Towards Novel and Intentional Cooperation of Diverse Autonomous Robots: An Architectural Approach

Niko Mäkitalo[1], Simo Linkola[1], Tomi Laurinen[1] and Tomi Männistö[1]

[1]*Department of Computer Science, University of Helsinki*

Abstract

In most autonomous robot approaches, the individual robot's goals and cooperation behavior are fixed during the design. Moreover, the robot's design may limit its ability to perform other than initially planned tasks. This leaves little room for novel dynamic cooperation where new (joint) actions could be formed or goals adjusted after deployment. In this paper, we address how situational context augmented with peer modeling can foster cooperation opportunity identification and cooperation planning. As a practical contribution, we introduce our new software architecture that enables developing, training, testing, and deploying dynamic cooperation solutions for diverse autonomous robots. The presented architecture operates in three different worlds: in the Real World with real robots, in the 3D Virtual World by emulating the real environments and robots, and in an abstract 2D Block World that fosters developing and studying large-scale cooperation scenarios. Feedback loops among these three worlds bring data from one world to another and provide valuable information to improve cooperation solutions.

Keywords

robot software architecture, robot cooperation, ontology-based reasoning, peer modeling, autonomous robots

1. Introduction

In autonomous robot cooperation, understanding the robots' context plays a key role. Situational context is a term used to describe why some phenomenon occurs in a specific situation and what actions can be associated with this situation [1]. This paper presents an architecture that fosters the robots' situational awareness in their present context. Central in our approach is the information that is relevant for the cooperation planning: A robot must be able to form an understanding of the other robots and their resources and an understanding of the environment where the cooperation is intended to take place. Hence our architectural approach does not provide a solution to form a complete or joint contextual understanding between the robots. Instead, the architecture enables each robot to form its own view of the situation. The robots then use their situational context model and understanding as a basis for forming joint action plans for meeting their own personal goals.

In most autonomous robot approaches, the goal of the individual robot and its cooperation behavior is fixed during the design. However, in heterogeneous encounters with diverse peers and other computational actors, this leaves little room for novel dynamic cooperation where new (joint) actions could be formed or goals adjusted after deployment. Nonetheless, this kind of creative use of complementary capabilities could highly benefit the whole robot population, especially when the population is sparse and consists of low-end consumer robots built for singular tasks, e.g., cleaning, with ample idle time to allocate to other goals.

To optimize the use of context and training the robots to understand their situation and cooperation possibilities, we propose a novel three-world development approach. The development approach involves Real World, 3D Virtual World, and 2D Block World, and an associated software architecture and frameworks that can operate in all these three different worlds, allowing to focus on different aspects of the development.

The 2D Block World works as a platform and test bed for developing the ontology-based understanding as it allows simulation of large number of diverse robots in different cooperation scenarios. Ontological reasoning and planning provide robots a shared understanding of "how the world works" and thus are crucial in our approach for multi-robot cooperation.

As our starting ontology, we adopt DUL (DOLCE+DnS Ultralite) ontology[1], which suits well for autonomous robot reasoning (see, e.g., KnowRob 2.0 [2]). It serves as a *top-level ontology*, which applications are supposed to extend by their own ontological concepts. For this work we have made a minimal extension to DUL to showcase the applicability of our approach.

In the Real World and 3D Virtual World implementation, we have focused on robots based on Robot Operating System (ROS). Briefly put, ROS is an open-source

Context-aware, Autonomous and Smart Architecture Workshop co-located with the 15th European Conference on Software Architecture 13-17 September, 2021, Virtual Event CASA@ECSA2021

niko.makitalo@helsinki.fi (N. Mäkitalo); simo.linkola@helsinki.fi (S. Linkola); tlaurinen@gmail.com (T. Laurinen); tomi.mannisto@helsinki.fi (T. Männistö)

[1]http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_Ultralite

robot development framework where different nodes, or programs, communicate asynchronously by *subscribing* and *publishing* to *topics* shared over a network. A single ROS node acts as the *server*, or *master*, to which other ROS-enabled devices can be connected to as *clients*, forming the network. Being a leading open-source project in robotics, ROS has an active development community, and a different, newer version, ROS2, is also seeing increasing amounts of use and development. In this work, we use ROS2 in our implementation.

Our approach aims to support ad hoc encounters of heterogeneous autonomous robots, which each have their own individual goals, which can be used to define various plans that include different types of tasks. Typically, the cooperation tasks can be categorized into loosely and tightly coupled cooperation tasks [3]: *Tightly coupled tasks* cannot be performed by one robot but require multiple robots working cooperatively; *Loosely coupled tasks*, on the other hand, can be performed by a single robot but the task can be performed more efficiently in cooperation.

The proposed software architecture enables cooperation in both tightly coupled and loosely coupled tasks mainly through *peer modeling*, which has been argued to be a requirement for cooperation [4]. The robots exchange, learn, use and evaluate models of themselves and their peers to identify and exploit cooperation opportunities. Although the architecture proposes means for coordination and communication, implementing tightly coupled tasks, however, requires more work from the developer.

