Task and Motion Planning for Extendable-Reach Robots

Sarah Mitchell
Stanford Autonomous Systems Lab, AA290 Independent Study
November 23, 2020

Abstract – ReachBot is a robot concept from the Stanford Autonomous Systems Lab which uses rollable extendable booms as long-reach manipulator arms, allowing the robot to remain light and compact under neutral or adverse gravity conditions. The question of how to control such a robot is addressed with Task and Motion Planning. Two task planning methods are explored in this paper. A STRIPS-like decision tree approach offers many theoretical guarantees, but is computationally intractable without modification. A greedy heuristic approach proves to be successful, fast, and scalable, but is not robust nor optimal. Future work suggests alternative decision tree methods such as iterative deepening.

GitHub: https://github.com/scmitch/ReachBot290_TMP

Introduction

ReachBot is a robot concept from the Autonomous Systems Lab at Stanford University, designed for a primary mission of exploring lunar lava tubes and a secondary mission of servicing the Lunar Gateway space station. While many robots use articulated limbs with a serial chain of actuators, ReachBot instead utilizes extendable arms for locomotion and mobile manipulation. These arms are in the form of rollable booms, with a swivel point at the shoulder and a swivel point at the end-effector. By using extendable arms, ReachBot has access to a much larger viable workspace when compared to an articulated-arm robot of the same body size, as shown in Figure 1. This provides ReachBot with the reach capabilities of large robots, while still maintaining a light and compact body core. Since ReachBot's main applications are in neutral-gravity and adverse-gravity environments, this is a key advantage.

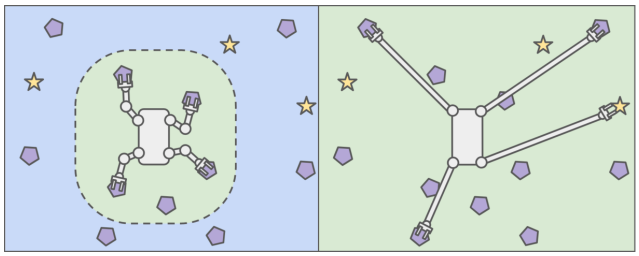


Figure 1: Comparison of articulated-arm climbing robots with ReachBot, showing reachable workspace in green [1]

ReachBot will travel by repeatedly ungrasping a foot, extending its boom toward a new anchor point, and then applying tension at this anchor point to bring its body into a new configuration with in-grasp manipulation. In this sense, ReachBot's locomotion is similar to a robotic hand with multiple fingers. Instead of the fingers manipulating the held object with in-grasp and re-grasp manipulation, ReachBot uses these same grasping techniques to move itself about the anchor points and hence traverse the terrain.

Motivation

We know that in order to move throughout these rough terrains and environments, we will need a climbing robot. There are currently climbing robots designs in development for similar approaches as ReachBot. One such robot is LEMUR 3, developed at the NASA Jet Propulsion Laboratory and shown in Figure 2 [2].

LEMUR 3 has four multi-joint climbing limbs, each of which is structured with a serial chain of actuators, as mentioned above. The end-effector for each limb is a microspine gripper, designed for grasping onto irregular rocky surfaces. LEMUR 3 has executed successful climbing tests scaling rocky surfaces, but its weakness is its small viable workspace for each limb movement.

The motivation for ReachBot's concept of extendable legs arises from our intended application to explore rough extreme terrains, specifically in unknown environments. For example, the geologic features that we must traverse inside an unexplored lunar lava tube are unpredictable. One such feature is shown in Figure 2. Although the surface im-

Figure 2: LEMUR 3 climbs along the face of a boulder, nearby a fissure that it may not be able to traverse [3]

mediately surrounding LEMUR 3 is easily climbable, next to it we see a large fissure dividing the right hand side of the boulder. If this gap is too wide for the robot to step over, and does not offer any means to circumvent the fissure, we would be stuck on one side of it. Without the ability to continue exploring on the other side, this could be costly to the mission if a point of interest is unreachable. A similar issue can occur in cluttered environments. Suppose that inside the lunar lava tube is a pile of large boulders, which are touching each other at irregular angles. LEMUR 3 may not be able to transfer its legs between the sharp angles of one boulder to the next.

In order to successfully and reliably maneuver around such obstacles, we want to create a robot with long-reaching arms at extendable lengths. This allows the robot to reach across gaps in the terrain, or reach high enough to climb up and over any large obstacles in its path. The ReachBot concept with extendable arms solves both of these precarious scenarios, providing a reliable method for traversing unpredictable environments.

