Multi-Robot Motion Planning for Quadruped Robots

Srikumar Brundavanam[1*], Akshay Raman[1*], and David Ologan[1*]

***Abstract*— This paper explores multi-robot motion planning strategies for quadruped robots. Here, we present three distinct methods: a sequential RRT-Connect (Rapidly Exploring Random Tree Planner), a joint state-space RRT-Connect Planner, and a Conflict Based Search. Our study showcases the success of each approach by generating kino-dynamically feasible and collision-free body trajectories for multiple legged robots navigating diverse terrain. Leveraging Quad-SDK, a open-source ROS-based framework designed for quadruped locomotion, we incorporate its footstep planner and low-level motor controller to implement these strategies. Applying all three global planner techniques proved successful in most scenarios with the conflict based planner approach having the lowest cost to goal. The sequential RRT had the quickest solve time due to its more simplistic approach.**

I. Introduction

In recent years, there have been notable advancements in the field of multi-agent motion planning. Motivated by the desire for increasingly collaborative and coordinated actions in robotics, researchers have attempted to solve this issue by employing a variety of sampling based algorithms. In spite of this recent success, quick and effective multi-agent planning for legged robots remains elusive.

Quadrupedal robots have demonstrated their ability to successfully navigate difficult environments particularly in those were wheeled platforms might fail. In order to realize the benefits of legged robots for tasks like exploration, warehouse robotics, and search and rescue, legged robots must be able to chart collision-free paths with each other while negotiating an increasingly complex and unpredictable environments. As such we explore a variety of common planning algorithms to see which presents the best performance in allowing multiple quadruped robots to interact and collaboratively plan efficient trajectories.

To tackle the complex multi agent trajectory planning problem as it applies to quadrupeds, our paper presents the following three methods:

1) Sequential RRT, that first solves for the path of an initial robot, and then constrains all subsequent solves with the sequence of states corresponding to its path
2) Joint Space RRT, which simultaneously plans in the state space of two robots
3) Conflict-Based Search, that iteratively calls a low-level planner, and re-plans with constraints to address path conflicts

The remainder of this work is organized as follows. Sec. II summarizes the related works. Sec. III-A, B, C, D describes the state representation of each respective planner, and their individualized implementation details. Subsequently, simulated results and solved trajectories are presented in Sec. IV, and V.

[1*]The authors contributed equally. Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15232, USA, vbrundav, akshayra, dologan@andrew.cmu.edu

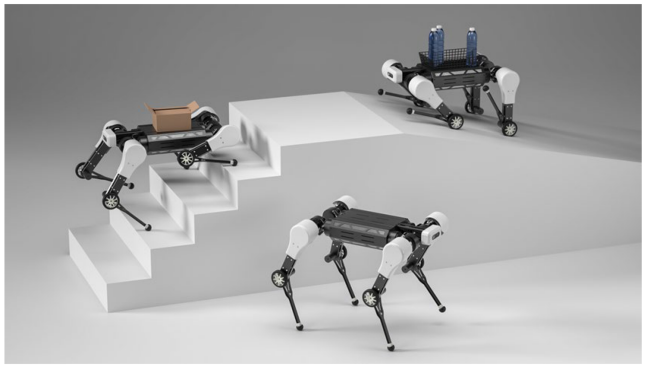

Fig. 1. Multiple quadruped robots working together in cohesion

II. Related Work

Prior work has demonstrated various approaches towards solving the multi-robot planning problem. Sampling based planners seem to be the common thread, as [1], implemented a kino-dynamic sequential and joint RRT* for UAV's. Similarly, [2], implemented an Multi-Robot RRT* that scaled well for up to 10 robots. These approaches are less feasible for quadruped robots given that they scale poorly as the number of robots increase, and the state space of a quadruped is much more complex than that of a UAV. Another common approach to multi-agent planning is Conflict Based Search [3], which avoids over-constraining any robots. [4] demonstrates the effectiveness of such an approach using non-linear dynamics and MPC as its low level controller.