The rest of this paper is structured as follows. In Section 2, we introduce concepts related to our architecture. In Section 3, we describe our solution – a software architecture that enables the developing, training, and testing cooperation of autonomous robots. In Section 4, we explain the current status of the architecture and what kind of experiments are currently possible with the architecture. In Section 5, we cover work related to our approach. In Section 6, we discuss how we plan to improve the solution in the future and what we are currently focusing on implementing. Finally, in Section 7, we draw some conclusions for this work.

2. Cooperation Concepts

To understand our architecture, we first introduce the ontological concepts we use to enable cooperation. The basic concepts introduced here are part of DUL ontology, but we extend them in our work to provide concrete solutions and a more fine-grained understanding of the situation at hand.

The robots' essential operation revolves around *goals* describing desirable situations, which we model as states of the environment and the robot should find itself in. A goal can be, e.g., to keep a room clean or deliver a package to a specific place. A robot may have multiple or even conflicting goals.

To achieve its goals (either by itself or in cooperation), robot forms *plans* which consist of *tasks*. A plan describes how a certain goal is achieved, i.e., which tasks should be done and their (partial) order. To make a plan concrete, each task needs to be assigned to a robot (or a set of robots). This concrete plan is called a *workflow*.

Tasks are the individual elements from which plans and workflows are composed of. Each task includes individual objects to be achieved, e.g., open a particular door, move to a specific place, etc. Tasks can be hierarchically nested in two ways. First, there can be general tasks (open a door) and refinements of those tasks (open a door by pulling the handle). Second, lower-level tasks may be combined to compose higher-level tasks, e.g., moving, opening a door, and moving again can be seen as one higher-level moving task. These task structures are used when generating and communicating workflows.

Tasks have defined start and end conditions. However, the actions (see below) can be partly responsible for checking these conditions. The start condition is checked before the task can be attempted, e.g., to open a door manually, the robot must be next to it. The end conditions are checked to see if the task was completed successfully, e.g., if the door is open. The task end conditions can be thought of as individual, low-level goals.

To achieve tasks, each robot has *actions* by which the tasks can be completed. The robot may have multiple (sets of) actions that achieve the same task, and an action may be utilized in multiple tasks. Where goals, plans, workflows, and tasks are platform-independent, actions should be implemented on each platform (and the world) separately.

To allow cooperation, robots communicate their goals, suggested workflows, and tasks to develop workflows, including multiple robots. To make this communication more fluent, robots maintain a model of themselves and each of their peers. In general, these models may hold any important information of the robot in question, such as their physical properties, *capabilities*, i.e., which tasks they can perform, the robot's goals, and the history of the workflows they have been included in and their success.

3. Software Architecture for Autonomous Robot Cooperation

At the core of our research is the *CACDAR architecture*. The architecture, with its components and the leveraged services, is depicted in Figure 1. The architecture can

