Motion Planning for Bipedal Robots

Aseem Saxena
Oregon State University

Mohitvishnu Gadde
Oregon State University

Ashish Malik
Oregon State University

***Abstract*—Using Sim2Real reinforcement learning, [1] has been able to demonstrate robust bipedal gaits such as standing, walking, hopping, running etc on bipedal robot Cassie. However, real world applications require robot autonomy above the level of heading velocity and direction to achieve meaningful goals. Motion and path planning using learned behaviors is still an open area of research in Robotics due to the computational requirements of planning algorithms and rapid updates required for real world applications. We tackle this problem for the Cassie Robot on simulated terrains using the RRT* algorithm to rapidly and reliable generate trajectory waypoints to reach the desired goals. These waypoints and Cassie's real-time pose are used by a controller to set the policy targets which performs the lower level locomotion behavior. We successfully demonstrate our system in simulation in a number of challenging scenarios.**

I. Introduction

Motion planning algorithms for robots have received widespread attention from the robotics community over the last couple of decades as robots gradually enter daily human lives. Modern robots differ tremendously in their morphology, sensor modalities, sizes, etc., but one thing which is common to all is to solve the problem of navigation or to operate in a complex environment. Broadly, the problem of motion planning is concerned about producing a sequence of targets such as control inputs, waypoints, sub-behaviors, etc for the robot to reach a goal configuration from a starting configuration while respecting the constraints put forward by the environment, or robot's morphology and configuration.

Previous research, on the bipedal robot - Cassie has produced various locomotion policies that can execute desired forward velocity and turn rate commands on the robot, both in simulation and in the real world [1]. However, these policies require a continuous input of velocity and turn rate commands to perform desired locomotion behaviors. In this work, we develop an autonomous system to generate these commands for the policy so as to meet a desired end-goal through locomotion (reach the goal/target position). The complexity of this work lies in two main domains: Computational complexity of the planning algorithms and conversion of the planner's output to low level locomotion policy's target. We mitigate the first challenge by using the RRT* motion planning algorithms to rapidly generate collision free waypoints. For the second challenge, we design a controller that takes in the waypoint and Cassie's current pose information to generate the policy targets. The controller autonomously decides to use a P controller or a fixed turn rate command to generate the policy targets based on the input information.

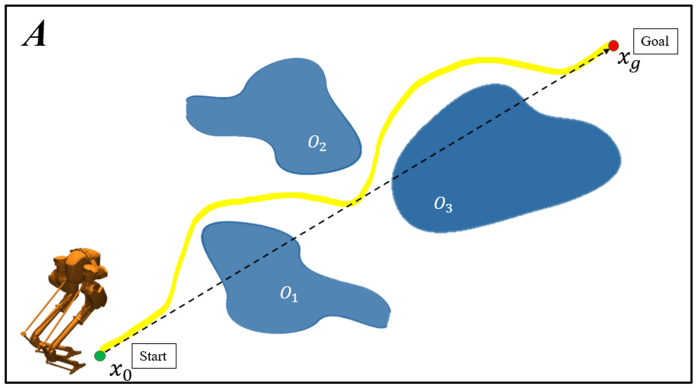

Fig. 1. An example unknown environment $\{A\}$ highlighting the obstacles, start position, goal position and the Cassie robot. The regions $O_1, O_2, ...O_i, ...O_j$ are the highlighted obstacles. The dotted line denotes the shortest path from X_0 (start point) to X_g (goal point). The yellow path depicts a feasible path for Cassie to follow to avoid the obstacles to reach the goal position.

II. Related Works

Research on motion planning began by formulating complete planners for polygonal robots moving in an environment with polygonal obstacles [2]. These algorithms suffered from high computational complexity.

Work on cell decomposition methods [3], potential fields [4] and roadmaps [5] heralded the onset of motion planning algorithms which could be applied practically. But, these algorithms suffered from intractability issues with high dimensional state spaces.

The most recent successful planners in the last decade are sampling based algorithms [6] [7]. Sampling based algorithms build a roadmap by randomly sampling and connecting points from an obstacle-free space in order to reason about the connectivity of the starting and goal configurations. This leads to computational savings since obstacles need not be constructed in state space. Instead of completeness guarantees, these algorithms offer probabilistic completeness guarantees, in other words, as the number of samples reaches infinity, the probability that the algorithm does not return a solution, when, in fact, a solution exists, tends to 0.

