Supporting AI Planning and Collaboration for Robotic Applications Using TÆMS*

Shelley Zhang[1,2], Alexander Moulton[2], Abhijot Bedi[2], Eugene Chabot [3]

[1] Gordon College, Wenham, MA
[2] University of Massachusetts Dartmouth, MA
[3] NUWC Division Newport, RI
shelley.zhang@gordon.edu, amoulton@umassd.edu, abedi@umassd.edu, eugene.j.chabot.civ@us.navy.mil

Abstract

This paper describes a general approach to integrating higher-level reasoning mechanisms including planning and scheduling methods with lower-level robotic control processes. We adopt a domain-independent task representation language TAEMS to describe the knowledge of tasks, resources, and their interrelationships. This TAEMS representation language serves as the input of the reasoning functions, which generate a schedule of executable methods to be executed by the robot in the physical world. In the execution process of this goal-directed plan, the robot also needs to attend to basic functions. The potential interactions between the plan and these basic functions would lead to interesting challenges that will be discussed. An integrated development platform with a simulator that supports real-world physics is also presented.

Introduction

Autonomous robots have been used to perform complex tasks in domains (Darmanin and Bugeja 2017) that have been deemed too dangerous, tedious, or out of reach for human intervention for nigh on 50 years. From the frontiers of space to the depths of the ocean, autonomous robots have been deployed to make the impossible possible by operating with decision-making capabilities to rival that of our own. Increasingly complex and expansive tasks require a higher order of planning and cooperation.

Autonomous robots have been researched and deployed in a multitude of various applications and capacities. Most of these approaches use a low-level approach to the performance of tasks such as behavior-based (Arkin 1998; Parker 1995) or rule-based decision-making (Marti et al. 2009). As researchers (Wahde 2009) have pointed out earlier, such basic control mechanisms are insufficient to solve realistic problems, nor to support efficient cooperation among multiple robots. Many cooperation approaches also rely on a centralized component to process and direct the actions of others such as swarms based on biological flocks (Lee and Chong 2008). Such centralized approaches usually suffer from single-point failure problems and lack the flexibility to handle environmental dynamics. A distributed multi-robot architecture DIRA has been proposed in (Simmons et al. 2001; Parker, Rus, and Sukhatme 2016), where each robot has three layers of control including planner, executive, and behaviors. Robots can interact with each other at each of these layers. This hierarchical idea resembles our approach through this early work is limited to fixed agent teams and task allocations, and the planning is restricted from supporting more complex tasks and constraints.

On the other hand, software agent control has been studied in the AI community from the deliberation perspective with planning and scheduling mechanisms. Additionally, the multi-agent community has examined the cooperation among multiple agents on topics including task allocations (Liang and Kang 2016), multi-agent planning, Generalized Partial Global Planning (Lesser and Corkill 2014), auction and HTN planning (Milot et al. 2021). Partially Observable Markov Decision Processes (POMDPs) are another framework to model dynamic processes with uncertainty used for robot planning (Castellini, Marchesini, and Farinelli 2019) though the computational cost is challenging. Most of these works focus on reasoning and deliberation of goals and tasks but leave out the physical world complexities such as communication limitations and environmental uncertainties. Therefore, the objective of this study is to develop efficient robot control and coordination mechanisms with both the higher-lever deliberations on goals and tasks as well as the lower-level consideration of physical world complexities.

Figure 1 describes the overview of this framework. First, we adopt domain-independent task representation language TÆMS to represent the knowledge of tasks, resources, and their interrelationships. This domain-independent representation language serves as the input of the reasoning tools

*This material is based upon work supported by the Office of Naval Research (ONR) through two contracts with UMass Dartmouth (N00014-21-1-2236 and MUST III S31320000053405). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of ONR or the U.S. Government.

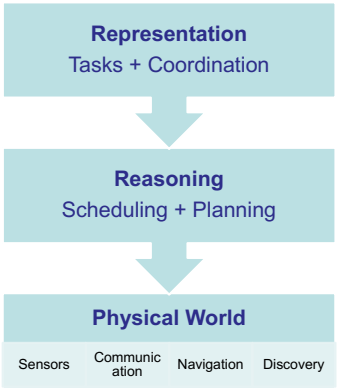

Figure 1: Framework Overview

including scheduling and planning functions. The output from the reasoning toolkit, which is a schedule of executable methods, is passed to the robot in the physical world. In the execution process of this goal-directed plan, the robot also needs to attend to basic functions such as sensing, communication, navigation, and discovery. The potential interactions between the plan and these basic functions would lead to interesting challenges that we will discuss later.