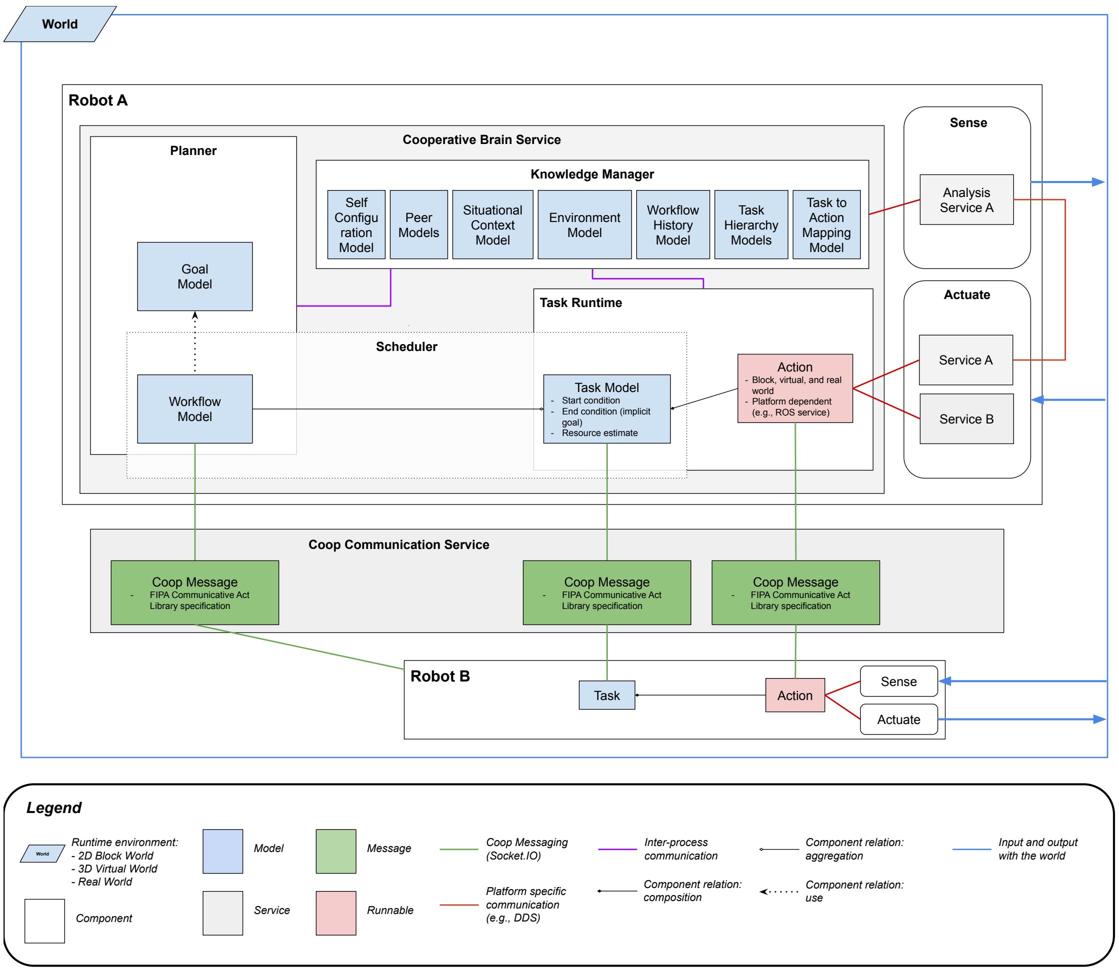

World
Robot A
Cooperative Brain Service
Planner
Goal Model
Knowledge Manager
Self Configuration Model
Peer Models
Situational Context Model
Environment Model
Workflow History Model
Task Hierarchy Models
Task to Action Mapping Model
Task Runtime
Scheduler
Workflow Model
Task Model
- Start condition
- End condition (implicit goal)
- Resource estimate
Action
- Block, virtual, and real world
- Platform dependent (e.g., ROS service)
Sense
Analysis Service A
Actuate
Service A
Service B
Coop Communication Service
Coop Message
- FIPA Communicative Act Library specification
Coop Message
- FIPA Communicative Act Library specification
Coop Message
- FIPA Communicative Act Library specification
Robot B
Task
Action
Sense
Actuate
Legend
World
Runtime environment:
- 2D Block World
- 3D Virtual World
- Real World
Component
Model
Service
Message
Runnable
Coop Messaging (Socket.IO)
Platform specific communication (e.g., DDS)
Inter-process communication
Component relation: composition
Component relation: aggregation
Component relation: use
Input and output with the world

Figure 1: CACDAR architecture.

operate in all three worlds, 2D Block World, 3D Virtual World, and Real World, and it also provides feedback loops between these three different worlds, allowing us to manually and automatically incorporate the insights in order to advance the situational context awareness that fosters the robot cooperation (see Figure 2).

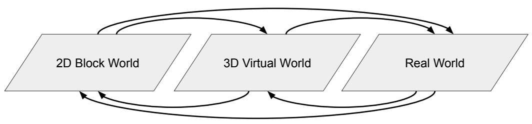


Figure 2: Lessons learned feedback loops between three development and evaluation stages.

3.1. Cooperative Brain Service

The critical enabling service for the novel and valuable cooperation is platform-agnostic *Cooperative Brain Service*, which encloses several components. The service is responsible for the high-level functionality of the robot such as planning of future tasks and cooperation (see *Planner*), scheduling of tasks to be executed (see *Scheduler*) by *Task Runtime*, and it gathers information from sensors, its operation and communication with other robots into *Knowledge Manager* which it uses in its reasoning. For cooperation, the service needs to be able to detect if there is a robot that has requested help, and then try to reason if it would a) have the missing resources, or b) would have free resources or less important tasks so that it could free up the resources for the cooperation. The availability of such resources (e.g., time and battery) are estimated in collaboration with *Scheduler* and *Task Runtime* components.

However, the most crucial responsibility of the service is to estimate whether it will meet its own goals. It constantly keeps track of its resources and what resources other robots have allocated for helping it to meet its goals. Hence, it leverages *Knowledge Manager* and *Task Runtime* components by observing changes in the models that represent the other robots and environment, and then notifies the *Planner* which can alter its workflow and tasks (e.g., by replanning tasks with missing resources or reorganizing tasks in its workflow).

3.2. Knowledge Manager

Knowledge Manager takes care of maintaining the robot's understanding of the world and the information associated with the cooperation. The main input source for the component is the robot's (platform-dependent) service components that the robot uses for observing and sensing. *Knowledge Manager* may also exchange information with other robots' *Knowledge Manager* components via respective *Cooperative Brain Services* with *Coop Messages*. *Knowledge Manager* maintains the following models that enable novel and valuable cooperation as well as robot's individual goal-oriented behavior:

Situational Context Model captures information considering the robot's current situation, e.g., where it and other robots currently are, what is the state of the environment objects near it, and other dynamic properties. The model's contents can be updated using feedback from sensors, *Environment Model* (e.g., by making queries of possible state changes in the physical objects represented in the ontology if they are not directly perceived), *Self* and *Peer Models*, and direct communication with other actors, such as robots, through *Coop Messages*. To this extent, *Situational Context Model* operates in tandem with the environment and peer models to provide a unified view of the most current understanding of the situation. This model can be used directly in *Planner*, whereas other models provide more fractured view of the situation.