Notably, these methods have yet to be more widely extended to legged platforms like quadruped robots. Available quadruped locomotion tools like Quad-SDK currently support multi-robot visualization [5], and fast global motion planning for a singular agent.[6] However, the platform lacks support for multi-robot planning and simulation, particularly as the number of agents increases beyond two. As such, this project aims to extend each of these methods into Quad-SDK, leveraging its modular structure, visualization tools, local footstep planner for our implementation.

III. Methods

A. State Formulation, RRT-Connect

In order to effectively plan trajectories for each quadruped robot, key simplifying assumptions are made. First, the mass

of the legs is assumed to be much less than that of the body, allowing for their exclusion during high level planning as the dynamics of the robot remain relatively unchanged. Next, we assume the nominal roll of the robot to remain relatively constant and that motion primitives for the robot during stance and flight phase can be approximated as piecewise polynomials. More details on this representation can be found here [6]. For our sampling based planner, the representation of discrete time robot states is defined as a vector of robot body position, orientation and velocity. Note here that in the joint case, the size of this vector is scaled appropriately to the number of robots. Start and goal states are defined similarly, except the velocity portion is 0;

$$s = \begin{bmatrix} q \\ \dot{q} \end{bmatrix} \tag{1}$$

$$s = \begin{bmatrix} q_x & q_y & q_z & q_p & \dot{q_x} & \dot{q_y} & \dot{q_z} & \dot{q_p} \end{bmatrix}^T \tag{2}$$

Given the complexity of the state space of the system, the low level planner runs an RRT-Connect for speed. The algorithm is as follows with a few notable differences. During connect and extend operations, intermediate states are interpolated, and checked for kino-dynamic feasibility. Additionally, in both the sequential and CBS approaches, the collisions with the other robot path or against constraints are checked for validity.

Algorithm 1 RRT-Connect

Input: Initial configuration q_{init}, goal configuration q_{goal}, search space X, step size Δq
Output: Path P from q_{init} to q_{goal} (or failure)
Initialize tree T_{init} with node q_{init}
Initialize tree T_{goal} with node q_{goal}
while $\neg$ReachedGoal$(T_{\text{init}}, T_{\text{goal}})$ **do**
    $q_{\text{rand}} \leftarrow$ RandomConfig(X)
    $q_{\text{nearest}} \leftarrow$ NearestNode$(T_{\text{init}}, q_{\text{rand}})$
    $q_{\text{new}} \leftarrow$ Steer$(q_{\text{nearest}}, q_{\text{rand}}, \Delta q)$
    if ValidEdge$(q_{\text{nearest}}, q_{\text{new}}, X)$ **then**
        Add q_{new} to T_{init} with q_{nearest} as its parent
    end if
    Swap T_{init} and T_{goal}
end while
$P \leftarrow$ ExtractPath$(T_{\text{init}}, T_{\text{goal}})$
return P

After each RRT call, path length and solve time are recorded, and the paths are post-processed. The path is then interpolated over time with a fixed dt. For the purposes of this report, 0.03 seconds was the assigned dt.

B. Sequential RRT Planner

The Sequential RRT planner is the simplest way to integrate multiple robots or agents in a single planning solve. This planner adopts a series based solving approach for each planner per robot. The planner provides all 'priority' to the first robot by solving it in a configuration space excluding all other robots. It employs the RRT algorithm to find an optimal path from the robot's starting point to its destination. This involves randomly exploring the space, growing a tree-like structure from the start towards the goal, and iteratively refining the path to optimize it.

Once an optimal path is determined for the initial robot, the next crucial step is path interpolation. Here, the planner increases the resolution of the path by adding intermediate way points. Before post processing the final path of the first robot, a linear interpolation is constructed between the waypoints of the first robots path [1]. This granulation of the path ensures a more detailed and smooth trajectory, essential for precise path planning of the sequential robots in the planner order.

