Transfer of Policies Based on Trajectory Libraries

Martin Stolle, Hanns Tappeiner, Joel Chestnutt, Christopher G. Atkeson
Robotics Institute, Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213
http://www.cs.cmu.edu/~mstoll
{mstoll,hanns,chestnutt,cga}@cs.cmu.edu,

***Abstract*— Libraries of trajectories are a promising way of creating policies for difficult problems. However, often it is not desirable or even possible to create a new library for every task. We present a method for transferring libraries across tasks, which allows us to build libraries by learning from demonstration on one task and apply them to similar tasks. Representing the libraries in a feature-based space is key to supporting transfer. We also search through the library to ensure a complete path to the goal is possible. Results are shown for the Little Dog task. Little Dog is a quadruped robot that has to walk across rough terrain at reasonably fast speeds.**

I. Introduction

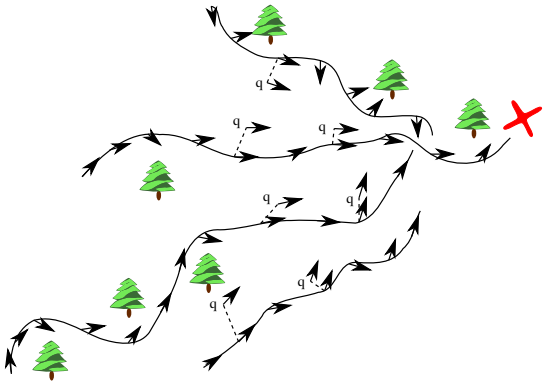


Fig. 1. Illustration of a trajectory library. When queried at any point (e.g. 'q'), the action (indicated by arrows) of the closest state on any trajectory is returned

A trajectory library [1] is a collection of state sequences annotated with actions. When controlling a system using a trajectory library as a policy, the current state is used to find the closest state on any trajectory. The action associated with the nearest state is used as the output of the policy (figure 1). We find trajectory libraries to be a useful tool in solving difficult control problems.

In many cases it is desirable to reuse an existing library of trajectories to solve new problems. This is especially true when the library is created manually, since there is no planner to fall back to. In other cases, it is desirable to augment a planner by adding special behaviors to a library that allow the agent to handle particularly tricky parts for which the planner alone cannot find satisfactory solutions. We would like to reuse these behaviors on new problems.

This material is based upon work supported in part by the DARPA Learning Locomotion Program and the National Science Foundation under grants CNS-0224419, DGE-0333420, ECS-0325383 and EEC-0540865.

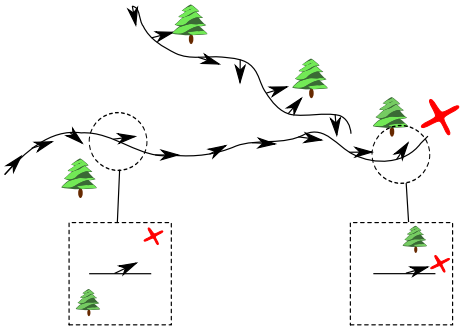


Fig. 2. Illustration of feature space. On the circled states as examples, we show how state-action pairs could be specified in terms of local features such as the relative positions of obstacles and the goal.

In this paper, we present a transfer algorithm. The algorithm has two key ideas. One key idea is to represent the library in a feature-based space. When using a feature-based representation, instead of representing the state of the system using its default global representation, we use properties that describe the state of the system relative to local properties of the environment. For example in a navigational task, instead of using global Cartesian position and velocity of the system, we would use local properties such as location of obstacles relative to the system's position (figure 2). This allows us to reuse parts of the library in a new solution if the new problem contains states with similar features. The transfer to a new environment is done by looking for states in the new environment whose local features are similar to the local features of some state-action pair in the source environment. If a match is found, the state-action pair is added into the transferred library at the state where the match was found.