In the class of sampling-based algorithms, two main subtypes exist. Probabilistic Roadmaps (PRM) [7] and Rapidly-exploring Random Trees [6].

PRMs [7] construct the roadmap graph first (containing most of the collision free trajectories). Once the roadmap is constructed, a path can be found by computing the shortest

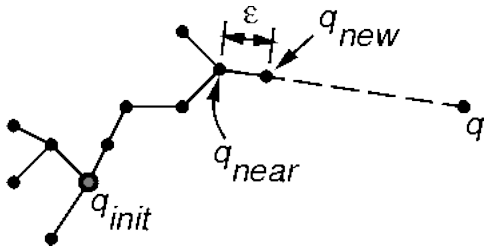

Fig. 2. Pictorial representation of the expand procedure in RRT* [14]

path in the graph from the start to goal configuration. One issue with PRMs is that the roadmap construction step could be computationally prohibitive.

To address concerns with PRMs, incremental sampling-based planning algorithms [6] such as RRT [8] have emerged as an online alternative. Due to their incremental nature, they can terminate as soon as a solution is foound as opposed to PRMs. Slight changes in the environment do not warrant planning from scratch since most trajectories are still collision free.

RRT has nice theoretical guarantees and can work reliably with discrete/continuous systems having non holonomic constraints, non linear dynamics and differential constraints.

The general formulation of the motion planning problem doesn't take into account the 'distance' or 'cost' of a solution. Various heuristics have been proposed to modify RRT in order to get 'better' solutions.

In [9], The authors prove that RRT almost always returns a sub-optimal solution and RRT* is asymptotically optimal. The authors also show that RRT* has the same computational complexity as RRT.

III. RRT* Algorithm

We will define the sub-procedures used by RRT* before we describe the algorithm.

- *Sampling:* The function $Sample$ returns independent and identically distributed (i.i.d) samples from X_{free} (points lying outside the obstacle space).
- *Steering:* given two points x and y, the function $Steer$ returns z which is closer to y than x. The distance metric is usually a Euclidean Norm.
- *Nearest Neighbor:* For a given graph $G = (V, E)$ and x both part of X_{free}, the function $Nearest$ returns the vertex V in G which is closest to x.
- *Near Vertices:* Similar to $Nearest$, $Near$ returns the n closest vertices in graph G closest to x.
- *Collision Test:* given two points x and x' in X_{free}, the function $ObstacleFree$ returns true if and only if the line segment connecting x and x' lies in X_{free}.
- *Parent:* given a vertex v, the function $Parent$ returns its unique parent v' in the tree.

RRT* starts from a graph with only the starting position vertex and no edges. Then it incrementally grows a graph in X_{free} by sampling a point x_{rand} and extending the graph towards it. The way it does this is by connecting a new vertex to the point which incurs minimum cost in the set X_{near}. It then connects the new vertex to all the vertices in X_{near} to adjust the tree. The algorithm of RRT* is put down below from the original paper [9] for comprehensive understanding.

Algorithm 1: Body of RRT Algorithm

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$; $i \leftarrow 0$;
while $i < N$ **do**
  $G \leftarrow (V, E)$;
  $x_{\text{rand}} \leftarrow \texttt{Sample}(i)$; $i \leftarrow i + 1$;
  $(V, E) \leftarrow \texttt{Extend}(G, x_{\text{rand}})$;

Algorithm 2: $\texttt{Extend\_}RRT^*$

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\min} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow (G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\min}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and*
     $\texttt{Cost}(x_{\text{near}}) >$
     $\texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$

IV. Locomotion for Cassie

The work [1] presents a principled way to design reward functions that implicitly capture all periodic bipedal locomotion gaits. The work is motivated by the observation that all periodic gaits can by defined by swing (foot in the air) and stance (foot on the ground) phases, and relative ratio of the stance and the swing phases. The phases differ in the presence/absence of foot forces and velocities.

The reward function enables the robot to learn to raise their feet by penalizing forces during swing phase and penalizing velocities during stance phase. Through command randomization during the learning phase, this allows the agent to learn different locomotion behaviors. These behaviors can be composed appropriately to quickly and flexibly adapt to changing environment conditions.

