An Advice Mechanism for Heterogeneous Robot Teams

by

Steven Daniluk

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
University of Toronto Institute for Aerospace Studies
University of Toronto

Abstract

An Advice Mechanism for Heterogeneous Robot Teams

Steven Daniluk
Master of Applied Science
University of Toronto Institute for Aerospace Studies
University of Toronto
2017

The use of reinforcement learning for robot teams has enabled complex tasks to be performed, but at the cost of requiring a large amount of exploration. Exchanging information between robots in the form of advice is one method to accelerate performance improvements. This thesis presents an advice mechanism for robot teams that utilizes advice from heterogeneous advisers via a method guaranteeing convergence to an optimal policy. The presented mechanism has the capability to use multiple advisers at each time step, and decide when advice should be requested and accepted, such that the use of advice decreases over time. Additionally, collective collaborative, and cooperative behavioural algorithms are integrated into a robot team architecture, to create a new framework that provides fault tolerance and modularity for robot teams.

Acknowledgements

I would like to thank my supervisor Prof. M. R. Emami for his guidance throughout my research. Additionally, I would like to extend my gratitude to those around me who provided support throughout my degree: my family, my girlfriend Minhthi, as well as my lab mates, and my friends, Houman, Michael, Colin, and Lijun.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Enabling teams of autonomous robots with a diverse range of capabilities to seamlessly work together on a task has been a long sought after goal [2]. The applications of an effective robot team range from manufacturing, to planetary exploration, or to life critical scenarios such as search and rescue in disaster zones. However, achieving a robot team that is practical for these applications poses many difficulties, several of which are still active areas of research [1].

There are two particular areas where further developments in robot teams are necessary. The first is the unification of the architecture for organizing the control system, and the learning algorithms responsible for the robot's abilities. To date these have primarily been researched independently, without demonstrating an architecture's ability to accommodate layers of learning algorithms that create *collective*, *cooperative*, and *collaborative* behaviour within robot teams, and the practical implementations of such algorithms (e.g. centralized or distributed). The second is improving the performance of each robot through collaboration with its teammates. Given the inherent complexity in practical tasks for a robot team, it is advantageous to expedite the learning process of each robot through collaboration. A variety of mechanisms for exchanging advice between agents (i.e., robots), or receiving advice from a human, have been developed [12], [21], [16], [15]. However, an advice mechanism has yet to be developed that specifically addresses the needs of a real world robot team. Particularly, a mechanism which does not rely entirely on heuristics to control the usage of advice, and the ability to use advice from heterogeneous advisers that have learned a slightly different behaviour than the robot receiving the advice.

Progress in these two areas would create a single framework that more fully satisfies

the requirements for a real world robot team, and would expedite the learning process for all robots. Both of which bring practical robot teams one step closer to reality.

1.2 Objectives

The purpose of this research is to advance the utility of robot teams for real world applications. With this aim, this thesis has two main components: bridging the gap between robot behaviour algorithms and team architectures, as well developing a method for collaborative behaviour between heterogeneous robots.

For the first component, the collective, cooperative, and collaborative robot behaviour in Robust Concurrent Individual and Social Learning (RCISL) will be integrated into the Host Avatar Agent (HAA) architecture which provides modularity, scalability, and fault tolerance. In pursuit of this goal, this section of the research has the following two goals:

1. Organize the algorithms for each level of behaviour into modular forms compatible with the HAA architecture
2. Develop strategies for the usage and interaction with each behavioural component to ensure fault tolerance

The second component of this work, the development of a new advice mechanism, is intended to create a form of collaborative behaviour that is suitable for use with robot teams. The applicability for real world robot teams places additional requirements on the advice mechanism. Specifically, the goals for the advice mechanism are:

1. Compatible with heterogeneous advisers
2. Guarantee convergence to an optimal policy
3. Diminish the usage and influence of advice over time

1.3 Outline

In the introduction (Chapter 1) the motivation and goals of the research have been presented. In addition to this, background material necessary for the remainder of this work will be presented. The topics include robot team architectures, reinforcement learning, and methods of incorporating advice from other agents or external sources.

A brief discussion of RCISL and the HAA architecture, as well as the motivation for combining the two, is presented in Chapter 2. This is followed by the methods for

structuring the algorithms in RCISL to ensure modularity, scalability, and fault tolerance, and incorporating them into the HAA architecture.

Chapter 3 deals with the formulation of the Preference Advice mechanism. This includes the methods for incorporating advice, deciding when to use advice, and a proof demonstrating that a robot operating under an advised policy, using the Preference Advice mechanism, will converge to a stationary and optimal policy.

Two case studies are used to evaluate the Preference Advice mechanism. Chapter 4 contains the first case study, which evaluates the Preference Advice mechanism with a heterogeneous robot team performing the same task. In Chapter 5 the second case study is presented, which adds heterogeneity to the tasks, incorporates a layer of team learning, as well as adds sensory noise for each robot.

Chapter 6 includes some concluding remarks about the integration of RCISL into the HAA architecture, as well as the Preference Advice mechanism. Some comments regarding future work are also made.

1.4 Background

1.4.1 Robot Team Architectures

The architecture for a robot team defines the organization of the control system, as well as the mechanisms and algorithms governing the communication and interactions between different elements of the control systems. Hence, it plays a vital role not only in the capabilities of the team, but also in practicality for using the team. This importance has been highlighted in the suitably named *1,001 Robot Architectures for 1,001 Robots* [25], which describes the need for modularity and re usability when developing capabilities for robot teams.

Two important components of a robot team architecture are team formation, and team capabilities. Team formation refers to how the roles within the team are created and organized, while team capabilities refers to the algorithms defining robot behaviour. Some approaches to robot team formation have utilized predefined roles to dynamically form teams and manage the members [7, 9], enabling new members to be added and removed. A role can entail a responsibility for a task, or a governing process, such as the processes for environmental mapping or computational resource allocation. When multiple tasks and robots exist, a strategy for allocating tasks to team members is used, and these are often either market-based or behaviour-based approaches [24, 10, 11, 27]. Team formation and capabilities are coupled together, since the method used for forming

teams must not only account for the capabilities of the team, but also be versatile enough to accommodate changing capabilities for new applications.

Another important component of a robot team architecture is how computational processes and memory are distributed amongst the hardware, as well as the impact of any hardware or software failures. Since robot teams are often performing computationally demanding tasks, and are also typically connected via a single network, it is possible to distribute the processing amongst the team's computational resources. Strategies for utilizing shared hardware resources amongst robots have been presented in [13] and [29]. Additionally, since robot applications frequently involve the storage of state data about the team, environment, or mission, efforts must be made to minimize the likelihood of losing important information, or crippling the entire team due to a single failure. Preserving robot behaviour and information is discussed in [26], where processes can be saved and migrated between machines (i.e., robots). When learning algorithms are used, the behavioural policy of a robot is continuously evolving, which makes preservation of behaviour and memory essential. This importance is emphasized for real world applications, where failures or loss of information must not compromise the mission.

1.4.2 Reinforcement Learning

A popular component within individual robot capabilities is the ability to learn to perform new tasks through experience. Reinforcement learning is a common method for achieving this, and a brief introduction to the subject will be outlined here.

For an autonomous agent that performs actions, a, in states, s, and receives a reward, R, it must learn an appropriate mapping between states and actions. The purpose of reinforcement learning is to develop such mapping, called the policy π, that will maximize the expected reward of the agent over its lifetime. The reinforcement learning problem can be formalized as a Markov Decision Process (MDP), which consists of a set of states S, actions A, transition probabilities between states $T(s', a, a)$, and rewards $R(s, a, s')$.

The value of a particular state can be estimated by the cumulative expected reward for the current and future states (given the Markov property, the expected value of a state does not depend on any previous states.) The Bellman Equation (eq. 1.1) provides the expected value of a state, as a recursive formulation depending on the value of the succeeding state [3]. The factor $\gamma \in (0, 1)$ discounts the value of future states, giving a lower value to the states further into the future.

$$\begin{aligned} v_\pi(s) &= \mathbf{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right] \\ &= \sum_a \pi(a|s) \sum_{s'} T(s', a, s) \left[R(s, a, s') + \gamma v_\pi(s') \right] \end{aligned} \tag{1.1}$$

The Bellman Optimality Equation (eq. 1.2) represents the value of state s, given that every subsequent action taken is the optimal action.

$$v_*(s) = \max_{a \in A(s)} \sum_{s'} T(s', a, s) \left[R(s, a, s') + \gamma v_*(s') \right] \tag{1.2}$$

An alternative formulation of the Bellman Optimality Equation is to estimate the value of a state-action pair, $q_*(s, a)$, as given in eq. 1.3. The benefit of such a formulation is that it is not required to know the transition function $T(s, a, s)$, and is hence referred to as model-free reinforcement learning.

$$q_*(s, a) = \sum_{s'} T(s', a, s) \left[R(s, a, s') + \gamma \max_{a'} q_*(s', a') \right] \tag{1.3}$$

A popular approach to estimating the state-action value function $q_*(s, a)$ is the temporal difference method, which iteratively updates the value of the previous state as the new state and the corresponding reward are experienced. One such temporal difference method is Q-learning, which uses the update rule given by eq. 1.4 [33]. In eq. 1.4, $\alpha \in (0, 1)$ is the learning rate, which approaches 0 as t approaches ∞, such that eq. 1.4 will arrive at a locally optimal policy [33].

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t \left(R_t \left(s_t, a_t, s_{t+1} \right) + \gamma \max_{a \in A} Q_t \left(s_{t+1}, a \right) - Q_t \left(s_t, a_t \right) \right) \tag{1.4}$$

An alternative to Q-learning is SARSA(0) [28], which updates the values of $Q(s, a)$, commonly called Q-values, based on the selected action in the next state, as given by eq. 1.5.

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t \left(R_t \left(s_t, a_t, s_{t+1} \right) + \gamma Q_t \left(s_{t+1}, a_{t+1} \right) - Q_t \left(s_t, a_t \right) \right) \tag{1.5}$$

The policy for selecting actions, π, can either be deterministic, e.g., greedily selecting the most optimal action, or probabilistic, which will form a probability distribution over all possible actions. The motivation for a probabilistic policy is that it provides a balance

between exploration and exploitation of the current value function. An optimal policy depends upon the value function being determined, which by eq. 1.2 requires every state to be visited. Utilizing a probabilistic policy enables a balance between exploration nd exploitation to be achieved. An example of a probabilistic policy is the Boltzmann distribution, given by,

$$p(a_i) = \frac{e^{\tau_t(s)Q(s,a_i)}}{\sum_{j=1}^{n} e^{\tau_t(s)Q(s,a_j)}}, \qquad i = 1, ..., n \tag{1.6}$$

where n is the number of actions, and $\tau_t(s)$ is a temperature parameter controlling the smoothness of the distribution, which can vary between states.

1.4.3 Advice Mechanisms

The concept of advice for a computer program has existed for nearly 60 years [19]. Much of the recent research into advice has been focused on uses for reinforcement learning algorithms, due to their popularity. In general, advice within a reinforcement learning context can take three forms:

1. incorporating prior information into the agent's policy;
2. sharing a set of experiences; and
3. recommending an action.

Incorporating prior information into the agent's policy is often referred to as *transfer learning*, and is achieved by either imposing human generated rules onto the policy, or creating a mapping from a previously learned policy for a related task [31]. Transfer learning can create a proper jumpstart in the learning process [4], and avoid much of the early exploration costs. However, this requires extensive prior knowledge about the task and agents in order to generate the appropriate mappings. Since there is a cost of human involvement to generate the mappings for each application, transfer learning is most applicable for tasks and agents that are not frequently subject to change, such as RoboCup competitions where it is commonly applied [32].

The second form of advice is sharing experiences between agents. After performing what is deemed as a successful series of actions, the adviser agent will share with the advisee the set of $\langle S_t, S_{t+1}, A_t, R_t \rangle$ tuples that led to success. The advisee then replays these experiences, with the intention that they will learn to replicate this behavior.

In [34], a theoretical analysis of exchanging experiences is performed, assuming that agents simply observe their adviser and replay each of the observed experiences as if they were their own. Further, it is shown that the use of experiences reduces the search time for an optimal policy to be independent of the state space size and only dependent on the length of the optimal solution, i.e., the number of consecutive actions.

In [12], a series of successful experiences from a knowledgeable advisor, i.e., an agent with a learned policy or a human controller, are replayed by a learning agent. Such demonstrations enabled the agent to not only learn more quickly, but also perform tasks that they were initially unable to learn on their own. As mentioned in [12], a limitation of advice in the form of sharing experiences is the potential for harmful over-training of the agents, where an agent too strongly prefers certain actions because it has been repeatedly taught to do so, resulting in poor generalization for the agent.

An important point to note about exchanging experiences as advice is that if there is any heterogeneity between agents, such as robots with different capabilities, the rewards and state transitions shared in the experience may not be attainable for the advisee. Without some method of determining if an adviser is relevant, this can potentially cause the advisee to learn an incorrect policy through harmful over-training.

The third form of advice is a recommendation about which action to perform. Advice in the form of action recommendations can be generated either by another agent or by human beings, and can be treated as a binary decision, i.e., accept or reject, or as a suggestion that acts to bias the agent's own decision.

The effect of biasing an agent's decision via a reward signal is analyzed in [34], where in each state an agent can receive an additional reward from the adviser. As identified in [34], biasing the reward signal provides feedback only after the action is performed, making the expert's advice not immediately available when it is first needed. However, it is shown that if advice can be provided immediately when it is needed, then the search time for an optimal policy will be independent of the state space size, and only dependent on the length of the optimal solution (identical to the use of experiences.) One disadvantage of supplementing the reward signal as a form of action recommendation is the possibility for an agent to learn the incorrect policy. Similar to the use of experiences as advice, this is a problem that can arise with the use of heterogeneous agents.

In the Reinforcement and Advice-Taking Learning Environment (RATLE) mechanism [15], human observers are used to generate action suggestions and decide when they should be used. Advice is formed as IF-THEN or WHEN-REPEAT-UNTIL statements, which are then incorporated as inputs into a connectionist Q-learning agent. By incorporating advice directly into the agent's neural network, it can be further processed and

potentially overruled by the agent's own policy. When compared to an agent that blindly obeys the same advice, the use of the RATLE mechanism results in faster learning. An important result from this work is that it demonstrates that the capability to further process advice can be beneficial.

The Skill Advice Guided Exploration (SAGE) mechanism [16] uses action suggestions from multiple advisers to bias the agent's policy. An agent selects an action based on its own action selection probability plus a weighted combination of all advisers' action selection probabilities. The weighting is determined by a shaping function that depends on the difference between the advisee's current state's utility and that of the adviser, as well as the frequency that each action is advised in the current state. Such approach is attractive, because all advisers are always utilized, and their influence can vary depending on the weighting attributed to them. The SAGE mechanism has the important ability to reject bad advice, i.e., an action suggestion with repeatedly low utility values, by attributing a very low weighting to it. However, the ability to reject advice happens on a per state basis. If the same adviser continues to provide poor advice they are not rejected entirely, but rather only in particular states which bad advice is repeatedly provided. This means that areas of the state space where an adviser has little experience can be identified, but outright rejection of an advisor will be very slow. Further, the benefit of using all available advisers comes with high computational and communication costs.

Building upon [15], Preference Knowledge-Based Kernel Regression (Pref-KBKR) provides another approach to using humans for biasing an agent's decision [14]. The advice is a preference for one action over another in a particular state, implying that the value of actions should have a specific hierarchy. Similarly to RATLE and SAGE, Pref-KBKR provides the possibility for an agent to reject the advice, if they strongly disagree with it, by expressing advice as a preference. The advice is pre-generated by a human, and takes the form of a series of IF-THEN rules, similar to the RATLE mechanism. Experiments were performed using agents foraging for food while avoiding enemies. Different pieces of advice were tested, and it was found that for all variations of advice the agents with advice outperformed those without.

An alternative to biasing an agent's decision is to provide an agent access to the policy of an adviser. Advice Exchange [21] is a mechanism that uses other agents as advisers, where each agent temporarily uses the policy of their adviser to select the current action. In the beginning of each new learning epoch, the adviser of the system is selected to be the best performing agent (determined by the average utility value of selected actions.) At each instant agents decide to accept or reject the advice by comparing their average

utility and the utility of each possible action to the adviser's. Advice Exchange along with SAGE are the only mechanisms that actively select their advisers. A limitation of Advice Exchange is that the choice of adviser only changes once per learning epoch. Additionally, since adviser selection is only dependent on the performance, it is possible for an irrelevant, yet well performing, advisor to be selected. The possibility of irrelevant adviser selection, combined with infrequent adviser selection, is likely to be problematic for agents with heterogeneous capabilities. Improvements to Advice Exchange have been made by incorporating a form of memory [22], but at the cost of additional rules and parameters for the mechanism.

Chapter 2

Integration of RCISL into the HAA Architecture

This chapter, and the following one (Chapter 3), focus on the integration of RCISL into the HAA architecture. The motivation for doing so is to create a single framework for robot teams that possesses each of the following capabilities:

- Robustness to uncertain and unpredictable situations
- Fault-tolerance for hardware and software failure
- Performance enhancement through learning
- Heterogeneity
- Modularity
- Scalability

2.1 The Host-Avatar-Agent Architecture

HAA [17, 18] is a modular hierarchical architecture consisting of three main components: *Hosts*, *Avatars*, and *Agents*. *Hosts* are the processors which perform all the calculations of the system, and may be located in either mobile or stationary hardware. *Avatars* are mobile platforms, equipped with sensors and manipulators for interacting with their environments. The architecture allows for diversity in terms of avatar capabilities, provided that a proper interface is used to communicate with the hardware. Finally, *Agents* are the software modules that control the system, residing in hosts. Each agent is responsible

for a specific task, and is represented by a set of state data, functions for manipulating that data, and functions for communicating with other agents in the system. New agents and tasks can be defined as required, forming a highly modular system. The separation of software and hardware additionally increases scalability, as agents are dynamically instantiated and removed, allowing for easy addition and removal of hardware.

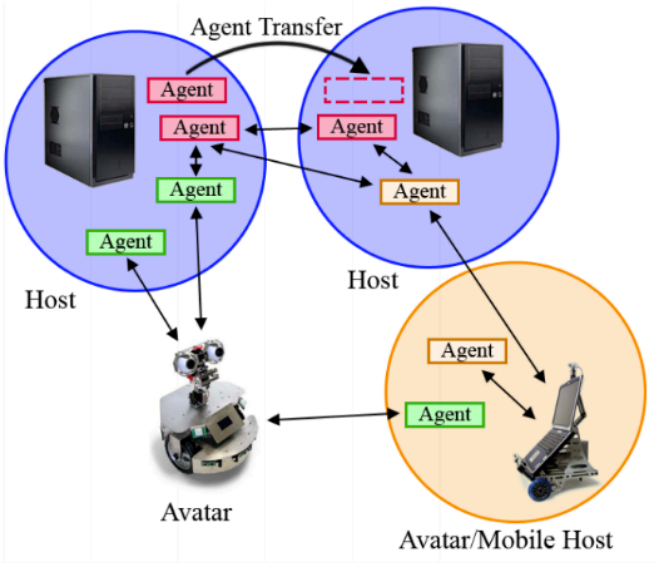


Figure 2.1: An example of agents residing on different hosts, and being transferred between hosts, in the HAA architecture.

Another main feature of the architecture is data and processing distribution [17], an illustration of which is shown in fig. 2.1. A distributed processing network allows agents to migrate freely between hosts through a freeze-thaw process, while a distributed database (DDB) provides storage of state data for all agents. This enables optimization and load balancing, as computationally intensive tasks can be migrated to hosts with spare capacity. The ability to conserve power on certain mobile hosts by reallocating processing tasks is another benefit. Fault tolerance is achieved through the sharing of state data via the DDB, as only local data which has not yet been distributed will be lost in case of hardware or software failures. In terms of a control system, the architecture offers a high degree of flexibility. Due to its modular nature, the degree of centralization can be freely chosen, as well as the level of agent interaction, all while benefiting from the fault tolerance inherent in the architecture. Scalable Hierarchical Control (SHC) is a control scheme that is implemented in the HAA architecture [17], and separates agents into the following different levels:

1. *Executive*: Handles overall task allocation, planning and scheduling, as well as

resource allocation and requests received from the Supervisor level.

2. *Supervisor*: Responsible for the completion of specific tasks allocated to it by the Executive level. Each Supervisor agent can either complete an assigned task directly, coordinate Avatar level agents to handle the task, or spawn additional Supervisor agents and divide the task. Supervisor agents are also responsible for requesting Avatar resources from the Executive level.

3. *Avatar*: Acts as an interface between the control system and the actual physical avatars. Each mobile platform is assigned an Avatar agent, that translates the functionality of the hardware into an internal representation which is then exposed to the system. Avatars can be assigned tasks matching their capabilities without the rest of the system knowing the specifics of how the capabilities are implemented, thus facilitating modularity.

Examples of agents in each level can be found in [17]. A schematic overview of the hierarchical structure with typical agents included is shown in fig. 2.2.

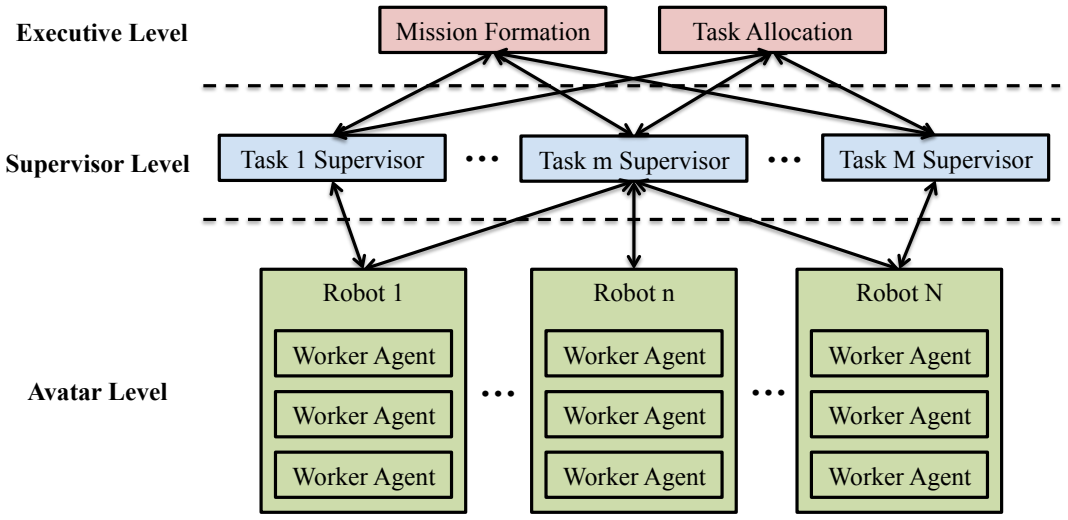


Figure 2.2: HAA control hierarchy displaying the executive, supervisor, and avatar levels of control.

2.2 Robust Concurrent Individual and Social Learning

RCISL [6] provides a method of efficiently implementing collective, collaborative, and cooperative behaviour within a robot team. Individual learning (collective behaviour)

and team learning (cooperative behaviour) are partitioned into two separate concurrent Markov Decision Processes (MDPs) [5]. The individual learning MDP, defined by the tuple $\langle S_I, A_I, T_I, R_I \rangle$, develops a policy for each robot to perform tasks, while the team learning MDP, defined by the tuple $\langle S_T, A_T, T_T, R_T \rangle$, develops a policy for allocating tasks to team members. The separation into concurrent MDPs prevents the *curse of dimensionality* as team members are added, often limiting robot teams.

The collective behaviour is achieved through individual learning, performed with a Q-learning algorithm [33] using the update rule in eq. 1.4. Each robot develops an individual behaviour policy π_I for performing their assigned task. Actions are selected from the policy π_I in a probabilistic fashion via a Boltzmann distribution (eq. 1.6).

Collaborative behaviour is achieved through the Advice Exchange algorithm [21, 5], which enables robot i to temporarily replace their own policy $\pi^i_{I,t}$ with the policy $\pi^k_{I,t}$ of advisor k for time step t. The following equation provides the conditions that must be met in order for the advised policy to be used:

$$\bar{\pi}^i_{I,t} = \begin{cases} & \text{IF } k = \underset{z \in Z^i}{\operatorname{argmax}} \left(\bar{q}^z_h \right), \text{ AND} \\ & \hat{q}^i_h < \hat{q}^k_h, \text{ AND} \\ \pi^k_{I,t} \quad , & \bar{q}^i_{M,h} < \bar{q}^k_h - \delta \left| \bar{q}^k_h \right|, \text{ AND} \\ & \sum_{a_t} Q^i \left(s^i_t, a \right) < \rho \sum_{a_t} Q^k \left(s^i_t, a \right) \\ \pi^i_{I,t} \quad & \text{, otherwise} \end{cases} \tag{2.1}$$

Where Z^i is the set of permissible advisers for robot i, $\bar{q}$ is the average quality of selected actions for the current epoch, $\hat{q}$ is the best average quality over all epochs, and h is the current epoch. $\bar{q}_{M,h}$ is the average quality for epoch h up to time step M, which is calculated with a simple moving average. Both $\hat{q}_h$ and $\bar{q}_h$ are updated with an exponential moving average at the end of each epoch. Since Advice Exchange directly adopts the policy of the adviser, the set of permissible advisers Z^i must be defined *a priori* such that the advisee will not adopt an inappropriate policy (i.e., a policy for a robot with substantially different capabilities).