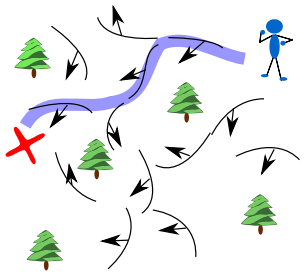


Fig. 3. Illustration of search through a trajectory library. For the given start state, we find a sequence of trajectory segments that lead to the goal

The other key idea is to ensure that the library produces goal-directed behavior by searching through the library.

Trajectory libraries are often used in a greedy manner to pick an action based on the state of the system. If the library was transferred to a new problem, there is no guarantee that greedily picking actions will get to the goal. Parts of the relevant state space might not even map to an appropriate goal-directed action. This is especially true if the feature-based representation used for transfer does not take into account progress towards the goal. Even if individual state-action mappings are goal-directed, it is still possible that following such a greedy policy gets stuck or loops. We search through the library to ensure that following the library will lead to reaching the goal (figure 3).

II. Related Work

Transfer of knowledge across tasks is an important and recurring aspect of artificial intelligence. Previous work can be classified according to the type of description of the agent's environment as well as the variety of environments the knowledge can be transferred across. For symbolic planners and problem solvers, high level relational descriptions of the environment allow for transfer of plans or macro operators across very different tasks, as long as it is still within the same domain. Work on transfer of knowledge in such domains includes STRIPS [2], SOAR [3], Maclearn [4] and analogical reasoning with PRODIGY [5]. More recent relevant work in discrete planning can be found in [6], [7].

In controls, research has been performed on modeling actions using local state representations [8], [9]. Other work has been done to optimize low-level controllers, such as walking gaits, which can then be used in different tasks [10]–[13]. In contrast, our work focuses on finding policies which take into account features of the specific task. Some research has been performed to automatically create macro-actions in reinforcement learning [14]–[17], however those macro actions could only transfer knowledge between tasks where only the goal was moved. If the environment was changed, the learned macro actions would no longer apply as they are expressed in global coordinates, a problem we are explicitly addressing using feature-based representations. Another method for reusing macro actions in different states using homomorphisms can be found in [18].

Bentivegna et al. [19] explore learning from observation using local features, and learning from practice using global state on the marble maze task. Our approach to learning from demonstration takes a more deliberate approach, since we perform a search after representing the learned knowledge in a local feature space.

Our approach is also related to the transfer of policies using a generalized policy iteration dynamic programming procedure in [20]. However, since trajectory libraries are explicitly represented as state-action pairs, it is much simpler to express them in a feature space.

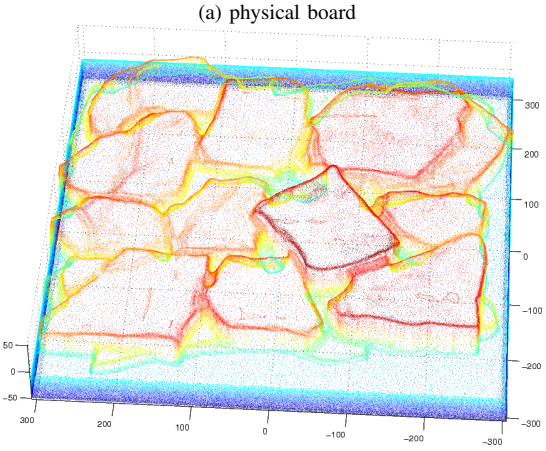
For an overview of uses of trajectory libraries in control and artificial intelligence, see [1].

III. Case Study: Little Dog

The domain to which we applied the algorithm is the Little Dog domain. Little Dog is a quadruped robot developed by Boston Dynamics (figure 4). It has four legs, each with three actuated degrees of freedom. Two degrees of freedom

Fig. 4. Little Dog robot

(a) physical board



(b) computer model

Fig. 5. Sample terrain board

are at the hip (inward–outward, forward–backward) and one at the knee (forward–backward). Torque can be applied to each of the joints. This results in a 12 dimensional action space (three for each of the four legs). The state space is 36 dimensional (24 dimensions for the position and speed of the leg joints and 12 dimensions for the position, orientation, linear velocity and angular velocity of the center of mass). The task to be solved in this domain is to navigate through rough terrain (figure 5).