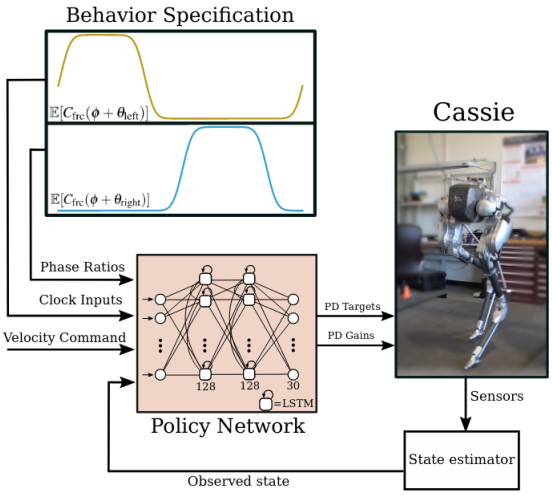

Fig. 3. System Diagram of [1]

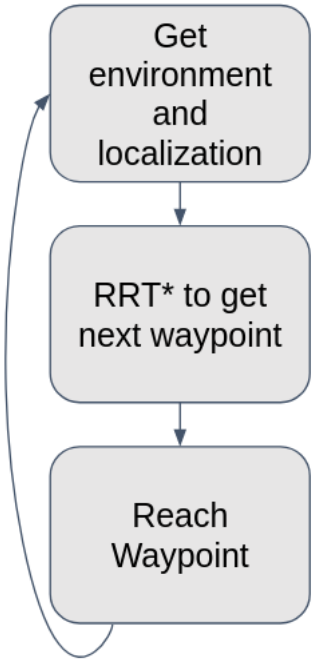

Fig. 4. System Diagram of our approach

The locomotion policies are parameterized using LSTM [10] neural networks that offer an internal memory. This internal memory serves to improve sim2Real performance, and smooth switching of the locomotion behaviours.

V. Methodology

We do all our experiments in simulation on the MuJoCo simulation environment [11]. Our code runs on a laptop with a Core i7-8th Gen processor, 16GB RAM and the NVIDIA GeForce MX150 GPU (2GB VRAM). Our implementation was tested on Ubuntu 20.04 OS.

Our environments are procedurally generated by randomly sampling different cuboids of varying sizes at randomized locations and randomly spawning Cassie and the goal location. We ensure that we spawn Cassie and the goal in the free region to avoid impossible scenarios.

We break down the motion planning problem into two parts. Firstly, we use the RRT* algorithm to generate the next waypoint to reach in the free space. We design a simple yet effective Waypoint Controller which enables the robot to reach the next waypoint.

The waypoint to policy's target controller uses Cassie's current pose and next waypoint location as input. It first matches Cassie'e heading direction towards the next waypoint (geodesic line between the current location and next waypoint's location). This is important as the RRT* ensures that the shortest path between the current robot position and the next waypoint is obstacle free. However, no such guarantees can be made for the other possible paths with the same starting and ending positions. Therefore, we wish to follow the shortest path as close as possible. When the error in the orientation is less than a fixed threshold, the controller commands a fixed forward velocity and an turn rate that is proportional to the error between the desired orientation and current orientation. This proportional component to control heading direction is essential as the policy, which being incredibly stable isn't absolutely reliable. So the second component ensures that the errors do not accrue into large deviations in the actual vs desired path to the waypoint. Once the robot successfully reaches the waypoint, RRT* is invoked again for retrieve the next waypoint towards the goal.

Proportional control [12] is a linear feedback control system wherein a correction is applied to the control input at all times. The correction is proportional to the difference of the desired input and the actual input. A drawback of P Control is that it cannot completely remove the error between the desired and actual input. P Control is mathematically expressed as

$$P_{out}(t) = K_p * (SP - PV)(t) + p0(t) \tag{1}$$

where p_0 refers to the controller input without any error, SP (desired input) - PV (actual input) being the instanteneous error at time t. K_p being a proportional term which is tuned manually to get good convergence. P_{out} is the output of the Proportional Controller. In our case, we have two P controllers, one for orientation and one for position.

VI. Results

We record qualitative demonstrations of Cassie successfully reaching the desired location Link1 Link2. We notice that cassie is successfully able to navigate narrow gaps in order to reach the goal which is interesting.