For the next robot, the planner utilizes this interpolated path of the initial robot as a temporary conflict to avoid collision. During the random state generation phase, this new path is fed as a vector of states and is added as a conflict. Secondly, during the extend phase, the extended path of the second robot is checked against the interpolated plan of the first robot. The planner checks for conflicts against this existing path. By considering the already planned path of the initial robot, the planner effectively guides subsequent robots away from the trajectories of previously occupied robots. This ensures their path is conflict-free. The psuedocode for this planner is shown below in alg 2.

The sequential planner falls short in 'tight' areas where only one robot can fit at time. The sequential planner gives priority to the first robot and excludes those points for the following robots. A time based conflict interpolated vector was created for the first robot. This tweak looked at the distance of the current state of the second robot to all interpolated states of the first robot. Any states outside the time range of the second robot distance was eliminated from the search of collision to first robot. This ensured a state was not blocked for the whole duration of the plan. This simplified time assumption allowed for robot paths to cross while ensuring the robot's bodies do not collide [6]. Sequential approach allows for coordinated planning among multiple robots, making it a robust solution for complex multi-robot systems. However, this approach still falls short in terms of path quality and in scenarios where two robot's goal position are very close. in this scenario, the path of the second robot could be cut off as the planner is trying to around tight spaces.

C. Joint Space RRT Planner

The Joint Space RRT Planner represents a pivotal approach in the realm of multi-robot coordination, specifically addressing the intricacies of planning in a shared environment. Unlike the sequential RRT planner, which handles robots individually in a priority order, the Joint Space RRT planner operates by considering the combined configuration space of both robots simultaneously. This approach begins by generating two random configurations, conceptualized as a single 'point' in the joint space of the robots. This point essentially represents a combined state where each compo-

Algorithm 2 Kinodynamic Sequential RRT-Connect Planner

$Q, n_{root}, p_1, p_2 \leftarrow \emptyset$
$p_1 \leftarrow$ RRTCONNECT
$FinePath_1 \leftarrow$ INTERPOLATEPATH(p_1)
$CollisionChecker \leftarrow$ CREATECOLLISIONCHECKER$(FinePath_1)$
$p_2 \leftarrow$ RRTCONNECT$(CollisionChecker, \emptyset)$
$n_{root} \leftarrow (p_1, p_2)$, cost
Q.push(n_{root})
while $\neg(Q$.empty()$), \neg$(goal reached) **do**
    $c_{node} \leftarrow Q$.top()
    $C \leftarrow$ checkCollisions(c_{node}.plans(), d)
    if C.empty() **then**
        return c_{node}.plans()
    else
        Q.pop()
        for each robot i in C_i **do**
            $K \leftarrow$ convertCollisionConstraint(C_i)
            $p_i \leftarrow$ RRTCONNECT$(M_i, O, x_{i,0}, x_{i,f}, K, d)$
            $s \leftarrow (p_i$, other plans$)$, cost
            Q.push(s)
        end for
    end if
end while

nent corresponds to the position and velocities of each of the robots. By solving the path planning problem in the joint space, the planner effectively coordinates the movements of both robots in a way that is not only efficient but also inherently mindful of the other's presence [5]. This leads to a more integrated and harmonious movement strategy, crucial for scenarios where robot interaction is constant and highly dynamic.

While proving effective, this approach increases the solve time substantially as a path needs to be found in the joint space; thus increasing the degrees of freedom from 6 to 12 for just 2 robots. As the number of robots grow, the solve time exponentially grows as the degree of freedom for the configuration space goes up by 6 per robot. As a result, hypothetically this planner will have the longest solve time.

To avoid the random configuration of both robots generating two points within conflict range, the random points were biased away from each other. In detail, the first robots configuration was first randomly found and the second robots state points were biased away from the first robot to ensure there was no internal collision when solving for the plan in the joint space. Additionally, the new struct incorporated both robots when extending to the final position and finding if there is a collision. To ensure kinodynamic extension, each robot was individually extended and if there was a collision with an object the final extended path is fed as the new state point to be created [7]. Hence, the joint space closely followed the RRT-Connect structure while adding extra paremeters for the additional robot states. The psuedo code for the implementation of this algorithm is given below and shown in alg 3.