Cooperative behaviour through team learning is achieved with the RL-Alliance algorithm [5], which is a modification of the L-Alliance algorithm [23]. RL-Alliance finds a policy, π^i_T, that chooses a task selection action, $(a \in A_T)$, for every team member i. Each team member has two metrics associated with it: impatience, and motivation. Impatience represents a team member's belief about their ability to perform each task, and is a function of their expected time to complete that task. Motivation represents a

team members desire to perform each task, and grows at each iteration as a function of that team member's impatience towards the task. During each task allocation round, every task is greedily allocated to the available team member with the highest motivation. Alternatively, tasks can be acquiesced by team members if they have had a sufficiently long attempt at the task and another available team member is expected to be able to complete the task quicker. The motivation for robot i to attempt task j is updated as,

$$m_t(i,j) = \left[m_{t-1}(i,j) + p_t(i,j)\right] g_t(i,j) \tag{2.2}$$

where $p_t(i,j)$ is the impatience robot i has towards attempting task j, and $g_t(i,j)$ is a gating multiplier ensuring robot i does not have any other tasks assigned, task j is incomplete, and no other robot is expected to complete task j quicker than robot i. The impatience factor is given by,

$$p_t(i,j) = \frac{\theta_1}{\tau_t(i,j)} \tag{2.3}$$

where θ_1 is a constant, and $\tau_t(i,j)$ is that average trial time for robot i to complete task j. The average trial time is updated according to,

$$\tau_t(i,j) = \beta(i,j)\left(\tau_{t-1}(i,j) + \frac{\theta_2}{f_{ij}}\left(I_{ij} - \tau_{t-1}\left(i,j\right)\right)\right) \tag{2.4}$$

where the parameter θ_2 denotes a constant learning rate, f_{ij} denotes the number of attempts member i has had towards task j, while I_{ij} denotes the time taken during the latest attempt by team member i on task j. Additionally, $\beta(i,j) = e^{(f_{ij}/\theta_4)}/(\theta_3 + e^{(f_{ij}/\theta_4)})$ is a *softmax* expression with constants θ_3 and θ_4 enabling a team member to make many attempts (for adequate training) towards one task. The above formulation for average trial time enables a pareto-optimal task assignment to be achieved.

Finally, robustness is achieved via a particle filter for robot state estimation. The particle filter estimates position and orientation information about the robot's environment that is necessary to form the individual learning state s_I. The formulation for the particle filter can be found in [6]. Filtering the robot's state information enables the individual and team learning algorithms to maintain their performance in the presence of noisy sensory data.

2.3 Integration

The primary tasks for integrating collective, collaborative, and cooperative behaviour into the HAA architecture are: determining the necessary information to be preserved at all times that will enable an agent to completely recover, and ensuring that all communication and dependencies for the newly created agents have the appropriate fall-back behaviour.

2.3.1 Collective Behaviour

Individual learning is implemented within the HAA architecture as a single agent (following the HAA terminology for agents) called *AgentIndividualLearning*, with one instance spawned by the Task Supervisor agent for each existing avatar. Although AgentIndividualLearning is unique for each avatar, the distributed processing capabilities of HAA enable each agent to reside on the hardware of any available host. AgentIndividualLearning makes requests to the DDB or other agents to obtain state information (e.g., the ExecutiveSimulation agent in [17]) for the Q-learning algorithm, while the controller responsible for issuing commands to the avatar requests movements from the individual learning agent (e.g., the AvatarSimulation agent in [17]). Hence, AgentIndividualLearning maintains a modular form, enabling it to be altered or its functionality to be replaced with little modifications to dependent agents.

For individual learning the primary data to be retained at all times is the developed behaviour policy π_I, and state information for the Q-learning update in eq. 1.4. The data transfers necessary to ensure fault tolerance are summarized below in Table 2.1. Agent transfer refers to planned migration of agents between hosts, agent recovery refers to the recovery process after an unexpected failure, while DDB backup corresponds to the routine data backups. During agent transfer, data is transferred directly between hosts, during agent recovery data is retrieved from the DDB, and during DDB backups data is sent to the DDB.

Due to the size of the table containing the Q-values, from which the behavioural policy is derived, only the single updated Q-value is saved after each Q-learning update with eq. 1.4. However, during agent recovery it is necessary to retrieve the entire set of Q-values for all states and actions from the DDB.

Table 2.1: Summary of data transfer for the collective behaviour agent

	Data Transferred	**Frequency**
Agent Transfer	$s_{I,t}$ $s_{I,t-1}$ $Q_t(s_{I,t}, a_I), \quad \forall a_I \in A_I$ $v_t(s_{I,t}, a_I), \quad \forall a_I \in A_I$	At agent freeze request
Agent Recovery	$Q_t(s_I, a_I), \quad \forall s_I \in S_I, \forall a_I \in A_I$ $v_t(s_I, a_I), \quad \forall s_I \in S_I, \forall a_I \in A_I$	After agent failure
DDB Backup	$Q_{t+1}(s_{I,t}, a_{I,t})$	After Q-learning update

Where $v_t(s_I, a_I)$ is the number of times action a_I has been performed in state s_I at time t.

2.3.2 Collaborative Behaviour

Collaborative behaviour through advice is also implemented as a single agent, called *AgentAdviceExchange*, for each avatar in the HAA architecture, and is spawned by the Task Supervisor agent. AgentAdviceExchange receives requests from the avatar's AgentIndividualLearning for advice, and issues requests to the appropriate AgentIndividualLearning of the adviser avatar (provided the conditions in eq. 2.1 are satisfied). Each AgentAdviceExchange routinely uploads its performance metrics (i.e., $\hat{q}$ and $\bar{q}$) to the DDB, and retrieves the performance metrics for all other avatars at the start of a new epoch. Additionally, to accommodate failures of other agents, each advisor AgentIndividualLearning must respond to the advice request within a predefined time period, beyond which they are assumed to have failed and will be ignored until the next DDB update.

The state information for the Advice Exchange algorithm is saved in the HAA architecture as outlined below in Table 2.2. Identical to Table 2.1, agent transfer refers to planned migration of agents between hosts, agent recovery refers to the recovery process after an unexpected failure, and DDB backup corresponds to the routine data backups. Given the small amount of data associated with AgentAdviceExchange, DDB backups are not event driven, but rather are performed on a routine schedule (e.g. every 100 time steps).

Table 2.2: Summary of data transfer for the collaborative behaviour agent

	Data Transferred	**Frequency**
Agent Transfer	Adviser Id $\bar{q}_h$ $\bar{q}_{M,h}$ $\hat{q}_h$	At agent freeze request
Agent Recovery	$\bar{q}_h$ $\bar{q}_{M,h}$ $\hat{q}_h$	After agent failure
DDB Backup	$\bar{q}_h$ $\bar{q}_{M,h}$ $\hat{q}_h$	Pre defined intervals

2.3.3 Cooperative Behaviour

The RL-Alliance algorithm for team learning is implemented within the HAA architecture in a distributed fashion, where one AgentTeamLearning will be spawned by the Task Supervisor agent for each existing avatar. Hence, all team learning agents are responsible for negotiating with each other while allocating tasks. Decentralizing task allocation prevents a single failure from disrupting all avatars by losing task information, and enables each avatar to determine its own task independent of other agent failures. The negotiation process is a crucial component in a distributed system, as it is responsible for resuming normal operation regardless of any party's failure. Task allocation is performed in synchronized ordered rounds, where the order of agent participation is determined by the round coordinator. Before the start of each new round, the round start time and order is distributed amongst the participating agents by the last agent to participate in the preceding round. Each agent is allotted a predefined amount of time to participate in the round, beyond which they are assumed to have failed. The negotiation process is outlined in Algorithm 1.

In the event that the last agent in the current round fails before or during the coordination for the next round, it will still be responsible for coordinating the round when it recovers on another host. The round information is routinely saved in the DDB, such that during recovery the failed round coordinator will receive the last round order, indicating that it is responsible for coordinating the next round. Algorithm 1 will then default to the timeouts for participating agents being surpassed, and the round coordination will proceed as intended. Additionally, since the RL-Alliance algorithm relies on the perfor-

Algorithm 1 Task Allocation Round Negotiation

```
1: if t > round_start_time and !participated then
2:     n ← Number of agents ahead on list
3:     if n == 0 or t > round_start_time + n * timeout then
4:         Perform RL-Alliance algorithm
5:         Upload data to DDB
6:         participated ← true
7:         if last_agent then
8:             Randomize agent list
9:             Broadcast new list
10:        end if
11:    end if
12: end if
```

mance metrics of all other robots, each AgentTeamLearning routinely updates the DDB with it's own metrics. Performance metrics for other agents are then received in DDB notifications. Therefore, during each task allocation round, the RL-Alliance algorithm will operate based on the most recent data received from the DDB for all other agents. The data transfers necessary to ensure fault tolerance for agent data and negotiations are summarized below in Table 2.3

Table 2.3: Summary of data transfer for the cooperative behaviour agent

	Data Transferred	**Frequency**
Agent Transfer	Id of each AgentTeamLearning Completion for each task Negotiation round number Negotiation round start time Negotiation round order Current task Id Current task state T_i*, $\forall i \in I$	At agent freeze request
Agent Recovery	Id of each AgentTeamLearning Negotiation round number Negotiation round start time Negotiation round order Current task Id Current task state T_i*, $\forall i \in I$	At agent freeze request
DDB Backup	Negotiation round number Negotiation round start time Negotiation round order Current task Id Current task state T_i*, i = Agent Id	After each negotiation round

* T_i is the set of all data for the RL-Alliance algorithm for the robot i, given as $T_i = \{\tau(i,j), \bar{\tau}(i,j), \sigma_\tau(i,j), f(i,j), m(i,j), p(i,j), \nu(i,j) | j \in J\}$, where $\bar{\tau}(i,j)$ and $\sigma_\tau(i,j)$ are the mean and standard deviation of task completion times for robot i attempting task j, and $\nu(i,j)$ is the time spent on the current task.

Chapter 3

Preference Advice

In this section the formulation for the Preference Advice Mechanism is presented. The motivation for developing the presented mechanism stems from a desire to develop a mechanism more suitable for robot teams that would have minimal restrictions on the advisers and the use of advice. Hence, we desire an advice mechanism with the following capabilities:

1. does not require advisers with full knowledge of the task;
2. is compatible with advisers of varying skill levels and relevance;
3. guarantees convergence to an optimal policy;
4. diminishes the influence and usage of advice over time.

The first capability, to not require advisers with full knowledge of the task, emphasizes the need for utilizing partial knowledge from other agents. During the initial stages of learning, each agent's knowledge of the task will be sparse, but the team collectively may have near complete knowledge. The second capability of compatibility with advisers of varying skill levels and relevance is essential for heterogeneous agents. As the robots' capabilities become more diverse, it becomes more important for an advice mechanism to identify the usefulness of each adviser on its own. The third capability of convergence to an optimal policy not only provides a theoretically sound mechanism, but is of great importance to heterogeneous robot teams. When discrepancy exists between robots, there is a risk that an adviser may guide an agent towards a different policy. Finally, the fourth capability for the influence and usage of advice to diminish over time serves to make the mechanism practical for real world applications, where each use of advice incurs a communication and computational cost.

The Preference Advice mechanism will be presented as follows: first, a method for characterizing the information possessed by an agent about a state, as well as incorporating information from an adviser, i.e. advice, into the advisee, will be shown. This will be followed by demonstrating that a reinforcement learning agent incorporating the advice of an adviser in the method shown will converge to a stationary and optimal policy. Finally, a formulation for an MDP will be presented that determines when an agent needs advice, and which advisers it should use.

3.1 Incorporating Advice

Since the purpose of a reinforcement learning agent is to develop an optimal policy by learning the value of each state and action, how knowledgeable that agent is about a state should be dependent on these values. However, instead of directly using utility values which can vary greatly in magnitude between applications, it is advantageous to use the probability values for selecting each action, which are bounded on the interval [0,1]. To assess how knowledgeable an agent is about a state via action selection probability values, it is assumed that a high certainty about which action(s) to select can be directly related to a high level of knowledge in the given state. Therefore, the greater the deviation from equal probability for all actions, the more knowledgeable an agent is likely to be.

To convert state-action values, $Q(s,a)$, to probability values for each action, $p(a_i)$, a Boltzmann distribution (eq. 1.6) can be used. In the context of action selection probabilities, an agent can be said to have zero knowledge about its current state when no single action is considered to be more valuable than any other. If there are n possible actions, then this corresponds to each action having an equal probability value given by

$$p(a_i) = \epsilon = \frac{1}{n}; \qquad i = 1, ..., n \tag{3.1}$$

Conversely, an agent can be said to have the maximum possible amount of knowledge about its current state when all probability is attributed to a single action, represented by

$$p(a_i) = 1, \quad \text{and} \quad p(a_j) = 0 \quad \forall i \neq j \tag{3.2}$$

Let $d(a_i) = p(a_i) - \epsilon$ define the distance between selection probability of an action and the base value. The preference level, $k(a_i)$ of action a_i is defined to be proportional to the square of the distance $d(a_i)$, as given by

$$k(a_i) = \text{sign}(d(a_i))d(a_i)^2, \qquad i = 1, ..., n \tag{3.3}$$

Therefore, the set $\boldsymbol{K} = \{k_1, ..., k_n\}$ represents the preference levels for the actions in a given state, where the magnitude of each element represents the magnitude of preference for each action, and the sign represents a "preferred to" ($k > 0$) or "preferred not to" ($k < 0$) action.

Let $\boldsymbol{K}_o$ denote an agent's initial set of preference levels for the actions in a state, and let $\boldsymbol{K}_m$ denote the preference levels of adviser m in the same state. For each action a, the advised preference level is a linear combination of the advisee and adviser preference levels, as given by

$$\hat{k}(a_i) = k_o(a_i) + \lambda_t(s)k_m(a_i); \qquad i = 1, ..., n \tag{3.4}$$

where $\lambda_t(s, m)$ controls the influence of advice from adviser m, and will be derived in section 3.2. The advised action selection probabilities, $\hat{p}(a_i)$, are then obtained by

$$\hat{p}(a_i) = \frac{\epsilon + \text{sign}(\hat{k}(a_i))\sqrt{|\hat{k}(a_i)|}}{\sum_{j=1}^{n} \hat{p}(a_j)}; \qquad i = 1, ..., n \tag{3.5}$$

Eq. 3.5 ensures that all selection probabilities will form a valid distribution. The advised policy then selects actions based on the advised selection probabilities $\hat{p}(a_i)$.

3.2 Advised Policy Convergence

Since the use of advice will alter the selection probabilities for each action, and hence the agent's policy, this can potentially prevent an agent from learning the true value of each $Q(s, a)$ and developing an optimal policy. Given the variety of methods available for updating the values of $Q(s, a)$, often having different convergence requirements, we will demonstrate convergence with an advised policy for two commonly used methods: Q-learning, and SARSA(0). Both methods update one $Q(s, a)$ at a time, however the Q-learning update (eq. 1.4) does not depend on the agent's current action selection policy, while the SARSA(0) update (eq. 1.5) does. Consequently, the SARSA(0) method has more strict convergence requirements.

As detailed in [30], the convergence of SARSA(0) can be proven by treating the update as a stochastic process described by Theorem 1 of [8]. The SARSA(0) method is shown to converge to the true values of $Q(s, a)$ when the policy for selecting actions guarantees infinite exploration (i.e., each action in every state is experienced infinitely

many times,) and that at time infinity the policy becomes greedy. Naturally, the policy at time infinity will be optimal given that the true values of $Q(s,a)$ have been reached and actions are greedily selected. The convergence of Q-learning only requires infinite exploration to reach the true values of $Q(s,a)$; however, the policy must also become greedy at time infinity if an optimal policy is to be reached.

Thus, in order for Q-learning and SARSA(0) to converge using an advised policy two requirements must be met. First, the advised policy must guarantee infinite exploration, and become greedy at time infinity. Secondly, the influence of advice must diminish to zero at time infinity, enabling the agent to converge to a stationary policy. The fulfillment of these two requirements will guarantee convergence to an optimal policy in the presence of advice.

Lemma 3.2.1. *Let D_i be an increasing sequence of σ-fields, and let A_i be D_i measurable. Then the following holds with probability 1,*

$$\left\{\omega : \sum_{i=1}^{\infty} P(A_i|D_{i-1}) = \infty\right\} = \{\omega : \omega \in A_i \quad i.o.\} \tag{3.6}$$

Lemma 3.2.2. *Consider a communicating MDP, and the reinforced decision process,*

$$(x_1, a_1, r_1, ..., x_t, a_t, r_t, ...) \tag{3.7}$$

Let $v_t(s)$ denote the number of visits to state s up to time t, $v_t(s,a)$ denote the number of times action a has been chosen in state s during the first t time steps, and $t_s(i)$ denote the time when state s was visited the i^{th} time. Assume that the action at time t, a_t, is selected purely on the basis of the statistics D_t:

$$P(a_t = a|D_t, a_{t-1}, D_{t-1}, ..., a_1, D_1) = P(a_t = a|D_t) \tag{3.8}$$

where D_t is computed from the full history $(x_0, a_0, r_0, ..., x_t)$. Further, assume that the action selection policy is such that

$$\left\{\omega : \lim_{t\to\infty} v_t(s)(\omega) = \infty\right\} \subseteq \left\{\omega : \sum_{i=1}^{\infty} P(a_{t,i} = a|D_{t,i})(\omega) = \infty\right\} \tag{3.9}$$

Then, for all (s,a) pairs, $v_t(s) \to \infty$, and $v_t(s,a) \to \infty$, with probability 1.

Lemma 3.2.3. *Consider an agent with a policy π_o given as a set of probabilities $P(a|s,t,Q)$ determined from a Boltzmann distribution (eq. 1.6). If the temperature parameter $\tau_t(s)$ is defined as,*

$$\tau_t(s) = \frac{ln(v_t(s))}{\max_{a \in A} |Q_t(s, a_{max}) - Q_t(s, a)|} \tag{3.10}$$

where $a_{max} = \text{argmax}_{a \in A} Q_t(s, a)$*, then policy* π_o *guarantees each action* a *in every state* s *is experienced infinitely often, and becomes greedy at time infinity.*

The proofs for Lemmas 3.2.1, 3.2.2 and 3.2.3 can be found in [30]. Lemmas 3.2.1 and 3.2.2 show that if the sum of selection probabilities for each action is infinite, then each action a in every state s will be experienced an infinite number of times (i.e., infinite exploration is achieved.) Lemma 3.2.3 provides a policy that achieves infinite exploration and becomes greedy at time infinity. Therefore, to achieve infinite exploration under an advised policy we wish to show that $\sum_{i=1}^{\infty} P(a|s, t_s(i)) = \infty$, where $t_s(i)$ is the time of the i^{th} visit to state s.

Theorem 3.2.4. *Let an agent determine its initial action selection probabilities,* $p_o(a_i)$*, via a method which ensures infinite exploration and becomes greedy at time infinity, such as Boltzmann exploration (Lemma 3.2.3). Let the advised policy* $\hat{\pi}$ *be obtained by incorporating advice via eq. 3.4 and having* $\lambda_t(s)$ *defined as,*

$$\lambda_t(s) = \left(\frac{\epsilon}{v_t(s)} - 1 \right)^2 + \frac{k_o(a_{min})}{\epsilon^2} \tag{3.11}$$

where $a_{min} = \text{argmin}_{a \in A} k_o(a)$*. Then, the advised policy* $\hat{\pi}$ *guarantees each action* a *in every state* s *is experienced infinitely often, and becomes greedy at time infinity. Additionally,* $\lambda_t(s) \to 0$ *as* $t \to \infty$*. Therefore, an advised agent using the Q-learning or SARSA(0) update method will converge to a stationary and optimal policy at time infinity.*

Proof. The influence of advice will alter the initial action selection probabilities, but it must not prevent infinite exploration. Utilizing lemmas 3.2.1 and 3.2.2, and the knowledge that $\sum_{i=1}^{\infty} c/i = \infty$, where c is a constant, the requirement of infinite exploration can be fulfilled if the following condition holds for the advised policy $\hat{\pi}$ with the action selection probabilities defined in eq. 3.5:

$$\hat{p}(a_i) \geq \frac{c}{v_t(s)}, \qquad i = 1, ..., n \tag{3.12}$$

The advised action selection probability $\hat{p}(a_i)$ will depend on the influence of the adviser in eq. 3.5, governed by $\lambda_t(s)$. Additionally, since we are concerned with maintaining a minimum action selection probability across all actions $a \in A$, then $\lambda_t(s)$ will be limited by the action that reaches the minimum selection probability given by the condition in eq. 3.12 with the least influence from adviser. This will occur when the actions with the

lowest preference level for the advisee and adviser are the same. Let this limiting action be denoted by a_{min}. An expression for $\lambda_t(s)$ which satisfies eq. 3.12 can be found by relating $\hat{p}(a_{min})$ to the advised preference level $\hat{k}(a_{min})$,

$$\begin{aligned}
\hat{p}(a_{min}) &\geq \frac{c}{v_t(s)} \\
\left[-\hat{k}(a_{min})\right]^{1/2} + \epsilon &\geq \frac{c}{v_t(s)} \\
\left[-k_o(a_{min}) - \lambda_t(s) k_m(a_{min})\right]^{1/2} + \epsilon &\geq \frac{c}{v_t(s)} \\
\frac{-k_o(a_{min}) - \left(\epsilon^2/v_t(s) - \epsilon\right)^2}{k_m(a_{min})} &\geq \lambda_t(s)
\end{aligned}$$

where the sign of $\hat{k}(a_{min})$ in the square root is set to negative, because $\hat{p}(a_{min})$ always results in a negative preference (prefer not to), and the constant c is selected to be equal to ϵ^2. We set the adviser preference $k_m(a_{min})$ to its lowest limit (strongly prefer not to) of $-\epsilon^2$, determined by setting $p(a_i) = 0$ in eq. 3.3, to come up with a conservative upper bound for $\lambda_t(s)$. This indeed corresponds to the adviser's agreement with the advisee on the least preferred action, resulting in a level of influence for which anything below will guarantee condition 3.12. Hence, we can express $\lambda_t(s)$ as,

$$\lambda_t(s) \leq \left(\frac{\epsilon}{v_t(s)} - 1\right)^2 + \frac{k_o(a_{min})}{\epsilon^2}$$

Therefore, by choosing $\lambda_t(s)$ as defined in eq. 3.11, each action a in every state s will be experienced infinitely often. Further, the influence of advice diminishes to zero, since for every state s, $\lim_{t\to\infty} v_t(s) = \infty$, resulting in $\lim_{t\to\infty} \lambda_t(s) = 0$. Since at time infinity the initial agent's action selection policy becomes greedy, and the influence of advice diminishes to zero, then the advised policy will also become greedy at time infinity. Thus, the convergence requirements for Q-learning and SARSA(0) have been met, and the advised agent will converge to a stationary and optimal policy. □

It can easily be verified that the advised preference levels produced by incorporating an adviser's advice via eq. 3.4 with $\lambda_t(s)$ defined in eq. 3.11, will always be within the bounds determined by eq. 3.3.

Lemma 3.2.5. *With $\lambda_t(s)$ defined by eq. 3.11, it will always be true that,*

$$-\epsilon^2 \leq \hat{k}(a_i) \leq (1-\epsilon)^2 \qquad i = 1, ..., n$$

Proof. The lower and upper bounds of $\hat{k}(a_i)$ are $-\epsilon^2$, and $(1-\epsilon)^2$, as determined by eq.

3.3 for $p(a_i) = 0$ and $p(a_i) = 1$, respectively. The proof that $-\epsilon^2 \leq \hat{k}(a_i)$ follows directly from theorem 3.2.4, since $\lambda_t(s)$ was derived such that $\hat{p}(a_i) \geq \epsilon^2/v_t(s) \quad i = 1, ..., n$, for any values of $k_m(a_i)$. For the largest advised preference level we consider the case that maximizes the increase in preference level due to advice. This occurs when the preference levels for each action of the advisee and adviser have the same sign, and when $n-1$ actions have equal preference levels of $k(\hat{a}_{min}) < 0$, and a single action has the preference level of $k(\hat{a}_{max})$, where $\hat{a}_{min} = \text{argmin}_{a \in A} \hat{k}(a)$ and $\hat{a}_{max} = \text{argmax}_{a \in A} \hat{k}(a)$, respectively. In this case, the advisee's preference level for action $\hat{a}_{max}$ can be expressed as $k_o(\hat{a}_{max}) = -(1/\epsilon - 1)^2 k_o(\hat{a}_{min})$, where $k_o(\hat{a}_{min})$ can be set to $-(\epsilon^2/v_t(s) - \epsilon)^2$, which comes from setting $p_o(\hat{a}_{min}) = \epsilon^2/v_t(s)$. Letting the adviser's preference be set to the maximum value of $(1-\epsilon)^2$, we can compare $\hat{k}(\hat{a}_{max})$ to the upper limit,

$$
\begin{aligned}
&(1-\epsilon)^2 \geq \hat{k}(\hat{a}_{max}) \\
&(1-\epsilon)^2 \geq k_o(\hat{a}_{max}) + \lambda_t(s)(1-\epsilon)^2 \\
&(1-\epsilon)^2 \geq -(1/\epsilon - 1)^2 k_o(\hat{a}_{min}) + \left(\left(\frac{\epsilon}{v_t(s)} - 1 \right)^2 + \frac{k_o(a_{min})}{\epsilon^2} \right) (1-\epsilon)^2 \\
&(1-\epsilon)^2 \geq (1/\epsilon - 1)^2 (\epsilon^2/v_t(s) - \epsilon)^2 + \left(\left(\frac{\epsilon}{v_t(s)} - 1 \right)^2 - \frac{(\epsilon^2/v_t(s) - \epsilon)^2}{\epsilon^2} \right) (1-\epsilon)^2 \\
&(1-\epsilon)^2 \geq (1-\epsilon)^2 (\epsilon/v_t(s) - 1)^2
\end{aligned}
$$

which will be true for any $v_t(s) \geq 1$ (and $n > 1$). □

An interesting property of the proposed method of incorporating advice is that the convergence of the advised policy does not depend on the adviser. Infinite exploration and greedy action selection can be achieved for any advice. A poor adviser may guide an agent towards imperfect actions and slow the rate at which the agent learns the task, but it will not prevent convergence to an optimal policy. This property is particularly useful for scenarios with multiple advisers in heterogeneous teams.

3.3 Determining When To Use Advice

With regards to utilizing the advice of an adviser, it is of course possible to use several advisers at each time step. However, each time advice is requested from an adviser, a communication cost is incurred. Further, the benefit of advice will diminish over time as the advisee learns its own task, which should be reflected in how frequently advice is

requested over time. Thus, polling every adviser for its advice, or polling a fixed number of advisers at each step, is not desirable, as it can result in receiving advice of little benefit and unnecessary communication costs. Our approach is to enable the advice mechanism to learn when it should accept or reject advice, and when it should continue seeking more advisers.