We wish to improve on certain aspects of our framework. The default velocity set for cassie to move is set conservatively, we noticed that our P controller cannot converge for higher velocities, this necessitates the need to design more sophisticated controllers like PID controllers [15] in the future.

The underlying python implementation of RRT* could be improved to run faster by leveraging Parallel computing.

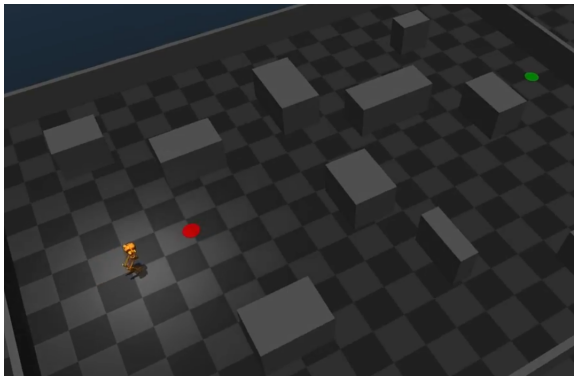

Fig. 5. Screenshot of cassie in the middle of reaching a waypoint(shown in red), Final goal shown in green and obstacles are shown in grey.

Another way to improve its runtime is to implement it in C/C++

The environments we created using simple 3d shapes are sufficient to demonstrate the effectiveness of our approach but do not correspond to how real world environments look like. [13]

VII. Conclusion and Future Work

This section describes the contributions, conclusions and future work for the project.

- In this work we demonstrated goal reaching abilities in randomly generated maps of the robot cassie in simulation
- We showed how traditional motion planning algorithms like RRT* produce reliable and reachable waypoints for cassie to target.
- We demonstrated that a simple P controller is sufficient in order to realize the desired motion albiet it adds additional constraints on how the motion is performed.

Directions for future work include removing our reliance on global information (known location and size of obstacles) and perfect localization. We wish to mount and use a RGBD camera on Cassie and decide what velocities and turn rates to reach at each time step without making any collisions and with the aim of reaching the goal in a sufficient amount of time. Rendering and using more realistic simulation environments [13] is also a viable and important future step in order to ensure that our policies transfer well to the real world.

VIII. Team contributions

- Mohit: Research and implementation of the RRT* algorithm, generating randomized obstacle maps in simulation and extracting obstacle information for the RRT* algorithm.
- Ashish: Design and testing of the controller in simulation. Interfacing the RRT algorithm, simulation and the controller to produce simulation demonstrations.
- Aseem: Setting up the simulation framework, Cassie's locomotion policy and visualization components of the demonstrations.

Acknowledgment

We would thank Helei Duan, Jeremy Dao and Bikram Pandit for their insightful comments and ideas.

References

[1] Jonah Siekmann, Yesh Godse, Alan Fern, Jonathan Hurst, "Sim-to-Real Learning of All Common Bipedal Gaits via Periodic Reward Composition" 2021 IEEE International Conference on Robotics and Automation (ICRA), 7309-7315.

[2] T. Lozano-Perez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. Communications of the ACM, 22(10):560–570, 1979.

[3] R. Brooks and T. Lozano-Perez. A subdivision algorithm in configuration space for findpath with rotation. In International Joint Conference on Artificial Intelligence, 1983.

[4] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. International Journal of Robotics Research, 5(1):90–98, 1986.

[5] Canny. The Complexity of Robot Motion Planning. MIT Press, 1988.

[6] L. Kavraki and J. Latombe. Randomized preprocessing of configuration space for fast path planning. In IEEE International Conference on Robotics and Automation, 1994.

[7] L.E. Kavraki, P. Svestka, J Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation, 12(4):566–580, 1996.

[8] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. International Journal of Robotics Research, 20(5):378–400, May 2001.

[9] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in Proc. Robotics: Science and Systems (RSS), 2010.

[10] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780

[11] Todorov, Emanuel, Erez, Tom, and Tassa, Yuval. Mujoco: A physics engine for model-based control. In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pp. 5026–5033. IEEE, 2012

[12] Proportional Control, Wikipedia Article, Link

[13] Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., Parikh, D., and Batra, D. (2019). Habitat: A Platform for Embodied AI Research. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV).

[14] Steven M. LaValle, Rapidly-Exploring Random Trees, 2001, Link

[15] Proportional Integral Derivative Controller, Wikipedia Article Link