Algorithm 3 Kinodynamic Jointspace RRT-Conmect

Input: Initial configurations $q_{\text{init1}}, q_{\text{init2}}$, goal configurations $q_{\text{goal1}}, q_{\text{goal2}}$, search space X, step size Δq
Output: Joint Path P from $(q_{\text{init1}}, q_{\text{init2}})$ to $(q_{\text{goal1}}, q_{\text{goal2}})$ (or failure)
Initialize trees T_{init1} with node q_{init1} and T_{init2} with node q_{init2}
Initialize trees T_{goal1} with node q_{goal1} and T_{goal2} with node q_{goal2}
while $\neg$ReachedGoal$(T_{\text{init1}}, T_{\text{goal1}}, T_{\text{init2}}, T_{\text{goal2}})$ **do**
    $q_{\text{rand1}} \leftarrow$ RandomConfig(X)
    $q_{\text{rand2}} \leftarrow$ BiasedRandomConfig(q_{rand1}, X)
    $q_{\text{nearest1}} \leftarrow$ NearestNode$(T_{\text{init1}}, q_{\text{rand1}})$
    $q_{\text{nearest2}} \leftarrow$ NearestNode$(T_{\text{init2}}, q_{\text{rand2}})$
    $q_{\text{new1}} \leftarrow$ Steer$(q_{\text{nearest1}}, q_{\text{rand1}}, \Delta q)$
    $q_{\text{new2}} \leftarrow$ Steer$(q_{\text{nearest2}}, q_{\text{rand2}}, \Delta q)$
    if ValidEdge$(q_{\text{nearest1}}, q_{\text{new1}}, X)$ **then**
        Add q_{new1} to T_{init1} with q_{nearest1} as its parent
    end if
    if ValidEdge$(q_{\text{nearest2}}, q_{\text{new2}}, X)$ **then**
        Add q_{new2} to T_{init2} with q_{nearest2} as its parent
    end if
    Swap T_{init1} and T_{goal1}
    Swap T_{init2} and T_{goal2}
end while
$P \leftarrow$ ExtractJointPath$(T_{\text{init1}}, T_{\text{goal1}}, T_{\text{init2}}, T_{\text{goal2}})$
return P

D. Conflict Based Planning

Conflict-Based Search (CBS) is a popular distributed solution to multi-agent path finding problem. The two-layered algorithm operates on the space of potential robot plans, solving for each robot independently, while using a high level planner to resolve conflicts between potential plans. The algorithm generates collision constraints for path conflicts between agents, rather than aggregating the agents into a single joint entity. In its high level search, the CBS grows a constraint tree, where each node contains a suggested robot path, a vector of inherited collision constraints, and an associated cost. In our implementation, these suggested robot paths are not necessarily collision free, and the conflict-based search relies on its low level sampling based planner, in our case RRT-Connect, to quickly re-plan with consideration to its constraints. Furthermore, the cost function of this high level search is composed of the aggregated path length of each robot in the search.

Our implementation of a Kino-Dynamic Conflict Based Search begins by initializing a priority queue Q, an empty constraint tree T and a root node n_{root}. The algorithm then sends a ros-service call to Quad-SDK's Global Body Planner with a robot model M_i, terrain map O, start and goal positions $x_{i,0}, x_{f,0}$, and an empty set of constraints. The returned trajectories from the RRT-Connected are stored within the root node of the tree, which is then pushed into the queue.