Similar Problems & Approaches

One of the most common approaches I found for problems similar to ReachBot was Task and Motion Planning (TMP). The idea behind Task and Motion Planning is to take a desired goal state of the world, and divide up the process of achieving this goal state into a series of tasks. Each task is a discrete action, which is used to change the world state. However, each task also requires a plan of continuous motion to execute.

A useful analogy for gaining intuition for this process is shown in Figure 3, the Flashlight Example [4]. Suppose that we would like a robot to load two batteries into a flashlight. Assuming the flashlight begins empty, the sequence of tasks to achieve this goal are as follows: uncap the flashlight, insert Battery 1, insert Battery 2, and then place the cap back on the flashlight. Once the robot knows the series of tasks that it must do, it can then begin to plan a continuous trajectory of motion to perform each task. For example, the first task is removing the cap of the flashlight. To complete the task, robot manipulators must pick up the flashlight, grasp the cap, and remove it, all of which are achieved by continuous commands of the low-level actuators.

For this project, I chose to focus on the task planning part of TMP. Continuous-motion constraints on the physical robot model itself, such as structural and kinematic considerations, will instead by tackled by another graduate student in the Autonomous Systems Lab, Randall Ticknor, for his lab rotation. Thus the task planning algorithms considered in this paper can abstract away most requirements and technicalities of continuous motion.

A common approach for tasking planning is to use decision trees. When given a problem solving scenario, a robot will not automatically know which tasks in which order will achieve the goal state. To solve the problem, we construct a decision tree where each node represents a state, with branches for all possible actions from that state. When the state of any node is equivalent to the goal state, we have solved the problem and can reconstruct the task sequence by backtracking up the tree. The depth of the tree corresponds to the total number of actions taken to reach a given node. If we find multiple goal nodes on the same depth level of the tree, then we have found multiple solutions that achieve the goal in the same number of actions. When this happens, we are free to choose whichever node has a sequence of tasks which is considered most optimal by continuous motion planning.

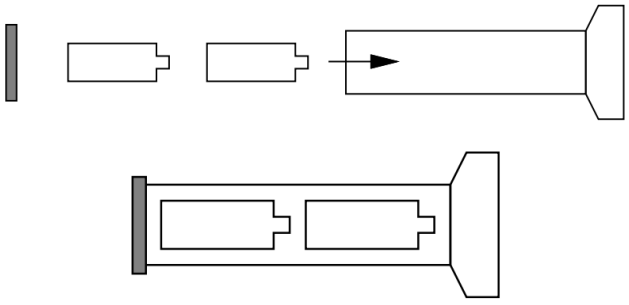
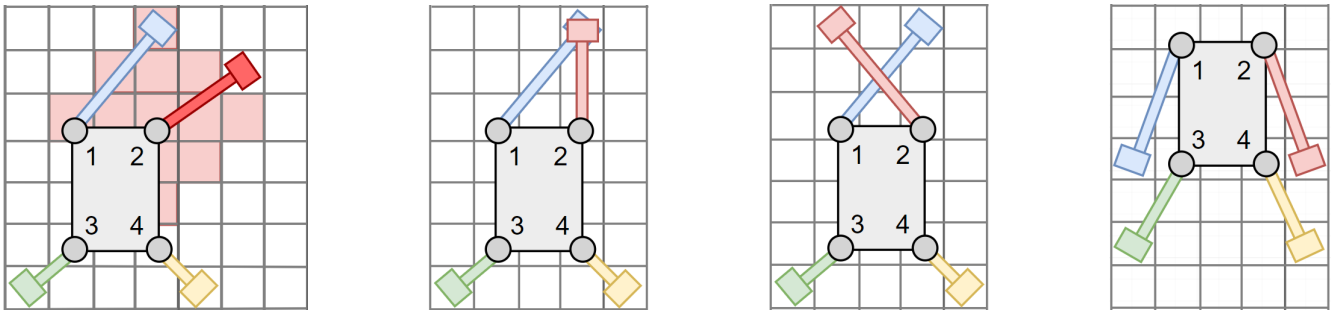


Figure 3: Task and Motion Planning Example: Loading two batteries into a flashlight [4]



(a) Feet must not exceed the maximum boom length

(b) Two feet cannot occupy the same grid square state

(c) Legs cannot intersect one another, nor the body

(d) Body must be positioned to keep all legs under tension

Figure 4: ReachBot basic task constraints on a discretized grid, shown as examples of illegal configurations

Methods - ReachBot Task and Motion Planning

I began by applying these Task and Motion Planning approaches to our extendable-reach locomotion model. Start by discretizing the two-dimensional grid world that the ReachBot model will be exploring. The robot's state is a defined by a list of five grid points: four represent the position of each of ReachBot's four feet, and the fifth marks the position of its body. We are not yet considering angular orientations of the body for this project, although this could be incorporated later. The goal state is a particular configuration of grid points which places ReachBot at a point of interest in the world. In TMP, the tasks for ReachBot are the sequential movements of its feet and body from one grid square to another. The motion planning is then conducted by the low level kinematics and actuators required to move the foot or body continuously from one point to the next.