The robot is controlled by sending desired joint angles to an on-board PD controller. The desired joint angles can be updated at 100Hz. The on-board PD controller computes new torque outputs at 500Hz. The robot is localized using a Vicon motion capture system which uses retro-reflective markers on the robot in conjunction with a set of six infrared cameras. Additional markers are located on the terrain boards. The proprietary Vicon software provides millimeter accuracy

location of the robot as well as the terrain boards. We are supplied with accurate 3d laser scans of the terrain boards. As a result, no robot sensor is needed to sense the obstacles.

For difficult terrains, we use a joystick together with inverse kinematics to manually drive the robot across the terrain and place feet. Sequences of joint angles together with body position and orientation are recorded and annotated with the stance configuration of the robot. The stance configuration describes which legs are on the ground and which leg is in flight. Several trajectories are recorded with the robot traversing multiple terrain boards in different directions. These trajectories are automatically segmented into individual footsteps according to the stance configuration. Our algorithm then takes this library of footsteps and applies it to new terrains.

IV. How the Library Works without Transfer

This section explains how the library works using global or absolute state space coordinates. This is helpful for understanding the implementation of the transfer algorithm, which is explained in the next section.

Action Selection: When learning from demonstration with Little Dog, the library contains sequences of steps that were manually taught using a joystick remote control. A new step is started whenever all four feet are on the ground. The global position and orientation of the robot together with the joint configuration forms a state on a trajectory. The sequence of joint angles through lift off until all four feet are back on the ground is the action associated with the state. By using a trajectory library to pick which step to take, the robot can succeed in traversing a terrain even if it slips or if it is just randomly put down near the start of any step. The global state-based lookup into the trajectory library works as follows: When started or after completing a step, the robot takes its current position, orientation and joint configuration and computes the position of the four feet in terrain coordinates. Then, for every step in the library, the sum of the Euclidean distances between the current position of the feet and their respective position at the beginning of the step is computed. The step with the minimum sum of the Euclidean distances is used as the next step to take.

Action Execution: In order to ensure smooth behavior, before the sequence of joint configurations of the chosen step can be played back, every joint is moved along a cubic spline from its current angle to its angle at the start of the chosen step. The time taken depends on how far the body is from it's intended position. The speed is chosen conservatively to avoid slipping. Furthermore, if the joint angles are just played back as they were recorded, errors in the position and orientation of the robot would accumulate. In order to solve this problem, an integral controller is added during playback to correct for errors in body position and orientation. This controller works as follows:

Every foot f has a three dimensional vector integrator $\boldsymbol{I}_f$ associated with it. Instead of playing back the recorded joint angles $\boldsymbol{j}_f$, we use the joint angles to compute the body-relative three-dimensional position of every foot: $\boldsymbol{x}_f = FK_f(\boldsymbol{j}_f)$. The integrated vector is added to the body-relative position of the foot to compute a new body-relative position of the foot: $\boldsymbol{x}'_f = \boldsymbol{x}_f + \boldsymbol{I}_f$. Inverse kinematics is used to compute appropriate joint angles for this new position: $\boldsymbol{j}'_f = IK_f(\boldsymbol{x}'_f)$. These are the new desired joint angles.

In order to update the integrators, we first compute the global terrain-relative position of every foot ($\boldsymbol{X}_f$) using the current joint positions and pose of the robot: $\boldsymbol{x}_f = FK_f(\boldsymbol{j}_f), \boldsymbol{X}_f = POSE(\boldsymbol{x}_f)$. We then hypothesize the robot being in the correct position and orientation and compute body-relative positions of the feet's current terrain-relative positions in the ideal body frame: $\boldsymbol{x}\_\boldsymbol{d}_f = POSE\_d^{-1}(\boldsymbol{X}_f)$. Some fraction of the difference between the actual body-relative position of the feet and the ideal body-relative position of the feet is added to the feet's integrators: $\boldsymbol{I}'_f = \boldsymbol{I}_f + k \cdot (\boldsymbol{x}\_\boldsymbol{d}_f - \boldsymbol{x}_f)$. Assuming no further slippage occurs, this control will move the body towards its correct orientation and position. In order to have the feet step into their intended locations, the integrator for each foot is decayed to zero while the foot is in fight. Since the foot is no longer on the ground in this case, the foot is no longer needed to correct the body position and orientation. Assuming the stance feet succeed in correcting the body's position and orientation, the flight foot, with zeroed integration term, will step into its intended location on the terrain.