We will describe this process in greater detail in the rest of this paper. We will also present an integrated development platform to support the development of agent prototypes with a simulator environment with real-world physics.

TÆMS Formal Representation of Tasks and Environment

TÆMS (Task Analysis, Environment Modeling, and Simulation) is a framework for modeling complex task environments from both an agent-centric subject view and an objective view of the real problem-solving situation. Figure 2 shows a TÆMS example graphically in Robot Rescue (Gohardani, Mehrabi, and Ardestani 2016), where a group of heterogeneous agents, Police Force, Ambulance, and Fire Brigate are working to rescue civilians in a disaster scenario.

Each node in TÆMS graph represents a *task*, which may be further decomposed as a set of subtasks. A task with no subtask is referred to as a *method*, the smallest element for scheduling and direct execution. For example, the task *Rescue Civilians in Building 1* has two subtasks, first *Pickup Civilians in Building 1* and then *Deliver Civilians to Hospital*. The quality accumulation function (QAF) associated with the task node *Rescue Civilians in Building 1* is *seq_min*, which specifies that the two subtasks, *Pickup* and *Deliver* need to be executed in a sequence order; in addition, the minimum quality associated with any of these subtasks would be counted as the quality achieved for the task node. TÆMS defines various QAFs to specify how the quality of a task's subtasks shall be used to calculate the quality of the task itself, including the following:

- *Max*: the quality of the task is equal to the maximum quality of any one of its subtasks;
- *Min*: the quality of the task is equal to the minimum quality of any one of its subtasks;
- *Sum*: the quality of the task is equal to the sum of the qualities of its subtasks;
- *Sum All*: similar to *Sum*, with the additional requirement that all the subtasks must be completed for the task to have quality.

In addition, *Seq_Max*, *Seq_Min*, and *Seq_Sum* are similar to *Max*, *Min* and *Sum All* described above except the subtasks need to be executed in a sequencing order.

The outcome of the execution of each method is described from three perspectives: quality, cost, and duration. Duration is the time length of the method execution, the quality and cost may be assigned to domain-specific meaning. For example, in this Robot Rescue domain, quality may be defined by the importance and contribution towards the final evaluation goal, which is to maximize the number of saved civilians and their health scores. Cost may be defined as resource consumption.

Given the stochastic nature of the environment, the method execution outcome is described as a probability distribution over possible values. For example, method *Remove Blockade on Road A* has its quality outcome specified as:
$(25\%\ \ 0)(75\%\ \ 10.0)$
which means there is a probability of 0.75 that this method execution would produce 10.0 units quality but it may fail with a probability of 0.25 hence producing 0 unit quality. Such probability distribution would be built in the agent as pre-knowledge if available; which could also be learned and updated by the agent from the real-world experience.

Time constraints may also be specified for task and method with *earliest_start_time* (est) and *deadline* (dl), indicating the earliest possible time at which the method can be executed and the latest possible time at which the method should be completed respectively.

Besides these task-associated constraints described above, there are also *non-local effects* to characterize the influence of task A on task B:

- enables: task B can only start after task A has succeed.
- disables: task B cannot be executed after task A succeed.
- facilitates: the execution of task A helps task B by increasing its quality, or reducing its cost or duration.
- hinders: the execution of task A makes task B more difficult by reducing its quality, or increasing its cost or duration.

In the Robot Rescue example, the Police Force needs to clear the road so that Fire Brigade may move to the building to extinguish fire and then the Ambulance may pick up civilians from that building. Such a complex task scenario is depicted in Figure 2 with *enables* relationship: both task *Remove Blockade on Road B* and task *Extinguish fire on Building 1* enable task *Pickup Civilian in Building 1*.

Though the initial task structure with outcome data is provided as pre-knowledge to each agent, this information may be updated by the agent during execution when the agent learns from its experience with the environment. New tasks and methods may also be created and incorporated into the current task tree according to dynamic needs.