I experimented with a STRIPS-like representation to implement the decision tree approach to task planning, because it's a logic-based representation that is common for these types of discrete planning problems [4]. STRIPS creates the decision tree in a breadth-first forward search. However, breadth-first tree representations are known to be computationally expensive. For this reason I also considered a greedy heuristic approach, which does not make use of a decision tree. In this method, the robot takes each action to move the corresponding foot or body as close to the goal position as possible.

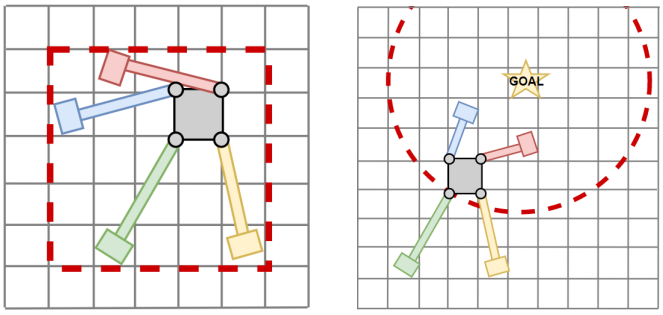
Task Constraints

To begin either one of these approaches for Task and Motion Planning, I first had to lay out the basic task constraints of the task planning problem. Note that these constraints do not take structural considerations of the robot into account. The task constraints only consider each configuration of the robot as a fixed point in time, without concern for the continuous motions required to transition from one fixed state to the next. An overview of the constraints is shown in Figure 4.

First we must take into account the maximum length of the booms. When moving a foot, this is used to limit how far away from the shoulder joint that it can reach. When moving the body, we need to ensure that it stays inside the intersection of reach spaces from each foot. When calculating the reach space of each foot, we'll make the assumption that every grid square contains a single graspable point. For this reason, no two feet can occupy the same grid square. Other illegal configurations are those in which any leg intersects another leg, or intersects the body.

The final constraint is slightly more nuanced, as it takes world physics into account. When operating a three dimensional world, ReachBot must apply tension along all of its booms in order to suspend itself above and around the terrain. We carry this constraint into our two dimension simulation by saying that all legs must be able to apply tension on the body in a given fixed configuration. For example, having all feet on the same side of the body, as shown on the right in Figure 4, is not a stable configuration. If tension were applied to the booms, then the body would be pulled into a different equilibrium – the body cannot suspend itself on one side of all the legs.

Since many physical aspects of ReachBot have not been finalized yet, I used simple geometric checks to approximate this constraint. One approximation is to constrain the body to be inside a box whose boundaries are defined by the maximum and minimum foot location in each dimension. This trims off many illegal tension configurations, but does not cover all cases. One counterexample is shown on the left



(a) Tension bounding box counterexample (body not under tension)

(b) Hind feet cannot move inside the dashed circle (closer to the goal than the body)

Figure 5: Two approximation methods, which together help enforce the tension task constraint

of Figure 5, where the body is within the boundary box yet not under tension. When using the heuristic greedy approach, I layered on an additional approximation in which the hind feet are constrained to be no closer to the goal position than the body is. Since the front feet are as far forward as possible, this will typically position the body somewhere between the front feet and the hind feet. The range of possible positions under this constraint is shown in Figure 5 on the right. Although there is no theoretical guarantee of these approximations, I have yet to see an obvious violation of the tension constraint when both are combined. Nevertheless, once the physical model of ReachBot's booms and grippers is more thoroughly developed, then these approximations will be replaced with a more robust constraint check.

STRIPS Decision Tree – Approach

STRIPS stands for the STanford Research Institute Problem Solver, and is a problem solving and planning system that can be tailored to many different scenarios [4]. It constructs the decision tree in a breadth-first forward search pattern. An advantage of this search pattern is that we are guaranteed to find an optimal solution, if one exists, where optimality is defined as taking the fewest number of actions to reach the goal. And since breadth-first searches consider all shallower tree depths before deeper tree depths, then we are guaranteed to first find the optimal solutions before suboptimal solutions. The major disadvantage of the STRIPS representation is that it creates a combinatorial optimization problem. For systems with many states and actions, this results in high computational complexity that may become intractable.