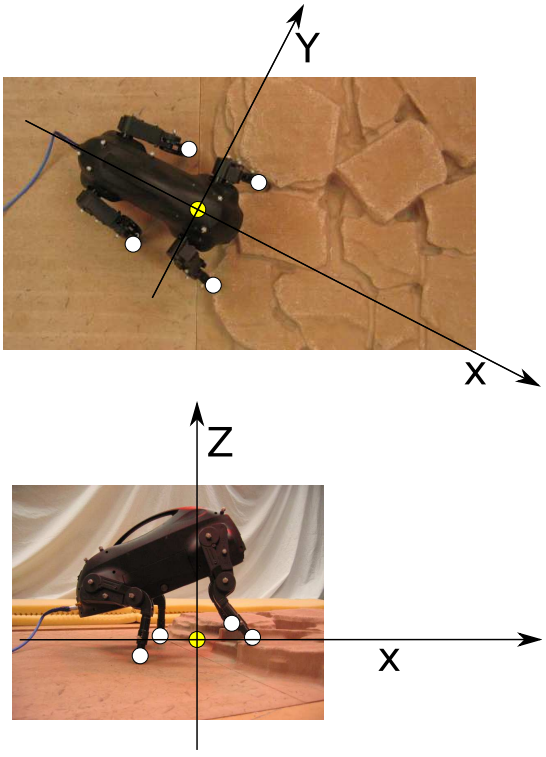


Fig. 6. Local frame

V. Library Transfer

Clearly, a global or absolute state space representation of the policy is limited to a particular task. If the terrain is moved slightly, the steps will be executed incorrectly. Furthermore, if parts of the terrain have changed, the policy cannot adapt to the changes. In order to solve these problems, we created an algorithm for transferring trajectory libraries to different environments. As the first part of the algorithm, a local feature-based representation of the environment is used

- transfer library
  - create height profile for every step
  - create height profile for a sampling of positions and orientation on the new map
  - for each step, find best match and transform step to the best match, discard if best match worse than some threshold
- find appropriate steps to get from s_{start} to the goal s_{goal} if p is a step, $s(p)$ is the start state of the step, $f(p)$ is the final state of the step.
  - add two steps, p_{start} and p_{goal} with $s(p_{start}) = f(p_{start}) = s_{start}$ and $s(p_{goal}) = f(p_{goal}) = s_{goal}$
  - $\forall p, p'$, find the Euclidean foot location metric between $f(p)$ and $s(p')$.
  - Define the successors of p, $succ(p)$ to be the n p' with the smallest distance according to the metric.
  - Create footstep plans between all $f(p)$, $s(p')$, s. t. $p' \in succ(p)$
  - Perform a Best-First-Search (BFS) through the graph whose vertices are the footsteps p and directional edges are defined from $p \rightarrow p'$ whenever $p' \in succ(p)$
  - The final library consists of all steps p on the path determined by the BFS as well as all generated footsteps by the footstep planner on that path.

