Efficient Task Execution and Refinement through Multi-Resolution Corrective Demonstration

Çetin Meriçli, Manuela Veloso, and H. Levent Akın

***Abstract*— Computationally efficient task execution is very important for autonomous mobile robots endowed with limited on-board computational capabilities. Most robot control approaches assume fixed state and action representations, and use a single algorithm to map states to actions. However, not all instances of a given task require equally complex algorithms and equally detailed representations. The main motivation for this work is a desire to reduce the computational footprint of performing a task by allowing the robot to run simpler algorithms whenever possible, and resort to more complex algorithms only when needed. We contribute the Multi-Resolution Task Execution (MRTE) algorithm that utilizes human feedback to learn a mapping from a given state to an appropriate detail resolution consisting of a state and action representation, and an algorithm. We then present Model Plus Correction (M+C), an algorithm that complements an existing robot controller with corrective human feedback to further improve the task execution performance. Finally, we introduce Multi-Resolution Model Plus Correction (MRM+C) as a combination of MRTE and M+C. We provide formal definitions of MRTE, M+C, and MRM+C, showing how they relate to general robot control problem and Learning from Demonstration (LfD) methods. We present detailed experimental results demonstrating the effectiveness of proposed methods on a simulated goal-directed humanoid obstacle avoidance task.**

I. INTRODUCTION

Computational footprint of a robot controller is often overlooked as long as the controller is able to perform as expected on the robot. As robots become ubiquitous and general-purpose, multiple software modules are likely to run on the robot simultaneously. Despite the continuous advancements in the computational technology that enables the mobile robots to be equipped with more and more powerful computers, the on-board computational resources are still to be shared among these multiple software components.

Most robot control approaches consider a fixed state representation computed using the sensory input, an action representation, and an algorithm for executing the task at hand through mapping states to actions. Employing a complex algorithm that uses the most detailed state representation and action definitions available might be computationally expensive and infeasible for continuous use. On the other hand, some instances of the task might be handled using simpler algorithms operating at a coarser detail resolution. However, using only a simple algorithm and less detailed representations might fail to capture the details in more complex situations, and might eventually lead to a failure in the task execution.

To learn from human demonstration how to change the internal state representation and the corresponding algorithm to perform the task according to the complexity of a given situation, we contribute Multi-Resolution Task Execution (MRTE), an algorithm that employs a set of detail resolutions, each having its own state and action representations, and a controller operating on these representations to perform the task. A policy is learned from human demonstration, and then used to determine which detail resolution to use in a particular state of the system during autonomous execution of the task. Next, we present Model Plus Correction (M+C), a formal model based on our previous work for improving the execution performance of an algorithm by augmenting it with corrective feedback received from a human teacher [1]. Finally, we combine MRTE with M+C, and present the Multi-Resolution Model Plus Correction (MRM+C) algorithm that allows the robot to learn how to dynamically change the detail resolution to operate at for a given state, and how to further improve the execution performances of individual algorithms running at different detail resolutions through multi-resolution corrective demonstration.

Over the course of a training session, the teacher observes the robot executing the task using hand-coded algorithms and intervenes either to deliver a corrective action at the current detail resolution, or to switch the system to a finer detail resolution. The robot learns a detail switching policy for deciding which detail resolution to use in a particular state while also building up individual corrective demonstration databases for the algorithms at each detail resolution. During the autonomous execution of the task, the robot first chooses the appropriate detail resolution to run at in the current state, and then decides whether to execute a demonstrated action or the action computed by the default controller.

II. BACKGROUND AND RELATED WORK

We define the general robot control problem formally as a tuple $\langle Z, A, \pi \rangle$. The world consists of states S, and A is the set of actions the robot can take. Transitions between states are defined with a probabilistic transition function $T(s'|s,a) : S \times A \times S \rightarrow [0,1]$. The state is not fully observable; instead, the robot has access to an observed state Z with the mapping $M : S \rightarrow Z$. The robot uses an execution policy $\pi : Z \rightarrow A$ for selecting the next action $a \in A$ based on the current observed state $z \in Z$.