Once the decision tree finds a goal node, we can reconstruct the task sequence and bring in kinematics and dynamics constraints to attempt a feasible continuous trajectory. If we cannot create a continuous trajectory without breaking one of the motion constraints, then we discard this task sequence and repeat the process on another goal node of the same tree depth. If there are no nodes remaining at that depth, then the tree continues searching until it finds a new goal node farther down the tree. This method is guaranteed to find a solution if one exists. If a solution does not exist, however, then the algorithm would continue searching deeper tree depths forever. For this reason, we typically place a maximum depth on the creation of the decision tree.

Although there are many varieties on STRIPS-like representations, they all tend to have similar characteristics. The STRIPS system typically represents the states in terms of propositional logic statements, and the actions are designed to switch a binary value in one of the state conditions. In the flashlight example, one of the binary state conditions is whether or not Battery 1 is currently inside the flashlight. An action is able to flip this bit, such as inserting or removing the battery.

For the implementation of this system for ReachBot, we could preserve the spirit of binary representation by considering the feet to be represented by singleton matrices. Each matrix would represent the grid space with a 1 in the foot's current position and 0's everywhere else. However, this isn't practical. It's also equivalent to simply storing the foot's current position as a tuple, which is what we do for the sake of storage and simplicity. Actions are defined by which foot or body they move, and also to which grid point that part of the robot is moved. To find the full set of legal actions from a given tree node, first consider the reach space within maximum boom length for each foot, as well as the body, and then remove all configurations which violate any of the task constraints

The STRIPS representation also maintains a list of preconditions that are required to be met before any given action is allowed. For example, a precondition for inserting

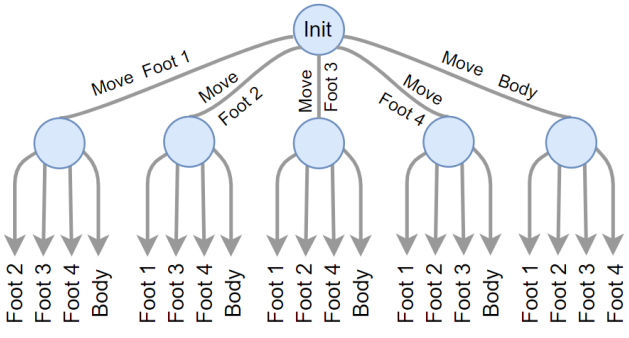


Figure 6: First two levels of ReachBot's decision tree using STRIPS; nodes do not branch on the same action twice

a battery is that the cap is not currently on the flashlight. ReachBot actions can also be associated with a set of preconditions using the STRIPS representation. One of the main preconditions for moving any given foot is that the remaining feet are located in such a way that ReachBot will be able to continue supporting itself once the anchor point at the current foot is removed. We could decide to either include this precondition check when building our decision tree of actions, or alternatively check for such physical stability when determining if a successful series of tasks on the decision tree is feasible in continuous space. Either way, these preconditions are covered here only for completeness in the discussion of the STRIPS approach and will not be implemented in this paper.

STRIPS Decision Tree – Results

Although the STRIPS approach is promising in theory, with many advantages, the high computational complexity of the decision tree for ReachBot turned out to be a major limiting factor. The method works for problems with very small state spaces and path lengths, but quickly becomes intractable for state spaces and path lengths of any meaningful size. Breath-first forward search on a tree like this grows exponentially with the depth of the tree. Specifically, the computational complexity is $O((|\mathcal{S}| \times |\mathcal{A}|)^d)$ where $|\mathcal{S}|$ is the cardinality of the state space, $|\mathcal{A}|$ is the cardinality of the action space, and d is the tree depth.

Before I calculated an approximate complexity for our problem, I first put in place one heuristic method for pruning the decision tree. Because we are interested in optimal solutions, we can remove all branches from the decision tree that result in moving the same foot twice in a row or the body twice in a row. Allowing such actions would effectively waste an action – if a foot is capable of moving to a more optimal position from any given configuration, then it would just move all the way there in one step. This is why the decision tree shown in Figure 6 allows all five possible actions from the root node, but only four actions from each remaining node. The previous action taken to reach a given node is not reconsidered for the immediate next branches.