Environment Model connects actions in the operating environment, e.g., moving or object manipulation, into state changes in ontological objects. The model should represent the environment and its objects in sufficient detail so that it can be used to derive reasonable *Situational Context Model* and reason about possible changes of certain actions in particular situations. It can be updated using feedback from the environment (either perceived or received through communication). The level of detail in *Environment Model* varies across the different *world types*. In 2D Block World, the model is sufficient to possess simple logical states, e.g., is the door open or closed, while in Virtual and Real World the model may be more elaborate, e.g., a door can be partially closed and currently opening. However, to keep the "backward functionality" intact from Real World back to 2D Block World, individual object states and actions that manipulate them in Real World model should be mappable into the 2D Block World model.

Self Model and Peer Models contain information about the robot itself and its peers. In general, each peer has its model, but aggregate models, e.g., considering certain classes of robots, are possible. Robots exchange basic information considering themselves (drawn from their *Self Model* and other knowledge sources) when they first meet their peers and update and replace this information through communication and observations. Where *Situational Context Model* offers current information of the state of the world, and *Environment Model* offers an understanding of how the world works, these models provide knowledge of what are each robot's goals, which tasks are possible for the robot, and what restrictions the robot may have for performing specific tasks, e.g., if the robot can only open specific types of doors. From the cooperation perspective, these models are highly relevant, as their information is needed in *Planner* when determining whether who can perform a particular *Task*.

Task to Action Mapping Models contains knowledge about mapping the task realizations to actions. This knowledge is mainly about the robot's tasks, but peers' tasks to action mapping information can also be partially stored. This applies especially to cases if the robots are of the same type. Additionally, other peers may provide some information about their action mapping for a particular task, e.g., resource estimates, timing information, or constraints that can be used in planning.

Workflow History Model contains the information on earlier cooperation situations, such as performed task hierarchies, their configurations, and execution results. The information is used for improving the quality of the cooperation by analyzing which workflows and roles have previously worked well and which ones have failed.

Task Hierarchy Models are used as configuration models for creating task hierarchies (consisting of task-goal-plan nodes), e.g., options for decomposing tasks or goals and constraints for valid hierarchy configurations. It can be used to determine whether a particular task hierarchy configuration is valid, and the hierarchies can, then, be used by *Planner* or other components in *Knowledge Manager*, e.g., to represent aggregated high-level capabilities of the peers.

3.3. Planner

Planner is responsible for constructing *Workflows* which are then, e.g., passed to *Scheduler* for execution or stored for later use. As input, *Planner* is given some starting situation, e.g. the current *Situational Context*, a desired end condition, e.g. the current *Goal*, and other related parameters, e.g. restrictions for the workflow. *Planner* leverages the information maintained by *Knowledge Manager* in its attempts to select the robot and its peers to specific roles and to assign them *Tasks*. For actually assigning *Tasks* for its peer robots, *Planner* negotiates with different robots' *Planner* components. The purpose is to ensure that the robot has a correct understanding of its peer's capabilities (i.e., *Tasks* it can perform) and that the peer has sufficient resources, e.g., time and battery power, to participate in the workflow.

Goal Model defines a single mission that is expected to be carried out by a single robot or a set of robots. However, it does not define how the actual plan and the mission is expected to be performed. Instead, a *Goal Model* can set some ground rules for the robot behavior, like time constraints or quality attributes. A *Goal Model* is used for deriving start and end conditions for specific tasks. It may also affect what types of robots get selected into the roles of the cooperation.

Workflow Model consists of a *Goal Model* and a partially ordered list of *Task Models* where each task is assigned to a (set of) robots. By default, *Planner* tries to put together a *Workflow Model* where the robot itself is in the primary role, and its peers are assigned only if the robot cannot meet the *Goal*. However, the *Goal Model* can affect how the workflow is put together: As the *Goal Model* contains information regarding a single robot's mission, it can then define the mission to be highly cooperative or act as a leader. For example, consider that one robot is expected to act as a supervisor for the other robots – its mission is then defined to coordinate the others and their cooperation.

3.4. Task Runtime

Different types of robots can feature very differing underlying platforms for development and interfacing in general. Therefore, the platform is essentially what dictates how actions have to be implemented. The *Task Runtime* is accordingly designed so that support for new platforms can be added at will, in the form of *platform modules*. Currently supported platforms are the older and newer versions of the Robot Operating System, ROS1 and ROS2, introduced in more detail later. However, as a particular measure stemming from the similarity of these platforms, actions are shared between both by abstracting implementation differences of subscribers and publishers using the respective *platform modules*.