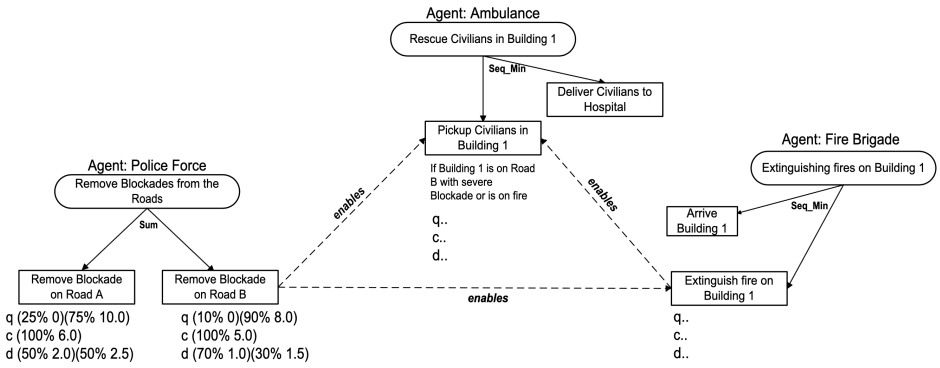

Figure 2: Robot Rescue Example in TÆMS Representation

Intelligent Scheduler and Collaboration Toolkit

As we presented in the previous section, TÆMS framework supports the modeling of complex scenarios with alternative ways to accomplish goals, the interrelationship among tasks and agents, soft-realtime constraints, and uncertainty description of the environment. Domain-dependent intelligent schedulers with planning capability may be developed to produce a sequence of methods with specified start and finish times, considering specific evaluation criteria. The complexities of the task representations prohibit an exhaustive search for an optimal outcome, hence the planning scheduler would exploit various heuristics to produce approximate solutions guided by the user-specified criteria (Wagner, Garvey, and Lesser 1998).

The top portion of Figure 3 depicts the TÆMS task trees for two agents: Police A and Police B in the Robot Rescue Scenario. Police A must first move to the target blockage in order to remove it. There are two ways to remove the blockage, either by itself (*Remove-Blockage-Self*) or as a team that requires help from Police B (*Remove-Blockage-Team*). The difference between these two approaches is that the team approach produces a higher quality (10 v.s. 6) with less cost and shorter duration (5 v.s. 10).

The left bottom portion describes the schedule evaluation criteria with three dimensions. In this shown example, quality is the most important factor with a weight of 0.5, followed by duration with a weight of 0.4 and cost is the least important factor. This criterion is used by the intelligent scheduler to rank alternative schedulers. The best schedules according to this criterion are returned.

The right bottom portion shows two schedules. The first schedule does not need coordination, finishing at time 11 with a quality of 7 and a cost of 11. The second schedule needs the collaboration from another Police B, it finishes at time 6, with a quality of 11 and a cost of 6. Noted this cost is only the cost of Police A. Additional cost would occur from Police B when it helps, but that amount is not known to Police A nor considered by Police A.

A collaboration toolkit would evaluate this collaboration schedule compared with the best alternative without coordination, and find out that, with this collaboration approach, the local utility is increased by 2 units. Therefore, a coordination request is generated shown in Figure 4.

This request is sent to Police B for asking help to remove the blockage as a team with a deadline of 6. Police B then uses its collaboration toolkit to evaluate this request by comparing its best plan with this request and without this request. Usually helping another agent may cause the loss of its local utility. However, if this loss is less than the gain of Police A, police B will grant this request. Otherwise, Police B will reject this request. In the latter case, Police B may also propose an alternative by offering to help at a time different from what is requested. Police A then re-evaluates this alternative to see if it still is valid.

It's worth noting that TÆMS framework adopts a soft real-time (Erickson and Anderson 2022) assumption which means that missing deadlines sometimes are permitted, though it may bring different utility in solving the problem. This is different from a hard real-time system where a valid solution must ensure no task misses a deadline. Hence Police A may evaluate the alternative completion time proposed by Police B to see how it fits into its local view of the problem-solving scenario.