In our case, the space of reachable legal states for each group of actions (i.e. for each movement of a foot or body) is different depending on the current limb configuration. To help illustrate the fast-growing complexity of these problems, I ran approximate numbers for some examples of a relatively small world. Suppose the maximum length of the booms is only two squares on the grid, similar to how the constraint is illustrated in Figure 4. Estimating by eye, I approximated that each foot would be able to reach an average of seven legal states for any given move. Each node can take four possible actions, as explained above. Although the feet are allowed to move in an order throughout the sequence, they will all have to move at approximately the same frequency. A depth of 5 in the tree allows each action, four feet and the body, to occur once. A depth of 10 in the tree allows each foot and the body an opportunity to move twice. The resulting number of tree nodes are shown below.

$$(|S| \times |A|)^d = (7 \times 4)^5 = 2 \times 10^8$$
$$(|S| \times |A|)^d = (7 \times 4)^{10} = 3 \times 10^{14}$$

We can see that even for these relatively simple examples, the number of nodes in the tree is already becoming unmanageable. This is not to say that decision trees are necessarily always intractable for ReachBot, but they will need to be carefully constructed and pruned in order to make the complexity more realistic. (See the Future Work section).

Fixed-Gait Greedy Heuristic – Approach

My next goal was to try a new approach to creating a task sequence for ReachBot, specifically one that avoids decision trees to achieve low complexity. The main reason that the STRIPS method is computationally expensive is because it allows every foot the opportunity to move on each turn, and it allows it to move to any legal square. To achieve low complexity, I streamlined the selection of actions and states by following a heuristic. The approach here is to choose a fixed order by which to move the feet and body, and cycle through this order of actions until the goal state is reached. This gait is recalculated for each problem based on the current robot configuration and goal state. The order starts with two “front" feet, who distances to the goal were measured to be the smallest. Next in the order is the body, followed by the two back feet. This heuristic is also greedy, meaning that every action will move its corresponding foot or body as close to the goal as possible, within the constraints. This approach should be sufficient for empty grids without obstacles.

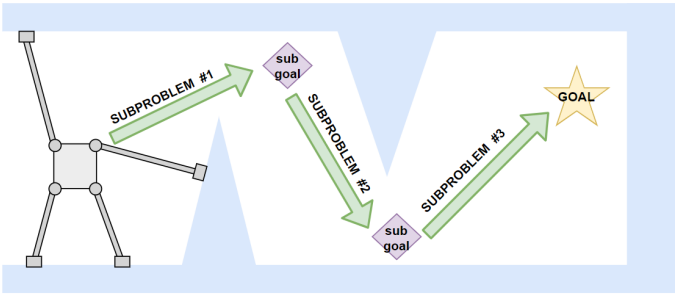
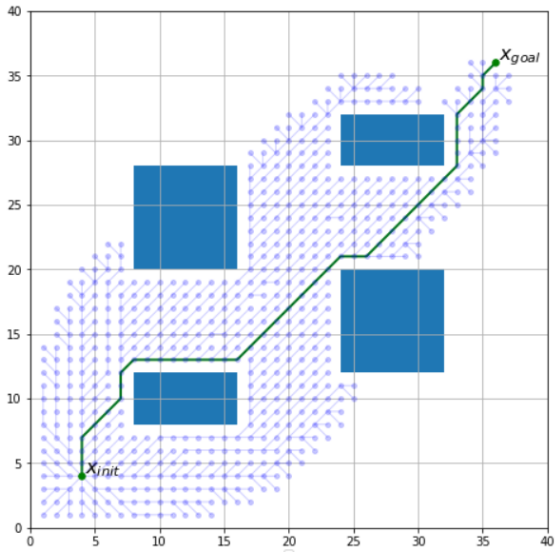
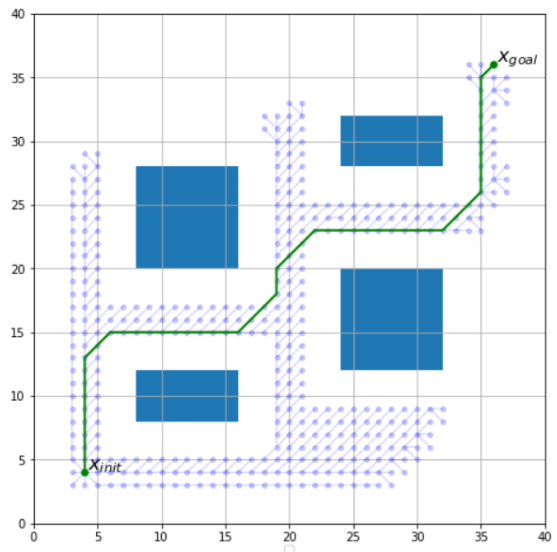