Task. The self-adaptive aspects of the architecture come into play when the autonomous operation or cooperation requires certain resources. Each robot describes its capabilities by communicating to others what kind of *tasks* they can execute. A task may consist of sub-level tasks, that is, a task may group other tasks into a higher-level behavior. As an example, consider that a robot can perform a task Guide. Such task then consists of other tasks, like Move, Turn, Navigate, etc.

Action is the mapping from the behavior modeled with tasks to the actual implementation of a specific task. Actions are generally platform-specific, but there can be alternative versions of actions for different robots even within the same platform. Similar to the tasks, also actions can consist of other sub-level actions. For instance, conforming Action: Guide may leverage various other action implementations.

3.5. Robot's Services

For actuating and sensing the events coming from the world, the architecture enables leveraging various services and communication between them. In Figure 1, such services have been illustrated: an imaginary actuating *Service A* is used, for example controlling the robot, and at the same time, it sends data to *Analysis Service A*. While we have mainly used ROS2 based services in our current implementation, the Cooperation Brain is not tied to any specific robot technology. Hence the services may also be realized as ROS1 services or any other type of service technology (e.g., as a Docker-based microservice).

3.6. Scheduler

Scheduler component is part of the Task Runtime component. Scheduler fetches the Tasks from the Planner components Workflow and delivers the runnable Tasks to the Task Runtime. Scheduler's primary duty is to maintain a Task list for execution in the robot, considering priorities and constraints of active goals. For this purpose, the Scheduler uses each task's start and end conditions to ensure that the situation is correct for running the task. The Scheduler also uses the resource estimates to ensure that the robot has the promised resources for performing the task.

3.7. Coop Communication Service

In order to cooperate effectively in varying situations and environments, the robots require a communication platform that can relay messages between the components deployed on various nodes. The base technology for inter-robot communication is Socket.IO. It provides a relatively reliable and fast enough communication channel for negotiating about the cooperation-related activities, like tasks and roles in workflows, and providing feedback.

In our present research, we mainly leverage ROS2-based robots. ROS2, on the other hand, leverages DDS technology for communication between the ROS2 services. Hence, in the future, our implementation may change using DDS also for the cooperation communication to make the architecture more streamlined. The downside, however, is that setting up a DDS-based communication infrastructure can be challenging for robots that lack the required resources, and as there are several different DDS implementations, incompatibility issues may emerge and issues with licensing. For this reason, the implementation yet relies on our service and Socket.IO technology. Additionally, to support also non-ROS2 based robots, we have been discussing implementing a communication bridge that would allow ROS and other types of robots and smart objects and resources (e.g., sensors, existing facility service systems, smart home systems, etc.) in the environments to participate and enhance the cooperation.

Coop Message is the base unit of the communication in the *CACDAR architecture*. Two other base message types – *BroadcastMessage* and *Direct Message* – are inherited from the base, and the idea is that the communication language is extended by inheriting new subtypes. The only requirement is that each message has a sender. The actual communication messages are based on FIPA Communicative Act Library Specification [5] from which we use a subset.

Broadcast Messages are sent publicly to all robots and services connected to the Coop Communication Service. Typical use cases for these messages are when a new robot arrives at a specific venue and then gets connected to the Coop Communication Service located at this venue. The robot may then greet the other connected ones by broadcasting its name and the tasks it considers capable of performing. The robot may also request help from other robots by trying to describe its goal to other robots.

Direct messages, on the other hand, are sent directly from one robot to a set of recipients. These messages are mainly used for negotiating a cooperation plan and communicating during the execution of the plan.

4. Current Status

In this section, we present the current implementation status of the architecture. We start by describing an example use case and then continue presenting some proof of concept implementations for the use case. Additionally, we report our experiences so far about the three-world approach and its benefits.

4.1. Example Use Case: Package Delivery

Throughout our implementation work and experiments, we have used the following as a base use case and scenario which has been adjusted and changed to different environments and worlds in our three-world approach: A delivery robot with a heavy package, e.g., a tool rented online, comes to a construction site previously unknown to it. It has a goal: deliver the tool to a specific location. There are also a number of other robots with different capabilities executing their tasks, such as cleaning the place, who have appropriate knowledge related to their responsibilities, such as the layout of the cleaning areas. As their responsibilities, e.g., cleaning, may leave time for other tasks, they can help the delivery robot.

While the above scenario seems very simple, there are almost unlimited possibilities to advance creative cooperation. The scenario requires the robots to a) become aware of each other skills, objectives, and knowledge; b) be able to define their joint problem: delivering the package; and c) together form and execute good enough plans for solving the joint problem. Hence, despite the limited domain, we believe that the above scenario can serve as a basis for numerous other cooperation applications for diverse autonomous robots.

4.2. Introducing ROS and Real-World Robots

To get started with implementing the package delivery scenario and prototyping the *Task Runtime*, we initially focused on physical robots of the Real World. We had an already available supply of small-sized research use robots, and with Real World being the most intricate of the three-world approach, it was deemed beneficial to get familiar with the particularities of physical robot development from early on. The robots chosen for these early implementation efforts were Rosbot 2.0 and TurtleBot3, both which used Robot Operating System, ROS, as their development platform. As ROS was to become the first platform the *Task Runtime* would support, familiarizing ourselves with it was necessary to get started.