The advice utilization process, namely learning the relationship between an agent's own stage in improving its performance and the benefit of advice, can be represented by an MDP, and hence a reinforcement learning algorithm can be applied. At each time step an agent can decide if it needs advice, if it should accept the adviser's advice, ask another adviser, or cease asking additional advisers. Thus, the mechanism must learn when it is most appropriate to request advice, and when the advice should be accepted. The advice utilization process is defined by the $\langle S_A, A_A, T_A, R_A \rangle$ tuple:

State $s_A \in S_A$: is defined by $s_A = \{\psi_o, \bar{\psi}_m, \gamma\}$, which contains information in the agent's current state about i) the advisee's confidence level, ii) whether or not the adviser's confidence level is greater than the advisee's, and iii) the advisee's experience. The elements ψ and $\bar{\psi}_m$ are defined as,

$$\psi = \sqrt{\frac{1}{1-\epsilon}} \left[\sum_{i=1}^{n} \left(d(a_i) \right)^2 \right]^{\frac{1}{2}} \tag{3.13}$$

$$\bar{\psi}_m = \begin{cases} 1 & \text{, when } \psi_m > \psi_o \\ 0 & \text{, otherwise} \end{cases}$$

The values of ψ_m and ψ_o are obtained using the action selection probabilities of the adviser and advisee, respectively. The value of ψ provides a single metric about the agent's confidence level in the current state, which is bound on the interval $[0, 1]$ for any number of actions. In the zero confidence case, where the selection probabilities of all actions are equal, $\psi = 0$; whereas for the maximum confidence case, when all probability is attributed to a single action, $\psi = 1$. The agent's experience in the current state is represented by $\gamma = 1/v_t(s)$.

Action $a_A \in A_A$= {*accept*, *skip*, *cease*}, where *accept* incorporates the advice with Eq. 3.4 and asks the next adviser for advice, *skip* ignores the advice and asks the next adviser for advice, and *cease* rejects the advice and ceases asking for any more advice in the current state.

Transition $T_A(s'_A, a_A, s_A)$: represents the probability of transitioning from state s_A to state s'_A after performing action a_A.

Reward $R_A(s_A, a_A, s'_A)$: defines the reward received by the mechanism after each action, and is given by

$$R_A = \begin{cases} \left(1 + \delta\left(\hat{\psi} - \psi_o\right)\right)^2 & , a_A = accept \\ \left(1 + \psi_o\right)^2 & , \text{otherwise} \end{cases} \tag{3.14}$$

where $\hat{\psi}$ is found with eq. 3.13 by using the advised action selection probabilities from eq. 3.5. The coefficient δ is a constant positive value to balance accepting and rejecting advice, and can be adjusted depending on the application.

At the beginning of each time step, before an agent takes an action, the advice mechanism will first decide if advice should be requested at all. Since the mechanism's state s_A depends on the adviser's advice, and no advisers have yet been polled, this decision-making is achieved by selecting an action from the mechanism's policy as if there were an adviser available with stronger confidence than the advisee. Hence, the first state consists of $s_A = \{\psi_o, 1, \gamma\}$, which corresponds to the mechanism being optimistic that a suitable adviser is available. If the selected action is *cease*, then the agent does not request any advice at the current time step. Otherwise, an adviser is polled for its advice, a new state s_A is formed, and the mechanism chooses between *accept*, *skip*, and *cease*. When the advice is accepted, the advisee's preference levels are updated with eq. 3.4, and advice is requested from a new adviser (presuming one is available.) When the *skip* action is selected, the advice is ignored, and advice from a new adviser is requested. Finally, when the *cease* action is selected, the adviser's advice is not used, and the mechanism ceases requesting any additional advice for this time step. Hence, the mechanism continues requesting advice, either accepting or ignoring the advice, until either the *cease* action is selected or there are no more available advisers.

The order in which advisers are polled for their advice is not part of the mechanism's decision-making process. Instead, a ranking scheme is used to determine the order in which advisers are polled. The optimal adviser for an agent is one with identical capabilities and who has learned the values of $Q(s, a), \forall s, a$. If any heterogeneity exists between agents, their transition probabilities between states will differ, and hence the true values of $Q(s, a)$ for agents will not necessarily be equal for each action a in every state s. As long as the method for learning the values of $Q(s, a)$ implemented by the agent is guaranteed to converge to the true values of $Q(s, a), \forall s, a$, then at time infinity $Q_o(s, a) = Q_{m'}(s, a)$, and hence $k_o(a) = k_{m'}(a)$, where the o and m' subscripts denote the advisee and an optimal adviser, respectively. Therefore, selecting the best available adviser is equivalent to finding the adviser with the greatest similarity, using a similarity

measure which is maximized when $Q_o(s,a) = Q_{m'}(s,a)$. The dot product between the unit vectors of the advisee's and adviser's preference levels, given by eq. 3.15, is one such measure. Using the preference levels, as opposed to directly using the values of $Q(s,a)$ or $p(a)$, is advantageous since the values of $k(a)$ are signed, so the dot product can also indicate when two agents oppose each other. Equation 3.15 reaches a maximum of 1 only when the preference levels are equal, is zero when either agent has zero preference across all actions, and can be negative when the preference levels oppose each other. The similarity measure is updated as an exponential moving average each time the adviser is polled for its advice, given by eq. 3.16, where ρ is a constant decay rate. During each advice round, advisers are then selected in the order of most to least similar.

$$\beta_m = \left(\frac{\boldsymbol{K}_m}{||\boldsymbol{K}_m||}\right) \cdot \left(\frac{\boldsymbol{K}_o}{||\boldsymbol{K}_o||}\right) \tag{3.15}$$

$$\omega_{m,t+1} = \rho\omega_{m,t} + (1-\rho)\beta_m \tag{3.16}$$

Lastly, the policy for the advice mechanism, $\pi_A(s_A)$, is developed through a Q-learning algorithm with the update rule defined in eq. 1.4. The steps to be performed by the mechanism to update an agent's preference levels at each time step are summarized in Algorithm 2.

There are two additional points to note about the proposed mechanism. First, the mechanism has only two parameters: δ in eq. 3.14 and ρ in eq. 3.15. Secondly, the state space for the advice mechanism is small. With three possible actions, two state elements in the range [0,1], and one binary state element, the state space size is $6 \times u \times v$, where u and v are the number of discrete intervals for the state elements ψ and γ, respectively. A small state space is intentional, since advice is most beneficial in the early stages of learning, and the state space of the advice mechanism must be significantly smaller than the agent's reinforcement learning problem if a benefit is to be received in the early stages of learning.

Algorithm 2 Seek Advice

$\hat{\boldsymbol{K}} \leftarrow \boldsymbol{K}_o$
Set state s_A
$a_A \leftarrow \pi_A(s_A)$
if a_A ! = *cease* **then**
    Set adviser order
    Poll next adviser m
    Set state s_A
    while *next_adviser_available* **do**
        Update similarity measure
        $a_A \leftarrow \pi_A(s_A)$
        if $a_A ==$ *accept* **then**
            Update $\hat{\boldsymbol{K}}$ with eq. 3.4
        end if
        if $a_A ==$ *cease* **and** *next_adviser_available* **then**
            Poll next adviser m
        end if
        Set state s_A
        Perform Q-learning update with eq. 1.4
        if !*next_adviser_available* **or** $a_A ==$ *cease* **then**
            Break
        end if
        $\boldsymbol{K_o} \leftarrow \hat{\boldsymbol{K}}$
    end while
end if
return $\hat{\boldsymbol{K}}$

Chapter 4

Case Study 1: Preference Advice With Individual Learning

4.1 Simulation Scenario

To demonstrate the effectiveness of the Preference Advice mechanism, a case study with a heterogeneous robot team is used. The team is composed of robots with varying capabilities, where each robot is individually learning a task with a Q-learning algorithm, while also receiving advice from its peers or virtual advisers. Each robot has the same goal, however their capabilities differ, resulting in each type of robot learning a slightly different policy.

A foraging scenario is used where the goal of the team is to collect all of the items in the area and bring them to a target zone. Robots must learn to navigate the world effectively, to collect the items in as few actions as possible. An illustration of the scenario is show in fig. 4.1. Each robot is initially assigned an item to collect randomly. The total foraging area is 10m by 10m, with 4 obstacles each having a diameter of 1m, and a single target zone with a diameter of 2m. The items to be collected are 0.5m in diameter, and their quantity is equal to the number of robots used in each experiment. Additionally, there is a circular area 4m in diameter in the center of the foraging area that represents rough terrain, which only certain types of robots can pass through. The rough terrain is intended to mimic real applications of heterogeneous robot team, such as search and rescue, where environmental conditions may be more unfavorable for certain robots. Four different types of robots are used, differing in their speed of movement and their ability to traverse the rough terrain: S-NR, S-R, F-NR, and F-R, where S, F, NR, and R represent slow, fast, non-rugged and rugged, respectively. Only rugged robots can

traverse the rough terrain. Each run begins with the items, obstacles, target zone, and robots, randomly positioned within the foraging area, and ends when either all items have been returned to the target zone, or 4000 iteration steps have been performed.

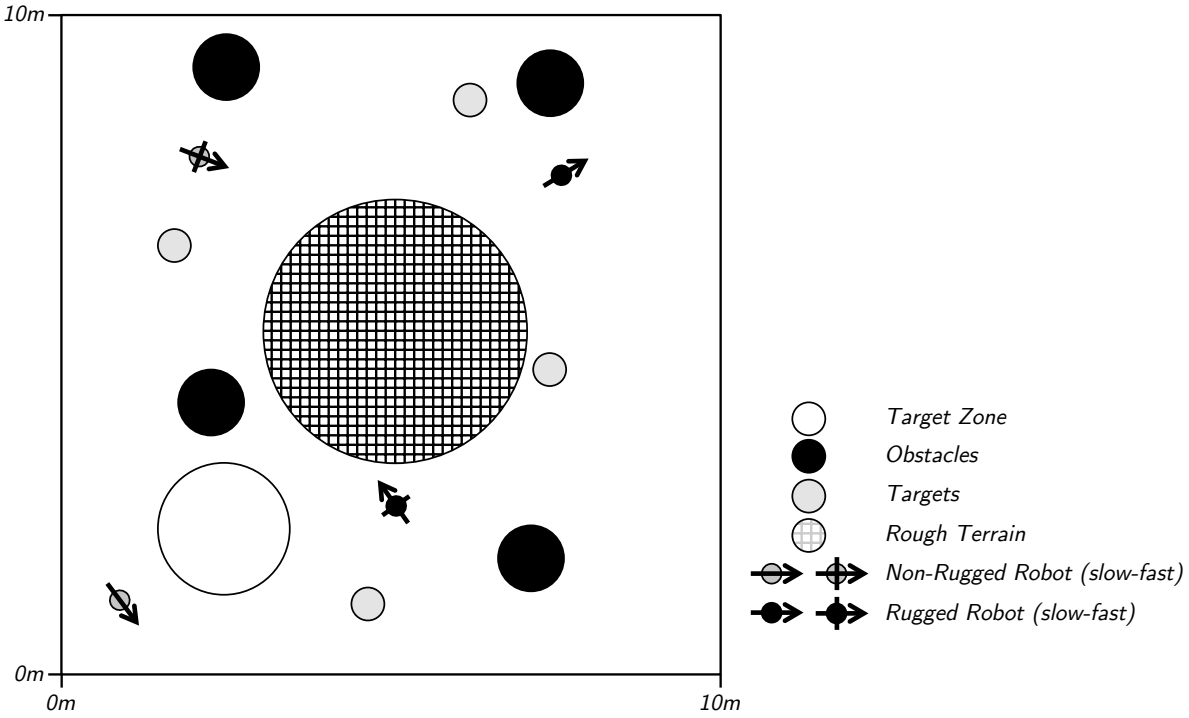


Figure 4.1: Sample foraging scenario displaying robots, items, obstacles, target zone, and the area of rough terrain

The learning process for each robot is an MDP with a $\langle S, A, T, R \rangle$ tuple defined as:

State $s \in S$: is defined by $s = \{t_d,\ t_\theta,\ g_d,\ g_\theta,\ o_d,\ o_t\}$, which contains the distance d and relative angle θ from the robot to center of the item t, goal location g, as well information about the closest obstacles. The obstacle states, $o_d = \{o_{1,d}, ..., o_{k,d}\}$ and $o_t = \{o_{1,t}, ..., o_{k,t}\}$, contain the distance d and type t of the closest obstacle along k equally spaced detection rays. Three detection rays are used, orientated with a separation of $\pi/10$ rads. The rigid obstacles, walls, and items (other than the one the robot is aiming at), are treated as the same type, while the rough terrain is treated as a separate type. All distances are limited to a maximum range of 2m and divided into 5 discrete intervals, while all angles are divided into 5 discrete quadrants within the interval $[0, 2\pi)$ rads.

Action $a \in A$: is defined by $[move\_forward, rotate\_left, rotate\_right, interact]$. The $move\_forward$ action will move the robot 0.2m for a slow robot and 0.4m for a fast robot, while the $rotate\_left$ and $rotate\_right$ actions will move the

robot $\pm 1/5\pi$ radians for all robots. *interact* attaches an item to the robot if the robot is within 0.5m of the item and the item is the robot's target item.

Transition $T(s', a, s)$: represents the probability of transitioning from state s to state s' after performing action a.

Reward $R(s, a, s')$: is the reward given to each robot for its action, and is defined in Table 4.1. A reward is given when the robot moves at least a threshold distance Δd (set to 30% of the robot's step size) towards or away from an item or goal area, or their target item is returned. When the robot does not receive any reward for its movement with respect to an item or for returning an item, a reward of 1.0 is given.

Table 4.1: Reward function for robot actions in Case Study 1.

Behavior	**Reward**
Robot moved at least Δd towards target item	5
Robot moved at least Δd away from target item	0.1
Item moved at least Δd towards target zone	5
Item moved at least Δd away from target zone	0.1
Item is returned to target zone	50
None of the above occurs	1

Each robot develops an individual policy π through a Q-learning algorithm with the update rule in eq. 1.4. The discount factor γ is held constant at 0.3, while the learning rate α decays to 0 as $t \to \infty$, governed by:

$$\alpha = 1/(1 + v_t(s))^{\sigma} \tag{4.1}$$

where σ is a constant that is set to 0.9. Each action has a probability of being selected defined by a Boltzmann distribution with a modified version of $\tau_t(s)$ in eq. 3.10, as given by,

$$\tau_t(s_I) = \frac{-\ln\left((1 - n\xi)/v_t(s) + n\xi\right)}{\max_{a \in A} |Q_t(s_I, a_{max}) - Q_t(s_I, a_I)|} \tag{4.2}$$

where $a_{max} = \text{argmax}_{a \in A}\, Q_t(s_I, a_I)$, n is number of actions, and ξ is a constant coefficient $[0, 1)$. The expression for $\tau_t(s_I)$ in eq. 4.2 is a modification of the formulation in [30], and enables actions to be selected in a probabilistic fashion that becomes more greedy over time, while maintaining a minimum probability of ξ over all actions. The value of ξ is selected to be equal to 0.02. Ensuring a minimum action selection probability

is necessary in a simulation environment where the state resolution is kept low enough to be tractable. Discretizing states can result in two very different scenarios having the same state values, such as a robot being directly against an obstacle, and a robot having a minimal amount of clearance to pass the obstacle. In such a scenario a policy which is too greedy could result in the robot getting stuck repeatedly performing an poor action. Finally, for the Preference Advice mechanism actions are selected via a ϵ-greedy policy, where a random action is selected with probability 0.05, otherwise the highest valued action is selected. The coefficient δ applied to the advice mechanism reward in eq. 3.14 is set to 2.5.

4.2 Experiments

A series of experiments using the Preference Advice mechanism have been formed to demonstrate its capabilities. In the following sections a *novice* robot refers to a robot that has had zero prior experience at the task, with zero initialized quality values. An *expert* robot is one which has previously performed and learned from the given task for a defined number of runs. Further, experts do not perform policy updates during simulations in order to perform a controlled analysis of the advice mechanism. Each expert is simply a saved policy from a previously trained robot.

4.2.1 Experiment 1: Homogeneous peers as advisers

When agents are concurrently learning a task, a difficult challenge is to use the partial knowledge of the other agents in a beneficial way. The ability of the proposed mechanism to do so is demonstrated by performing the foraging scenario with 4 novice agents of the same type (chosen to be S-NR). Each agent will inevitably learn different portions of the state space before others, so the mechanism will need to use the partial knowledge of others in order for the agent to develop the appropriate policy more quickly. When the advice mechanism is used, each agent should learn its task more quickly when compared to the case without the advice mechanism. This can be reflected by the simulation time to complete the task (number of iterations) and total team effort (number of actions) at each run, as well as the average reward received by the team during each run. We also compare the performance of the proposed mechanism to the team performance using the Advice Exchange algorithm [21].

4.2.2 Experiment 2: Heterogeneous peers as advisers

We extend experiment 1 to use heterogeneous advisers to demonstrate the mechanism's performance with advisers possessing different capabilities. Four robots, one of each S-NR, S-R, F-NR, and F-R type, perform the foraging scenario. In the case of non-rugged robots, the rugged advisers will attempt to guide them through the rough terrain, which they are incapable of moving through. Such a scenario will illustrate that the biasing effect of the advice in the proposed mechanism can influence an agent without aggressively forcing it to perform detrimental actions.

4.2.3 Experiment 3: Expert advisers of varying skill level

Due to the random nature of action selection (as well as scenario initialization), certain agents may learn the task faster or more slowly than others, resulting in the usefulness of advice varying between agents. When advisers of the same type, but different expertise, are made available to the advisee, the advice mechanism must be capable of recognizing varying levels of knowledge about the task. To demonstrate this, a simulation is performed with one novice S-NR robot having access to expert advisers of the same type trained for 10, 50, and 100 runs. Since all robots are homogeneous, the advice mechanism must evaluate advisers based on their skill at the task. The relevance of each adviser is indicated by the similarity measure in eq. 3.16. The appropriate behaviour of the mechanism is expected to attribute a greater similarity to the experts with more experience as time proceeds.

4.2.4 Experiment 4: Expert advisers of varying capabilities

As previously stated, for heterogeneous robot teams the suitability of advice depends not only on the expertise of the adviser, but also on how similar the adviser is to the advisee. When advisers of different types but similar expertise are made available to the advisee, the advice mechanism must still be capable of determining the relevance of each adviser. To demonstrate this, a simulation is performed with a novice S-NR robot having access of expert advisers of each possible type (i.e., S-NR, F-NR, S-R, and F-R), all previously trained for 100 runs. In this scenario the rugged advisers will attempt to guide the robot across the rough terrain, which it is incapable of doing, while the fast advisers will have learned a different sequence of motions than the advisee due to the larger movement during each action. Again, the relevance of advisers is indicated by the similarity measure with eq. 3.16.

4.2.5 Experiment 5: Supplement a team of novices with a partially trained adviser

An interesting use of advice for robot teams is when a group of novice robots can have access to at least one expert robot. Even if the expert robot is only partially trained, its availability should still accelerate the learning process for the entire team. To investigate this, 4 S-NR novice robots perform the foraging scenario. Each novice has access to the advice of its peers as well as a previously trained S-NR robot. The supplementary expert adviser does not participate in the scenario. The experiment is repeated using the supplementary expert adviser, which is trained for 10, 50, and 100 runs, to illustrate the effects of varying levels of expertise made available in the early stages.

4.3 Results

During each experiment, the simulations are limited to 200 runs when 4 robots are used, and 100 runs when 1 robot is used, with each run limited to 4000 iterations. The reduction in runs when a single robot is used is due to the robot's performance converging more quickly when other robots are not present to impede its motion. Each experiment is repeated 15 times, and the data is averaged over all 15 trials. Additionally, a 10 point moving average (i.e. averaged over 10 runs) is applied to the averaged results from the 15 trials. The following metrics are utilized to measure the team's performance: simulation time, total, effort, standard deviation of simulation time, standard deviation of total effort, and the average team reward. Where simulation time is the total number of iterations made for each team member, and total effort is the number of actions towards the task taken by each team member.

4.3.1 Experiment 1: Homogeneous peers as advisers

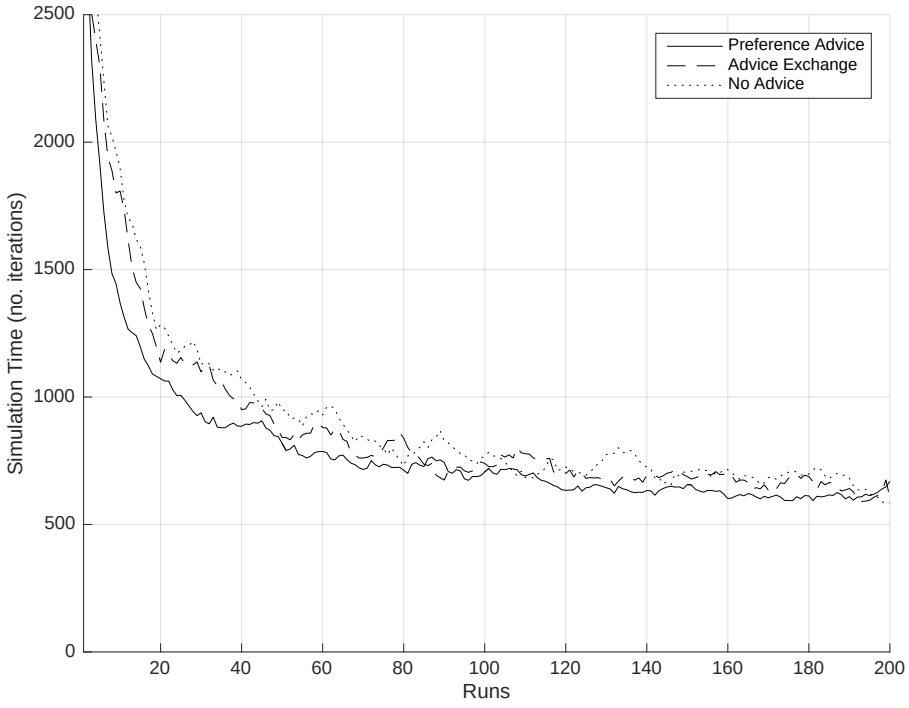
Experiment 1, the use of homogeneous peers as advisers with 4 S-NR robots, is considered first. Figs. 4.2a and 4.2b shows the simulation time (number of iterations) and total effort (number of actions) required by the team at each run to complete the task without advice, with the Preference Advice mechanism, and with the Advice Exchange mechanism for comparison. It is apparent from the figures that the use of advice provides a consistent reduction in the mean values of both simulation time and total effort. This is especially evident during the transient stage of learning, considered as the first 50 runs where rapid convergence occurs. For both metrics, the use of the Preference Advice mechanism provides a greater improvement than the Advice Exchange mechanism. The benefit of

Advice Exchange is most noticeable within the first 100 epochs for simulation time and total effort, but becomes indistinguishable from the no advice case beyond that. With the Preference Advice mechanism, the mean values of simulation time and total effort appear to be consistently less than both the Advice Exchange and no advice cases for the first 100 runs, beyond which only the reduction in simulation time is discernible.

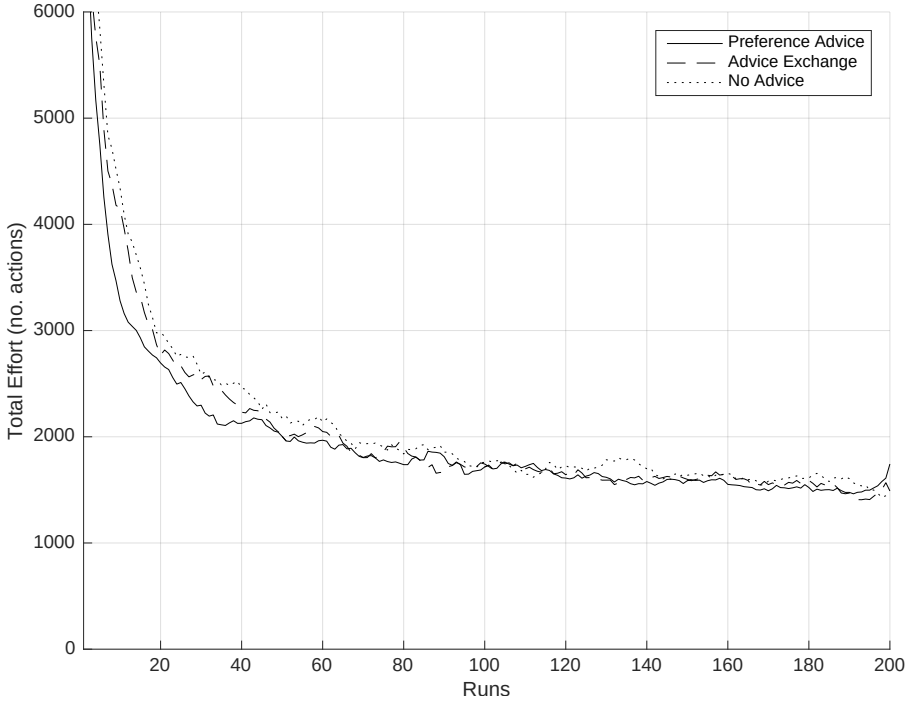
The standard deviation of simulation time and total effort at each run is presented in figs. 4.3a and 4.3b. With the Advice Exchange mechanism there is no discernible reduction in the standard deviation of either simulation time or total effort. However, with the Preference Advice mechanism there is a clear reduction in the standard deviation of both simulation time and total effort. This reduction is most prominent during the transient stage, but is present for all 200 runs. Reducing the standard deviation of simulation time and total effort with the Preference Advice mechanism indicates that it increases the consistency in the team's performance at the task. Again, this improvement is the result of utilizing multiple advisers at each time step.