Çetin Meriçli and Manuela Veloso are with the Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213 {cetin,veloso}@cmu.edu

H. Levent Akın is with the Department of Computer Engineering, Boğaziçi University, Istanbul, Turkey akin@boun.edu.tr

A. Learning from Demonstration

Learning from Demonstration (LfD) is a supervised learning approach for transferring task or skill knowledge to an autonomous robot by means of the demonstrations of the task or skill execution [2]. The LfD methods make use of a teacher who demonstrates the robot how to perform the task or skill at hand as the robot records the demonstrated actions associated with the perceived state of the system synchronously. The robot then uses the stored state-action pairs to derive an execution policy for reproducing the demonstrated task or skill.

We define LfD formally as an instance of general robot control problem with a tuple $\langle Z, A, \pi_{demo} \rangle$, where the execution policy $\pi_{demo} : Z \rightarrow A$ is extracted from a demonstration dataset D consisting of teacher demonstrations $d \in D$, where $d = \langle z, a_{demo} \rangle, z \in Z, a_{demo} \in A$, and a_{demo} is the action demonstrated by the teacher in the observed state z. During execution, the robot uses π_{demo} for selecting the next action a based on the current observed state z.

B. Related Work

Being a very active field of research, LfD has been utilized in many different learning scenarios, focusing on different aspects of the learning. Here, we present a few representative studies and state how our approach relates to them.

A method for for utilizing human feedback as the reward signal for Reinforcement Learning (RL) has been proposed in [3]. Our approach differs with the way the human feedback is incorporated. We use human feedback to update the existing task definitions while their method utilizes the received feedback for training a RL system. A sliding-autonomy approach for learning behavior policies from human demonstration called "Confidence Based Autonomy (CBA)" has been introduced in [4]. The main difference between CBA and our proposed research is that instead of starting from scratch, our approach needs teacher feedback only when the existing algorithms fail. A method for refining a demonstrated skill execution policy using kinesthetic feedback from the teacher during the execution of the skill using the execution policy extracted from the demonstration examples has been proposed in [5]. Their work shares a similarity with our approach as both systems utilize provided human feedback interleaved with the task or skill execution, but their method uses a single detail resolution. A hierarchical apprenticeship learning approach for learning complex skills which are non-trivial even for the domain experts by providing isolated advice at different hierarchical levels has been introduced in [6]. Our multi-resolution approach differs from their work with its utilization of multiple algorithms with different computational complexities to handle different situations of the same task. An algorithm for incrementally learning subtasks in a task hierarchy by automatically partitioning a given demonstration for the full task has been proposed in [7]. Despite being hierarchical, their method uses a single state and action definition in contrast to our approach.

III. APPROACH

A. Multi-Resolution Task Execution (MRTE)

We define Multi-Resolution Task Execution (MRTE) as a tuple $\langle \pi_{arbitrator}, \{c_1, c_2, ..., c_N\} \rangle$, where c_r is the controller defined for the detail resolution $r \in R$, and $\pi_{arbitrator}(z) : Z \rightarrow R$ is the detail resolution selection policy. A controller for the detail resolution r is defined as a tuple $c_r = \langle Z_r, A_r, f_r^{state}, f_r^{action}, \pi_{model}(r) \rangle$ where $f_r^{state} : S \rightarrow S_r$ is the function for mapping the global state to the state definition at the detail resolution r, and $f_r^{action} : A_r \rightarrow A$ is the function for mapping the action computed at the detail resolution r into an action representation at the finest detail level.

The detail resolution selection component acts as an arbitrator among the different detail resolutions. Given the current observed state of the system, it decides which detail resolution to run at for computing the next action. Unless stated otherwise, the system always runs at the coarsest detail resolution. A human teacher provides demonstrations to teach the robot when to switch to a finer detail resolution. The task execution policy $\pi_{model}(r)$ for the controller of the detail resolution r is provided by a hand-coded algorithm. Fig. 1 shows the schematic representation of MRTE.

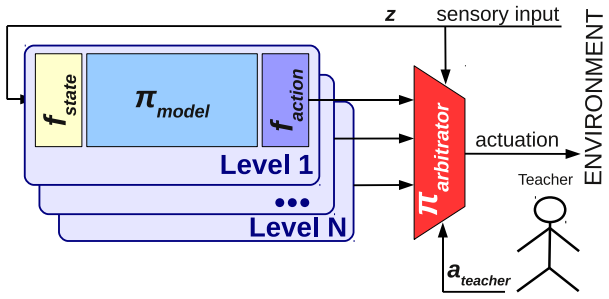


Fig. 1. The schematic representation of MRTE.