Algorithm 4 Kino-Dynamic Conflict Based Search

$Q, n_{root}, p_0 \leftarrow \emptyset$
for each robot i **do**
  $p_0 \leftarrow p_0 \cup RRTConnect(M_i, O, x_{i,0}, x_{f,0}, \emptyset, \emptyset)$
end for
$n_{root} \leftarrow p_0$, cost
Q.push(n_{root})
while $\neg(Q)$.empty(), $\neg$(goal reached) **do**
  $c_{node} \leftarrow Q.top()$
  $C =$ checkCollisions(c_{node}.plans(), d)
  if C.empty() **then**
    return c_{node}.plans()
  else
    Q.pop()
    for each robot i in C_i **do**
      $K \leftarrow$ convertCollisionConstraint(C_i)
      $p_i \leftarrow RRTConnect(M_i, O, x_{i,0}, x_{f,0}, K_i, d)$
      $s \leftarrow p_i$, cost
      Q.push(s)
    end for
  end if
end while

At every iteration of the CBS, we pop the node with the smallest associated cost. Given the discretized path from each robot, we check every pair of robots for conflicts, by checking their relative COM positions, with a predefined safety threshold d. If the paths return conflict free, each corresponding plan is published and sent to its respective local planner for foothold resolution. [5].

Path conflicts are defined in a method similar to [3], in which each collision is described as $C = \langle i, j, [t_s, t_f] \rangle$ where, i, j, correspond to the agents involved in the collision, and t_s , t_f corresponding to the start and end times of the collision. In our case, since the algorithm operates on discretized body trajectories, these also happen to correspond to plan indices. A simplifying assumption was made that robots that have reached their goal positions remain in stance phase until the completion of all plans.

After generating a vector of robot conflicts, we arbitrarily select a conflict C_i, and for each robot involved generate a corresponding successor s. Each constraint is represented as a collection of static obstacles, of size d, and the function convertCollisionConstraint takes the collision C_i and converts it into vector K, composed of the body positions of the other robot. Essentially, robot 1 must avoid the set of points correlated to the relative position of robot 2 at the time of collision. This vector of collisions is passed to the RRT Connect which in addition to its kino-dynamic checks, also rejects any potential sample or connect operation that leads to a state overlapping with our collision constraints. Both of these successors are pushed into the queue and the process repeats until a collision free node is found.

As currently implemented, the algorithm cannot guarantee completeness, given the current representation of the collision constraints, as well as our handling of scenarios where robot behavior appears to be coupled (e.g. a tight corridor, where one robot is required to wait for the other to pass). Representing collision constraints as static obstacles may over constrain the space of viable robot positions in the map, preventing us from finding a solution, even if one exists. As such a great deal of tuning is needed to minimize these effects.

IV. Results

To test the three planner methods, a quad-SDK simulation was used where two robots were dropped at a set start point. Random goal points were fed into both robots and all three planners were run on the same set of start and goal points. The joint and sequential were run a total of 16 times while the CBS planner was only run 5 times. Of the 5 times the CBS planner was run, the same start and goal points were fed into the all three planners to keep consistency. The average performance of the three planners are shown in Table I. Each planners path observation is discussed in more detail.

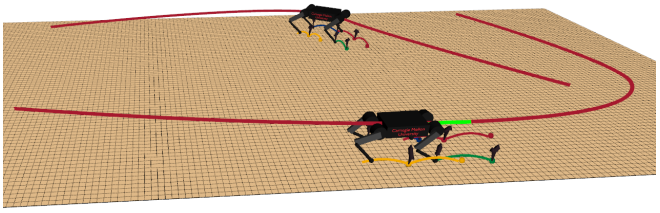

Fig. 2. Sequential Planner trajectory planning results

Planner	Avg. Path Length	Avg. Planning Time	Success
Sequential	18.2	0.094	11/16
Joint	16.85	1.067	13/16
CBS	15.12	0.254	5/5

TABLE I
Simulation Results on Flat Terrain

The Sequential RRT planner demonstrated its quick ability to arrive at a solution. However, in couple scenario the robot was not able to find a path for the second. This scenario largely rose when trying to fit the second robot in a corner position while it is blocked by the first robot. Since the planner is solved sequentially, the first robot interpolated path blocks the second robot. The CBS planner was able to send the second robot to the goal position before the first robot reached the goal position. Figure 2 shows the sequentially RRT shining where there is a lot of space for the second robot to travel around the trajectory of the first robot as highlighted by the red planning lines. Figure 3 shows where the sequential RRT takes a submoptimal path to the goal. This is possibly due to the fact that the robot cannot directly cross first robots path as the time distance from robot ones position is the same as robot two to reach that point. Hence,

