selected to move, and the direction of movement determined by the RRT process, with "direction" defined as the vector between the quadruped's torso and the end state.

However, following regular PRM planning would be redundant in our setup, if not detrimental, to planning. PRM works to discretize a continuous space, thus enabling time-efficient search methods over this discretized space. However, our setup is already discretized since it is made of a set of pegs, and thus probabilistic sampling of pegs does not need to be done. Further, randomly sampling over a set of pegs would be less likely to generate a valid state of 4 pegs, and less likely to generate states that involve transitions of one or two legs only. Increasing the number of pegs and specializing the state sampling methods can address the former problem, but not the latter, ultimately making naive PRM intractable for this problem.

Instead, we treat our discrete configuration space much like a continuous one, (motivated by its high density) and leverage a local planner to decide if sampled states are connected. Here, we leverage an A* search with an extremely pessimistic heuristic. This results in searches narrowly remaining around the path-of-steepest descent for the heuristic, which is a straight line between the start and goal in physical space. Overly-pessimizing the heuristic allows A* to find a valid path in a reasonable time, provided a path exists nearby to the path-of-steepest descent, a straight line in physical space. If such a path does not exist, then the search must consider all possible states nearby to this line, before moving onto the location of the real path, making the search intractable. Thus, by imposing search-time limits, we can use A* search over the configuration space, but only as a local planning primitive. The search decides if a straight-line path of short enough distance exists between two states. Usage of this local planner makes ordinary PRM planning practical.

Using this combination, we are able to find paths in all scenarios one would expect PRM to succeed in while maintaining the discrete nature of valid foot placements, which is required to model arbitrary terrain.



Here we see a raw, pessimistic A* performed in two different environments. When there is a straight-line path, it is found (left). When there is not, (right) the search will get stuck in a "local minima" and evaluate many thousands of states in that area in R^2, ultimately failing to find a path in reasonable time.

3) Technical Details

3.1) State Definitions

The quadruped's design involves four locations of interest for planning—the locations of the four feet tips. Thus, each state is defined as a 4-tuple of the pegs that the quadruped stands on, with each leg tip being assigned a determined peg within the state. Note that while the state exists in a discrete space, the foot placements are also occasionally brought into task space, where it would consist of four points in R^2.

3.2) Freespace Determination - Heuristic

As stated above, the configuration space is a 4-dimensional lattice, with the free space being a subset thereof. Determining if a point is free requires determining of the corresponding four points in R^2 represent a feasible position for the quadruped. To ensure overall computational tractability, there are strict performance requirements for this test. Determination for an arbitrary quadruped is, however, non-trivial, particularly if there are asymmetries in the legs or variation in the COM height is allowed.

To meet performance requirements, then, we implemented a heuristic check for viability for use in initial path generation. This heuristic conceives of the quadruped in two dimensions, with each leg emanating from the vertex of a rectangle. The feet are modeled to be able to independently reach any point within a specified radius of the leg origin. With this model, the free space test is made tractable. The center of the robot is placed at the average of the four peg positions in R^2. Orientation is selected by averaging the necessary rotations to align each leg origin with the corresponding peg. Selecting an orientation and position fixes the leg origins, so a simple distance test for each foot tells us if the state is in free space.

While this function works well for a large number of potential states, it is clear that compared to reality it is lacking. For instance, real quadrupeds have coupling between foot placements, in that placement of foot relative to leg origin affects the height of the leg origin, which in turn affects the height of the other leg origins, which in turn affect the reachable radii of the other legs. While developing a full model for this behavior is out of the scope of this project, we wanted a planner that could theoretically use a higher fidelity, less performant model.

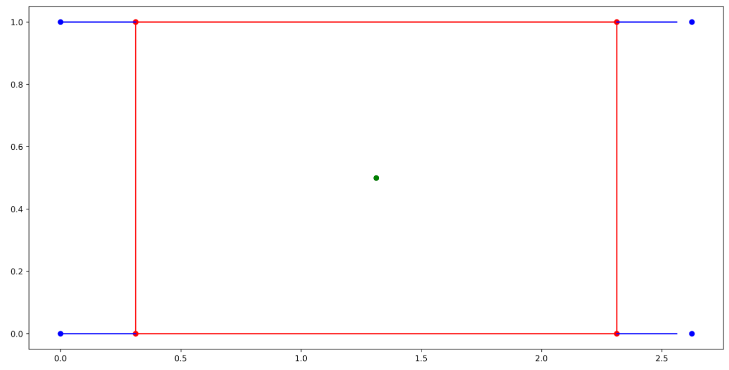
3.3) Freespace Determination - Numerical

As was discussed, there are cases where the basic feasibility function fails, so flexibility to include a more complex model is important. A higher-fidelity model is very important in situations where there are fewer pegs or only certain paths that can be followed that require the robot to stretch to its maximum range of motion. The specifics of our higher-fidelity model are unimportant, it merely represents a higher-cost calculation that can not be used in the planner inner-loop, yet is important to validate our final paths against.

