From: Eli Case
Date: November, 5, 2023
Subject: MECH 450 – Homework 4

---

1. (a) We can determine the configuration space of the robot by examining each component of the state vector, x, and its corresponding topology. The state vector is written as $x = \begin{bmatrix} p & v & q & \omega \end{bmatrix} \in \mathbb{R}^{13}$. For the position, we have $p \in \mathbb{R}^3$. For the rotation, $q \in SO(3)$. The robot velocity is represented by $v \in \mathbb{R}^3$. Lastly, for the angular velocity, $\omega \in \mathbb{R}^3$. Putting this together, we have that the configuration space is $C = \mathbb{R}^3 \times SO(3) \times \mathbb{R}^3 \times \mathbb{R}^3 = SE(3) \times \mathbb{R}^6$.

(b) The control space can be represented as $U = \mathbb{R}^3 \times \mathbb{R}^3 = \mathbb{R}^6$.

(c) This is a holonomic robot, since there are not constraints on possible configurations of the robot, as it can rotate and translate freely in 3D space.

(d) The constraint described by $v_x^2 + v_y^2 + v_z^2 = 1$ prescribes that the tangential speed of the speed is constant and equal to 1. Physically, the robot is constrained to a sphere of constant radius in $\mathbb{R}^3$. This system is non-holonomic, since the robot is constrained to configurations that place the robot on the spherical surface.

(e) The constraint described by $v_x^2 + v_y^2 \leq 1, v_z = 0$ prescribes that the robot an only have velocity in the x and y directions that is less than or equal to 1, and can not have velocity in the z-direction. Physically, this means the robot is constrained to move within a plane in $\mathbb{R}^3$. This system is non-holonomic, since the robot can only attain configurations that physically place it on the plane.

2. (a) In RRT*, the function used to extend the graph to the randomly sampled state is modified. In RRT, we have a function $Extend(G, x_{rand})$ that steers the extends the graph, G by steering it from $x_{nearest}$ to x_{rand}, where $x_{nearest}$ is the node on the graph nearest to x_{rand}, the randomly sampled state. In RRT* we perform this procedure in the $Extend(G, x_{rand}$ function to add new state to the graph, and then we attempt to simplify the existing graph with the new node in the graph, x_{new}. To simplify the graph, we consider x_{new} and a set of nodes $x_{near} \in X_{near}$ that are within a distance of specified radius r from x_{new}. We want to find shorter paths between x_{new} and $x_{near} \in X_{near}$, according to a cost function that accounts for the edge distance and an addition heuristic. We accomplish this task in two steps.

i. Loop over all $x_{near} \in X_{near}$ and see if current cost of x_{new} is higher than the cost of the obstacle-free path with one edge between x_{near} and x_{new} pllus the cost of x_{near}. If the existing cost is higher, we add the edge from x_{near} to x_{new} and remove the edge from the node x_{min} to x_{new}, where x_{min} is the nearest node in X_{near} to x_{new} in the original path from x_{near} to x_{new}.

ii. We then loop over all $x_{near} \in X_{near}\ x_{min}$. If there is an obstacle free path from x_{new} to x_{near}, we check if the cost of x_{near} is higher the cost of the obstacle-free path from x_{near} to x_{new} plus the original cost of x_{new}. If the cost is higher, then we remove the edge from the parent of x_{near} and x_{new} and add the edge from x_{near} to x_{new}.

(b) RRT* could be used to plan the motion of a second-order car, if we add functionality to propagate the car state from x_{near} to x_{rand} by sampling controls and integrating the equations of motion that describe the car system to find a control set that allows the car to move from x_{near} to x_{rand}, given its dynamic constraints. With the functionality added, we have the basic RRT algorithm for planning with controls. For RRT*, we need to add some additional functionality to the new $Extend(G, x_{rand}$ function to perform the path simplification steps based on the newly added state x_{new}. With controls, this process is much more complicated, as if we consider adding an edge to the graph, we much ensure that we can apply controls to propagate to match this new edge.

From the discussion in part a), we will replace existing paths from x_{near} to x_{new} with multiple edges to a path with one edge from x_{near} to x_{new} if it is shorter and collision-free with RRT*.When we plan with controls, we must randomly sample control to propagate the state directly from x_{near} to x_{new}, to see if this new edge on the graph is attainable given the dynamic constraints of the car.