Being small, economical robots for education and research use, Rosbot 2.0 and TurtleBot3 were at the time equipped simply with wheels for movement and LIDARs for scanning surroundings, with the Rosbot 2.0 also featuring a camera. The fundamental premises of the project also meant further restrictions: we could not have the robots share any common understanding of the world, not a common map or even coordinate system. Inspired by these limitations, the very first mutually coordinated *Action* made was that of Rosbot 2.0 following Turtlebot3 with the help of QR codes. Due to both robots using ROS, the development of *Task Runtime* began ROS support first, but care was taken in making it possible to add support for other robot platforms also.

The QR code method is straightforward in principle: A QR code stand is propped up on the Turtlebot3. When Rosbot 2.0 sees the QR code on Turtlebot with its camera, it tries to move itself so that the QR code is centered on the image at a direct angle and a certain distance. The movement direction is based on distance, derived by comparing the QR's width in the image to a predetermined expected width, and rotation, derived from the homography matrix between the image and the QR code within it.

However, this method alone does not make for an adequate following logic. When the guide, Turtlebot3, moves, it is easy for the follower, Rosbot 2.0, to lose sight of the QR code. As a solution, we implemented a communication protocol specific to this `Follow` action: If the guide goes out of view, the follower asks the guide to stop. The follower then moves to where it last saw the guide, and begins a search process that uses rotation data exchanged between the follower and the guide. This approach utilizes the fact that we can calibrate and compare the rotations of the robots, even when the coordinates cannot be shared (as both have their own map). As an additional measure, QR codes are added on all sides of the guide. If the follower sees a code other than the one behind the guide, the guide will attempt to rotate so that the follower is lined right behind the guide again.

As a result, the `Follow` and `Guide` actions were created successfully on the physical robots alone. Yet, many realities of the Real World hampering robot development became apparent: Lighting conditions would affect the detection of QR codes greatly, even the slightest of obstacles such as cables were insurmountable for the TurtleBot3, and having to reset the positions of the robots manually every attempt was also rather inconvenient in the long run. We also did not have the equipment or means for complicated feats such as having the robots to carry objects. To top it all off, the worsening COVID-19 situation meant that work would remain remote for the foreseeable future, so we started to look into the robot simulation environments next.

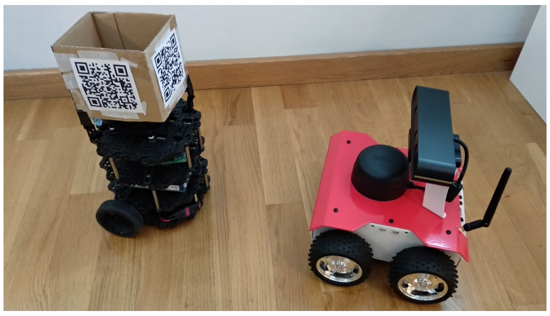


Figure 3: Rosbot 2.0 following a QR code-equipped TurtleBot3.

4.3. Migrating from ROS to ROS2

Before moving from the Real World to the 3D Virtual World, we also changed the primary development platform from the original ROS, also known as ROS1, to ROS2. Switching to the newer platform was not entirely trivial due to major design differences between the two, but we deemed it worthwhile for a number of reasons: ROS1 uses a client/server architecture, where all machines have to be registered on the server machine. ROS2 is instead an entirely peer-to-peer solution based on the DDS middleware, which would quite intuitively appear a better fit for cooperation of autonomous robots. Our project was also at an early stage at the time, and ROS2 exhibited more promising prospects for the future. In contrast to ROS, which was designed back in 2010 for research and educational purposes, improvements in aspects such as real-time programming claim to bring ROS2 closer to applicability even for industry use. For us, this heavily implied that any interesting future developments would most likely focus on ROS2. Therefore, we are now using the newly released stable release of ROS2, Foxy Fitzroy, as the platform for Turtlebots and Gazebo simulation. The support for ROS2 on Rosbot 2.0 has been more limited so far, so have instead kept it at ROS1 to demonstrate how the *Task Runtime* is made to support both ROS1 and ROS2.