B. Model Plus Correction (M+C)

We introduce *Model Plus Correction (M+C)*, a complementary corrective demonstration algorithm that makes use of an available controller, and utilizes corrective human demonstration to further refine the performance of the controller.

We define M+C formally as a tuple $\langle Z, A, \pi_{demo}, \pi_{model}, f_{reuse} \rangle$. The LfD definition is extended with a model-based algorithm $\pi_{model} : Z \rightarrow A$, and a correction reuse function $f_{reuse}(z, a_{demo}, a_{model}) : Z \times A \times A \rightarrow A$, where a_{demo} is the action computed by π_{demo}, and a_{model} is the action computed by π_{model}. f_{reuse} is a domain and task specific function that computes the final action to be executed as a function of the current observed state, and the actions computed by the model-based and the demonstration policies. During the training process, the human teacher observes the robot performing the task using the model-based policy and corrects the action of the robot if the computed action is erroneous. During the execution,

if a demonstration received in a similar situation is found in the database, the f_{reuse} function replaces the action provided by the controller with the demonstrated action. Fig. 2 shows the schematic representation of M+C.

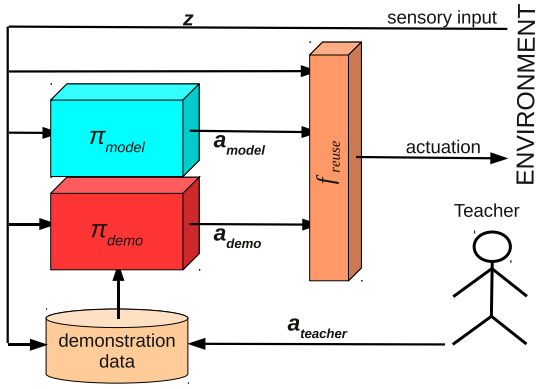


Fig. 2. The schematic representation of M+C.

C. Multi-Resolution Model Plus Correction (MRM+C)

We combine M+C with MRTE, and define Multi-Resolution Model Plus Correction (MRM+C) as a tuple $\langle \pi_{arbitrator}, \{c_1, c_2, ..., c_N\}\rangle$, where c_r is an M+C instance defined as a tuple $\langle Z_r, A_r, f_{state}(r), f_{action}(r), \pi_{demo}(r), \pi_{model}(r), f_{reuse}(r)\rangle$ at resolution $r \in R$. Fig. 3 shows the schematic diagram of MRM+C.

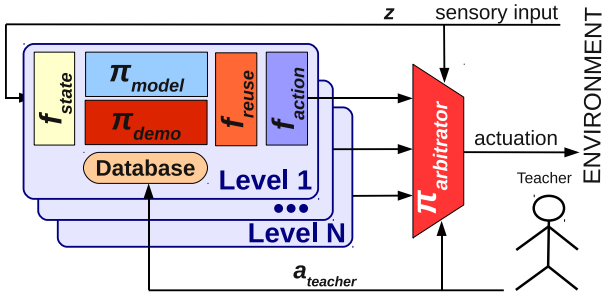


Fig. 3. The schematic representation of the MRM+C algorithm.

D. Demonstration Delivery: Training the System

During the training, the teacher uses a custom software to access the current detail resolution as well as the observed state represented at that resolution. The same user interface is also used to deliver the correction actions and issuing detail refinement commands. Along the course of a demonstration, the teacher observes the robot as it executes the task, and intervenes as needed by means of the following feedback types:

- The **elaborate** command switches the system to the next finer detail resolution.
- The **correct** command replaces the computed action to be executed with another action defined in the same detail resolution.

If an **elaborate** command is received, the system checks if there is a finer detail resolution available. If such a resolution is found, the received elaborate command is stored with the current state of the system at the finest detail resolution available. The robot then switches to the specified detail resolution, and recomputes an action. If a **correct** command is received, the provided substitute action is stored with the current state of the system at the current detail resolution. Finally, the action to be executed by the robot is replaced with the received corrected action.