A path that is "shorter" from x_{near} to x_{new} given the heurisitcs used in geometric planning, may not be shorter based on the dynamic constraints of the car, potentially adding a path that is longer than the original path. Additionally, given that it is expensive to propagate the system with randomly sampled controls in RRT, performing the propagation several times for each new node added to the graph would be very expensive. Thus, using RRT* to plan for a second-order car is theoretically possible but would be very expensive and may not even produce an asymptotically optimal path based on the heuristics used to simplify the graph in the graph extension function.

3. (a) The first sequence is as follows.

i. $MOVE(R_1, C, E)$
ii. $MOVE(R_2, B, D)$
iii. $MOVE(R_3, A, F)$
iv. $MOVE(R_1, E, A)$
v. $MOVE(R_2, D, C)$
vi. $MOVE(R_1, A, B)$

A longer sequence to achieve the same result is the following.

i. $MOVE(R_1, C, F)$
ii. $MOVE(R_2, B, D)$
iii. $MOVE(R_3, A, E)$
iv. $MOVE(R_1, F, A)$
v. $MOVE(R_3, E, F)$
vi. $MOVE(R_2, D, C)$
vii. $MOVE(R_1, A, B)$

(b) In this problem, a centralized planner will likely be more useful. This is because the robots need to know each others positions so that a robot is not in collision during movement or at the destination node.If we create a discrete configuration space of all the robot states on the graph, we will see that many configurations are invalid to avoid collision, and the validity of valid configurations change as the robot itself moves, and as the other robots in the system move. As a result of having a dynamic configuration space, it will likely be more suitable to plan over the entire configuration space (since it is relatively low dimensional) for the super robot compute the individual robot priority at each time step to find the shortest path, rather than having to compute the individual robot priority as each move command is executed for the individual robots in the decentralized planner.

(c) Because the configuration space is now high-dimensional, a decentralized method will likely be more suitable, as constructing and searching the entire configuration space will be very expensive. We can use the decentralized method and compute the priority using an efficient method for each robot movement command.

4. (a) We can generate the following task plan to solve the stated problem.

i. **True Predicates:** (ontable a) (ontable b) (on c a) (clear b) (clear c) (holding d)
Action: putdown(d)

ii. **True Predicates:** (ontable a) (ontable b) (on c a) (clear b) (clear c) (ontable d) (clear d) (handempty)
Action: unstack(c a)

iii. **True Predicates:** (ontable a) (clear a) (ontable b) (clear b) (ontable d) (clear d) (holding c)
Action: putdown(c)

iv. **True Predicates:** (ontable a) (clear a) (ontable b) (clear b) (ontable d) (clear d) (ontable c) (clear c) (handempty)
Action: pickup(b)

v. **True Predicates:** (ontable a) (clear a) (ontable d) (clear d) (ontable c) (clear c) (holding b)
Action: stack(b c)

vi. **True Predicates:** (ontable a) (clear a) (ontable d) (clear d) (ontable c) (handempty) (clear b) (on b c)
Action: pickup(a)

vii. **True Predicates:** (ontable d) (clear d) (ontable c) (clear b) (on b c)(holding a)
Action: stack(a b)

viii. **True Predicates:** (ontable d) (clear d) (ontable c) (handempty) (on b c) (on a b) (clear a)
Action: pickup(d)

ix. **True Predicates:** (ontable c) (on b c) (on a b) (on d a) (clear d) (handempty)
Action: done

(b) Since only boolean statements can be encoded in PDDL, one cannot add additional information about the objects involved in the task planner. For example, if one block has a larger size than the other block, if one block is much further away from the other blocks and should be moved last, etc. As a result one cannot create heuristics to optimize the solution of a task planning problem in PDDL for certain problem conditions.

Regardless, PDDL is very flexible, as an problem can be encoded into PDDL, if it is constructed with the proper logical framework. For a task planning problem, we need to decompose the problem into individual actions that require logical preconditions and generate logical effects to apply in a certain sequence to complete each task. Thus, PDDL encodes a series of logical steps to complete a problem that are generalizable to all problems, but the proper logical framework for a task planning problem can be difficult to construct in some cases.