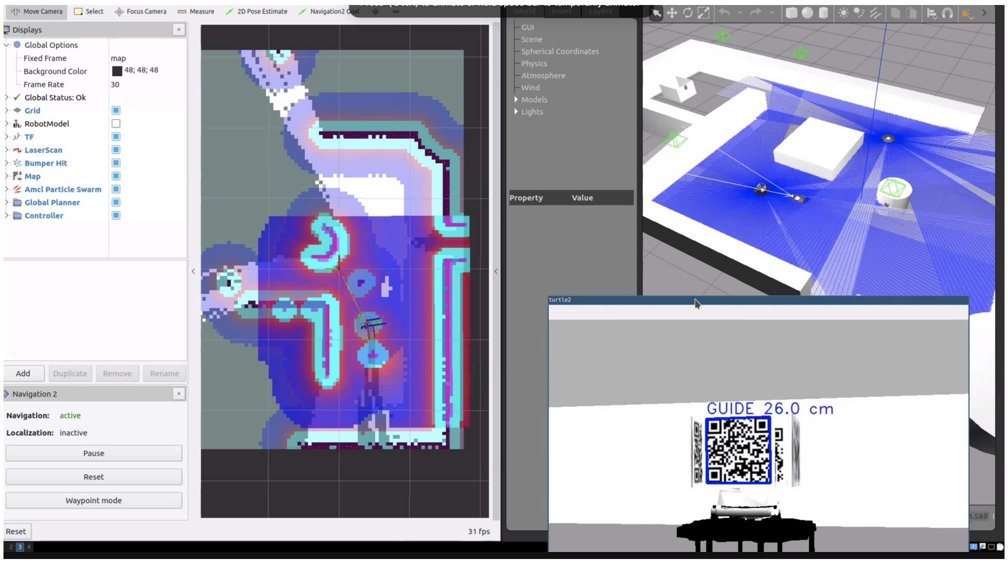
4.4. Bring Cooperation from Real World to 3D Virtual World

As the intermediate world of the three-world approach, we chose Gazebo[2] for our first simulation environment. Gazebo's close integration with ROS matches well with our work on real-world ROS robots up to that point, and it being the de facto standard for robot simulation on ROS also means that ample support is available from the open source community.

In many aspects, making the jump from Real World to Gazebo was quite straightforward. A model of Turtlebot3 was already available for Gazebo, and it controlled effectively the same as in the Real World. Case in point, the QR code following method implemented in Real World worked as-is in the simulated world too. What proved difficult instead was having multiple robots in the same Gazebo simulation. In a Real World environment, different robots can be assigned different domain IDs to avoid topic overlaps in messaging. In Gazebo, however, all simulated robots belong to the same domain by design, so so-called namespaces have to be used differentiate the topics. Unfortunately, as namespaces are no longer the preferred solution like they were in ROS1, many ROS2 components do not work very smoothly with them, resulting in some hack-like approaches required. Yet in the end, we have managed to get multiple Turtlebots running in the simulation, each complete with their own namespace and navigation stack.

For all that, running multiple robots in Gazebo presents us with a certain reality specific to the simulated world.

[2] http://gazebosim.org/



Fixed Frame map
Background Color 48; 48; 48
Frame Rate 30
Navigation: active
Localization: inactive

Figure 4: Cooperation running in the 3D Virtual World: Gazebo.

Increasing the number of robots simulated also increases the processing power required quite significantly. So far, we have been running Gazebo in a virtualized Ubuntu 20.04 on work use laptops. Just with three robots present in the simulation, the simulation runs at anything between 0.4–0.7 times of the ideal normal speed. Evidently this would indicate a need for a more powerful, possibly distributed solution, which we are looking into.

Though it can now be said that robot simulation too certainly has its own set of challenges, getting the Gazebo setup working has been quite beneficial in the end. The simulation environment can be edited at will, and resetting the simulation is naturally much simpler. Although, for reasons yet unclear, we are experiencing bugs with both to much inconvenience. Particular to our scenario, the light conditions are no longer an issue when it comes to QR code detection, and item delivery actions can be simulated simply by spawning and despawning items. Currently, Gazebo is our main platform for development, as seen with our newest demo.

4.5. Ontology Status

In 2D Block World, we have currently implemented a minimal extension to DUL with concepts related to cooperation and planning, i.e. goals, plans, workflows, tasks and actions, accompanied by a few physical object imitations residing in the environment, such as doors, which can be manipulated by completing the tasks (using particular actions).

Although the first simulations in 2D Block World seem promising, it is still unclear how much work one needs to do to provide similar functionality in 3D Virtual World or Real World for the physical objects. Sensing the states of the objects, e.g. is the door open or closed, may become a problem especially in Real World if the environment does not provide any support for it. However, this is not a problem that is unique to our approach as any autonomous robot encounters similar hardships in understanding its current situation.

5. Related work

In this section, we discuss on related research work on architectures enabling autonomous robot cooperation, leveraging ontologies for forming an understanding of the cooperation possibilities and situations, as well as task planning and decision making in the context of autonomous robot cooperation.

5.1. Architectures for Autonomous Robot Cooperation

Autonomous robots cooperating in uncertain and constantly changing environments have been studied for many years. The general interest in the overall topic has spawned several research subfields, e.g., swarm robotics [6], collaborative robotics (cf. [7]) and unmanned autonomous vehicles (UAV) (cf. [8]).

We find that the closest works related to our work from the architectural perspective are related to tightly coupled multi-robot cooperation. For example, Chaimowicz et al. [9] have studied architecture in which the key feature is flexibility which enables changes in leadership and assignment of roles during the execution of a task. While the approach allows dynamical behavior, the cooperation is yet tightly coupled. In our approach, each robot is expected to individually execute their tasks and then ask for help when needed. Hence the cooperation is less tightly coupled. In addition, the aim is not to jointly execute predefined tasks but instead, enable the robots to learn from their environment and their peers so that they could independently form new plans and meet their personal goals.