Figure 7: Dividing a feasible path through obstacles into line-of-sight subproblems for greedy heuristic approach



(a) A* path planning without obstacle inflation



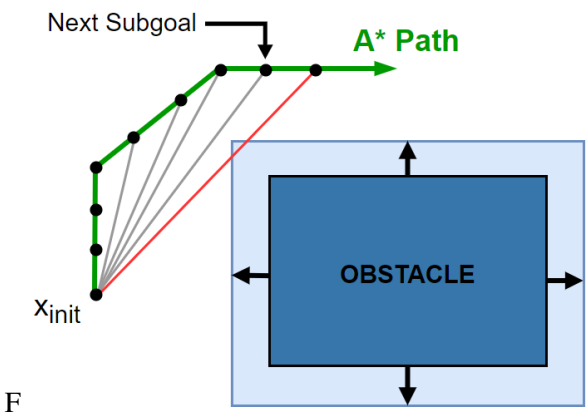
(b) A* path planning with obstacle inflation of 2 grid squares

Figure 8: A* search to find a feasible path through a cluttered environment

A more challenging test for this approach is to modify it to work with obstacles, since the ultimate goal is for ReachBot to be capable of traversing cluttered and unpredictable environments. One of the weaknesses of the greedy movement approach is that moving directly toward the goal can cause the robot to get stuck in local optimums, such as behind obstacles. This problem can be solved by dividing a feasible path to the goal into subproblems, where the beginning of each subproblem has a clear line-of-sight to its respective subgoal. If a sufficiently wide line-of-sight is presented to this algorithm, then any greedy movement toward to the global optimum (the goal state) will successfully avoid local optimums. A visualization of this segmentation is shown in Figure 7.

A feasible path can be created from any traditional path planning algorithm. For my approach, I used the A* search method. Before planning a path through the cluttered workspace though, I first inflated all the obstacles with a sufficiently wide buffer. The subgoal points defined along this feasible path will represent the desired location of the body. The legs can then move wherever they need to go in order to get the body to the right subgoal point location. However, we need to make room for the legs to move around the feasible path, else they snag on an obstacle which is the original problem we're trying to avoid. In Figure 8 we can see how path planning with no inflation leads to the path closely following the contours of the obstacles, leaving no room for the feet. In comparison, the path with inflated obstacles is sufficiently wide.

The next step is distributing subgoals along the feasible solution path, using as few subgoals as possible while still maintaining a line of sight between each one. To do this, I considered a line between the initial point to each subsequent point on the A* path. Each line is checked for a clear line-of-sight before moving on to the next, as shown in Figure 9. Once a path without a clear line-of-sight is reached, then the point immediately before it on the path is designated as the next subgoal. This point is then marked as the start of the next subproblem, and the process is repeated until it reaches the end of the path. Note that this line-of-sight algorithm should also use the inflated obstacles.



F

Figure 9: Determine subgoal locations by examining line-of-sight for successively further away path points

Fixed-Gait Greedy Heuristic – Results

Before adding obstacles to the workspace, I first tested the baseline fixed-gait greedy heuristic approach on an empty and relatively small 10x10 grid, as shown in Figure 10. The robot begins in the upper left corner of the grid and the goal is in the lower right, with the goal configuration lightly shaded in gray. The figure will link to an animation of the solution sequence. This test was a success, and illustrates the performance of the greedy heuristic on a grid that is already too computationally expensive for the STRIPS method to handle without further decision tree modification.

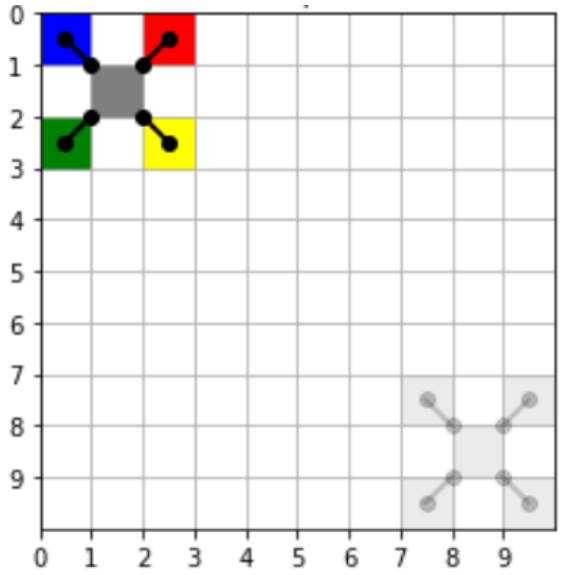


Figure 10: Set up for 10x10 empty grid test of greedy heuristic. Click here to view the animation.