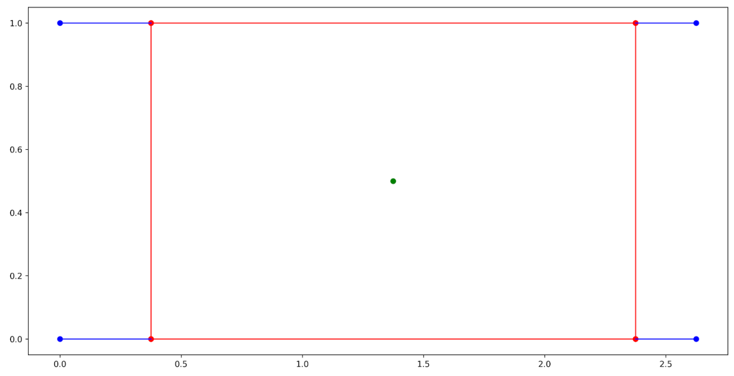
To limit scope, our higher-fidelity model again exists in two-dimensional space, but has varying leg radii, meaning optimal placement is no longer at the average of the four peg positions. The implementation uses the Scipy `minimize` function, which searches for a center position and orientation that minimizes shoulder-foot distances. Each distance is scaled inversely to the leg radii to account for the variable reaches. The initial guess for the minimization uses the heuristic selection method described above. From the result of this minimization, we determine if given the center and angle of rotation, all of the legs are able to reach the desired peg position and if so, we allow for this state to be considered for the path. This method is able to accurately classify as free some additional states (e.g. a state with all variable-length legs fully extended), while critically implementing the drop-in interface for a more complex model.

Critically, we also select the parameters of the heuristic model relative to those of the high-fidelity model such that the heuristic model should include almost all valid states, ensuring no possibilities are missed when planning.

To see the benefits, let us look at the following example. In the original implementation, if we were to have the set of pegs (given in blue in the images below), it would choose the following center (shown in green):



The robot's legs (blue lines) are able to reach the back with some slack (cannot be seen in a two-dimensional representation) but the front legs are unable to reach the pegs. As such, the configuration would not have been considered feasible. However, with this methodology:



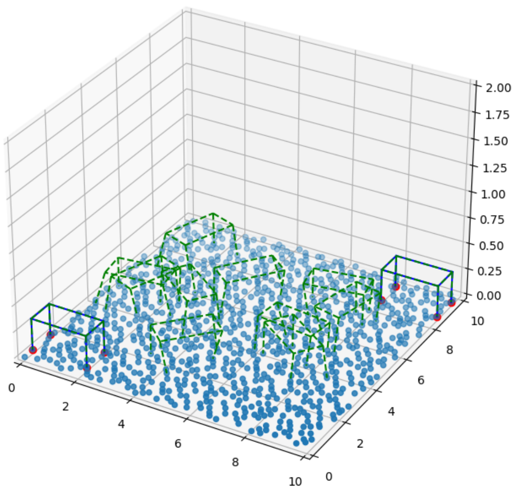
We can choose the more optimal center that is further along the x-axis to allow for the front legs to reach while allowing the back legs to utilize their full length. If we saw this in a three-dimensional representation, the robot would be fully splayed out, with the legs all in the same plane as the body.

3.4) Initial Planning

As briefly described above, planning is done using a PRM approach, which requires us to implement for our problem a sampling method, a free space test, and a connection test. The free space test was already discussed.

Randomly sampling the free space is non-trivial, because a random set of four pegs is highly unlikely to be a valid state. Thus, we instead randomly select a single peg and an angle. We then place an imaginary heuristic quadruped, with one shoulder centered over the peg and facing the specified direction. We then compute the centers of the other legs and use a KD-tree to sample nearby pegs, randomly choosing candidates from each of the three locusts to form a candidate set of four pegs. Then, as is normal, we perform a freespace test.