The ability of the Preference Advice mechanism to incorporate advice from multiple advisers at each time step is a key factor in the improvement in both the mean and standard deviation of simulation time and total effort. In the early stages of learning, a robot's experience in the state space will be sparse, hence several advisers may need to be polled before one with sufficient experience in the desired state is found. If only a single adviser can be utilized at each time step, which is the case for Advice Exchange, it severely limits the likelihood of obtaining useful advice. Additionally, it is likely that each robot will have some experience in the state, but not a large amount. This will frequently prevent advice from being used if it is required that the adviser's performance or experience exceeds the advisee's (as evaluated by certain conditions). Conversely, if an adviser's input can be utilized, regardless of the magnitude of their contribution, the benefits of advice can be redeemed more frequently and provide a more consistent performance improvement

Lastly, the average reward of all robots in the team is displayed in fig. 4.4. Again, the use of advice appears to provide a slight, yet consistent, improvement in the mean value of reward obtained compared to without advice. Acquiring reward in larger quantities indicates that the use of advice encourages the selection of more favorable actions earlier. Interestingly, the distinction between the Preference Advice and Advice Exchange mechanisms is only apparent within the first 50 runs. A likely cause of this is that the reward function only provides a large, or small reward when the robot moves some threshold distance towards or away from the target item or the target zone. This results in a small subset of actions having equivalent rewards, yet varying in their contribution to the goal.

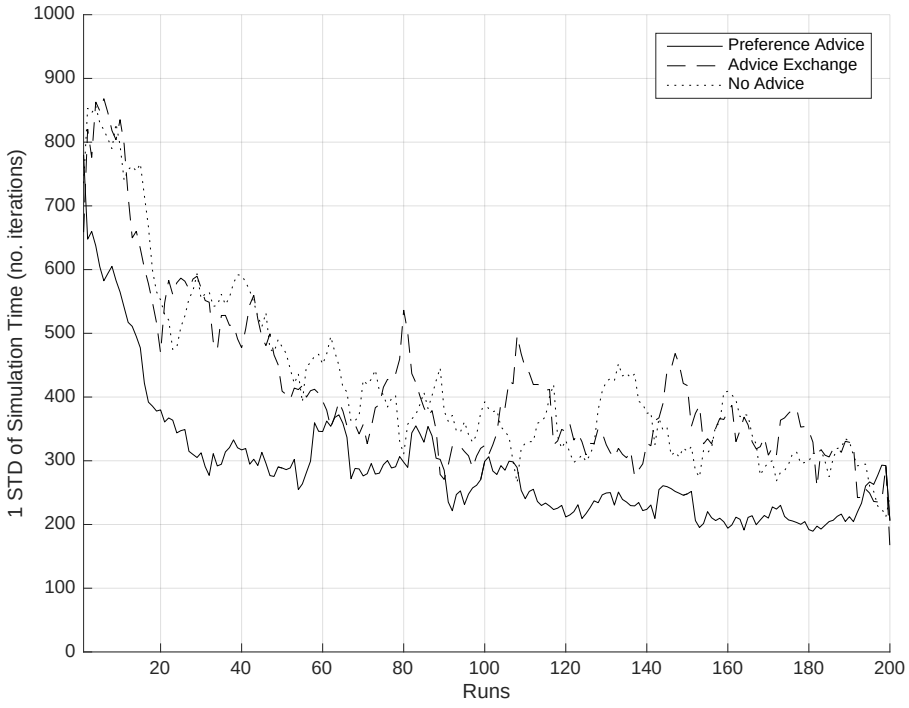


(a)

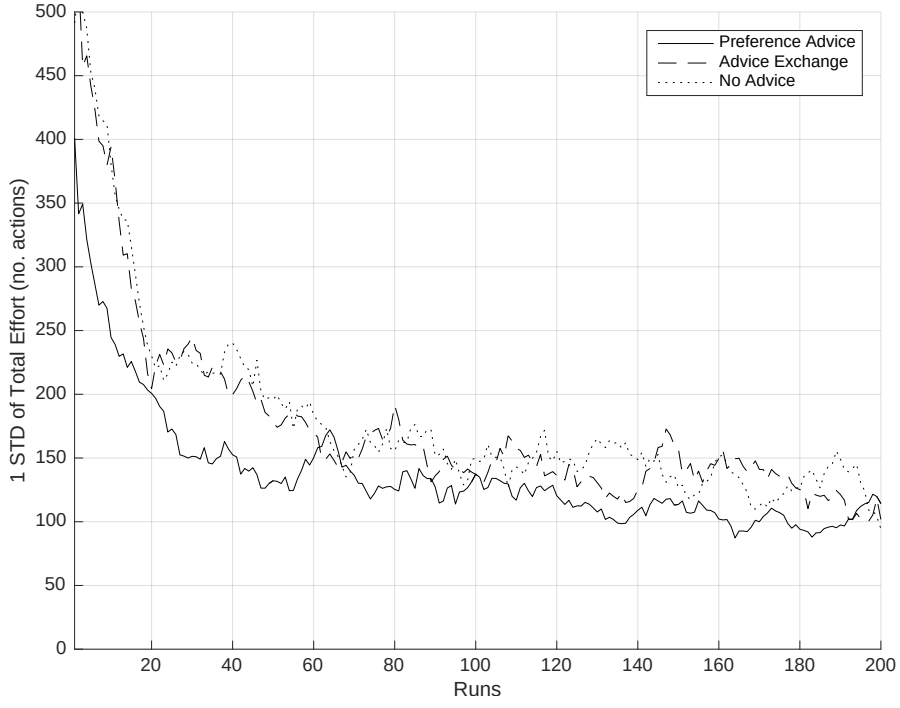


(b)

Figure 4.2: Performance with the Preference Advice mechanism, with the Advice Exchange mechanism, and without advice, for 4 S-NR robots in experiment 1: (a) simulation time, (b) total effort



(a)



(b)

Figure 4.3: Standard deviation of performance at each run for experiment 1: (a) simulation time, (b) total effort

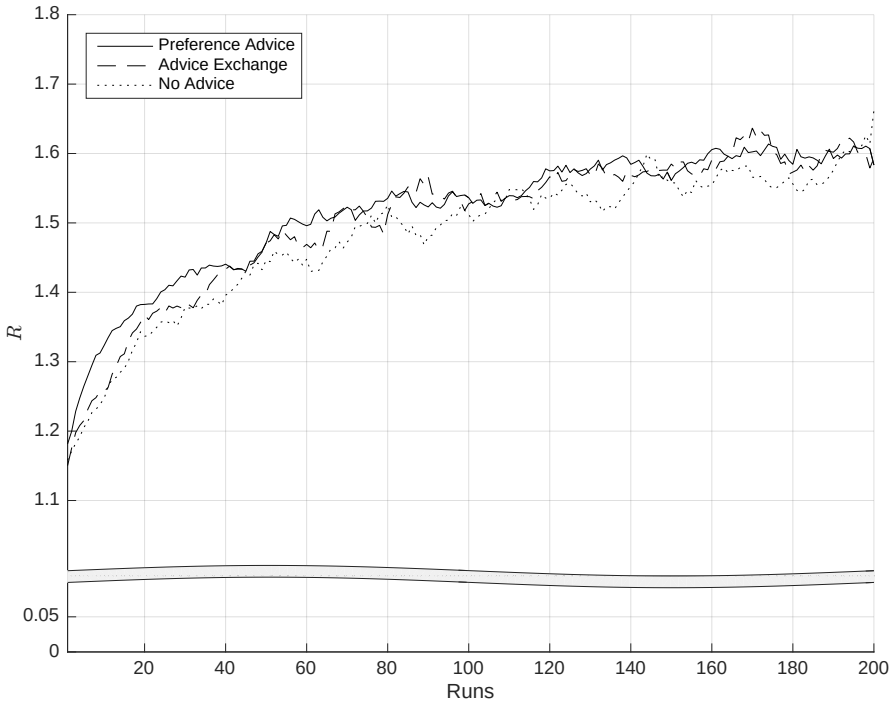


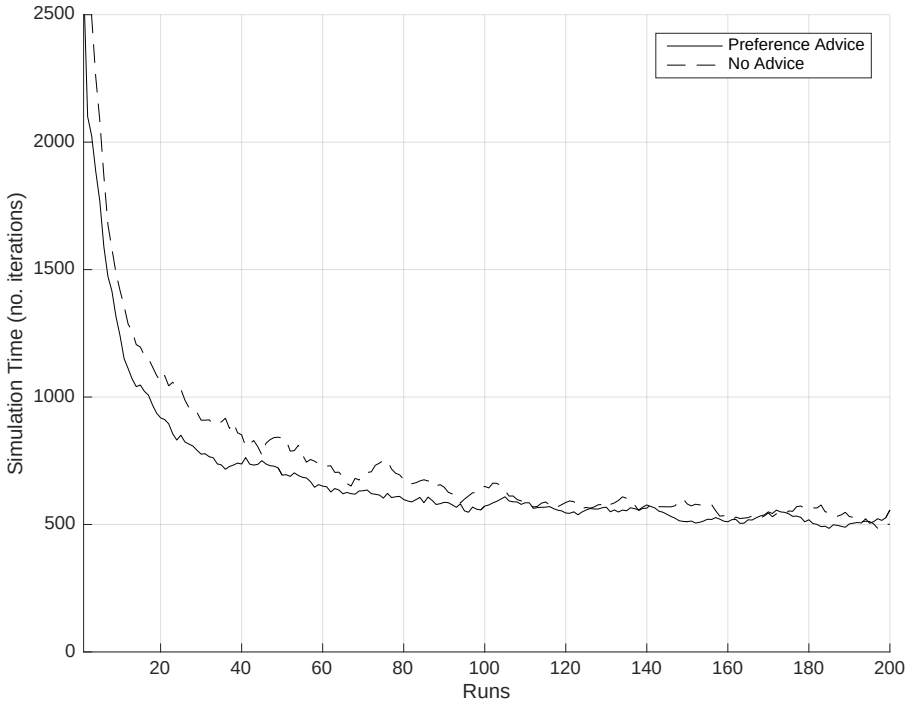
Figure 4.4: Average reward obtained between 4 S-NR robots in experiment 1

4.3.2 Experiment 2: Heterogeneous peers as advisers

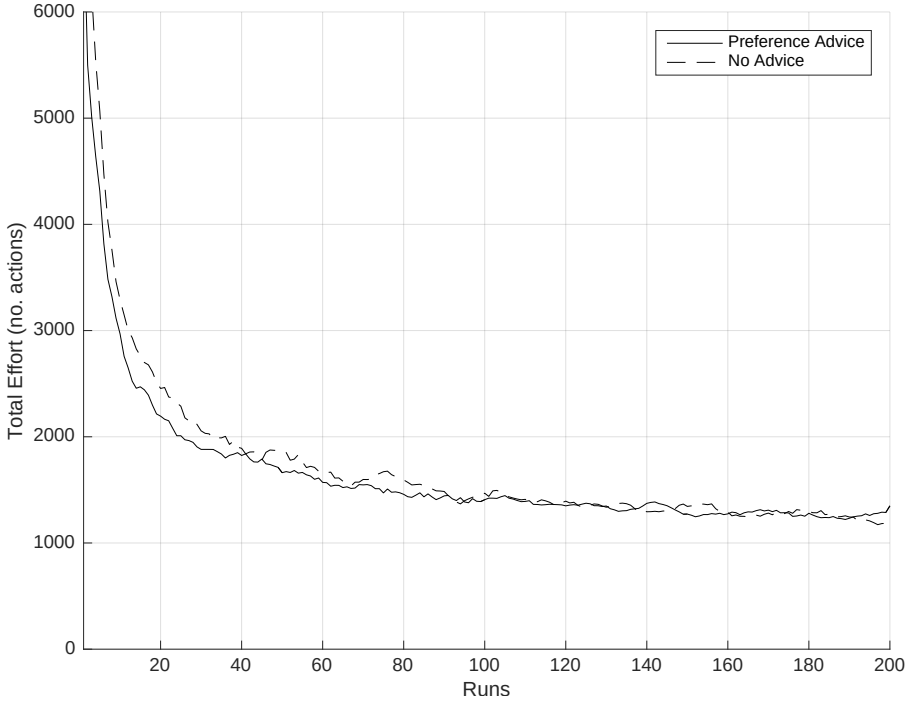
The second experiment demonstrates the mechanisms compatibility with heterogeneous advisers, where one of each type of robot (i.e., S-NR, F-NR, S-R, and F-R) perform the foraging scenario together. Hence, no two robots performing the foraging task have identical capabilities. The simulation time and total effort for the team to complete the task at each run, with and without advice, is shown in figs. 4.5a and 4.5b. The comparison to Advice Exchange is not made here, since it is not compatible with heterogeneous advisers.

Again, the mean values of simulation time and total effort are consistently lower with the Preference Advice mechanism than without advice, particularly during the transient stage of learning. In this scenario, the advice from rugged robots would be to cross the rough terrain, which non-rugged robots are incapable of doing, while the advice from non-rugged robots would be to go around the terrain, which is an inferior policy for rugged robots.

If unsuitable advice were being used, we would expect to see an increase in iterations accompanied by an increase in the standard deviation of mission iterations, as a result of robots getting "stuck" performing actions they are incapable of doing. However, there is a decrease simulation time and total effort, as well as a decrease in their standard

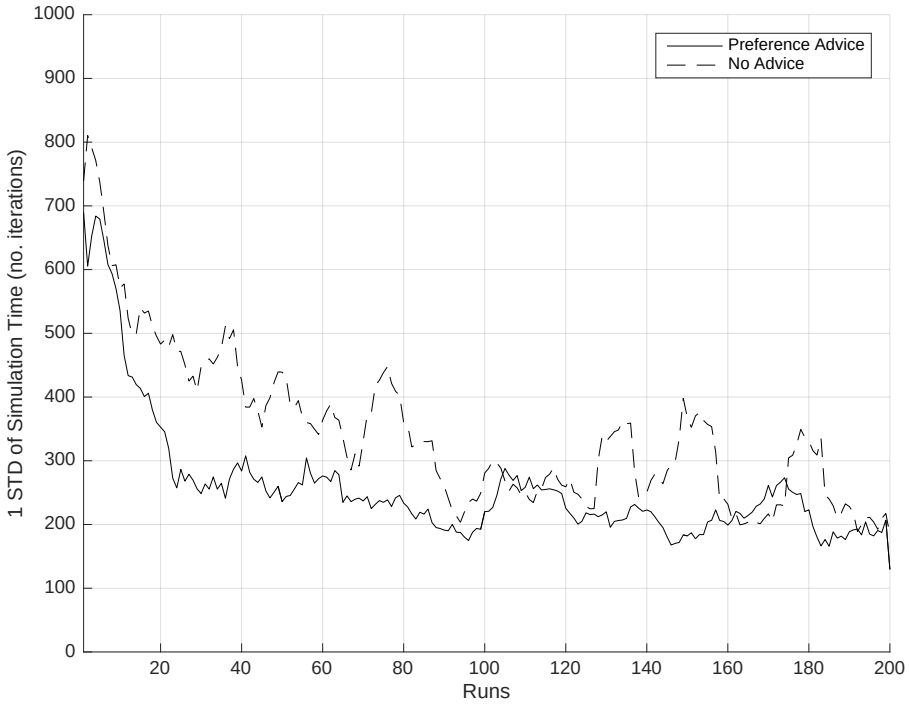


(a)

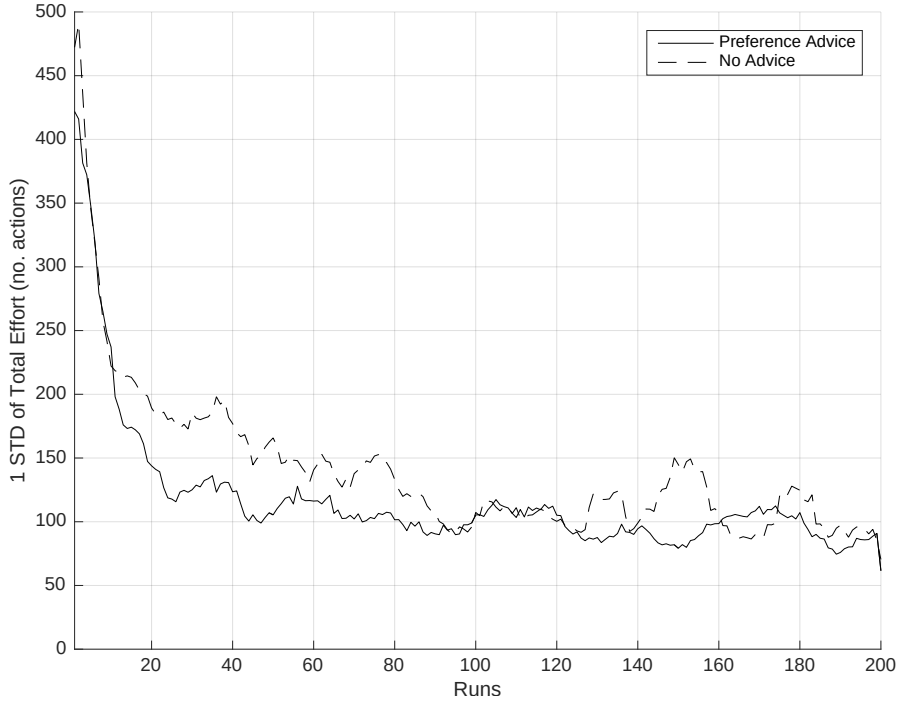


(b)

Figure 4.5: Performance with the Preference Advice mechanism and without advice for 4 heterogeneous robots (S-NR, F-NR, S-R, and F-R) in experiment 2: (a) simulation time, (b) total effort



(a)



(b)

Figure 4.6: Standard deviation of performance at each run for experiment 2: (a) simulation time, (b) total effort

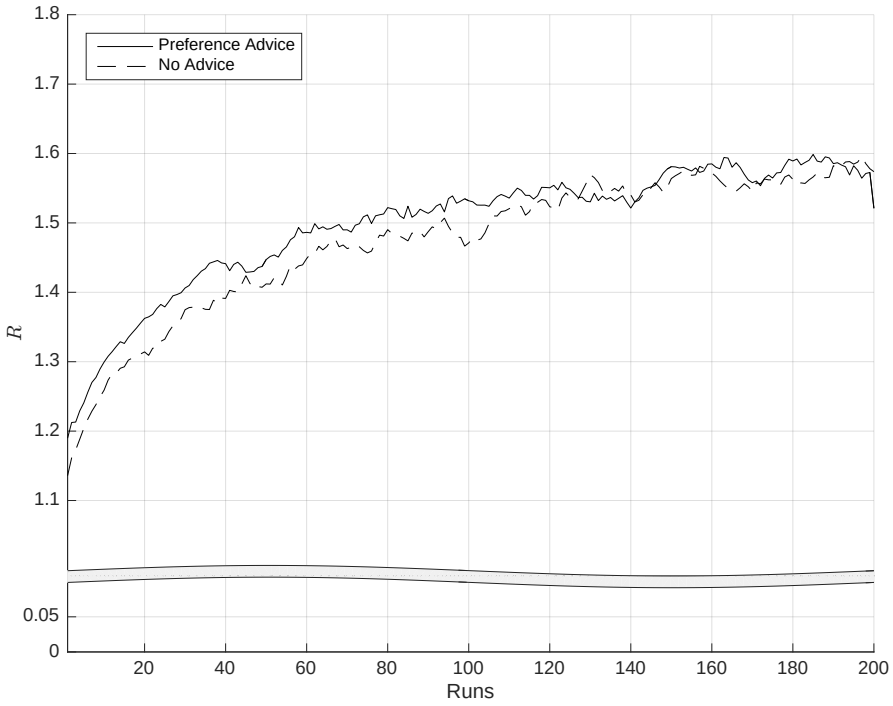


Figure 4.7: Average reward obtained between 4 4 heterogeneous robots (S-NR, F-NR, S-R, and F-R) in experiment 2

deviations (figs. 4.6a and 4.6b) that is comparable to the reduction in standard deviation from experiment 1 with homogeneous advisers. Therefore, this is a strong indication that the Preference Advice mechanism is compatible with heterogeneous advisers. Such compatibility is due to the adviser influence $\lambda_t(s)$ in eq. 3.11 being derived to ensure a conservative use of advice that will not bias the advisee's policy too strongly. The average reward of all team members is shown in fig. 4.7, where a similar improvement as in experiment 1 is observed.

4.3.3 Experiment 3: Expert advisers of varying skill level

Experiments 3 and 4 use virtual expert advisers that do not participate in the task, where the single robot performing the foraging task simply has access to their policies. Such a scenario enables the use of advice over time with Preference Advice mechanism to be studied under static adviser conditions, as well as demonstrating how the Preference Advice mechanism could be used with alternative advice sources, such as humans. In experiment 3, virtual expert advisers with different amounts of experience are made available to a S-NR robot. Figure 4.8 shows the percentage which the Preference Advice mechanism requests and accepts advice at each time step. As the robot learns the task, the frequency which advice is requested rapidly diminishes. Reducing the use of advice over time is a desirable property for real world robot teams, where unnecessary communication and computational costs should be avoided. The acceptance of advice at each time step increases to a peak near run 30, and steadily diminishes thereafter. The increase in advice acceptance between run 1 and run 30 indicates that the Preference Advice mechanism quickly learns the value of advice, while the decline afterwards paired with the small request occurrence indicates that it also becomes more selective with advice over time. The relevance of each adviser, as measured by eq. 3.16, is displayed in fig. 4.9. The relevance of an adviser increases with its experience, which is the appropriate behavior, resulting in the adviser with the most experience being polled first for its advice. Therefore, despite the mechanism not directly learning the relevance of each adviser, the ranking scheme implemented has worked successfully.

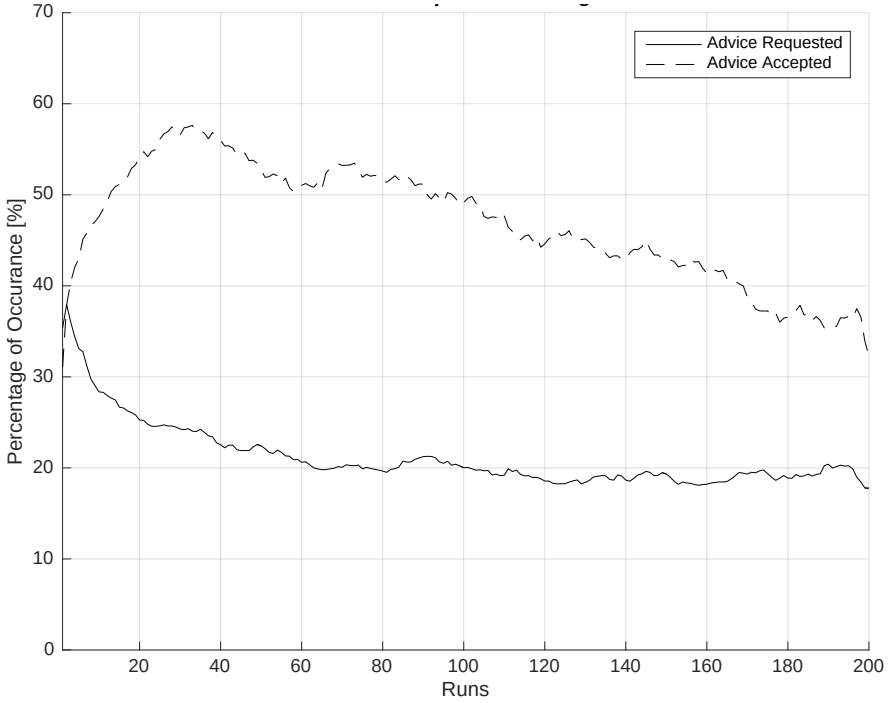


Figure 4.8: Percentage which advice is requested and used during each run in experiment 3

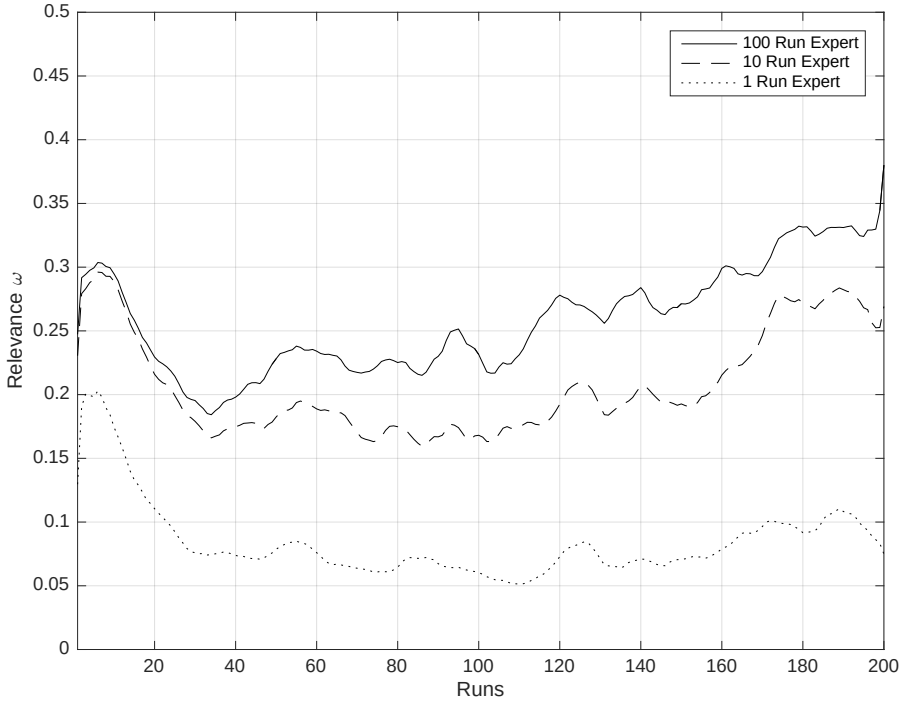


Figure 4.9: Relevance for advisers of varying skill in experiment 3

4.3.4 Experiment 4: Expert advisers of varying capabilities

Experiment 4 uses expert advisers differing in terms of capabilities. A S-NR robot having access to a virtual expert adviser of each type (S-NR, F-NR, S-R, and F-R) is used for the simulation. The relevance of each type of adviser, as determined by eq. 3.16, is shown in fig. 4.10. The two advisers of the slow type, namely S-NR and S-R, appear to be equally the most relevant, while the advisers of the fast type, F-NR and F-R, have lower, yet similar, levels of relevance. The Preference Advice mechanism differentiates between the policies of different adviser types, although the results indicate that the difference in policies between rugged and non-rugged robots is smaller than the difference between fast and slow robots. The results from this experiment highlight the importance of having an advice mechanism capable of determining the relevance of advisers on its own, since prior to operation the similarity in policies may not be obvious enough to generate static rules regarding the use of advisers.