Next I tested the greedy heuristic on a grid with a cluttered environment. I used the same path found in Figure 8 with inflated obstacles, and then divided the path into subproblems. The subgoals are all lightly shaded along the path in Figure 11, which again has a link to the animation solution sequence.

Note that dividing the A* path into line-of-sight segments only provides a series of single points along the path, each of which is designated as a subgoal point for the body. A full goal configuration, however, also requires a defined state for each of the four feet. For now, I've set each subgoal configuration to have an arrangement of feet closely surrounding the body, the same as I have used for the initial configuration and final goal configuration. Ideally, the algorithm should tailor the position of the feet around each subgoal separately, in order to smooth out the previous subpath with the next subpath as seamlessly as possible. Also, note that the robot is not required to exactly follow the A* path on the segments between the subgoals, as long as the final position lines up with each subgoal as a waypoint.

The resulting sequence of actions in the cluttered environment was also a success, making its way from the initial state to the end of the path without getting stuck behind any obstacles. Once I took out all of the plotting code, this method created the solution sequence in 1.2 seconds. This is a testament to the low complexity of this method, and shows that it would be effectively scalable to problems with bigger state spaces, longer paths, and more obstacles.

The main downside to this method is that it's not robust. Because we are fixing a gait and picking a greedy action, we are not considering all possible sequences of actions to reach the subgoal. There is no theoretical guarantee that we will find a solution if one exists, and moreover if we do find a solution, it will not necessarily be optimal. The greedy actions also have occasional issues where the robot will have boxed itself in with its own legs and get stuck, especially when it ends up crawling near the edge of an obstacle or the boundary of the grid world. A certain configuration is declared "stuck" if the method passes through an entire cycle of the gait without making any updates to the configuration. In order to make this method more robust, one could implement a recovery routine that untangles the legs before resuming to the gait process. I believe that often the robot could become unstuck by relocating even just the forward-most leg.

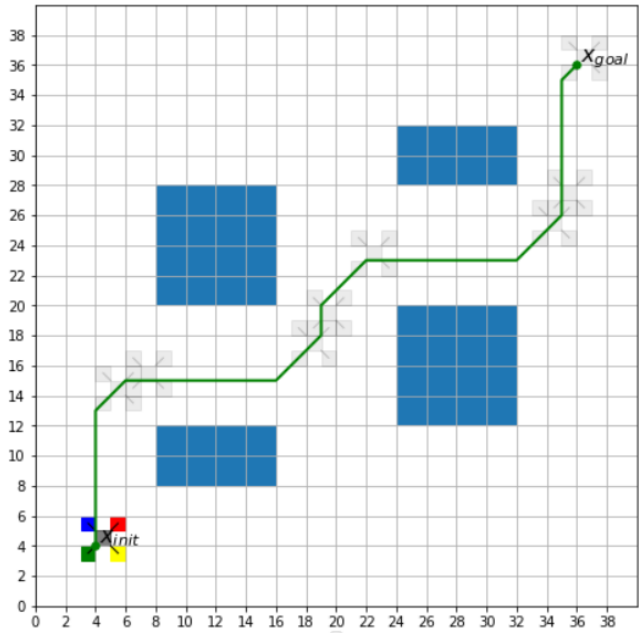


Figure 11: Set up for the complete test of greedy heuristic method on a 40x40 grid in a cluttered environment. Click here to view the animation.

Conclusion

This paper studied two methods of Task and Motion Planning to create a valid series of tasks to transport ReachBot through a discretized grid world. One approach used a STRIPS-like approach to decision trees, while the other used a greedy heuristic with a fixed cycle of actions. In the end, I wasn't able to create a working ReachBot model using the decision tree approach with the STRIPS representation. This system does have many theoretical benefits, though. By considered all legal actions from each state, and by searching the tree in a breadth-first forward search, this method is guaranteed to find an optimal solution (fewest actions taken) if one exists. Furthermore, if more than one node for a given tree depth matches the goal state, then we have the flexibility to choose whichever branching path works best for creating a feasible continuous trajectory plan. Ultimately though, the computational complexity of the decision tree without modification rendered this problem intractable for state spaces of meaningful size.