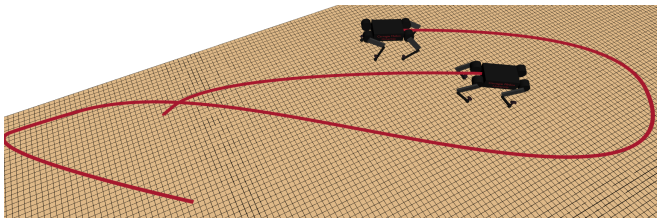

Fig. 3. Sequential Planner Suboptimal Trajectory

it is blocked for the second robot initially and the planner connects a more suboptimal path.

The Joint Planner overall performs much better than than the sequential planner due to the solving of both robot states. In this case, trajectory overlapping is not an issue. However there were instances this planner failed. Although the random points were biased away from each other, there were times where the robots collided when traveling from point to another as there was no real time internal collision checking between other robots. Additionally, the solve time for the Joint planner was substantially higher than the other two methods. Proving that joint solving was infeasible for more than two robots. Figure 4 below shows the joint planner in action.

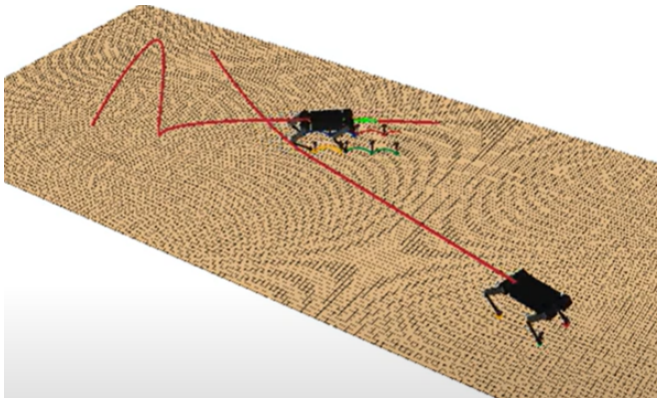

Fig. 4. Joint Planner Trajectory

The Conflict Based Search demonstrated its ability to chart body trajectories in a series of trials with randomized start and goal locations. It retains a relatively short average planning time of 0.254 s, demonstrating its ability to quickly determine collision free paths. The method was only tested up until three robots, so determining its scalability is left to future work.

The Conflict Based Search experienced difficulties when robot start and goal states are in close proximity, a consequence of the thresholding used to identify collisions. For instance, if two robots start positions are less than threshold d from one another, the planner may interpret that as a collision and invalidate the start state of both robots, causing the planner to fail.

Additionally, the importance of collision threshold tuning was found to be quite important, as sometimes the robot may opt for a flight phase over another robot as opposed to taking less direct route to goal.

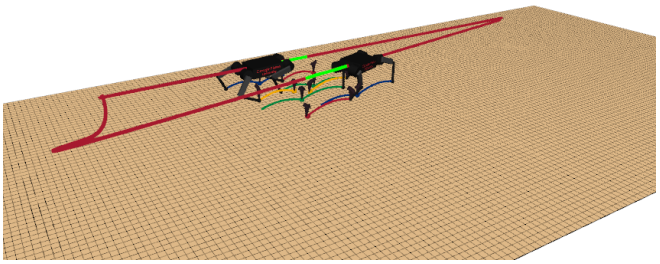

Fig. 5. Conflict Based Search successfully solving head to head planning problem, where robots exchange start states

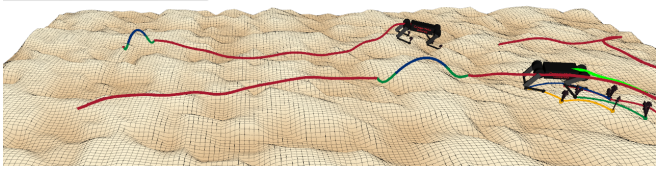

Fig. 6. CBS trajectory planning on terrain with features $\approx 25cm$, Walking Phases are plotted in Red, Flight Phases are shown in blue

Furthermore, the arbitrary conflict selection method greatly impacted the success of the conflict based search, because continuously selecting a given index of the conflict list may result in behavior similar to the sequential RRT, where one robot becomes heavily constrained and the other goes directly to goal.