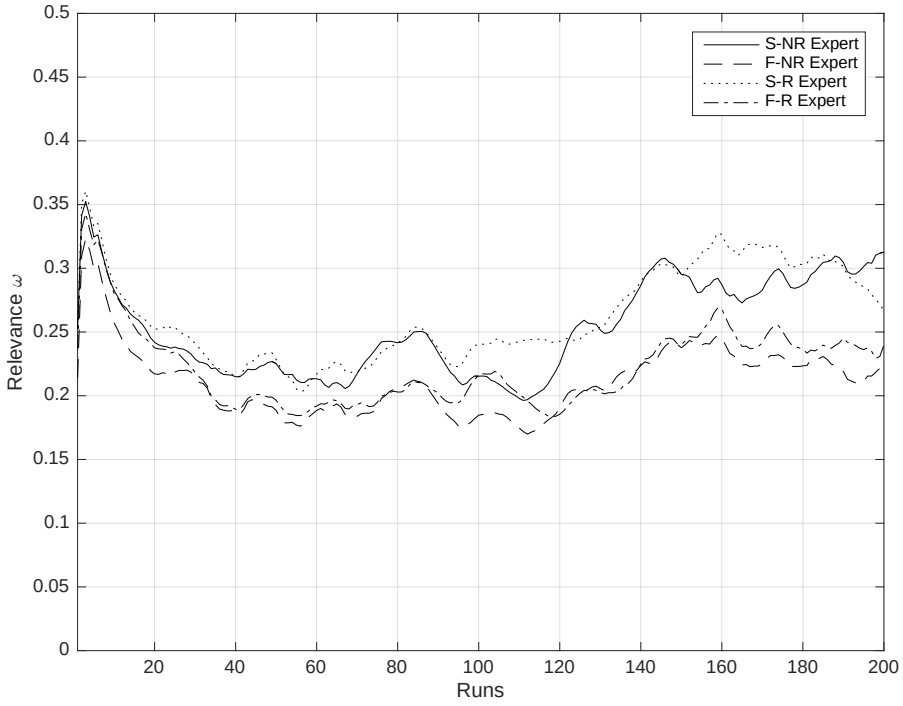


Figure 4.10: Relevance of advisers of varying capabilities in experiment 4

Fig. 4.11 shows the number of iterations for the single S-NR robot to complete the foraging task without advice, with advice from advisers of varying skill levels, and with advice from advisers of varying capabilities. Both variations in advisers provide similar improvements in simulation time, especially during the transient stage of learning. Again, since the supplementary expert adviser is a virtual adviser, it could also be a human

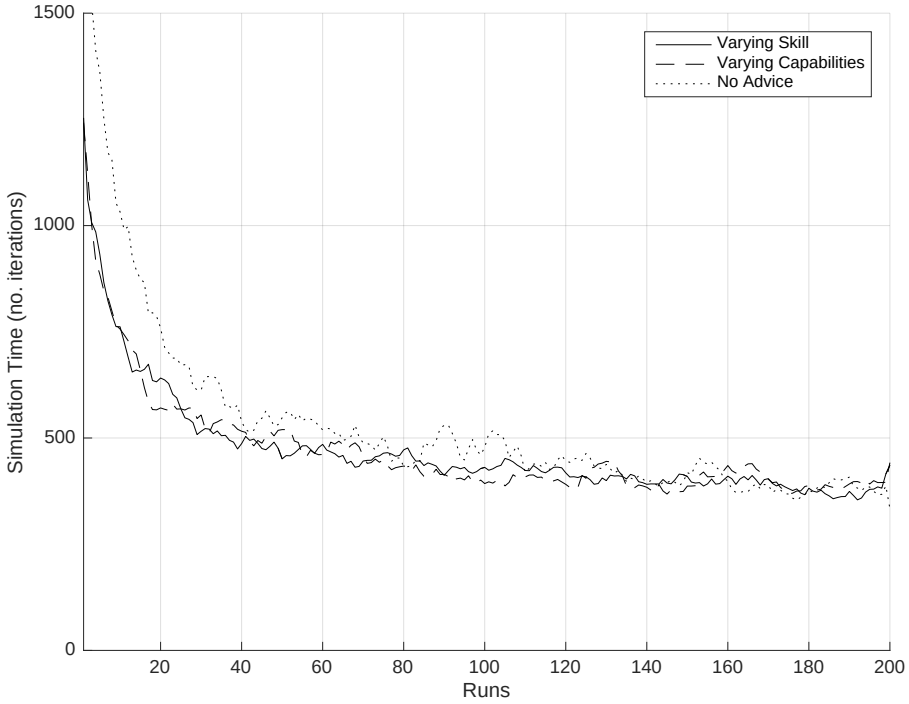


Figure 4.11: Simulation time for 1 S-NR robot with advisers of varying skill (experiment 2) and advisers of varying capabilities (experiment 4), compared to without advice

providing advice to the team. Such a reduction in iterations with a virtual expert adviser can provide significant benefits to real world robot teams, where the use of a single human can greatly reduce the time to learn the task at hand, as well as reduce operational time for the robots, and hence costs. Based on the adviser relevance values, it appears that for experiment 3 the 100 run expert (of the same type as the advisee) was consistently polled first. Additionally, for experiment 4 the relevance of the S-NR and S-R robots were the largest and nearly equal, indicating those two advisers were consistently polled first. These two observations imply that having two expert advisers with high relevance is not distinctly more beneficial than having a single expert adviser available.

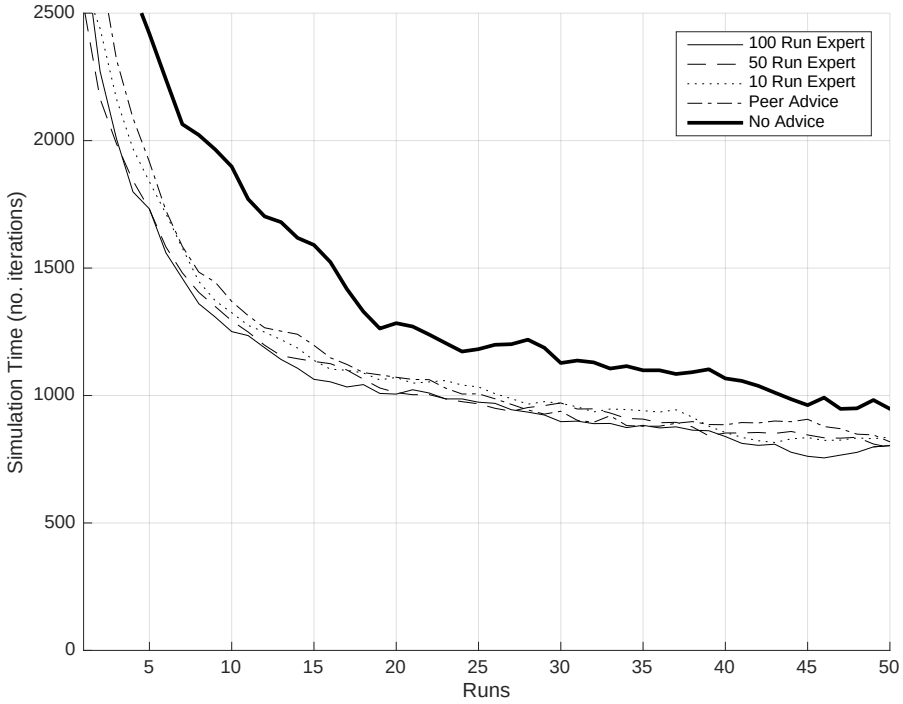


Figure 4.12: Simulation time for 4 S-NR robots with a supplementary expert adviser in experiment 5, compared to peer only advice and no advice

4.3.5 Experiment 5: Supplement a team of novices with a partially trained adviser

Lastly, we consider experiment 5, where a team of four S-NR robots use peers as advisers, plus one supplementary expert adviser. The simulation time for each case of adviser expertise (10, 50, and 100 runs) is shown in fig. 4.12, as well as peer only advice (identical to experiment 1) and no advice for comparison. Given the large number of curves on the plot, for clarity only the simulation time during the transient stage is shown. Relative to peer only advice, the supplementary adviser provides an additional reduction in simulation time, with the advisers having 50 and 100 runs of experience providing the most improvement. Despite the improvement obtained from having a supplementary expert adviser in addition to peers being relatively small, it does indicate that all four robots are successfully able to leverage the additional source of advice early in the learning process. Similar to the balance between exploration and exploitation with reinforcement learning, there is a similar balance between exploiting an experienced adviser and enabling a robot to experience a sufficient amount of exploration. The policy of the expert adviser with 100 runs of experience could be directly adopted, but it may completely prevent the novice robots from experiencing large areas of the state space.

Chapter 5

Case Study 2: Preference Advice Within RCISL

5.1 Simulation Scenario

The second case study serves to demonstrate the use of the Preference Advice mechanism within the RSICL algorithm, which is described in chapter 2. Namely, the use of advice in the presence of team learning and state uncertainty. Uncertainty in observations and state estimates is an inevitable aspect of real world robot systems, so it is important to test any new developments under such circumstances. A foraging scenario with a heterogeneous team of robots is used, but differs from Case Study 1 in the types of tasks to be performed, and the lack of the rough terrain. The additional variation in tasks to be performed adds an additional layer of heterogeneity that necessitates the use of a task allocation algorithm.

The goal of the team is to collect a set of items within a foraging area, and deposit them in a collection zone. The foraging area is 10m by 10m, containing 4 obstacles that are 0.5m in diameter, and a single collection zone that is 1m in diameter. Four different types of robots are used, differing in their speed of movement and their strength: Slow-Weak, Slow-Strong, Fast-Weak, and Fast-Strong. There are also two possible types of collectable items: light and heavy. Only the strong type robots are capable of moving the heavy items. All robots and items are 0.25m in diameter. In the presented case study, 4 robots are used (one of each type: Slow-Weak, Slow-Strong, Fast-Weak, and Fast-Strong), and 4 items are to be collected (two light and two heavy). Each run begins with the items, obstacles, collection zone, and robots, randomly positioned within the foraging area. A run is complete when either all items have been returned to the collection zone,

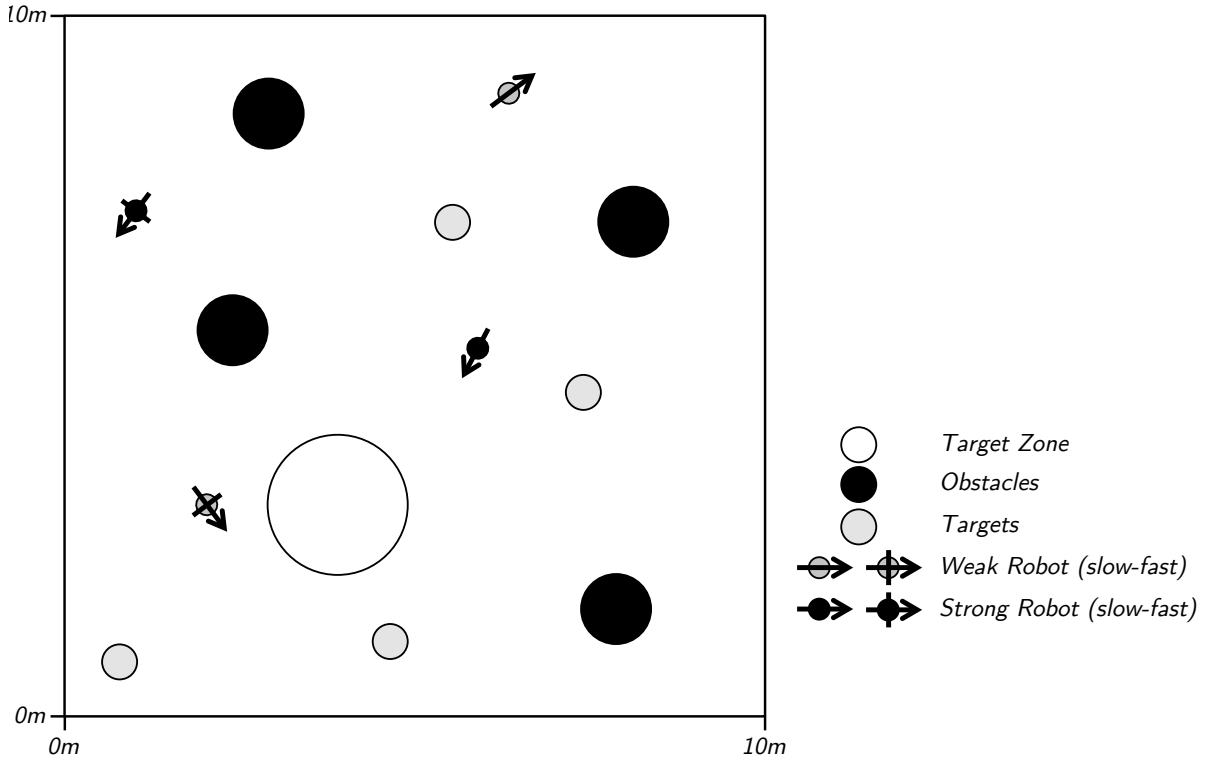


Figure 5.1: Sample foraging scenario displaying obstacles, target zone, and each type of robot and item to collect

or 6000 actions by each robot have been performed. An example initial configuration for the simulation scenario is shown in fig. 5.1.

As discussed in section 2.2, individual and team learning are partitioned into two separate, concurrent, MDP's. The individual learning process for each robot is an MDP with a $\langle S_I, A_I, T_I, R_I \rangle$ tuple defined as:

> **State $s_I \in S_I$**: is defined by $s_I = \{t_d, t_\theta, t_t, g_d, g_\theta, o_d, o_\theta\}$, which contains the distance and relative angle from the robot to the target t, goal location g, and closest obstacle o. The subscripts d and θ indicate that the variable represents the distance and angle to the object respectively. The state t_t indicates the type of target to be collected. All distances are limited to a maximum range of 3m and divided into 3 discrete intervals, while all angles are divided into 5 discrete quadrants within the range $[0, 2\pi)$ rads. The obstacle state o represents the single closest obstacle to the robot, which can be a fixed obstacle, a wall, items, or another robot.

Action $a_I \in A_I$: is defined by [$move\_forward$, $rotate\_left$, $rotate\_right$, $interact$]. The $move\_forward$ action will move the robot 0.2m for a slow type robot and 0.4m for a fast type robot, while the $rotate\_left$ and $rotate\_right$ actions will move the robot $\pm 1/5\pi$ radians for all robot types. $interact$ attaches an item to the robot if the robot is within 0.5m of the item and the item is the robot's target item.

Transition $T_I(s'_I, a_I, s_I)$: represents the probability of transitioning from state s_I to state s'_I after performing action a_I.

Reward $R_I(s_I, a_I, s'_I)$: is the reward given to each robot for its action, and is defined in Table 5.1. A reward is given when the robot moves at least a threshold distance Δd (set to 30% of the robot's step size) towards or away from an item or goal area, or their target item is returned. When the robot does not receive any reward for its movement with respect to an item or for returning an item, a reward of 1.0 is given.

Table 5.1: Reward function for robot actions in Case Study 2.

Behavior	Reward
Avatar moved at least Δd towards target item	5
Avatar moved at least Δd away from target item	0.1
Item moved at least Δd towards collection area	5
Item moved at least Δd away from collection area	0.1
Item is returned to collection area	50
None of the above occurs	1

Q-learning is used to develop the individual learning policy, π_I, in the same fashion as for Case Study 1. In eq. 1.4 the discount factor γ is held constant at 0.3, while the learning rate α decays to 0 as $t \to \infty$, governed by eq. 4.1 with the constant σ set to 0.9. Each action has a probability of being selected defined by a Boltzmann distribution with the temperature parameter $\tau_t(s_I)$ defined by eq. 4.2 with the value of ξ being set to 0.02 to ensure all actions maintain a minimum selection probability.

Team learning is achieved with the RL-Alliance algorithm, which is responsible for appropriately allocating tasks to robots with sufficient capabilities (e.g. only strong robots are capable of collecting heavy items). The parameter values for the impatience update (eq. 2.3) and trial time update (eq. 2.4) are selected to be $\theta_1 = 1.0$, $\theta_2 = 15.0$, $\theta_3 = 0.3$, and $\theta_4 = 2.0$. Additionally, the motivation for each robot is updated after every 5 time steps, and the maximum number of time steps before a robot must acquiesce its

task is selected to be 1500, enabling a task to be assigned multiple times if it is not yet completed.

Finally, for the Preference Advice mechanism actions are selected via a ϵ-greedy policy, where a random action is selected with probability 0.05. The coefficient δ applied to the advice mechanism reward in eq. 3.14 is set to 4.0. It is worth noting that no restrictions are placed upon which robots can be used as advisers (i.e. a robot can utilize advice from another robot of any type).

5.2 Experiments

The RCISL algorithm builds upon individual learning by adding additional layers of team and social learning, while also making each robot robust to measurement uncertainty via a state estimator. Investigating the use of the Preference Advice mechanism within RCISL provides additional insight into the mechanism's behaviour in more complex in realistic applications. Particularly, how the performance benefit with the Preference Advice mechanism is influenced by the presence of state uncertainty. With this goal in mind, three experiments have been formed.

5.2.1 Experiment 1: Preference Advice with team learning

The first experiment serves to demonstrate the use of the Preference Advice mechanism in an application that utilizes team learning. This repeats the experiments performed for CISL in [5], with the Preference Advice mechanism replacing the use of Advice Exchange. The foraging scenario is performed with and without the use of the Preference Advice mechanism to observe the impact on the team's performance.

5.2.2 Experiment 2: Preference Advice with team learning and measurement uncertainty

Real world robot systems inevitably have noise present in every measurement. Consequently, it is important to observe the influence of noise on each aspect of the system. In this experiment, zero mean Gaussian noise is added to each state variable for a robot (excluding the target type variable which is binary). The foraging scenario is performed with measurement noise having standard deviations of 0.05m, 0.20m, and 0.40m. Additionally, the scenario is performed for each noise level with and without the the use of the Preference Advice mechanism.

5.2.3 Experiment 3: Preference Advice with team learning, measurement uncertainty, and a state estimator

As in [6], a particle filter is implemented to estimate each of the state variable for every robot. Each simulation from experiment 2 is performed again, but this time with the particle filter utilized. This experiment serves to provide the most realistic usage of advice within a robot team, since uncertainty will always be present, but so will methods of eliminating that uncertainty.

5.3 Results

During each experiment, the simulations are limited to 200 runs, with each run limited to 6000 iterations. Each experiment is repeated 15 times, and the data is averaged over all 15 trials. Additionally, a 10 point moving average (i.e. averaged over 10 runs) is applied to the averaged results from the 15 trials. Identical to the experiments in chapter 4, the following metrics are utilized to measure the team's performance: simulation time, total, effort, standard deviation of simulation time, standard deviation of total effort, and the average team reward.

5.3.1 Experiment 1: Preference Advice with team learning

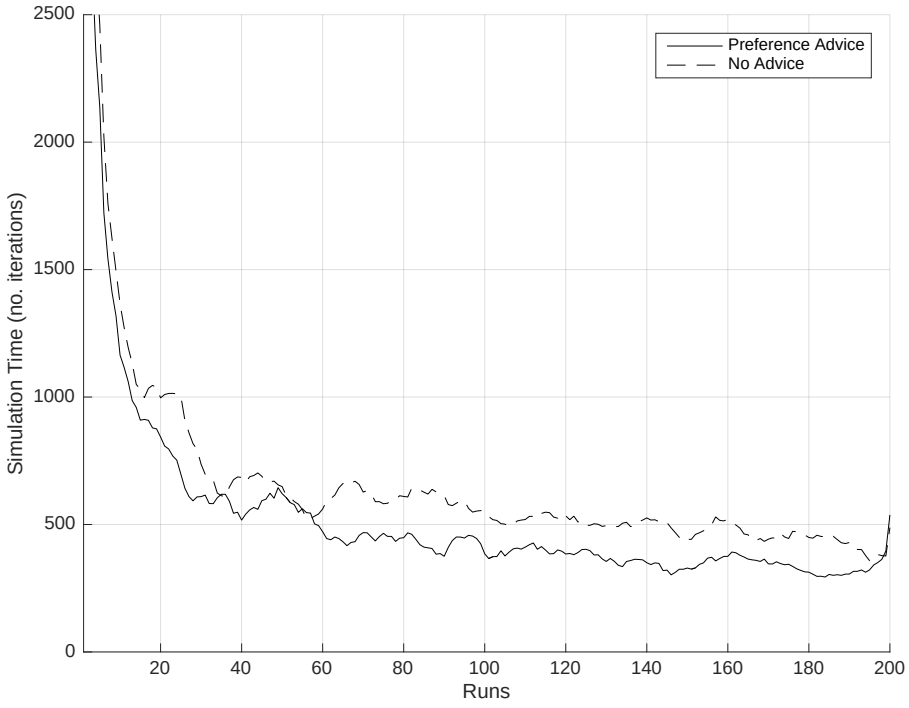
The results from experiment 1 are presented in figures 5.2 to 5.4. From figures 5.2a and 5.2b it can be seen that the Preference Advice mechanism has improved both the simulation time and total effort for the team consistently throughout the learning process. A notable difference with respect to the results without team learning (chapter 4) is that there is no immediate jump start in performance from the first epoch, but rather a gradual improvement. This can be attributed to the nature of the L-Alliance algorithm, which inevitably allocates tasks to the incorrect robots during the initial epochs, until it appropriately identifies the proper robot for each task. This results in both simulation time and total effort being dominated by the time L-Alliance allows each robot to attempt a task, regardless of each robot's performance at its task. It is likely that a prior heuristic incorporated into the team learning algorithm could greatly reduce the dependency of early performance on team learning.

A second noticeable difference to the results in case study 1 is that the performance improvement with the use of advice is still noticeable after learning has converged (i.e. from 50 up to 200 runs). An important difference between the two case studies is that in the current case study each robot developing is developing the same policy about how

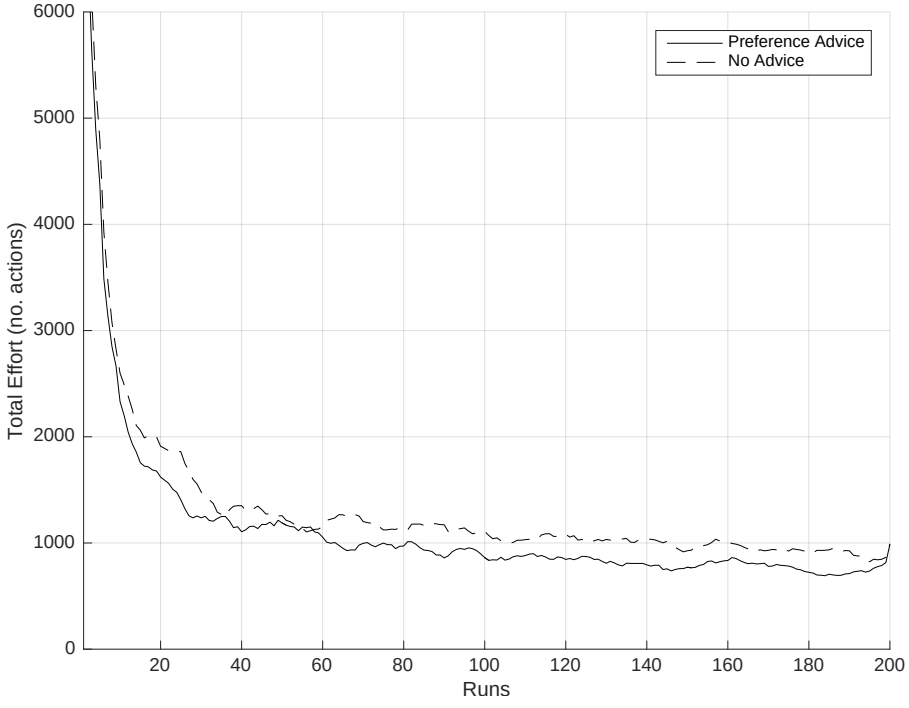
to move throughout the world, whereas in case study one the Non-Rugged robots were required to learn to avoid the rough terrain, and hence making their policy less applicable to the remainder of the robots on the team.

The standard deviation of simulation time and total effort over all 15 trials is shown in fig. 5.3a and fig. 5.3b. Similar to the experiments without team learning, the Preference Advice mechanism has reduced the standard deviation of both simulation time and total effort, indicating an improvement in consistency in addition to the improvement in performance. This is an intended and expected result for the use of advice that biases a robots decision, since the advice is not a strict acceptance, but rather a consensus on a decision. It is natural that utilizing more robots to arrive at a consensus for a decision will lead to more consistency in selecting the appropriate decision.

The average team reward for all robots is shown in fig. 5.4, where it can be observed that the Preference Advice mechanism provides a noticeable and consistent improvement compared to without advice. This indicates that the use of advice more quickly encourages the robots to select the more highly rewarded actions. Interestingly, the increase in reward due to advice is more prominent when team learning is used compared to case study 1. This is likely due to the increased similarity between robot policies with regards to navigating the environment. Similar to the results for simulation time and total effort, the initial rapid improvement in reward is not present for that case study, as it was for case study 1.