On the other hand, the fixed-gait greedy approach was very successful. By dividing a feasible path in a cluttered environment into line-of-sight subproblems, the greedy heuristic efficiently guided the robot through each subgoal configuration and ultimately to the final goal at the end of the path. Although not robust, this solution proved to be fast and widely scalable to bigger problems.

Future Work

Regardless of which task planning method is utilized, future development on the ReachBot project will need to combine the discrete task planning approaches with optimal continuous trajectory paths to execute each action in the task sequence. This will require adding additional physical constraints of the robot model, such as structural considerations or actuator limitations, among others.

Another major point of interest in future work on this project is further experimentation using decision trees for task planning. If using a breadth-first search such as STRIPS, then we would want to want to conduct a more thorough survey of potential heuristic methods for pruning the tree. If we can prune enough subtrees with sacrificing optimal solutions, then this method may become tractable. These pruning methods may benefit from a learning approach, such a reinforcement learning [5].

Decision tree methods other than STRIPS may also be considered. For example, one method that looks promising for our problem is iterative deepening. Iterative deepening is a heuristic depth-first algorithm which is asymptotically optimal for exponential trees in terms of time, space, and cost of solution path [6]. In fact, the feasible path produced from the greedy heuristic method in this paper may be useful as a seed to initialize an alternative, more complex algorithm that is designed for optimality.

For both the decision tree approaches as well as the greedy heuristic method, future developments should consider sparse anchor points in the workspace. In many cases this is a more realistic interpretation of a terrain environment, where not every reachable surface will be graspable. For applications in man-made environments such as the ISS or Lunar Gateway, which are static over time and have sparse grasping points, we may be able to make further simplifications. For example, we could have known map of the space station with various task plans predetermined and accessible in a look-up table.

If the method decided upon ends up planning with subproblems akin to the greedy heuristic approach, then future work would have to revisit the issue of optimizing where to place the feet surrounding each subgoal location. Some papers on Task and Motion Planning address this exact issue. One method is to represent the feasible space for each subtask as a manifold, as shown in Figure 12. To move from one subtask to the next, the trajectory must cross the transition space at the intersection of the two manifolds [7]. It is from this space that the method can select a transition configuration, with flexibility to choose a configuration that optimally smooths the trajectory between the two subproblems along the path.

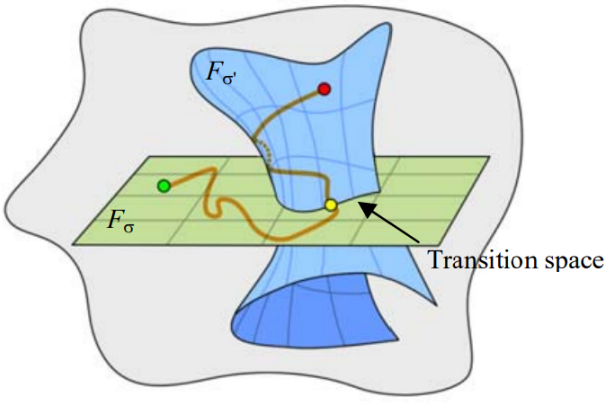


Figure 12: Transition space illustrated as the intersection of two subtask feasibility manifolds [7]

References

[1] M. Pavone, M. Cutkosky, and M. Lapôtre, "Reachbot: Small robot for large mobile manipulation tasks in neutral- and adverse-gravity environments," *NASA Innovative Advanced Concepts (NIAC)*, 2019.

[2] A. Parness, N. Abcouwer, C. Fuller, N. Wiltsie, J. Nash, and B. Kennedy, "Lemur 3: A limbed climbing robot for extreme terrain mobility in space," in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 5467–5473, IEEE, 2017.

[3] K. Uckert, A. Parness, and N. Chanover, "Deployment of an instrument payload on a rock-climbing robot for subsurface life detection investigations," *LPICo*, vol. 2197, p. 1031, 2020.

[4] S. M. LaValle, "Planning algorithms," *University of Illinois*, vol. 2004, 1999.

[5] R. Chitnis, D. Hadfield-Menell, A. Gupta, S. Srivastava, E. Groshev, C. Lin, and P. Abbeel, "Guided search for task and motion plans using learned heuristics," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 447–454, IEEE, 2016.

[6] R. E. Korf, "Depth-first iterative-deepening: An optimal admissible tree search," *Artificial intelligence*, vol. 27, no. 1, pp. 97–109, 1985.

[7] K. Hauser and J.-C. Latombe, "Integrating task and prm motion planning: Dealing with many infeasible motion planning queries," in *ICAPS09 Workshop on Bridging the Gap between Task and Motion Planning*, Citeseer, 2009.