Fig. 7. concise transfer and planning description

to find appropriate places for state-action pairs. In the Little Dog domain, we create a local height profile for each step. The origin of the local frame (figure 6) for the profile is the centroid of the global foot positions at the beginning of the step. The x-axis is aligned with a vector pointing from the XY-center of the rear feet towards the XY-center of the front feet. The z-axis is parallel to the global z-axis (aligned with gravity). The height of the terrain is sampled at 464 positions on a regular grid (0.35m $\times$ 0.20m with .012m resolution) around this origin to create a length 464 vector. The grid is normalized so that the mean of the 464 entries is zero. In the same way, we then create local terrain representations for a sampling of all possible positions and rotations around the z-axis. The rotations around the z-axis are limited to rotations that have the dog pointing roughly to the right ($\theta \in \{-\pi/4, \pi/4\}$). For every step in the library, we then find the local frame on the new map that produces the smallest difference in the feature vector. If this smallest difference is larger than some threshold, the step is discarded. The threshold is manually tuned to ensure that steps do not match inappropriate terrain. Otherwise, the step is transferred to the new location by first representing the position and orientation of its start state and all subsequent states in the local frame of the step. We then translate these local positions and orientations back into global coordinates based on the best local frame in the new map.

For performance reasons, after creating the feature vectors for the matching of steps, we used principal component analysis (PCA) to project the vectors into a lower dimensional space. The PCA space was created beforehand by creating feature vectors for one orientation of all obstacle boards we had. The first 32 eigenvectors, whose eigenvalues summed to 95% of the total sum of eigenvalues, were chosen as the basis for the PCA space.

Once all steps have been discarded or translated to new appropriate positions, we perform the search through the library. Due to the relocation, there is no guarantee that the steps still form a continuous sequence. Depending on the size and diversity of the source library, the steps of the new library will be scattered around the environment. Even worse, some steps might no longer be goal directed. In some sense, the steps now represent capabilities of the dog. In places where a step is located, we know we can execute the step. However, it is unclear if we should execute the step at all or in what sequence. We solve this problem by performing a search over sequences of steps. In order to connect disconnected steps, we use a footstep planner [21]. Given the configuration of the robot at the end of one step and the beginning of another step, the footstep planner can generate a sequence of steps that will go from the first to the latter. A heuristic algorithm is used to control the body and the actual leg trajectories online while executing the footsteps from the footstep planner.

For the search, we generate a topological graph. The nodes of the graph are the start state and the goal state of the robot, as well as every step in the transferred library. Edges represent walking from the end of the pre-recorded step represented by the start node to the beginning of the pre-recorded step represented by the target node. The cost of every edge is roughly the number of additional steps that have to be taken to traverse the edge. If the foot locations at the end of the source pre-recorded step are close to the foot locations at the beginning of the target pre-recorded step of the edge, no additional steps are necessary. In order to know the number of additional steps, the footstep planner is used at this stage to connect the gaps between steps when we generate the topological graph. Since the steps that are output by the planner are considered risky, we assign a higher cost to planned steps. (If the planner created reliable steps, we could just use the planner to plan straight from the start to the goal.) In order to reduce the complexity of the graph, nodes are only connected to the n-nearest steps based on the sum of Euclidean foot location difference metric. We then use a best-first search through this graph to find a sequence of footstep-planner-generated and pre-recorded steps. This sequence is added to the final library.

VI. Experiments

We performed several experiments to verify the effectiveness of the proposed algorithms. For all experiments we started with 7 libraries that were created from two different terrains. Using a joystick, one terrain was crossed in four different directions and the other terrain was crossed in three different directions (figure 8). The combined library contained 171 steps.

In order to test transfer using terrain features, we first looked at transferring the steps from these seven libraries to one of the original terrains. In theory, the steps from the library that was created on the same terrain should match perfectly back into their original location. Some spurious steps from the other terrains might also match. This is indeed

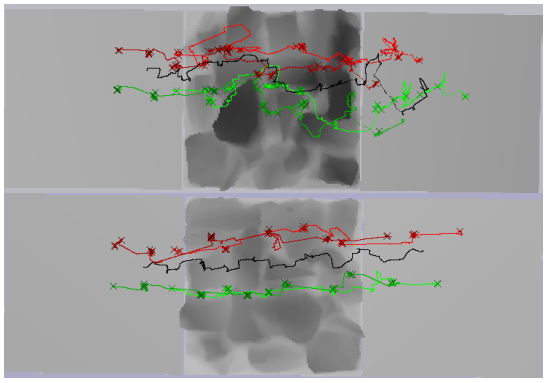


Fig. 8. Excerpts from the trajectory library: The black line marks the trajectory of the body while the colored lines correspond to the four feet (red=left side, green=right side; bright=front, dark=back). Crosses mark the position of the feet at the start of every step. The dog moves from left to right