Both the intelligent scheduler and the collaboration toolkit are domain-independent and, therefore may be applicable to various applications as long as the problems are described with the TÆMS framework.

Robot Control and Challenges

The integration of higher-level planning, scheduling, and collaboration reasoning processes with and robot's lower-level functional behaviors brings quite some exciting challenges to this research. Figure 5 depicts the overview of this integrated architecture for autonomous robot control. The robot receives new information from its sensors, and new messages arrive through communication channels. This new information and new messages are used to update the robot's world model, which contains the knowledge of the environment and also about other robots in this environment. In the robot rescue example, the world model stores

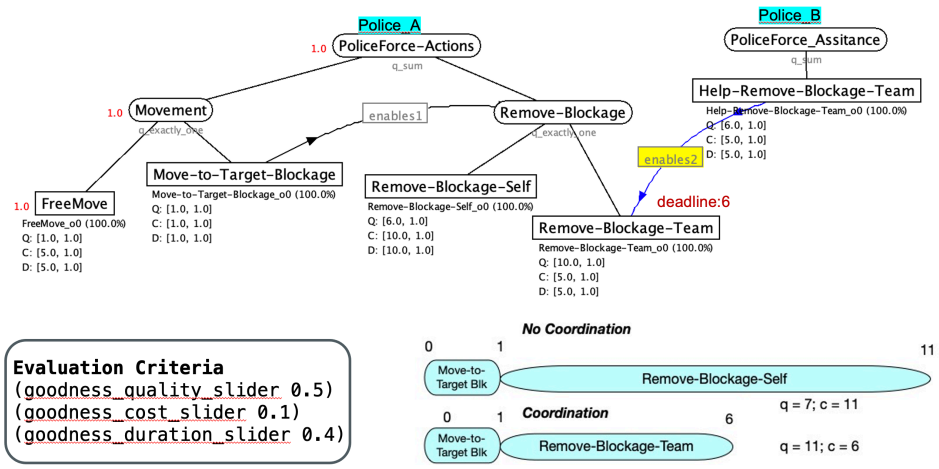

Figure 3: Schedule Example with Coordination Choices
Generated by a task structure visualizer with runtime information. Each method's outcome is described as quality Q, cost C, and duration D distribution. *D:[5.0, 1.0]*: with a probability of 1.0, the duration is 5.

```
Coordination Request
 from: Police_A
 to:   Police_B
 task: Help-Remove-Blockage-Team
 deadline: 6
 gain: 2.0
```

Figure 4: Coordination Request Example

information about the map, the blockage location, the discovered civilians and their locations, and the closeby agents and their locations. Based on the updated world model, the TÆMS model may also be updated. For example, a new task is created to help remove a blockage resulting from a message received. The expected outcome of a task may also be specified with more accurate information. For example, the time duration of arriving at a specific location may be updated when the road situation is known better.

Based on the TÆMS model, the intelligent scheduler may produce a soft real-time (Erickson and Anderson 2022) plan, a sequence of executable methods, each with a specified start time and end time. When applicable, the collaboration toolkit would also generate collaboration requests for other robots, which will be sent out as new messages by the communication module.

The first challenge is to decide whether the robot needs to interrupt the execution of a low-level method and report to the upper-level controller with newly discovered information and new messages. For example, in the movement process of picking up a civilian, which is a scheduled method for the ambulance robot, it discovers another location with trapped civilians inside. Should the ambulance stop its current execution and report this new piece of information and wait for a potentially modified plan or continue its current movement to pick up the assigned civilian as previously planned? In application domains where the computing deliberation time is ignorable compared to the execution time, the robot does not need to stop its execution while waiting for possible revision of the plan. Otherwise, this decision needs to be made more carefully.

Besides whether the robot shall stop its execution when new information is discovered, the second challenge is whether the robot shall replan given new information discovered or a new message received. This is also referred to as the commitment problem (Wooldridge 1999). Overcommit to a predetermined plan may lead to failure to respond to a changed environment or missing new opportunities. However, too frequently replanning also may lead to failure to accomplish any goal. In order to make a balanced decision, the robot needs to carefully evaluate the impact of the newly discovered information on its current plan and also assess the opportunity cost of each choice.