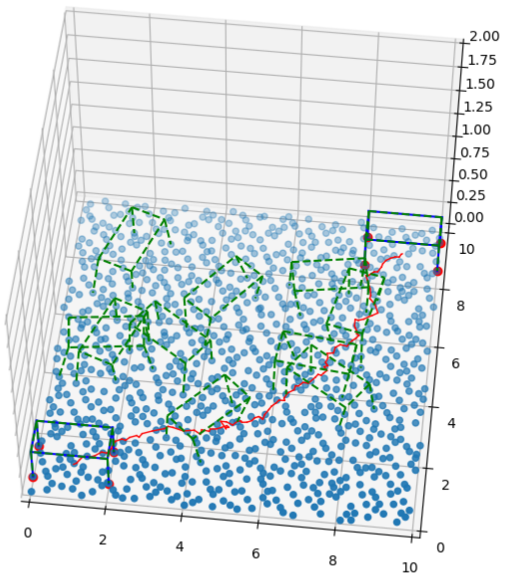
Connection testing is done using an A* search over the 4-dimensional configuration lattice. Adjacency testing in the search must be done on-the-fly, as we cannot explicitly generate the intractably-sized search graph. Adjacent states are generated from the current state by again using a KD-tree to list all the pegs within a reachable distance of the quadruped, and generating the list of all states consisting of a single foot movement to one of the pegs from the current state. Then, only freespace states are considered.



A sampling of states on a uniform peg distribution.

We consider the overall cost of a path to be the total distance moved by the legs, plus a constant penalty for each step taken. Thus, the heuristic chosen for A* search is the euclidean distance in R^2 from each foot in state one to the corresponding foot in state two. As mentioned before, thus heuristic is penalized by a large factor (50 to 100 per our tuning) so that this A* is tractable.

From this local planning and connection test, we obtain an actual path with a concrete distance between sampled states. These form the edges of the roadmap upon which the final path is computed using ordinary A* with a far-less-penalized heuristic. Local paths are concatenated to form the global path.



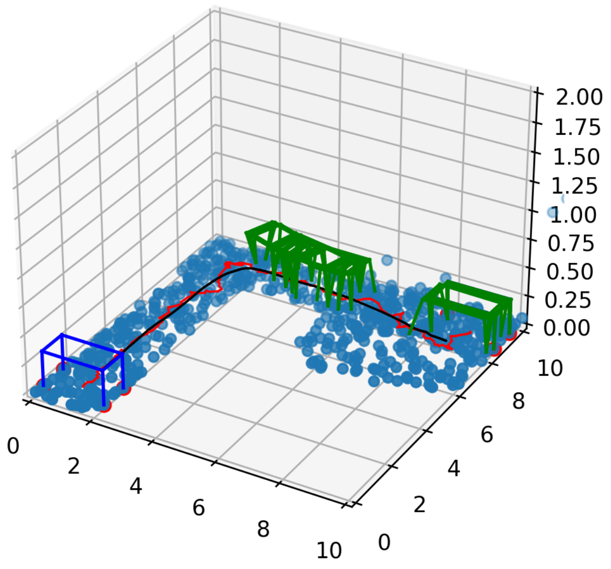
The resulting path consists of local paths between the sampled states.

3.5) ROS Visualization

To allow physical manipulations of legs simultaneously while keeping track of the relevant coordinate transforms, each leg is made to manage itself, with individual lists containing function calls to apply some transformation on a leg root or tip for each time step. In this way, all transformations can be done individually and at separate locations in the code/within each timestep. Before the final coordinates are sent to ROS, each leg iterates through the transforms and aggregates the effects. These function calls are ordered by when they should be applied, in order to maintain validity on all transforms. Thus, multiple legs can be easily manipulated simultaneously, and optimizations can then be done for walking time or other real-world constraints.

3.6) Post-Processing and High-Fidelity Freespace Testing

Since the A* searches used did not incorporate orientation into the segment weightings, post-processing of the paths was developed to avoid unusual orientations where the quadruped's orientation deviates largely from the overall path when there is no need to.