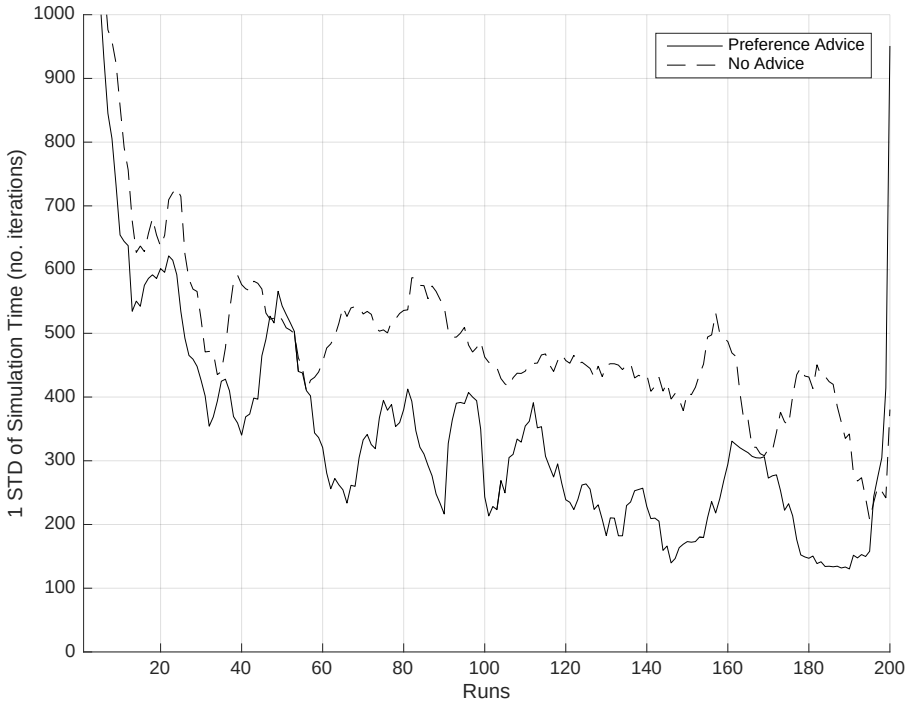


(a)

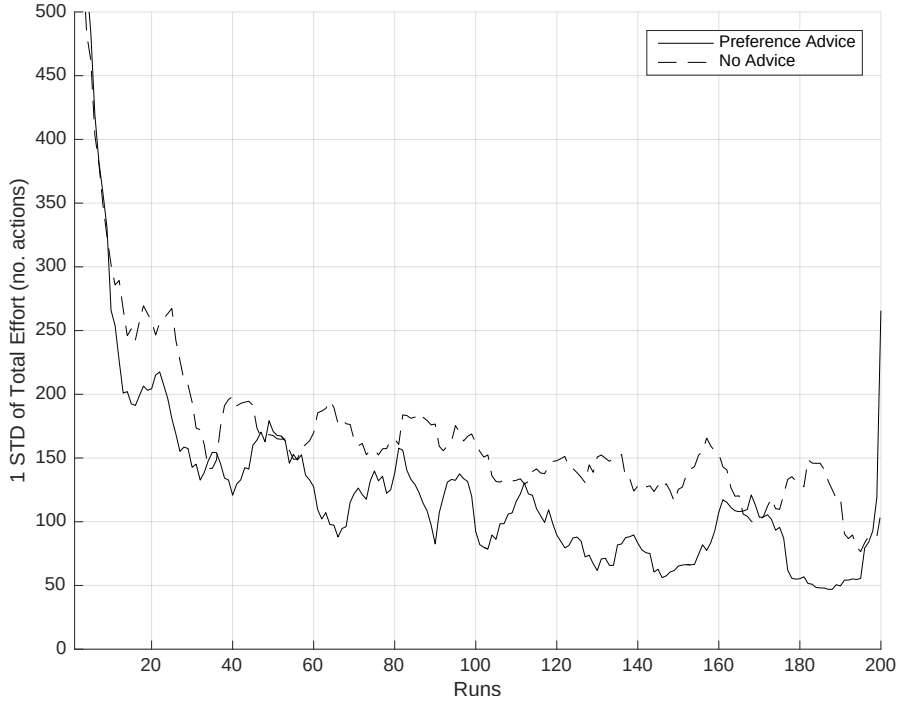


(b)

Figure 5.2: Performance for individual and team learning, with and without the Preference Advice mechanism in experiment 1: (a) simulation time, (b) total effort



(a)



(b)

Figure 5.3: Standard deviation of performance at each run in experiment 1: (a) simulation time, (b) total effort

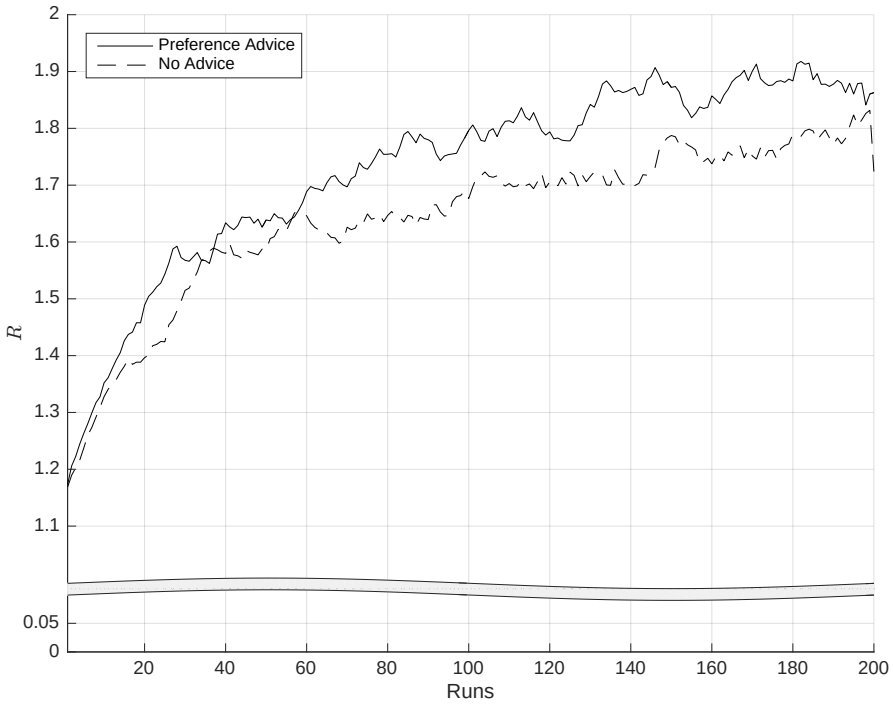


Figure 5.4: Average reward obtained between the 4 robots in experiment 1

5.3.2 Experiment 2: Preference Advice with team learning and measurement uncertainty

The results for experiment 2, which adds a zero mean Gaussian noise to all robot state variables, are shown in figures 5.5 to 5.10. Similar to experiment 1, the simulation time, total effort, and standard deviation of each metric are shown for each level of noise (standard deviations of 0.05m, 0.20m, and 0.40m). To accompany the results, each figure also contains the results from experiment 1 without noise and without advice for reference. It is worth noting that figures 5.7 through 5.10 have different axis scales than the remainder of figures within this case study, due to the substantial decrease in performance for large amounts of state uncertainty.

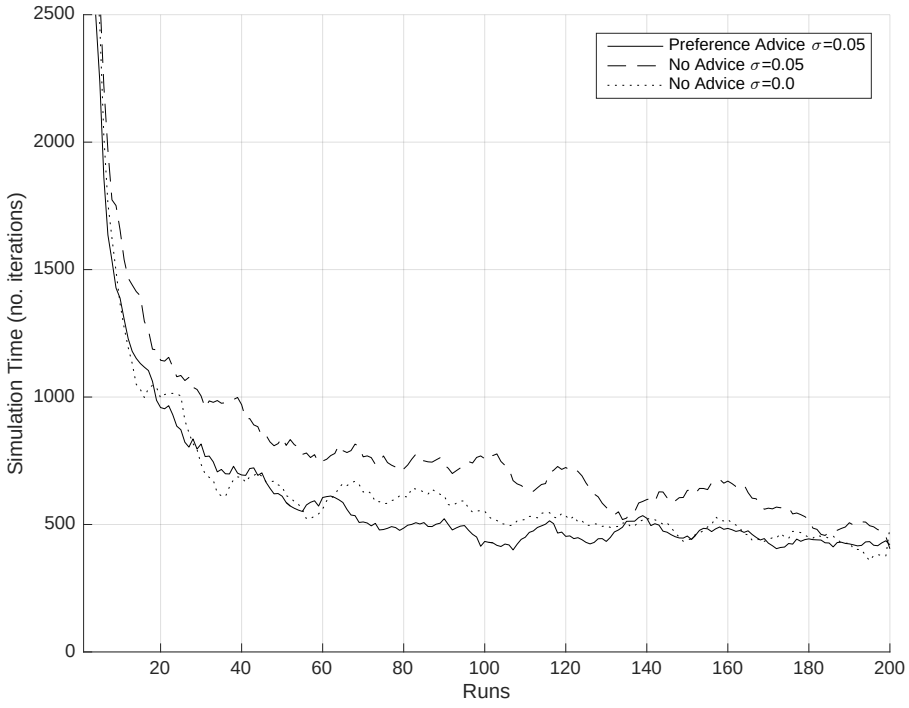
By observing the standard deviations of simulation time and total effort in figures 5.5b, 5.6b, 5.7b, 5.8b, 5.9b, and 5.10b, it is apparent that there is minimal distinction between the Preference Advice mechanism and no advice cases. Consequently, advice has little influence on the consistency of the team in the presence of uncertainty. However, there is a very noticeable improvement in both simulation time and total effort when the Preference Advice mechanism is used. Despite the variability in performance not being improved, there is still a large improvement in the mean performance. With 0.05m standard deviation noise, the use of advice improves the performance (simulation time

and total effort) to be approximately equivalent to the no advice case with zero noise (figures 5.5a and 5.6a).

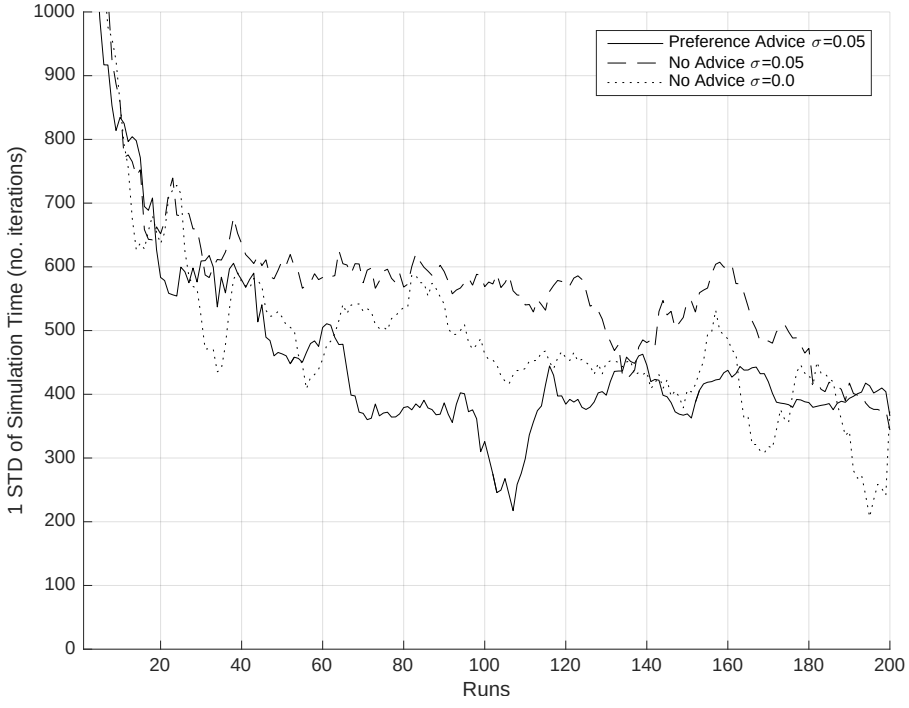
As the state uncertainty increases, the relative performance improvement with advice also increases, as can be seen in figures 5.7a, 5.8a, 5.9a, and 5.10a. A hypothesis for why the benefit of advice increases with state uncertainty is that the addition of state uncertainty has an equalizing effect on the reward and transition functions for each experience (s_I, a_I, s'_I). This results in a robot having less certainty about which action to select. In such a scenario the use of advice has an opportunity to substantially improve the confidence in action selection, This is in contrast to the case with no state uncertainty, where a robot can more easily converge to the true reward and transition functions, and hence have higher confidence in action selection.

The results from this experiment are the most significant for realistic robot teams, which will inevitably have some degree of uncertainty in every state estimate. It highlights in interesting observation that an advice mechanism that amalgamates the advice of multiple advisers can be analogous to a state estimator utilizing observations from multiple sources.

Additionally, it is worth noting that in figures 5.7b and 5.9b, the standard deviation of simulation time is extremely small during the first 5 runs. This is due to a limit being placed on the total allowed iterations, and the team consistently reaching this limit with large amounts of state uncertainty.

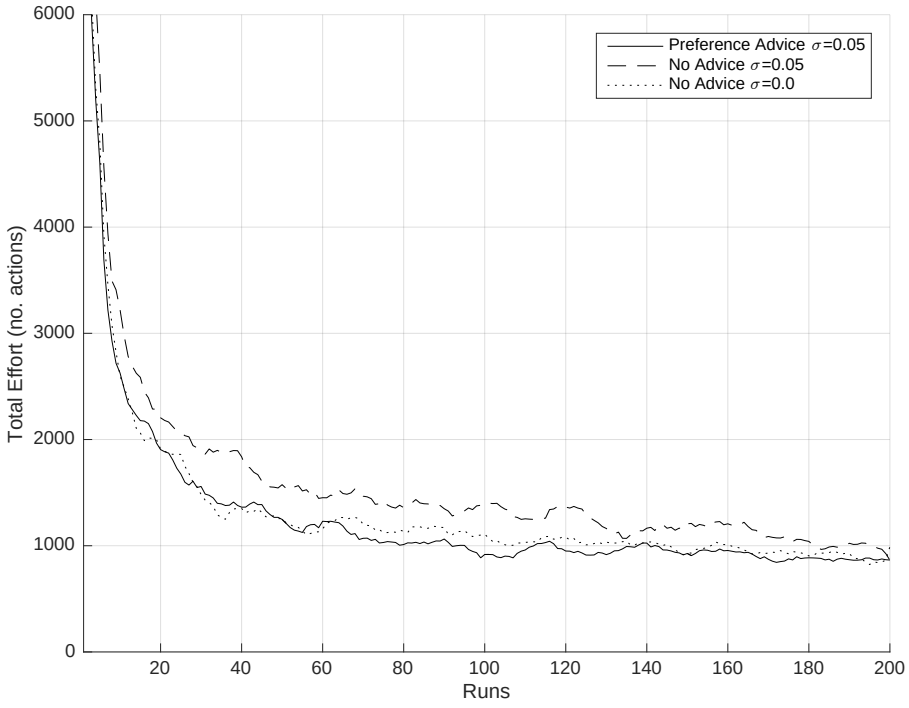


(a)

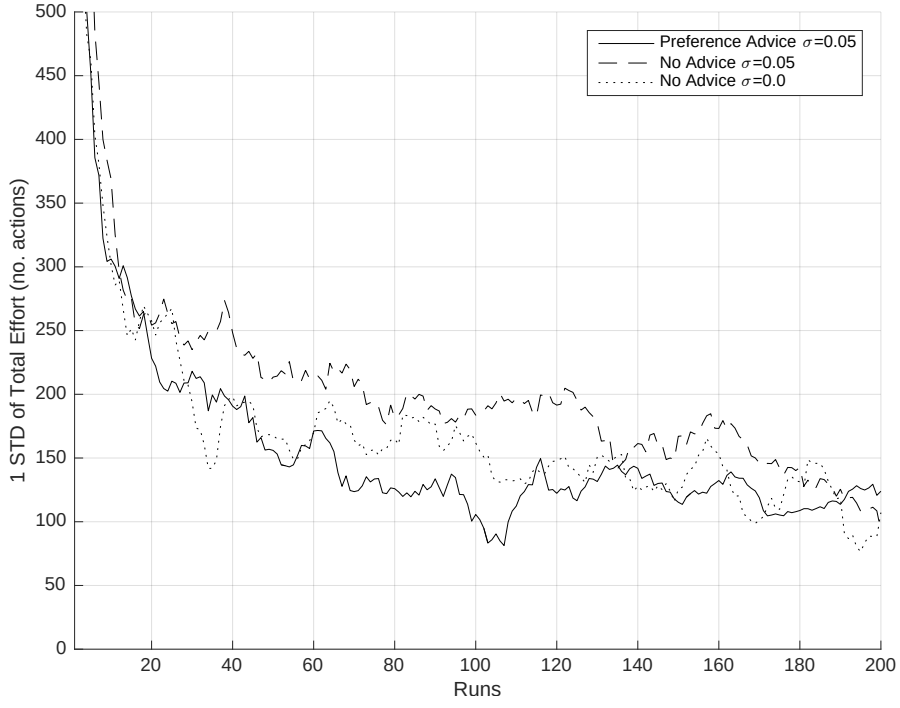


(b)

Figure 5.5: Simulation time with and without the Preference Advice mechanism with zero mean Gaussian noise with a standard deviation of 0.05m (experiment 2): (a) simulation time, (b) standard deviation of simulation time

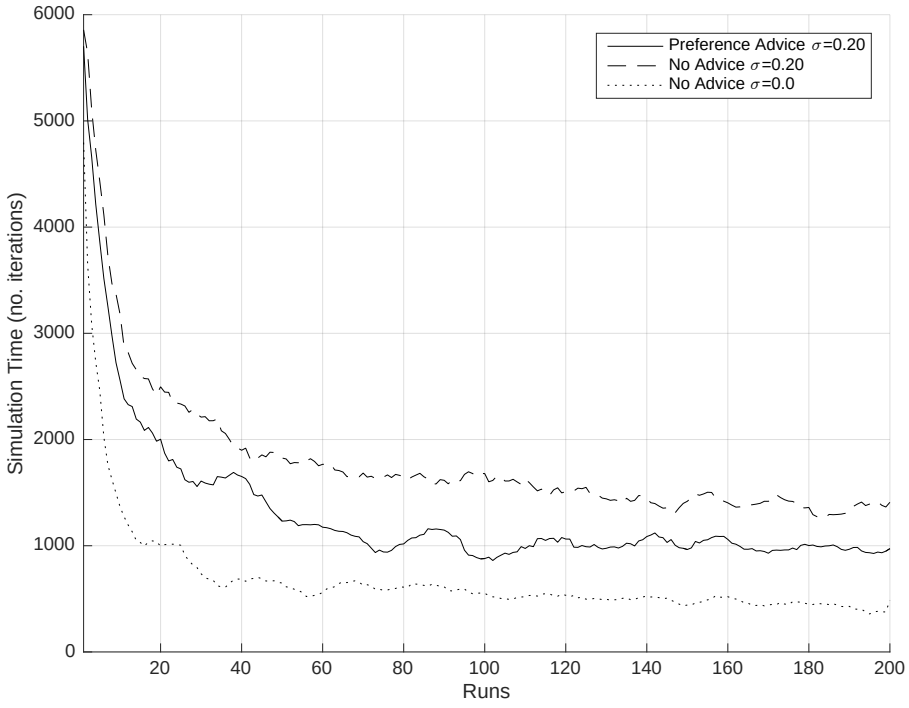


(a)

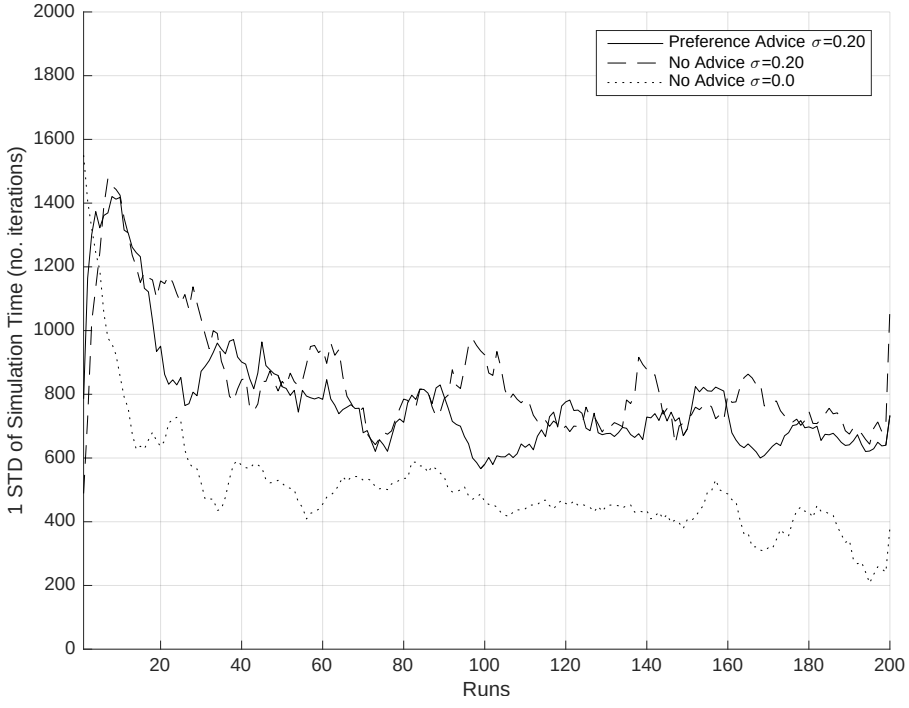


(b)

Figure 5.6: Team total effort time with and without the Preference Advice mechanism with zero mean Gaussian noise with a standard deviation of 0.05m (experiment 2): (a) simulation time, (b) standard deviation of simulation time

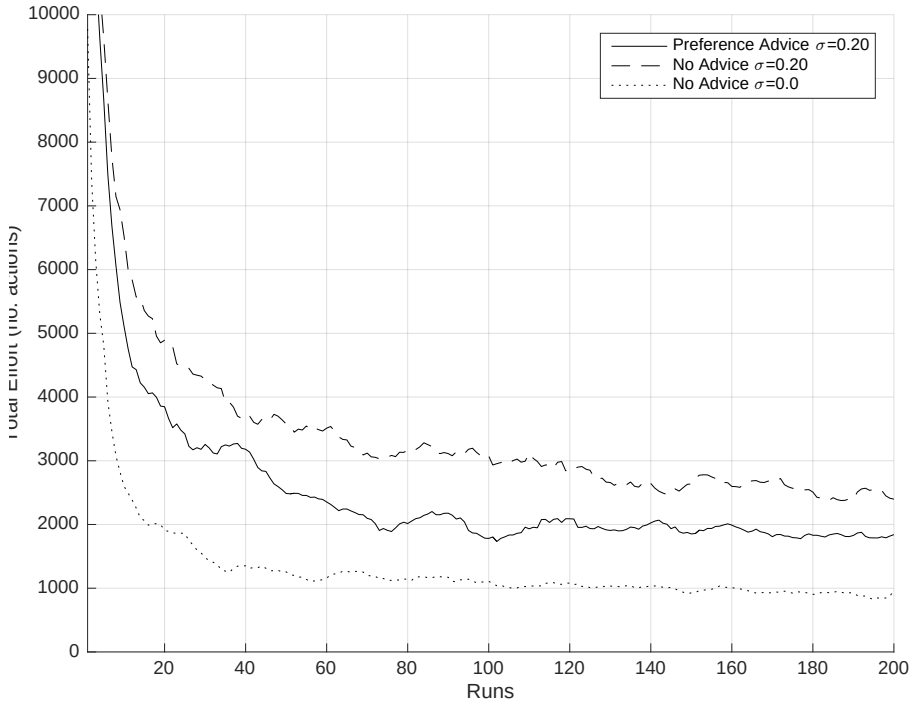


(a)

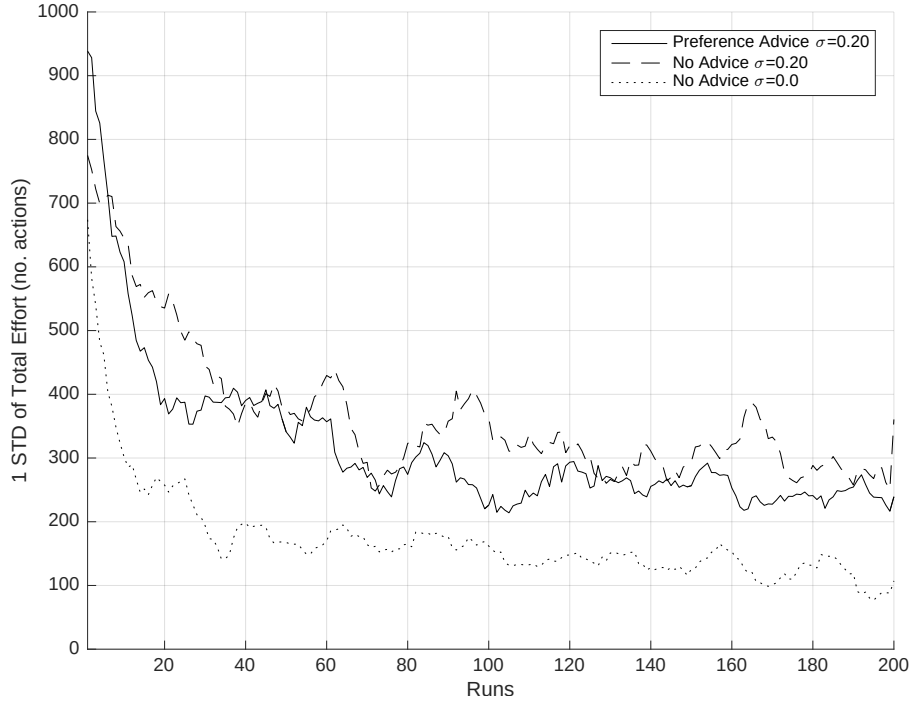


(b)

Figure 5.7: Simulation time with and without the Preference Advice mechanism with zero mean Gaussian noise with a standard deviation of 0.20m (experiment 2): (a) simulation time, (b) standard deviation of simulation time