E. Demonstration Reuse: Autonomous Execution

Each time the robot reaches a decision point during the autonomous task execution, the MRM+C algorithm switches to the most coarse detail resolution. Then, the robot starts searching in this particular order: i) a correction sample in the corrective demonstration database for the current detail resolution, ii) an elaborate command in the elaboration demonstration database for switching to the next detail level with finer resolution. If a demonstration is found in the correction database that is received in a state *similar enough* to the current state, the demonstrated correction action is selected as the next action. If an elaborate command is received in a state *similar enough* to the current state, the robot switches to the next finer detail resolution and starts over with searching. If no appropriate elaborate or correct commands are found, the algorithm for the current detail resolution computes the action to be executed. We use a domain and task specific metric with empirically determined parameters to compute the similarity of given two states. Alg. 1 shows the algorithm for the MRM+C execution.

IV. HUMANOID OBSTACLE AVOIDANCE TASK

We apply the proposed multi-resolution task execution and refinement approach on a humanoid obstacle avoidance task in a robot soccer environment. We define the obstacle avoidance task for a humanoid soccer robot as the problem of walking to a given goal location without bumping into the obstacles placed on the field. The robot starts in its own goal area, and the goal of the task is to reach within 1 meter distance of the opponent goal. The quantity, the shapes, and the locations of the obstacles on the field are not known. Fig. 4 presents an example instance of the humanoid obstacle avoidance task with three obstacles. The dashed lines represent an example traversal, and the yellow circles show the intermediate destination points selected by the robot.

To reach the goal position without hitting the obstacles, the robot has to have a way of modelling the occupancy status of the environment. We model the free space in front of the robot using a variant of the Visual Sonar approach [8]. We process the images seen through the color cameras of the robot to distinguish the soccer field and any non-field object lying on it, resulting in an occupancy map for the 180^o space in front of the robot. For each detail resolution, we divide the free space in front of the robot into a set of slices called

Algorithm 1 MRM+C execution

```
1: resolution ← COARSEST
2: state ← computeState(resolution)
3: mostSimilar ← ∅
4: maxSim ← 0
5: for each demo ∈ correctionDB_resolution do
6:    sim ← getSimilarity(state, demo(state))
7:    if sim > maxSim then
8:       maxSim ← sim
9:       mostSimilar ← demo
10:   end if
11: end for
12: threshold ← getCorrectionThreshold(resolution)
13: if maxSim > threshold then
14:   action ← demo(action)
15: else
16:   mostSimilar ← ∅
17:   maxSim ← 0
18:   for each demo ∈ elaborationDB do
19:      sim ← getSimilarity(state, demo(state))
20:      if sim > maxSim then
21:         maxSim ← sim
22:         mostSimilar ← demo
23:      end if
24:   end for
25:   threshold ← getElaborationThreshold()
26:   if maxSim > threshold then
27:      if resolution < FINEST then
28:         increaseResolution()
29:         goto 2
30:      else
31:         action ← computeAction(state)
32:      end if
33:   else
34:      action ← computeAction(state)
35:   end if
36: end if
37: executeAction(action)
```

free-space slots, where each slot has an occupancy indicator computed using the Visual Sonar occupancy data.

We calculate the similarity of two given states s_1, s_2 as $e^{-K|s_1-s_2|^2}$, where K is a coefficient for shaping the similarity function. In our implementation, we empirically selected $K = 5$. We define three detail resolutions $R = \{coarse, medium, fine\}$ for the humanoid obstacle avoidance task. In the remainder of this section, we explain the state and action definitions, and the destination point selection algorithms for each detail resolution.

A. Coarse Detail Resolution

At the coarse detail resolution, the 180^o space in front of the robot is divided into five free space slots, each covering 36^o. A boolean value indicates the existence of an obstacle along a free space slot in the state vector where **true** indicates the slot is occupied by an obstacle. If the average distance of the most detailed free space representation slots that falls within a free space slot at this resolution is less than a certain threshold, that slot is marked as occupied. In our implementation, the threshold for considering a free-space slot as occluded is 120 centimeters. Fig. 5(a) shows an example visualization of a coarse detail resolution state representation.