The third challenge is to decide when to communicate and to whom to communicate. Such decisions could be made as part of the collaboration plan at the upper level, however, more reactive and dynamic communications are also needed to share new discoveries with other robots or to reattempt sending messages when no response is received for previously sent messages. The question is how frequently such update messages shall be sent and how persistent the robot shall be to resend an unresponded message. Considering communication also carries a cost, channels may be unreliable, robots may move out of range, and information may be inaccurate or outdated, the decisions regarding communication demand further deliberation.

Simulation Environment and Development Interface

To verify the proposed framework as shown in Figure 1 and explore the challenges described in the previous section, we build a platform for developing software agents with AI technologies in a simulation environment in which they can be tested with real-world physics. This platform is built on the integration of an agent development framework and

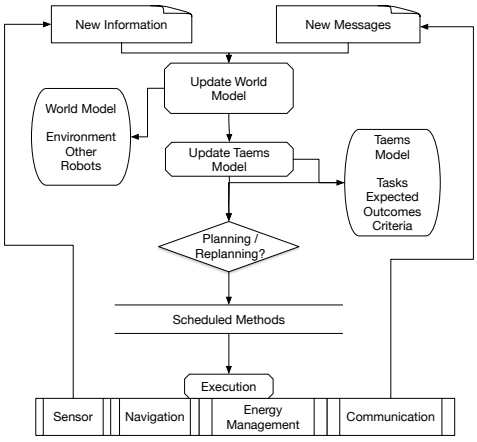

Figure 5: Integration of High-Level Reasoning and Low-Level Functions

a simulation environment. The agent development environment supports the pre-deployment prototyping of agents for customizable use cases, and the simulation environment can be altered to present obstacles to navigate, resources to mine, entities to rescue, and environmental forces such as tides or gravity to adapt to. We assert that such simulation testing is paramount to the success of real-world-deployed multi-agent systems used in collaborative autonomous robots.

The agent development framework chosen here is the Java Agent DEvelopment Framework, referred to as JADE (Bellifemine, Caire, and Greenwood 2007). This decision is made for the following two reasons. First, JADE has an integrated communication protocol, which is consistent with FIPA's ACL Message structure, and therefore simplifies the implementation of any communication between agents. Secondly, JADE also has support the use of TÆMS framework because both use the Java language for implementation. The widely-used open-source Robot Operating System, ROS (Quigley et al. 2009; Koubâa et al. 2017), is not chosen in this project because of the difficulty of integrating it with the Java-based TÆMS related AI technologies. However, as a future direction, it shall be considered to build a bridge between this set of AI technologies and ROS.

The construction of a simulator environment requires two principal elements: a graphics library and a physics engine. The natural place to start with such a search is where these tw o elements married most prominently, LibGDX (Stemkoski 2018) combines multiple graphics libraries and one of the most widely used two-dimensional physics libraries, Box2D (Parberry 2017). As Box2D is natively a C++ library, the thin wrapper that the LibGDX framework provides makes it usable in the native Java project architecture (libGDX community 2023). While elements of the graphic libraries it contains are used to draw the window, and render the actual scene, Box2D is responsible for most of the features concerning the agents.

Integrated Architecture Initial prototyping of both frameworks was straightforward but integrating them into a single project structure posed some challenges. To instantiate JADE agents, the JADE platform needs to be running. This feat was previously accomplished by running the *jade.Boot* class as the main class with command line arguments to specify the agents. To launch the simulator, its own main class must be declared in the run configuration. Fortunately, JADE supplies the ability to launch the platform programmatically which we can integrate into the setup of the simulator environment.

To truly emulate the partial observability of the environment that the robot would have in a real-world application, a wrapping layer called *AgentWorldPerspectiveLayer* is needed between the instance of the world (environment) and the agent core, because the agent should not be given direct access to the instance of the world that contains all environmental information. The introduction of the *AgentWorldPerspectiveLayer* class will act as an intermediary class between the agent core and the simulated environment. This will also grant us the ability to model communication constraints. For example, a message is delivered only when by the desired recipient is within the communication range before sending the message. Conceptually, the agent core wrapped with this *AgentWorldPerspectiveLayer* simulates the robot with limited and specified sensor/communication capability. By keeping the reference to the world encapsulated in a separate class from the agent, we can ensure that everything that the agent knows about the world is entirely derived from its own perception experiences and the messages of other agents.