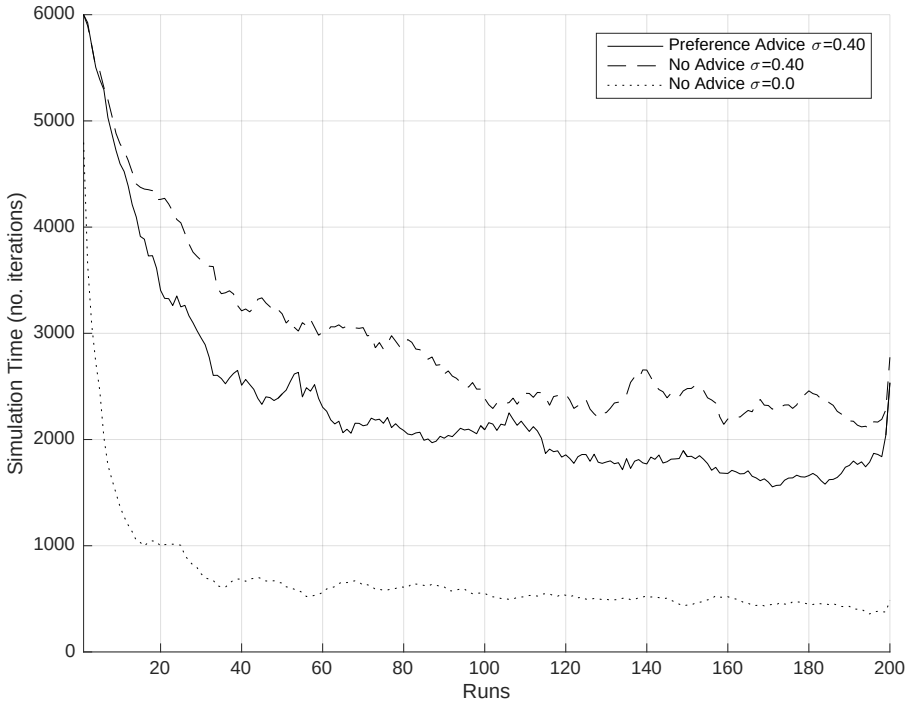


(a)

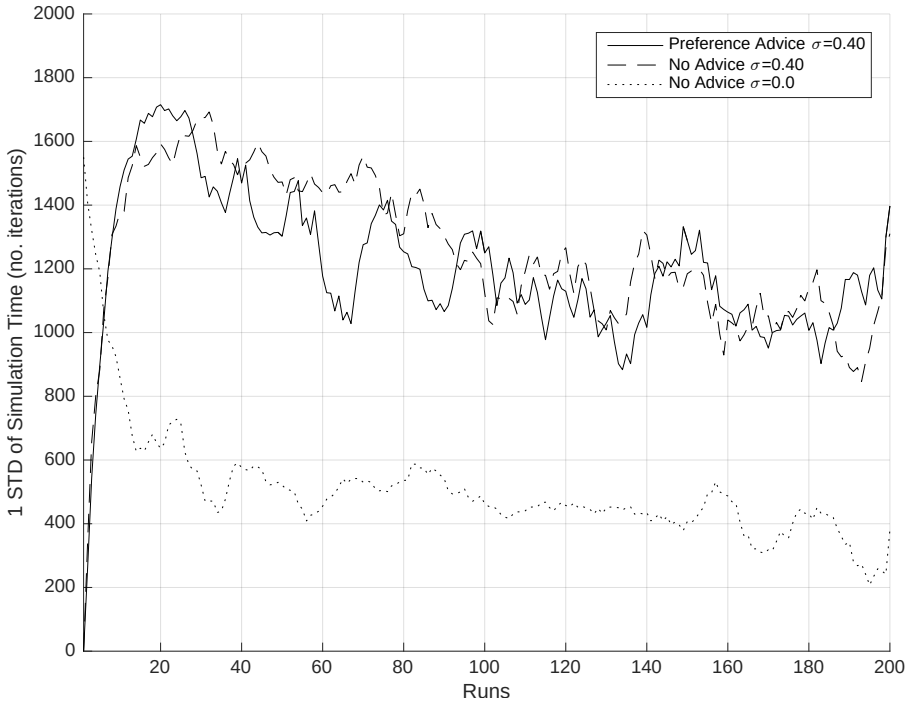


(b)

Figure 5.8: Team total effort time with and without the Preference Advice mechanism with zero mean Gaussian noise with a standard deviation of 0.20m (experiment 2): (a) simulation time, (b) standard deviation of simulation time

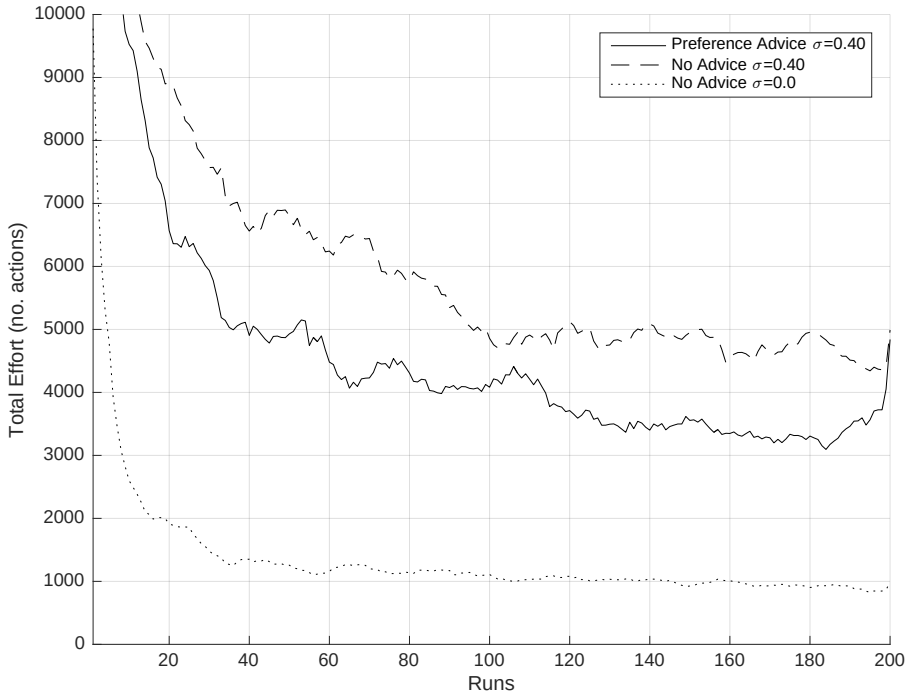


(a)

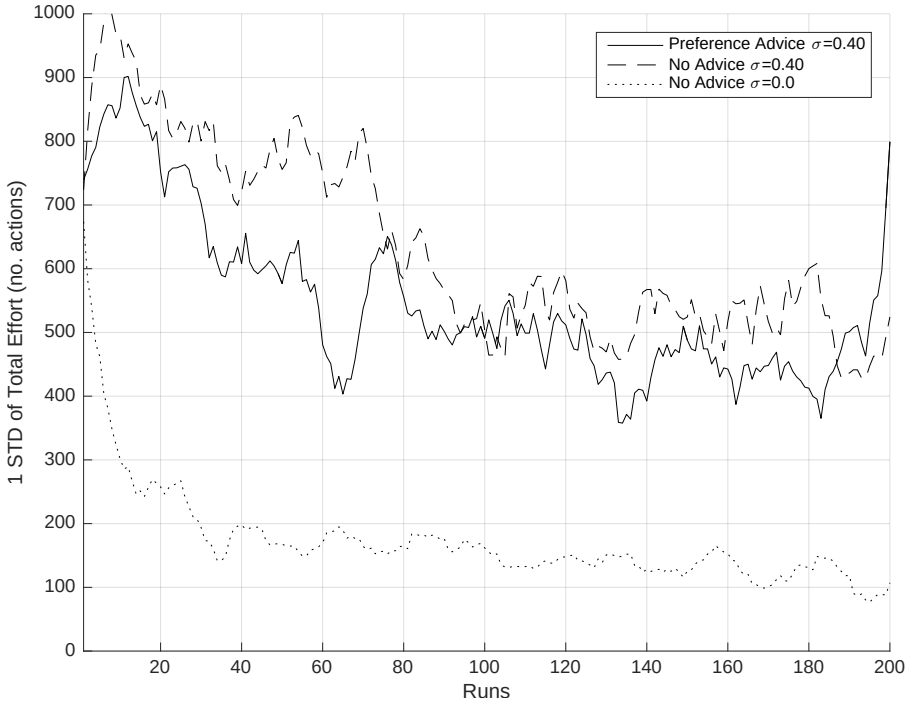


(b)

Figure 5.9: Simulation time with and without the Preference Advice mechanism with zero mean Gaussian noise with a standard deviation of 0.40m (experiment 2): (a) simulation time, (b) standard deviation of simulation time



(a)



(b)

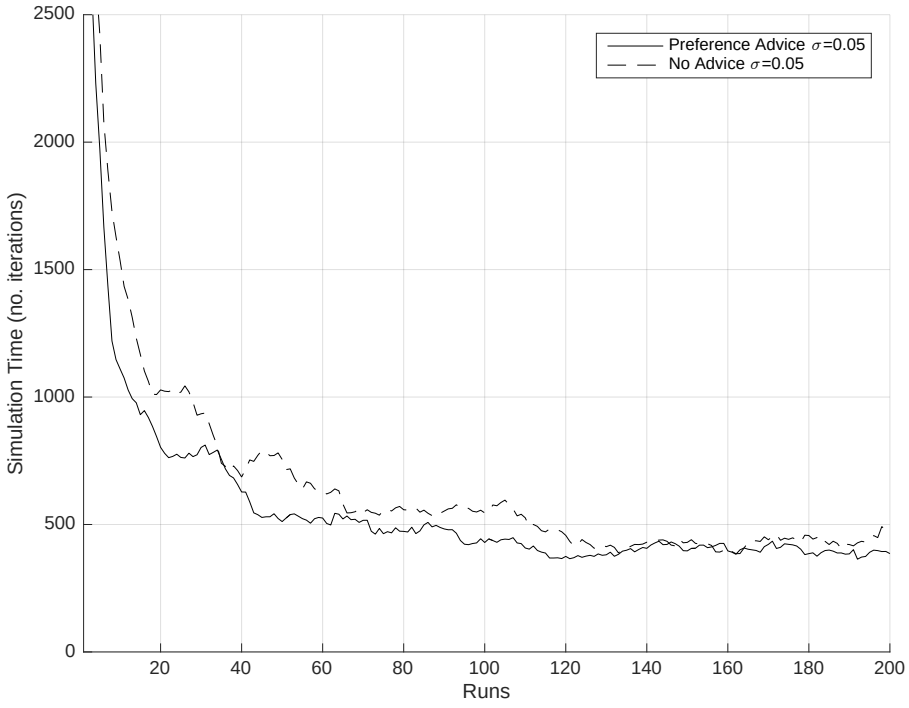
Figure 5.10: Team total effort time with and without the Preference Advice mechanism with zero mean Gaussian noise with a standard deviation of 0.40m (experiment 2): (a) simulation time, (b) standard deviation of simulation time

5.3.3 Experiment 3: Preference Advice with team learning, measurement uncertainty, and a state estimator

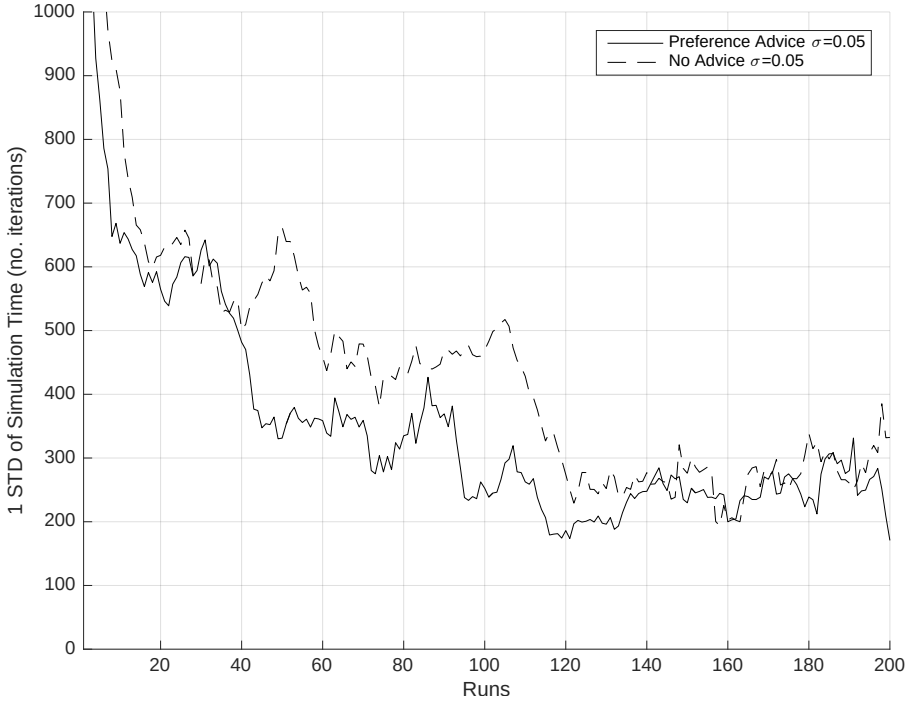
In experiment 3 a particle filter is used as a state estimator for each the robot's state variables that are subject to noise. Identical to experiment 2, the results for the team performance with and without advice for each level of sensory noise are shown in figures 5.11 to 5.16.

It is immediately apparent from the simulation time and total effort that the results for experiment closely mimic those of experiment 1. Particularly, the relative performance improvement achieved with the Preference Advice mechanism in the transient phase of learning, and the approximately equal performance during the steady state phase. Additionally, the relative performance improvement for simulation time and total effort with the Preference Advice mechanism is consistent across all three levels of uncertainty. Similar amount of improvement as in experiment 1 are also observed for the standard deviations of simulation time and total effort. The similarity in results to experiment 1 indicates that the particle filter is acting as an effective state estimator, and that the advice mechanism remains beneficial under such circumstances.

To compare the differences in performance for each level of uncertainty, the simulation time for all levels of uncertainty are plotted in fig. 5.17, where the no advice case is shown in fig. 5.17a, and the Preference advice case is shown in fig. 5.17b. For brevity, only the simulation time is shown. A slight increase in simulation time can be observed as the uncertainty increases, as expected since the effectiveness of a state estimator is a function of the amount of uncertainty in the system. It is worth noting that the performance difference between no noise, and noise of 0.40m standard deviation is more exaggerated for the no advice case than it is for the use of advice. Hence, the Preference Advice mechanism compensates for the shortcomings of the state estimator in highly uncertain situations.

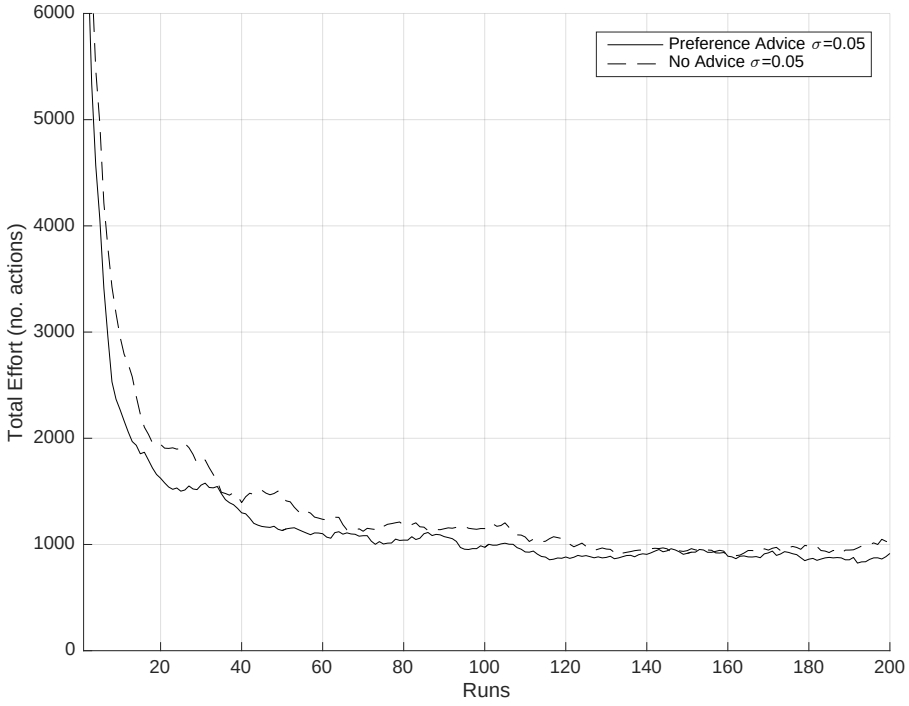


(a)

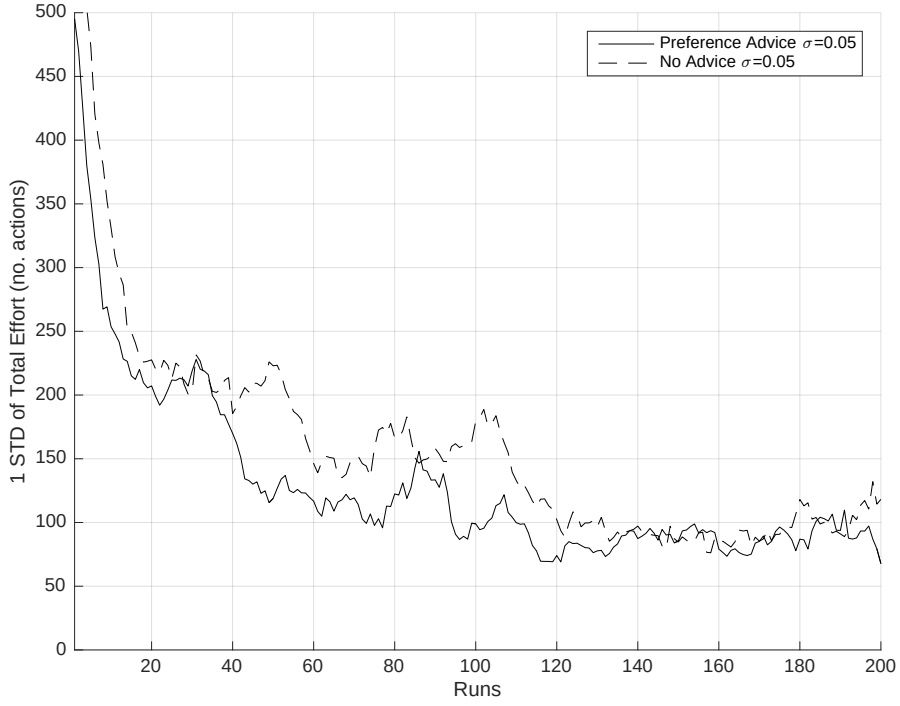


(b)

Figure 5.11: Simulation time with and without the Preference Advice mechanism, using a state estimator in the presence of zero mean Gaussian noise with a standard deviation of 0.05m (experiment 3): (a) simulation time, (b) standard deviation of simulation time

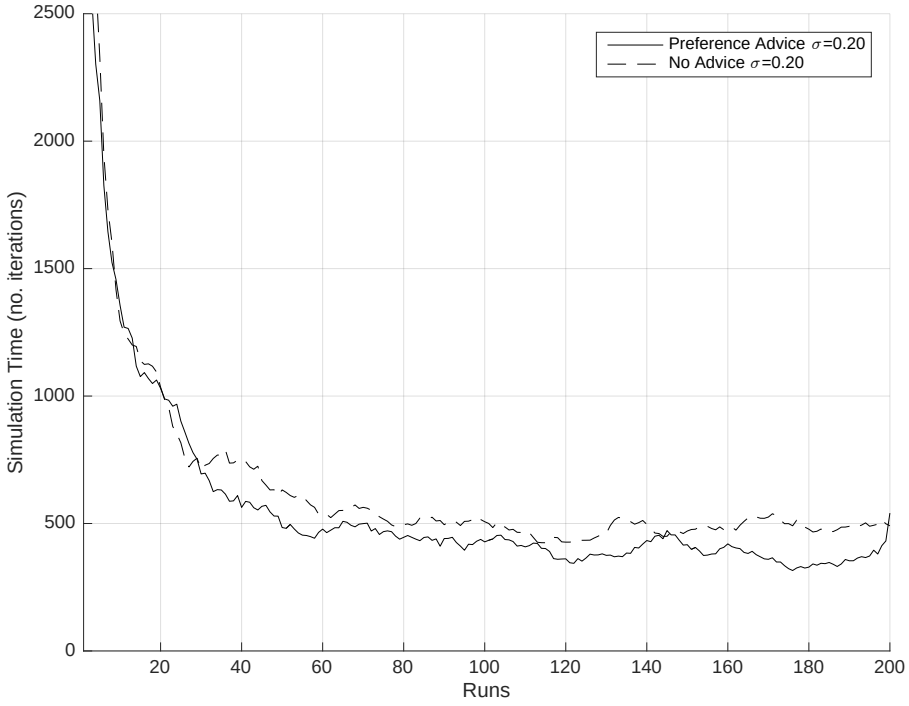


(a)

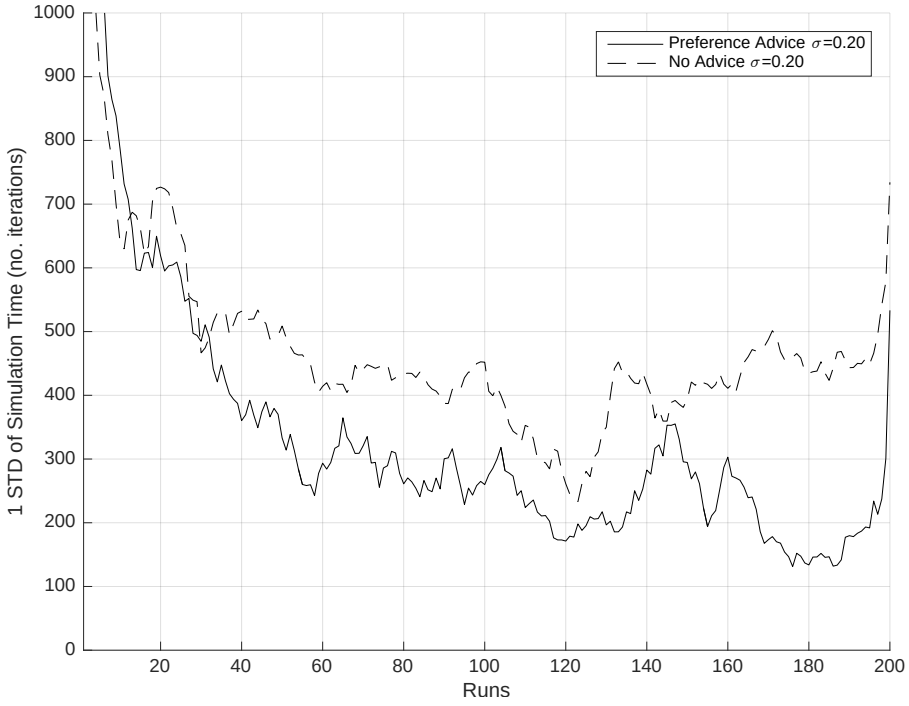


(b)

Figure 5.12: Team total effort time with and without the Preference Advice mechanism, using a state estimator in the presence of zero mean Gaussian noise with a standard deviation of 0.05m (experiment 3): (a) simulation time, (b) standard deviation of simulation time

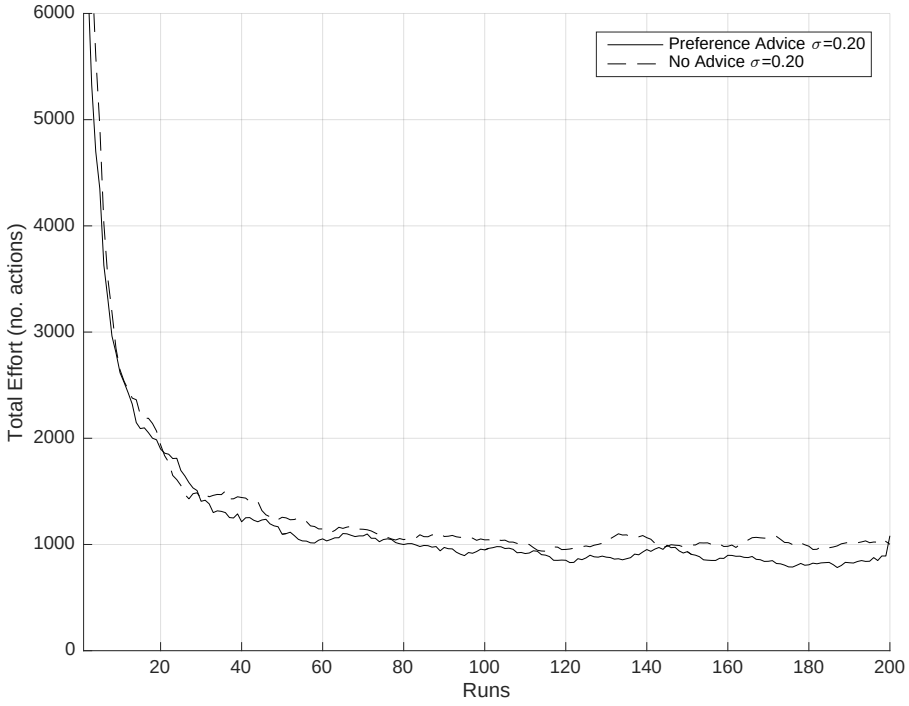


(a)