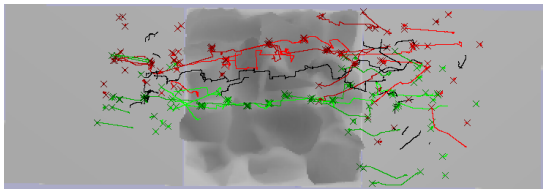


Fig. 9. Library matched against one of the source terrains. Some crosses do not have traces extending from them, since they are the starting location for a foot from a step where one of the other three feet was moving.

the case as can be seen in figure 9. The spurious matches are a result of some steps walking on flat ground. Flat ground looks similar on all terrains.

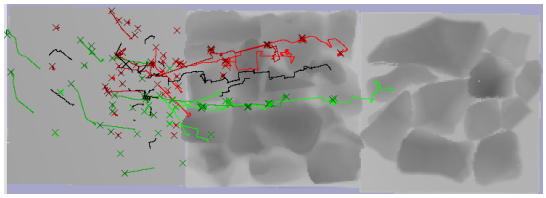


Fig. 10. Library matched against new, modified terrain

When modifying the terrain, we expect the steps to still match over the unchanged parts. However, where the terrain has changed, the steps should no longer match. For this experiment we modified the last part of a terrain to include new rocks instead of the previously flat part (figure 10). The matching algorithm correctly matches the steps that are possible and does not incorrectly match steps on the modified parts.

While the matching correctly identifies where to place steps in the library, the resulting library needs to be improved, as anticipated. There are large gaps between some steps. Moreover, some spuriously matched steps do not make progress towards the goal but can lead the robot away from it, if they happen to be matched greedily. We now use the search algorithm described earlier to postprocess the resulting library. The resulting plan should select the right steps, throwing out the spurious matches. Furthermore, by invoking the footstep planner to connect possible steps together, it will also fill in any gaps. This happens correctly

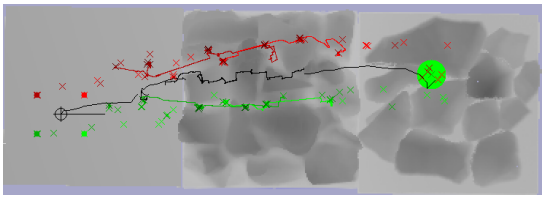


Fig. 11. Result of searching through the library on modified terrain with the green spot as the goal. The steps coming from the footstep planner do not show traces from the starting places of the feet (crosses), since the foot trajectories are generated on the fly during execution. The body trajectory for planned steps are only hypothetical trajectories — the on-line controller is used for the actual trajectories during execution.

for the modified map (figure 11).

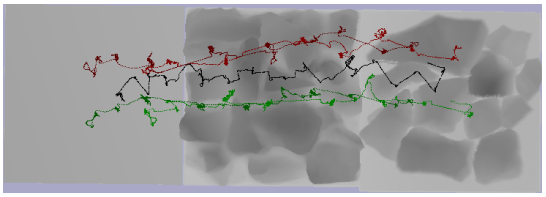


Fig. 12. Trace of Little Dog executing the plan

Finally, in order to validate the transfer algorithm, we executed the resulting library on the terrain with the modified end board. A plan, from a slightly different start location but otherwise identical to figure 11, was executed on the robot and the robot successfully reached the goal, switching between steps from the source library that were created by joystick control and the synthetic steps created by the footstep planner (figures 4,12).

VII. Discussion

The current algorithm allows each state-action pair in the library to match only once. However, it is quite conceivable that there are multiple states in a new task where an action could be executed. Finding a suitable second location is not trivial, though, since the second best match will usually be right next to the first match. One possible solution is to look for local maxima only or to introduce a minimum distance that matches have to be away from each other for any particular state-action pair.