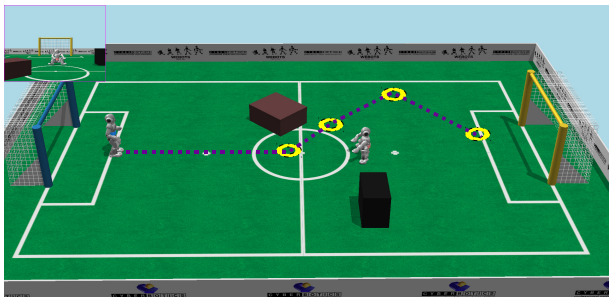


Fig. 4. An example instance of the humanoid obstacle avoidance task.

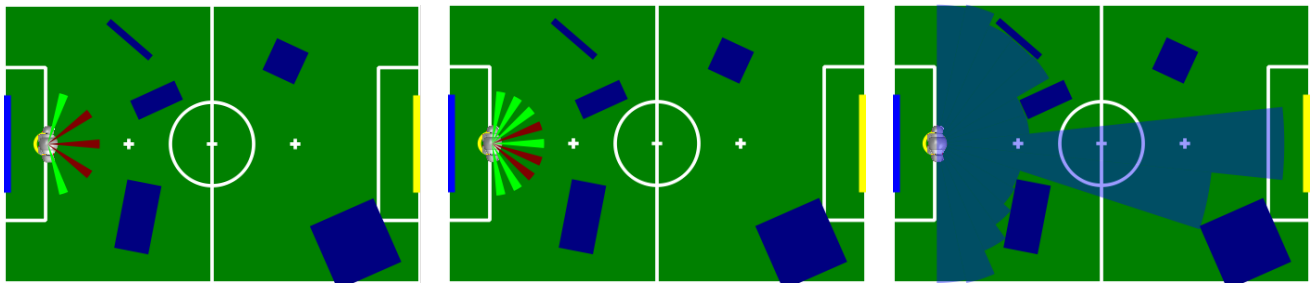
At this detail resolution, the destination point can be selected from among the five free space slot directions with a distance of 120 centimeters. However, the hand-coded algorithm for this resolution only selects from the three walking directions: *forward*, *left*, or *right*. If the middle slot (the slot number 2) is free, the algorithm selects the forward direction, otherwise it checks the right and left slots to decide. The algorithm also favors the left direction over the right direction, if the leftmost free-space slot (the slot number 4) is free. Alg. 2 shows the destination point selection algorithm for the coarse detail resolution.

Algorithm 2 Destination point selection for the coarse detail resolution.

```
1: slot ← −1
2: state ← getBooleanState(COARSE)
3: if ¬state(2) then
4:    slot ← 2
5: else
6:    if ¬state(0) then
7:       slot ← 0
8:    else
9:       slot ← 4
10:   end if
11: end if
12: angle ← calculateDirection(slot)
13: distance ← 120
14: return calculateGlobalPoint(angle, distance)
```

B. Medium Detail Resolution

The state representation for the medium detail resolution uses the same principles as the coarse state representation with the exception of using nine slots instead of five. Fig. 5(b) shows an example visualization of a medium detail resolution state representation.



(a) (b) (c)

Fig. 5. Example visualizations for the state representations at different detail resolutions for the same situation. a) coarse detail resolution, b) medium detail resolution, and c) fine detail resolution.

The hand-coded algorithm for this resolution goes over each free-space slot and selects the direction of the closest available slot to the opponent goal as the destination direction, using a fixed walking distance of 120 centimeters. Alg. 3 shows the destination point selection algorithm for the medium detail resolution.

Algorithm 3 Destination point selection for the medium detail resolution.

$state \leftarrow getBooleanState(MEDIUM)$
$goal \leftarrow getGoalSlot()$
$closestSlot \leftarrow 0$
$minDist \leftarrow 9$
for $slot \leftarrow 0; slot < 9; i \leftarrow slot + 1$ **do**
  if $|goal - slot| \leq minDist$ **and** $\neg state(slot)$ **then**
    $minDist \leftarrow |goal - slot|$
    $closestSlot \leftarrow slot$
  end if
end for
$angle \leftarrow calculateDirection(closestSlot)$
$distance \leftarrow 120$
return $calculateGlobalPoint(angle, distance)$

Algorithm 4 Destination point selection for the fine detail resolution.