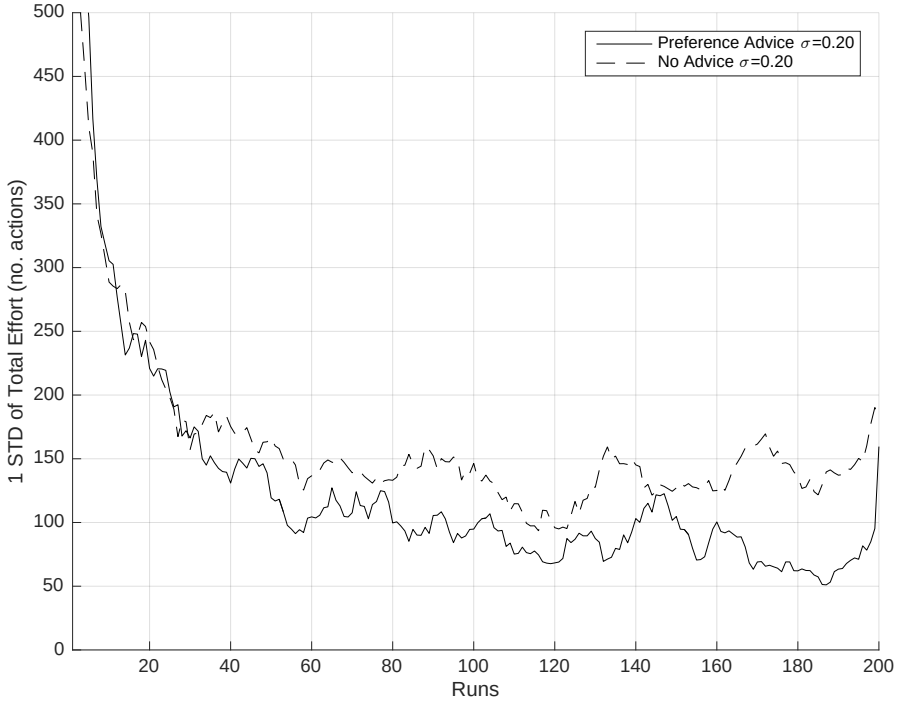


(b)

Figure 5.13: Simulation time with and without the Preference Advice mechanism, using a state estimator in the presence of zero mean Gaussian noise with a standard deviation of 0.20m (experiment 3): (a) simulation time, (b) standard deviation of simulation time

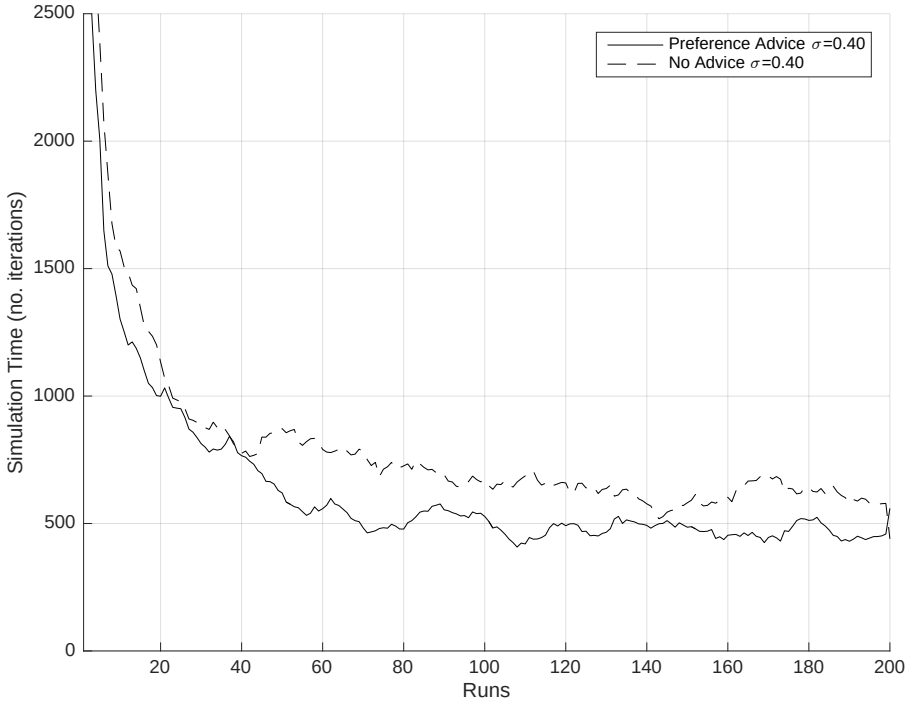


(a)

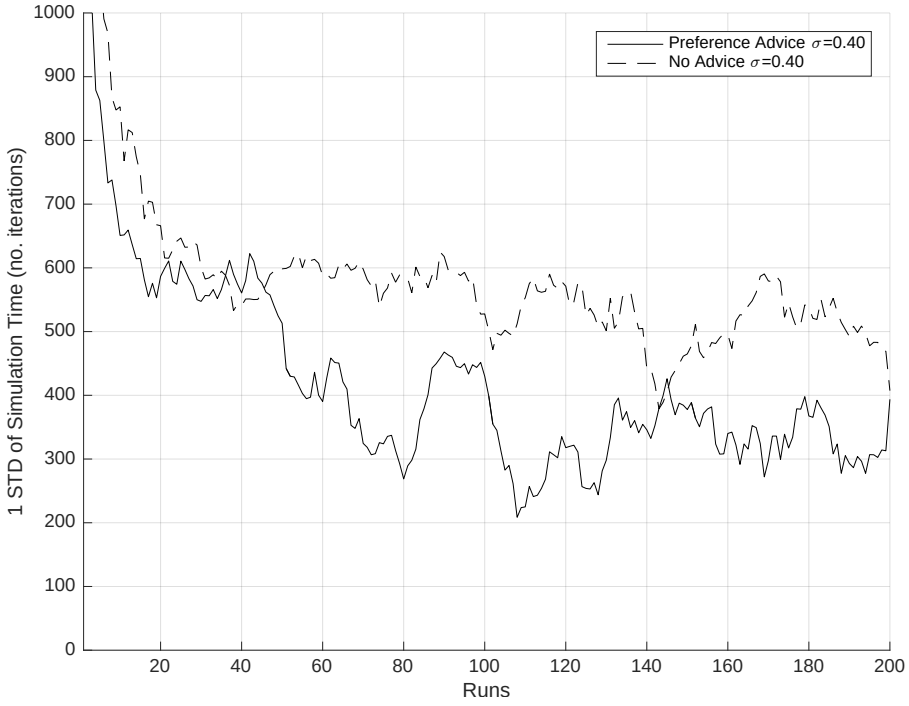


(b)

Figure 5.14: Team total effort time with and without the Preference Advice mechanism, using a state estimator in the presence of zero mean Gaussian noise with a standard deviation of 0.20m (experiment 3): (a) simulation time, (b) standard deviation of simulation time

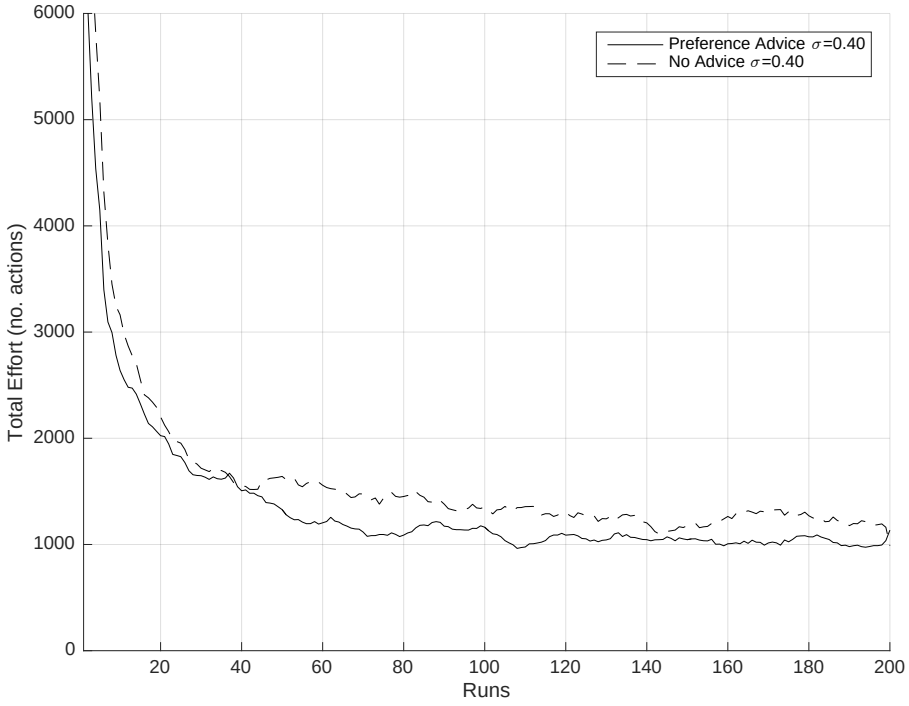


(a)

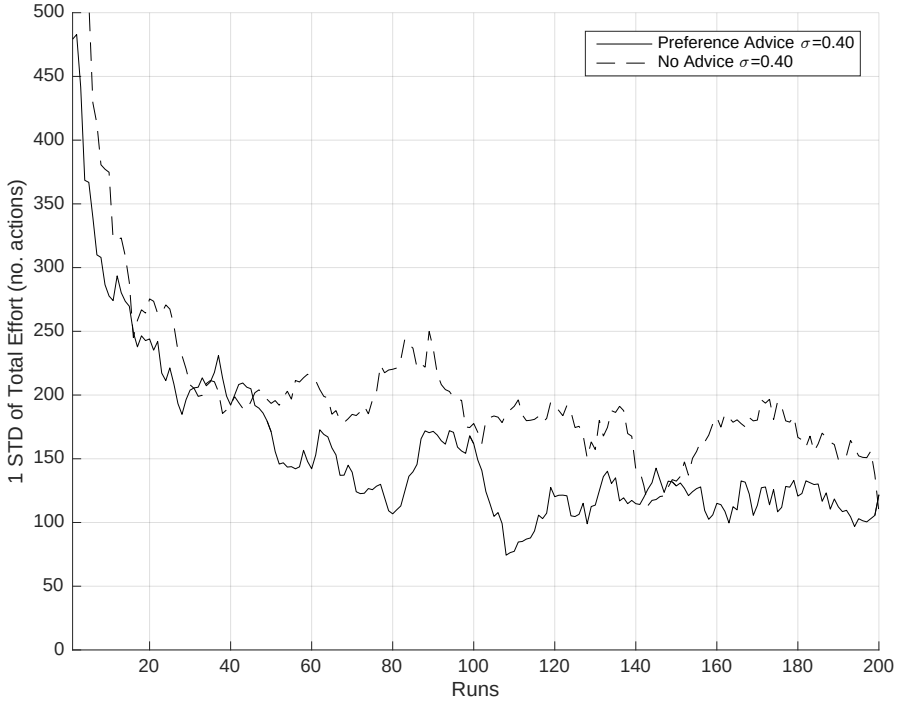


(b)

Figure 5.15: Simulation time with and without the Preference Advice mechanism, using a state estimator in the presence of zero mean Gaussian noise with a standard deviation of 0.40m (experiment 3): (a) simulation time, (b) standard deviation of simulation time

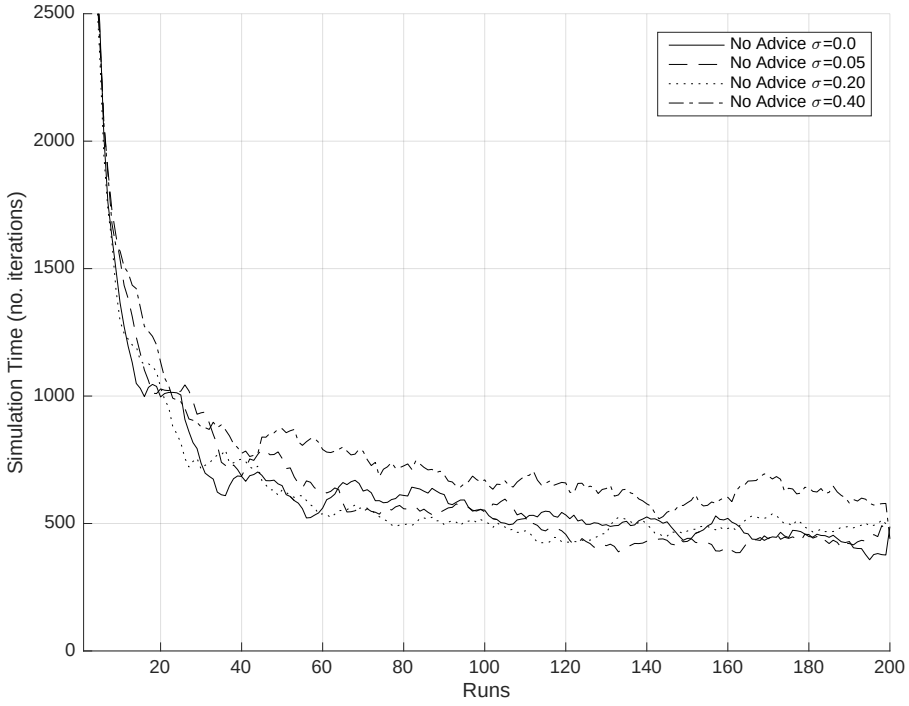


(a)

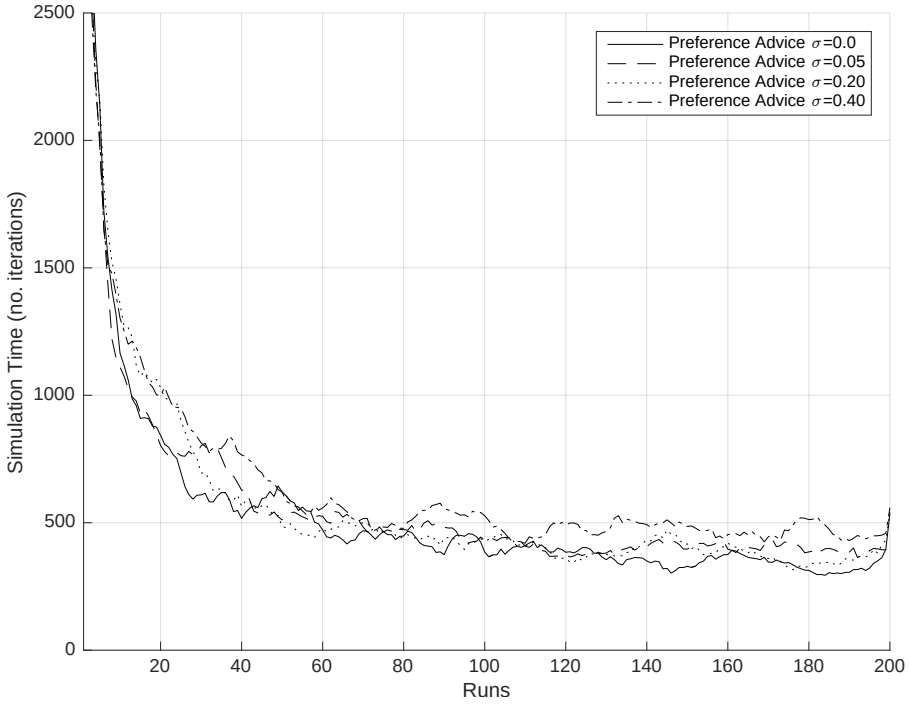


(b)

Figure 5.16: Team total effort time with and without the Preference Advice mechanism, using a state estimator in the presence of zero mean Gaussian noise with a standard deviation of 0.40m (experiment 3): (a) simulation time, (b) standard deviation of simulation time



(a)



(b)

Figure 5.17: Comparison of simulation time performance using a state estimator for all noise levels: 0.05m, 0.20m, and 0.40m (experiment 3): (a) No advice, (b) Preference Advice

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis has made two contributions to the progress of robot teams: the integration of collective, collaborative, and cooperative behaviour into an existing robot team architecture, and the development of an advice mechanism for heterogeneous robot teams.

The RCISL algorithms consisting of Q-learning for collective behaviour, Advice Exchange for collaborative behaviour, and RL-Alliance for cooperative behaviour, have been implemented within the HAA architecture by forming a new agent for each behavioural algorithm. Each behavioural algorithm has been implemented in a distributed fashion, and organized into a modular form that enables easy instantiation and removal from the control system for dynamically scaling the team. Additionally, each algorithm has been made robust to faults through routine backups of necessary state data in the DDB, and appropriate fallback behaviour for communication failures with other agents.

Experiments with the newly developed framework were performed with a foraging case study using a simulated team of heterogeneous robots. In the case study, 4 avatars would each develop a policy for collecting and delivering items, while receiving advice from other avatars, as well as develop a policy for appropriately allocating tasks. In each experiment, the rate of failures for agents and hosts were varied to observe the frameworks ability to handle faults, as well as the performance of the team at the foraging task in the presence of faults.

The new advice mechanism, called Preference Advice, has been developed for real world use of heterogeneous robot teams. Advice is incorporated into the advisee's policy by biasing the selection probability of each action. An important feature of this method is that it guarantees both convergence to an optimal policy at time infinity, and that the influence of advice diminishes to zero at time infinity. The Preference Advice mechanism

also has the capability to decide when advice should be requested and accepted, and to use multiple advisers at each time step. A ranking scheme is implemented to determine the order which advisers should be polled, and is based on the similarity between the advisee's and adviser's policies.

Experiments with the Preference Advice mechanism were performed with a simulated team of robots performing a foraging task. When a team of homogeneous robots with zero prior experience perform the foraging task, the mechanism accelerates the initial stage of the learning process in comparison to without advice, by reducing the simulation time (number of iterations) and total effort (number of actions) to complete the task. The improvement in performance observed also exceeds the improvement obtained with an alternative advice mechanism that uses a form of direct policy adoption. A reduction in the standard deviation of both simulation time and total effort was observed, indicating an improvement in the team's consistency. When the foraging task was performed with a team of heterogeneous robots, each possessing different capabilities, similar levels of improvement relative to the no advice case were observed for simulation time, total effort, and their respective standard deviations. The use of the Preference Advice mechanism was also studied with a single robot having virtual advisers of varying skill levels and experience. In both cases, the mechanism differentiated between advisers of different skill levels and capabilities through the implemented ranking scheme. Additionally, it was observed that the frequency which advice is requested decreases steadily from the first run, and the acceptance of advice reaches a peak during the transient stage of learning and steadily decreases afterwards.

The experiments have shown that the Preference Advice mechanism has three important attributes for use with real world robot teams. First, the convergence guarantees of the mechanism as well as the adviser ranking scheme enable a performance improvement to be obtained when heterogeneous advisers are used. Second, the usage of advice decreases over time, which reduces the communication and computational costs for the team. Third, an experienced adviser can further improve the benefit of advice. The experienced adviser could be a human being, which can help reduce the risk of an entire team performing potentially hazardous exploration.

The Preference Advice mechanism was also tested within the RCISL framework, i.e. with team learning and state uncertainty. A foraging case study was used, where heterogeneity exists in both the tasks to be performed as well as the robot capabilities. No restrictions were placed on which advisers could be utilized by any robot. Three experiments were performed: Preference Advice with team learning, Preference Advice with team learning and measurement uncertainty, and Preference Advice with team

learning and measurement uncertainty and a state estimator.

The Preference Advice mechanism was shown to have a consistent improvement in both simulation time and total effort, for all three experiments. The amount of improvement in performance (simulation time and total effort) with the use of advice was approximately equal between the experiments without noise, and with noise and a state estimator. However, a much larger performance improvement was observed for the case with noise but without a state estimator. Hence, when a robot can easily converge to the appropriate reward and transition functions through limited experience, advice has limited benefit. But for the cases where the reward and transition functions may take longer to converge to, such as having high state uncertainty, there is more potential for improvement and the effect of advice is more pronounced. An improvement in the standard deviations of simulation time and total effort were observed for the experiments without noise, and with noise and a state estimator. However no such improvement was observed for the case with noise but without the state estimator. Hence, the Preference Advice mechanism improves the mean performance under all three circumstances tested, but it only improves the team's consistency in the scenarios with low state uncertainty.

6.2 Future Work

There are a variety of extensions to the Preference Advice case studies that would provide additional insight. One extension would be to increase the amount of heterogeneity within the team and observe if the performance with the Preference Advice mechanism ever becomes worse than the no advice case. This would of course require maintaining one robot of each type to ensure all potential advisers are different from the advisee. Another extension would be to add increasing amounts for complexity to the environment, resulting in a more difficult task to learn, and observe the relationship between task difficulty and the performance benefit with advice. Lastly, it would be interesting to observe the performance for a larger team of robots, multiple robots of each type. Providing advisers that are learning the identical policy should further enhance the benefit of advice.

The presented mechanism is accompanied with a proof of convergence. However, it would be interesting to attempt to theoretically demonstrate an improvement in the rate of convergence, or simply to identify the constraints that would ensure an improvement in the convergence rate. The rate of convergence for Q-learning has been investigated in [20], which could provide a useful starting point.

Regarding the incorporation of advice into the advisee's policy, it would be desirable to

discover a more theoretically grounded method of biasing an agent's policy. An interesting approach to advice incorporation would be a form of Bayesian update, utilizing the advisee's initial information as the prior, and forming a posterior after incorporating the adviser's input. A potential approach to this may be to view action selection probabilities as a multinomial distribution, and use the Dirichlet distribution as a conjugate prior.

Bibliography

[1] T. Arai, E. Pagello, and L. E. Parker. Guest editorial advances in multirobot systems. *IEEE Transactions on Robotics and Automation*, 18(5):655–661, Oct 2002.

[2] T. Balch and R. C. Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, Dec 1998.

[3] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[4] Georgios Boutsioukis, Ioannis Partalas, and Ioannis Vlahavas. *Transfer Learning in Multi-Agent Reinforcement Learning Domains*, pages 249–260. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[5] Justin Girard and M. Reza Emami. Concurrent markov decision processes for robot team learning. *Engineering Applications of Artificial Intelligence*, 39:223 – 234, 2015.

[6] Justin Girard and M. Reza Emami. A robust approach to robot team learning. *Autonomous Robots*, pages 1–17, 2015.

[7] Tyler Gunn and John Anderson. Dynamic heterogeneous team formation for robotic urban search and rescue. *Procedia Computer Science*, 19:22 – 31, 2013. The 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013), the 3rd International Conference on Sustainable Energy Information Technology (SEIT-2013).

[8] Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Comput.*, 6(6):1185–1201, November 1994.

[9] E. G. Jones, B. Browning, M. B. Dias, B. Argall, M. Veloso, and A. Stentz. Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks. In

Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on, pages 570–575, May 2006.

[10] Nidhi Kalra, Robert Zlot, M. Bernardine Dias, and Anthony Stentz. Market-Based Multirobot Coordination: A Comprehensive Survey and Analysis. *Robotics Institute Carnegie Mellon University Tech Rep CMURITR0516*, 94(December 2005):48, 2006.

[11] G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, 2013.

[12] Long-Ji Lin. Programming robots using reinforcement learning and teaching. In *Proceedings of the Ninth National Conference on Artificial Intelligence - Volume 2*, AAAI'91, pages 781–786. AAAI Press, 1991.

[13] Andrey Loukianov, Masanori Sugisaka, Hidenori Kimura, Msanori Sugisaka, and H. Kimura. Implementing distributed control system for intelligent mobile robot. *Artificial Life and Robotics*, 8(2):159–162, 2004.

[14] Richard Maclin, Jude Shavlik, Lisa Torrey, Trevor Walker, and Edward Wild. Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2*, AAAI'05, pages 819–824. AAAI Press, 2005.

[15] Richard Maclin, Jude W. Shavlik, and Pack Kaelbling. Creating advice-taking reinforcement learners. In *Machine Learning*, pages 251–281, 1996.

[16] R. J. Malak and P. K. Khosla. A framework for the adaptive transfer of robot skill knowledge using reinforcement learning agents. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 1994–2001 vol.2, 2001.

[17] Adrian Martin and M. Reza Emami. A fault-tolerant approach to robot teams. *Robotics and Autonomous Systems*, 61(12):1360–1378, 2013.

[18] Adrian Martin and M. Reza Emami. A dynamically distributed control framework for robot teams. *International Journal of Robotics and Automation*, 29(3):312–318, 2014.

[19] John McCarthy. Programs with common sense. In *Semantic Information Processing*, pages 403–418. MIT Press, 1968.

[20] Larry Ng. *Concurrent Individual and Social Learning in Robotic Teams*. PhD thesis, University of Toronto Institute for Aerospace Studies, Toronto, Canada, November 2012.

[21] Luís Nunes and Eugénio Oliveira. *Cooperative Learning Using Advice Exchange*, pages 33–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[22] Luís Nunes and Eugénio Oliveira. *Exchanging Advice and Learning to Trust*, volume 2782 of *Lecture Notes in Computer Science*, pages 250–265. Springer-Verlag, 2003.

[23] L. E. Parker. Alliance: an architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, Apr 1998.

[24] Lynne E. Parker. Distributed intelligence: Overview of the field and its application in multi-robot systems. *Journal of Physical Agents*, 2(1):5–14, 2008.

[25] E. Prassler and K. Nilson. 1,001 robot architectures for 1,001 robots [industrial activities]. *IEEE Robotics Automation Magazine*, 16(1):113–113, March 2009.

[26] Himanshu Raj, Balasubramanian Seshasayee, Keith J. O'Hara, Ripal Nathuji, Karsten Schwan, and Tucker Balch. *Spirits: Using Virtualization and Pervasiveness to Manage Mobile Robot Software Systems*, pages 116–129. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[27] Florian Rohrmüller, Dirk Wollherr, and Martin Buss. MuRoCo: A framework for capability- and situation-aware coalition formation in cooperative multi-robot systems. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 67(3-4):339–370, 2012.

[28] Gavin Adrian Rummery. *Problem solving with reinforcement learning*. PhD thesis, University of Cambridge Ph. D. dissertation, 1995.

[29] P. E. Rybski, S. A. Stoeter, M. Gini, D. F. Hougen, and N. P. Papanikolopoulos. Performance of a distributed robotic system using shared communications channels. *IEEE Transactions on Robotics and Automation*, 18(5):713–727, Oct 2002.

[30] Satinder Singh, Tommi Jaakkola, Michael L. Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308, 2000.

[31] Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.

[32] Lisa Torrey, Jude Shavlik, Trevor Walker, and Richard Maclin. *Skill Acquisition Via Transfer Learning and Advice Taking*, pages 425–436. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[33] Christopher J.C.H. Watkins and Peter Dayan. Technical note: Q-learning. *Machine Learning*, 8(3):279–292.

[34] Steven D. Whitehead. A complexity analysis of cooperative mechanisms in reinforcement learning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 607–613, Anaheim, 1991.