The above green quadrupeds showcase states in which the quadruped maintains its orientation with respect to the path. Noting these regions are useful, as they help identify where turns occurred. In some cases, A may be able to eliminate these unnecessary turns, resulting in a much cleaner path.*

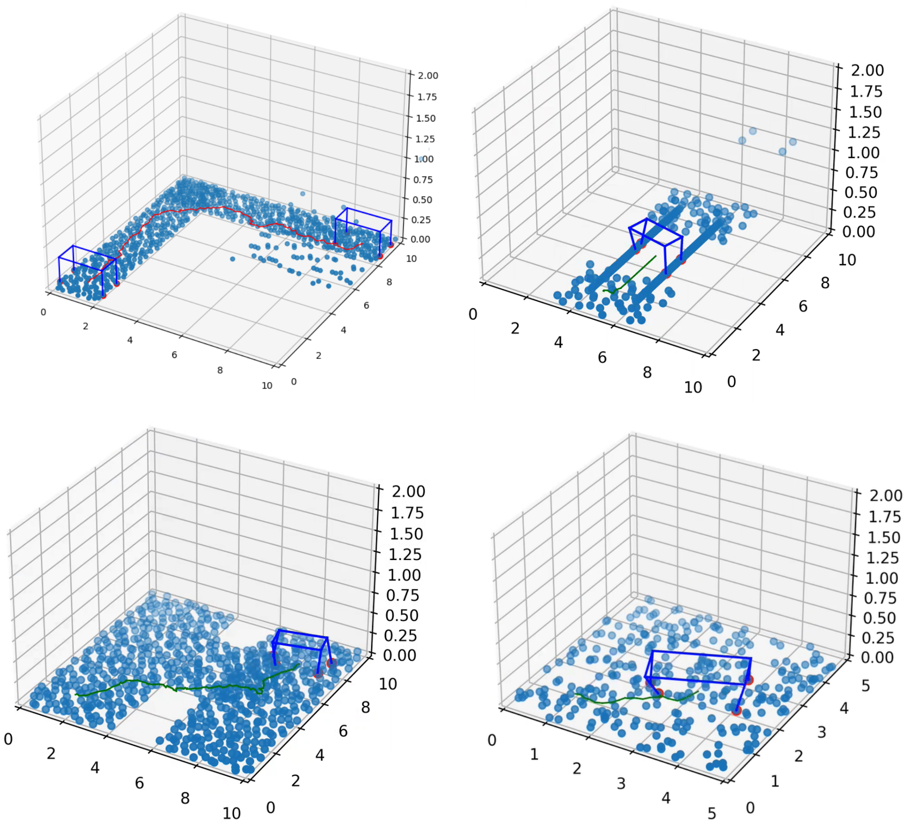
Thus, the post-processing function attempted to correct for these by iterating through the solved path and computing the orientation of the robot in addition to the orientation of the path around it. If these angles deviated significantly, then the surrounding states could be tested to skip the transition, or A* search run again between the previous and next states to attempt to find a better intermediate state or states. This enabled straight-line sections to be found, and thus the pure A* method (without hierarchical planning) can be used as it excels at straight-line solutions.

Separately from the above method, post-processing was also explored with methods similar to those covered in class, where states that are not immediately connected are tested to potentially eliminate states.

Additionally, the path is checked against the high-fidelity freespace model to ensure its validity in reality. A freespace check is made at every state along the path, and local A* searches are done using the high-fidelity freespace model to “patch” the resulting gaps. These searches are in between very close states, so there are relatively few freespace checks required. Thus, employing the high-fidelity model is tractable, and the result is a path in agreement with the high-fidelity model.

3.7) Peg Sampling Methods

When using pegs to model a continuously-navigable terrain, a naive uniform random sampling is non-optimal, because it is likely to leave gaps in the sampling unless the number of pegs is increased. Computation time can be lowered by sampling pegs using a lower-discrepancy method and lowering the number of pegs. For ease-of-implementation, we chose to use a randomized two-dimensional Halton sequence (a generalization of the van der Corput sequence), which will pseudo-randomly sample the space, with a uniform distribution of gaps between pegs regardless of sample size. Some of our test environments leverage this sampling technique, and others use ordinary uniform sampling.



4) Conclusions

This project represents an extension of the planning problems considered in class to a new medium, that of the quadruped. Our chosen simplification of the general navigation problem to a finite set of valid foot locations resulted in a high-dimensionality search space with no possible closed-form local planner. Moreover, the problem did not come with a closed-form (or even defined) freespace definition and required us to come up with our own.

We found employing overly-pessimistic A* search as a local planner to be fairly effective. This approach could generalize to other planning problems with discrete or hybrid configuration spaces. Others interested in solving similar problems may also find success with using A* as a local planner, but perhaps combining it with a different global method, such as RRT.

Overall, our approach was moderately successful and a fulfilling conclusion to the 133 series. 10/10 (squirrels are awesome).