$goalAngle \leftarrow getGoalAngle()$
if $goalAngle < -\frac{\pi}{2}$ **or** $goalAngle > \frac{\pi}{2}$ **then**
  if $|angle_0 - goalAngle| < |angle_{N-1} - goalAngle|$ **then**
    $destAngle \leftarrow angle_0$
  else
    $destAngle \leftarrow angle_{N-1}$
  end if
  $destDistance \leftarrow 120$
else
  $maxDist \leftarrow 0$
  for $i \leftarrow 1; i < N - 1; i \leftarrow i + 1$ **do**
    $distance \leftarrow 0.25dist_{i-1} + 0.5dist_i + 0.25dist_{i+1}$
    if $distance > maxDist$ **then**
      $maxDist \leftarrow distance$
      $maxSlot \leftarrow i$
    end if
  end for
  $angle \leftarrow angle_{maxSlot}$
  $distance \leftarrow maxDist$
end if
return $calculateGlobalPoint(angle, distance)$

C. Fine Detail Resolution

At the fine detail resolution, the free space is divided into 15 slots and the occupancy status for each slot is represented with a distance value in centimeters, denoting the distance to the closest detected obstacle lying along that particular free-space slot. Fig.5(c) shows an example visualization of the state representation for the fine detail resolution.

In addition to the destination direction, the destination selection algorithm for the fine detail resolution also determines the walk distance towards that direction. The algorithm goes over each free-space slot, and for each slot computes a weighted distance value using a sliding window of size 3 with the weights 0.25 at both ends and 0.5 for the center. The direction of the free-space slot with highest weighted distance is then selected as the walking direction and the computed weighted distance is used as the walking distance.

V. RESULTS

We evaluated the proposed algorithms using a simulated version of the obstacle avoidance task. We used two different obstacle configurations with three obstacles each where we randomly distorted the obstacle locations and orientations at every run.

We compared the performances of the hand-coded algorithms for each detail resolution (M), the improved algorithms using corrective demonstration (M+C), the multi-resolution task execution using the hand-coded algorithms (MRTE), and the multi-resolution corrective demonstration (MRM+C) algorithm. During the training session, 23 coarse, 8 medium, and 14 fine resolution level demonstrations were collected for the action correction, and 22 demonstrations were recorded for the detail resolution selection part.

We ran 100 experiments for each algorithm. Fig. 6 shows

the success rates of the algorithms. The blue bar in the Multi Resolution group is the success rate for MRTE, and the red bar in the same group is the success rate for MRM+C. As expected, the success rate of the algorithms increase as the algorithm gets more complex and runs at a finer detail resolution. Also, M+C outperformed the algorithms alone, and MRM+C outperformed MRTE, showing the impact of corrective demonstration in performance refinement. Fig. 7 shows the composition of executed actions per evaluated algorithm, and Fig. 8 shows the computational cost versus the performance evaluation. We approximated the computational complexity of an algorithm as the total number of comparison and assignment operations in the worst case execution scenario. In both the MRTE and MRM+C evaluations, the majority of the executed actions were computed by the coarse detail resolution algorithm and the coarse detail resolution demonstration database. Yet, the success rates for MRTE and MRM+C were better than the coarse and medium resolution algorithms, close to the fine detail resolution algorithm.

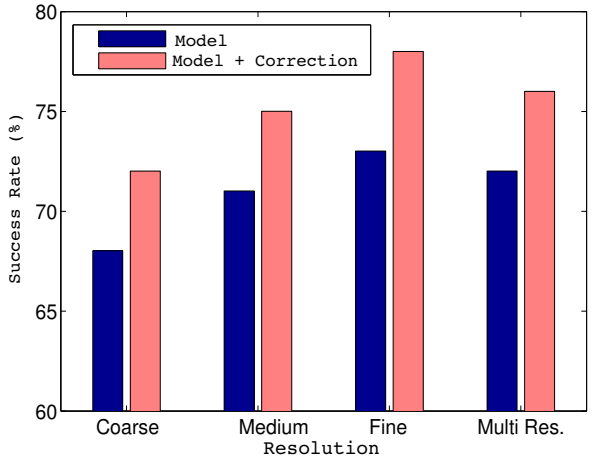


Fig. 6. The impact of corrective demonstration on the performances of the individual algorithms and MRTE. The performance is measured with the percentage of succeeded runs.