V. Conclusion

In this paper we have evaluated the performance of three distinct planners: the Sequential Planner, the Joint Space Planner, and the Kino-Dynamic Conflict-Based Search (CBS). The choice of planner must be informed by the specific requirements of the environment and the task at hand.

Our first approach, The Sequential Planner, solves the path finding problem in a linear, step-by-step manner, addressing each agent's path independently. This approach, while simpler and potentially more efficient in less crowded environments, often falls short in complex scenarios where interactions between agents significantly influence the overall system's behavior. Its primary limitation arises in scenarios where the sequential nature leads to sub-optimal paths for subsequent robots due to the constraints imposed by earlier decisions. On the other hand, the Joint Space Planner takes a more holistic approach, considering all agents simultaneously as a single entity. This methodology is particularly advantageous in tightly coupled scenarios, where the actions of one agent directly impact the others. However, this approach can become computationally intensive as the number of agents increases, making it less feasible for larger systems. Incorporating dynamic collision checking with other robots as points are sampled will allow for even better performance of the Joint RRT Planner.

While the Sequential and Joint Space Planners have their merits in certain contexts, our Kino-Dynamic CBS, with future enhancements, has the potential to offer a more

versatile and efficient solution for a wide range of multi-robot coordination scenarios. With regards to the low-level RRT, biasing the sampling of point in the RRT away from the applied constraints may improve both the path quality and the computation time. Additionally, simplifying assumptions, i.e treating collision constraints as static obstacles that exist for the entirety of the path duration, could be improved by representing the constraint instead as a dynamic obstacle over a given time period which may afford the afford the planner more leeway in ascertaining a viable plan, and improve its performance on smaller maps. In situations where robot motion appears to be coupled, solving the joint problem for two robots as described in [3] may be optimal, and remains future work. Finally, incorporating time into the search would allow for more fine tuned collision checking, but would also greatly complicate the search especially since velocities are a part of our state space.

VI. Code Repository

All of the code for this project can be found at the repository listed below. Each planner is implemented its respective branch, namely, sequential_rrt_planner, joint_rrt_planner, and devel_cbs.

References

[1] B. Cain, M. Kalaitzakis, and N. Vitzilaios, "Mk-rrt*: Multi-robot kinodynamic rrt trajectory planning," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2021, pp. 868–876.

[2] M. Čáp, P. Novák, J. Vokřínek, and M. Pěchouček, "Multi-agent rrt*: Sampling-based cooperative pathfinding (extended abstract)," 2013.

[3] J. Kottinger, S. Almagor, and M. Lahijanian, "Conflict-based search for multi-robot motion planning with kinodynamic constraints," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2022. [Online]. Available: http://dx.doi.org/10.1109/IROS47612.2022.9982018

[4] A. Tajbakhsh, L. T. Biegler, and A. M. Johnson, "Conflict-based model predictive control for scalable multi-robot motion planning," 2023.

[5] J. Norby, Y. Yang, A. Tajbakhsh *et al.*, "Quad-SDK: Full stack software framework for agile quadrupedal locomotion," in *ICRA Workshop on Legged Robots*, May 2022, workshop abstract.

[6] J. Norby and A. M. Johnson, "Fast global motion planning for dynamic legged robots," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 3829–3836.

[7] M. Čáp, P. Novák, J. Vokřínek, and A. Komenda, "Prioritized planning algorithms for trajectory coordination of multiple mobile robots," *IEEE Transactions on Automation Science and Engineering*, vol. 10, no. 3, pp. 593–607, 2013.