Agent Anatomy The simulator includes a multitude of visualization options for rendering object bodies including those representing the agent body. The full spectrum of these features can be seen in Figure 7. The agent bodies and sensors are given a basic circular shape to simplify movement by being able to ignore complicating factors such as orientation. The perimeter of the body boundary is drawn in white. The orientation of the agents body is indicated by the same color line emanating from its center. This gives an indication of rotation which can be especially useful in shapes such as circles where rotation can be hard to discern. The direction and magnitude of a shapes velocity is indicated by a red line also emanating from the bodys center in the direction of its movement. Bounding boxes are also drawn around each object in magenta to indicate the coordinate minima and maxima in both dimensions. Bounding boxes and velocities can be turned off when the renderer is instantiated.

Scenario The scenario that was used in the demonstration consisted of two agents initialized on opposite sides of the simulator environment. The communicating agent *commAgent* first sends a CFP message to the receiving agent *recAgent* with a list of potential coordinates to navigate to. The *recAgent* parses this list and calculates which of the coordinates has the smallest Euclidean distance to its current location. It then replies with a Proposal message with these coordinates. The *commAgent* replies with an Accept message and begins moving to the destination. Once the *recAgent* receives this proposal acceptance, it also begins moving toward the destination. To make it more explicit, a static object at the destination point was added to the simulator environment. As an interesting additional feature, the *recAgent* was also outfitted with a sensor to demonstrate that these objects can

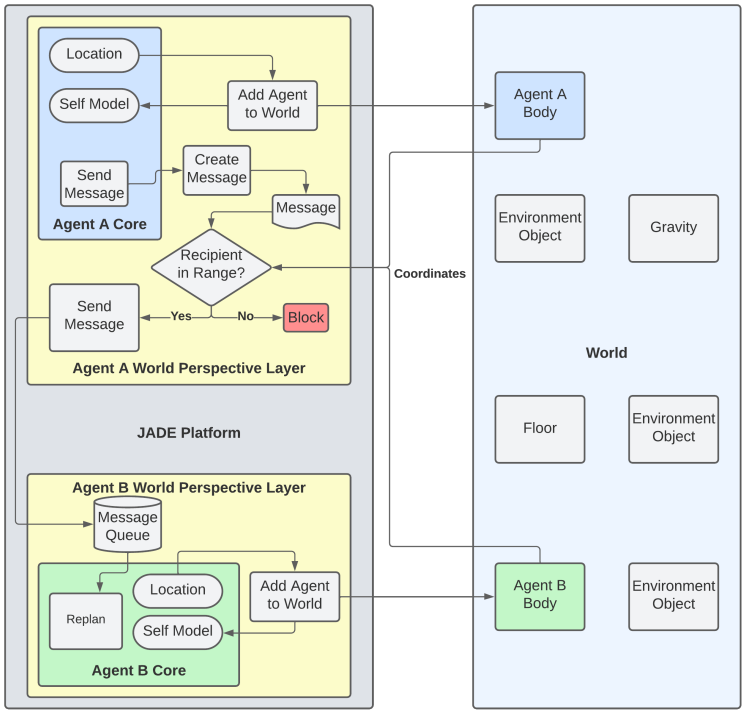

Figure 6: Functionality of Agent-World Perspective Layer

detect other objects within its boundaries without colliding with them.

Conclusion and Future Work

We presented an integration approach to developing robots with AI technologies for planning, scheduling, and coordination for complex soft real-time application problems with interrelated task relationships. We described the TÆMS language framework and showed how it may be used to model a sophisticated problem scenario - Robotcup rescue. An intelligent scheduler and a collaboration toolkit may be developed based on TÆMS framework to generate a plan for the robot and handle necessary collaboration with other robots. Three major challenges in this integration approach have been discussed. In the end, we presented a platform to support the development of software agents and testing them in a simulated environment with real-world physics, which would facilitate the transfer into real robot architecture.

We hope to continue this work to further explore the three challenges presented in this paper. It is important to test this approach with complex problem scenarios on the integrated simulation platform to better understand the feasibility and limitations of this approach. Furthermore, we would like to transfer this approach to real robots hopefully to work with RoS as a starting point.