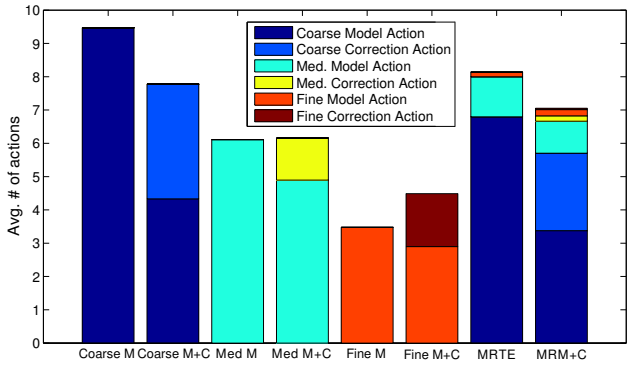


Fig. 7. The average number of actions executed per individual algorithms, M+C, MRTE, and MRM+C (best viewed in color).

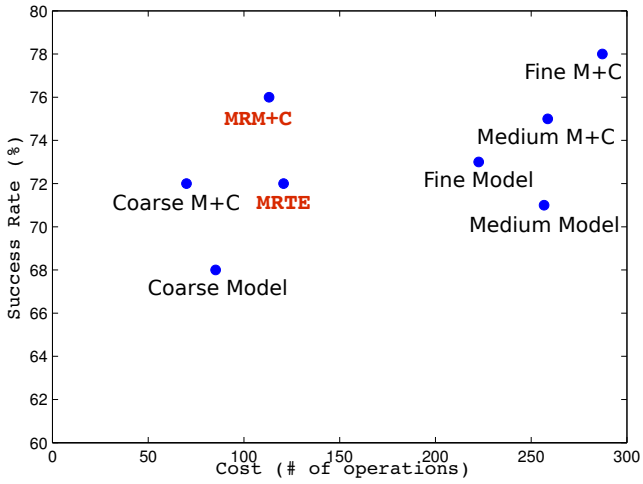


Fig. 8. The cost vs. performance for individual algorithms, M+C, MRTE, and MRM+C.

VI. CONCLUSION

In this paper, we presented a novel method for executing different parts of a task through running different algorithms with varying complexities and using state and action representations at different detail resolutions. We presented formal models for the proposed approaches that relate the presented models with each other, and with the traditional LfD methods. We provided experimental evaluation of the proposed approaches in a simulated obstacle avoidance task, demonstrating the computational efficiency of multi-resolution approach compared to controllers with fixed state and action definitions.

Possible future work includes evaluating the proposed approaches in a task with higher dimensional state and action space, investigating how the number of detail resolutions affect the performance, and examining the robustness of the system in presence of uncertainty.

REFERENCES

[1] Çetin Meriçli, M. Veloso, and H. L. Akin, "Task refinement for autonomous robots using complementary corrective human feedback," *International Journal of Advanced Robotic Systems*, vol. 8, no. 2, June 2011.

[2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Automation Systems*, vol. 57, no. 5, pp. 469–483, 2009.

[3] A. L. Thomaz and C. Breazeal, "Reinforcement learning with human teachers: evidence of feedback and guidance with implications for learning performance," in *Proceedings of AAAI 2006*.

[4] S. Chernova and M. Veloso, "Interactive policy learning through confidence-based autonomy," *Journal of Artificial Intelligence Research*, vol. 34, pp. 1–25, 2009.

[5] B. Argall, E. Sauser, and A. Billard, "Tactile feedback for policy refinement and reuse," in *Proceedings of ICDL 2010)*.

[6] J. Z. Kolter, P. Abbeel, and A. Y. Ng, "Hierarchical apprenticeship learning with application to quadruped locomotion," in *Proceedings of NIPS 2007*.

[7] D. H. Grollman and O. C. Jenkins, "Incremental learning of subtasks from unsegmented demonstration," in *In Proceedings of IROS 2010*.

[8] S. Lenser and M. Veloso, "Visual sonar: Fast obstacle avoidance using monocular vision," in *Proceedigns of IROS 2003*.