Another limitation of the current algorithm is that given a start state, the search through the transferred library only yields a single trajectory. In order to increase the number of trajectories in the final library, one could perform multiple searches from different start states. Alternatively, a backwards search from the goal could be performed and the complete search tree added to the library. Finally, instead of searching once, it is possible to continuously search through the library during execution. Since the search is on a topological graph, this search would be much faster than the search performed by a path planning algorithm in the continuous state space. The gaps between steps are already filled in when creating the topological graph and do not have to be replanned during the continuous search process.

A more radical departure from the current algorithm would be to do away with explicitly finding global states where the features of the state-action pairs from the original library

match. Instead, one could greedily match actions from the library based on local features of the start state and its vicinity. After executing the action, this can be repeated. Applying the library greedily based on local features does not allow for searching and might result in dead-ends. Also, it will not allow the robot to cross large gaps in the library if it is not in the vicinity.

Alternatively, one could search for a sequence of steps leading towards the goal, performing a local search at every expansion to find one or more suitable successor steps in the vicinity of the termination of the previous step. However, this will not work if the local searches fail to find matching steps because of gaps — large areas where no steps in the library match. One could extend the local search area until, in the limit, the complete relevant state space is searched at every expansion. This would essentially be the algorithm that is presented here.

In the Little Dog domain, we have shown that we can transfer trajectory libraries to modified versions of the terrain where the source libraries came from. However, the use of sum-of-squared errors of the feature vectors results in a very restrictive matching. For Little Dog, it is important that the terrain supports the stance feet and that neither the body nor the flight foot collide with the terrain. Hence, there are certain variations of the terrain (lower terrain in parts where the stance feet are not supported or higher terrain in parts which are not occupied by any part of the robot) that can be easily tolerated. However, the current metric would not match when there are inconsequential terrain changes. Unfortunately, due to inaccuracies in the robot model, it is not possible to just compute the swept volume of the robot's body and check for collisions: in many steps, feet move close enough to the terrain that such a collision checker would detect collisions and reject the step, even if the step is executable.

VIII. Future Work

In our future work, we would like to address some of the issues mentioned in the Discussion section. In particular, we would like to explore matching algorithms that more closely match the capabilities of the dog. This means that matching needs to take into account collisions in a meaningful way, so that changes in the terrain that don't affect the executability of a step do not count against the step.

Furthermore, we would like to allow for a step to be matched multiple times. We hope to obtain denser libraries after the transfer so that we are less reliant on the footstep planner. Ideally, if we have enough steps in our source library, no footstep planning should be necessary.

IX. Conclusion

We introduced a method for transferring libraries of trajectories to new environments. We have shown that the method correctly transfers libraries in the Little Dog domain based on terrain features. Furthermore, a search is used to effectively find relevant steps on the new terrain and fill in gaps in the library using a footstep planner.

References

[1] M. Stolle and C. G. Atkeson, "Policies based on trajectory libraries," in *Proceedings of the International Conference on Robotics and Automation (ICRA 2006)*, 2006. [Online]. Available: http://www.cs.cmu.edu/~mstoll/publications.shtml

[2] R. E. Fikes, P. E. Hart, and N. J. Nilsson, "Learning and executing generalized robot plans," *Artificial Intelligence*, vol. 3, pp. 251–288, 1972.

[3] J. Laird, P. Rosenbloom, and A. Newell, "Chunking in Soar: The anatomy of a general learning mechanism," *Machine Learning*, vol. 1, pp. 11–46, 1986.

[4] G. A. Iba, "A heuristic approach to the discovery of macro-operators," *Machine Learning*, vol. 3, pp. 285–317, 1989.

[5] M. M. Veloso, "Learning by analogical reasoning in general problem solving," Ph.D. dissertation, Carnegie Mellon University, 1992.