References

Arkin, R. C. 1998. *Behavior-based robotics*. MIT press.

Bellifemine, F.; Caire, G.; and Greenwood, D. 2007. *Developing Multi-agent Systems with JADE*. John Wiley & Sons. ISBN 9780470057476.

Castellini, A.; Marchesini, E.; and Farinelli, A. 2019. Online Monte Carlo Planning for Autonomous Robots: Exploiting Prior Knowledge on Task Similarities. In *AIRO@AI*IA*.

Darmanin, R. N.; and Bugeja, M. K. 2017. A review on multi-robot systems categorised by application domain. In *2017 25th mediterranean conference on control and automation (MED)*, 701–706. IEEE.

Erickson, J. P.; and Anderson, J. H. 2022. *Soft Real-Time Scheduling*, 233–267. Singapore: Springer Nature Singapore. ISBN 978-981-287-251-7.

Gohardani, P. D.; Mehrabi, S.; and Ardestani, P. 2016. RoboCup Rescue Simulation System 2016 Champion Team Paper. In Behnke, S.; Sheh, R.; Sariel, S.; and Lee, D. D., eds., *RoboCup 2016: Robot World Cup XX [Leipzig, Germany, June 30 - July 4, 2016]*, volume 9776 of *Lecture Notes in Computer Science*, 565–576. Springer.

Koubâa, A.; et al. 2017. *Robot Operating System (ROS).*, volume 1. Springer.

Lee, G.; and Chong, N. Y. 2008. Flocking Controls for Swarms of Mobile Robots Inspired by Fish Schools. In Lazinica, A., ed., *Recent Advances in Multi Robot Systems*, chapter 4. Rijeka: IntechOpen.

Lesser, V.; and Corkill, D. 2014. Challenges for Multi-Agent Coordination Theory Based on Empirical Observations. In *Proceedings of the 2014 International Confer-*

ence on Autonomous Agents and Multi-Agent Systems, AAMAS '14, 1157–1160. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450327381.

Liang, H.; and Kang, F. 2016. A novel task optimal allocation approach based on Contract Net Protocol for Agent-oriented UUV swarm system modeling. *Optik*, 127(8): 3928 – 3933.

libGDX community. 2023. libGDX Wiki Page. https://libgdx.com/wiki/. Accessed: 2023-10-10.

Marti, I.; Tomas, V. R.; Saez, A.; and Martinez, J. J. 2009. A Rule-Based Multi-agent System for Road Traffic Management. In *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, volume 3, 595–598.

Milot, A.; Chauveau, E.; Lacroix, S.; and Lesire, C. 2021. Market-based Multi-robot coordination with HTN planning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2606–2612.

Parberry, I. 2017. *Introduction to Game Physics with Box2D*. CRC Press.

Parker, L. E. 1995. On the design of behavior-based multi-robot teams. *Advanced Robotics*, 10(6): 547–578.

Parker, L. E.; Rus, D.; and Sukhatme, G. S. 2016. *Multiple Mobile Robot Systems*, 1335–1384. Cham: Springer International Publishing. ISBN 978-3-319-32552-1.

Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3.

Simmons, R.; Singh, S.; Hershberger, D.; Ramos, J.; and Smith, T. 2001. First results in the coordination of heterogeneous robots for large-scale assembly. In *Experimental Robotics VII*, 323–332. Springer.

Stemkoski, L. 2018. *Java Game Development with LibGDX: From Beginner to Professional*. Apress. ISBN 9781484233245.

Wagner, T. A.; Garvey, A. J.; and Lesser, V. R. 1998. Criteria Directed Task Scheduling. *Journal for Approximate Reasoning (Special Scheduling Issue); a version is also available as UMass Computer Science Technical Report 1997-59*, 19: 91–118.

Wahde, M. 2009. A General-Purpose Method for Decision-Making in Autonomous Robots. In Chien, B.-C.; Hong, T.-P.; Chen, S.-M.; and Ali, M., eds., *Next-Generation Applied Intelligence*, 1–10. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-02568-6.

Wooldridge, M. 1999. Intelligent agents. In *Multiagent systems: a modern approach to distributed artificial intelligence*, 27–77. The MIT Press.