[6] E. Winner and M. Veloso, "Automatically acquiring planning templates from example plans," in *Proceedings of AIPS'02 Workshop on Exploring Real-World Planning*, April 2002. [Online]. Available: http://www-2.cs.cmu.edu/~coral/publications/b2hd-02aipsw-elly.html

[7] A. Fern, S. W. Yoon, and R. Givan, "Learning domain-specific control knowledge from random walks." in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2004, pp. 191–199.

[8] S. Mahadevan, "Enhancing transfer in reinforcement learning by building stochastic models of robot actions," in *Proceedings of the Ninth International Conference on Machine Learning*, 1992, pp. 290–299. [Online]. Available: http://www.cs.umass.edu/~mahadeva/organized-pubs-by-year.html

[9] S. Chernova and M. Veloso, "Learning and using models of kicking motions for legged robots," in *Proceedings of the International Conference on Robotics and Automation (ICRA 2004)*, May 2004. [Online]. Available: http://www-2.cs.cmu.edu/~coral/publications/b2hd-icra04-chernova.html

[10] N. Kohl and P. Stone, "Machine learning for fast quadrupedal locomotion," in *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, July 2004, pp. 611–616. [Online]. Available: http://www.cs.utexas.edu/~nate/pubs/b2hd-kohlaaai04.html

[11] S. Chernova and M. Veloso, "An evolutionary approach to gait learning for four-legged robots," in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS 2004)*, September 2004. [Online]. Available: http://www-2.cs.cmu.edu/~coral/publications/b2hd-iros04-chernova.html

[12] T. Röfer, "Evolutionary gait-optimization using a fitness function based on proprioception," in *Eighth International Workshop on Robocup 2004*, 2005. [Online]. Available: http://www.informatik.uni-bremen.de/~roefer/public_e.htm

[13] J. D. Weingarten, G. A. D. Lopes, M. Buehler, R. E. Groff, and D. E. Koditschek, "Automated gait adaptation for legged robots," in *International Conference in Robotics and Automation*. New Orleans, USA: IEEE, 2004.

[14] A. McGovern, "Autonomous discovery of temporal abstractions from interaction with an environment," Ph.D. dissertation, University of Massachusetts Amherst, 2002. [Online]. Available: http://www.cs.ou.edu/~amy/pubs.html

[15] M. Stolle and D. Precup, "Learning options in reinforcement learning," *Lecture Notes in Computer Science*, vol. 2371, pp. 212–223, 2002. [Online]. Available: http://www.cs.cmu.edu/~mstoll/publications.shtml

[16] Ö. Şimşek and A. G. Barto, "Using relative novelty to identify useful temporal abstractions in reinforcement learning," in *Proceedings of the Twenty-First International Conference on Machine Learning*, 2004. [Online]. Available: http://www.cs.umass.edu/~ozgur/

[17] S. Mannor, I. Menache, A. Hoze, and U. Klein, "Dynamic abstraction in reinforcement learning via clustering," in *Proceedings of the Twenty-First International Conference on Machine Learning*, 2004.

[18] B. Ravindran and A. G. Barto, "SMDP homomorphisms: An algebraic approach to abstraction in semi Markov decision processes," in *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. AAAI Press, August 2003. [Online]. Available: http://www.cs.iitm.ernet.in/~ravi/

[19] D. C. Bentivegna, C. G. Atkeson, and G. Cheng, "Learning similar tasks from observation and practice," in *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems.*, Beijing, China, October 2006, pp. 2677–2683.

[20] M. Stolle and C. G. Atkeson, "Knowledge transfer using local features," in *Proceedings of the IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)*, 2007. [Online]. Available: http://www.cs.cmu.edu/~mstoll/publications.shtml

[21] J. Chestnutt, M. Lau, K. M. Cheung, J. Kuffner, J. K. Hodgins, and T. Kanade, "Footstep planning for the Honda ASIMO humanoid," in *Proceedings of the IEEE International Conference on Robotics and Automation*, April 2005. [Online]. Available: http://www.ri.cmu.edu/pubs/pub_4970.html