While Chaimowicz et al. also use the transportation of objects as an example, the same use case has been studied many times during the years. Recently, Zhang et al. [10] as well as Manko et al. [11] have studied control architecture that is using deep reinforcement learning in the transportation of large or heavy objects with a particular focus on decentralized decision making. While these approaches have similarities to our work, our work aims more for enabling individual robots to fulfill their personal goals instead of the group's goal. Hence our architecture would likely not be well-suited for such tightly coupled cooperation. However, we can learn from their experiences on how they use deep learning technologies and Q-learning-based algorithms for training the robots to execute a tightly coupled task, and in the future, we could try a similar approach in our 2D Block World.

5.2. Ontologies for Cooperation

Ontologies have been widely used to make agents and robots understand the structures of the physical and social world around them (see, e.g., Olivares-Alarcos et al. [12], Beetz et al. [2]), and initiatives considering their usage to build robot collectives that can communicate and cooperate have been suggested before, e.g., RoboEarth [13]. In contrast to RoboEarth, cooperation understanding and planning take place inside the individual robots in our architecture. The robots do not share their world views in general as they are assumed to hold also information that should not be shared with others, such as maps of restricted areas or passwords. Instead, they will only exchange information relevant to the current situation and goals directly with each other. That said, cloud-based solutions, such as RoboEarth, could be integrated into the architecture as optional components.

Mainly due to the advent of IoT, ontologies prove to be an exciting starting point for robots to understand the world as a built environment is getting populated with intelligent devices capable of communicating with other computational actors. This means that, e.g., a door can be opened using software communication alone and does not have to rely on physical door manipulation, and that sensors and other IoT devices may send information of their physical composition, purpose, and capabilities using ontological representations. This aids cooperation, especially on low-end robots, as the robot does not need to perceive these attributes from its raw sensor outputs such as camera streams.

5.3. Planning for Agents and Robots

Single robot planning may be approached from multiple perspectives. Two often used ones are heuristic shortest path search, such as the famous A* algorithm and its dynamic counterparts, and solutions used for logical optimization problems, e.g. (weighted) maximum satisfiability solvers. The shortest path search provides (estimates) for moving from one node to another in a graph and aims to find the path of nodes with the shortest length, and logical optimization aims to find a (maximal or minimal) set of clauses that satisfy certain conditions. Dynamic shortest path algorithms fit well in environments where the robot may not fully understand its situation, e.g., a complete map and logical optimization excels in cases where it is crucial to ensure the correctness of the solution beforehand.

However, our goal is to provide a planner that uses both logical verifications of the workflows through the fulfillment of each tasks' start and end conditions and a heuristic estimate of its execution resources through peer models and communication. Our approach differs from typical multi-robot task planning (see, e.g., Yan et al. [14]) in that one robot initiates the planning of the workflow phase (task decomposition), and it communicates, based on its peer models, with other robots to find suitable members to execute the tasks (task allocation).

6. Discussion

The implementation work has brought us numerous insights into the realities of cooperation in both real and simulated worlds. We have not yet faced any truly insurmountable issues, but many aspects make working with these environments not entirely straightforward.

In Real World, there are innumerable factors that can potentially affect the robots' ability to perform, such as the lighting as mentioned above conditions and cables on the floor. Of course, for our project's purposes, this would not seem a significant issue, as we can perform our tests in a carefully designed, controlled environment.[3] However, this does not remove the fundamental issue of unexpected factors. How would this uncertainty be dealt with within a hypothetical practical environment? One possible approach would be introducing some degree of "self-healing" properties in the design, both in terms

[3]In fact, we have long had plans to set up such an environment in the university campus, but the ongoing COVID-19 situation means these plans are still postponed.

of the robots' performance and the cooperation context. Currently, extensive work on this aspect is beyond the scope of this project, however.

In contrast to the unpredictable Real World, the simulated 3D environments are inherently about control and thus easier to work with. Nevertheless, there can still be considerable effort to set up a simulation the desired way, as seen with the difficulties in simulating multiple robots simultaneously in Gazebo. It also became apparent that multi-robot simulation can involve substantial hardware requirements. Still, we have found the Gazebo 3D simulation fulfills its purpose satisfactorily as a platform where cooperative actions can be developed for Real World (ROS2) robots in a more controlled manner.

However, it could also be noted that while the usage of 3D simulation does simplify some aspects, designing *Actions* for the *Task Runtime* remains an endeavor that relies on detailed knowledge in leveraging a particular robot's inner workings. Contrary to how the primary interests of this project are in the dynamic and creative aspects of robot cooperation, there remains a nontrivial effort necessary in creating the actual units of implementation, *Actions*. Future work could explore how to design the *Actions* more efficiently.

7. Conclusions

In this paper, we presented a new software architecture and development approach for diverse multi-robot cooperation. The core idea of the new approach is improving the situational context by developing and training peer models and an ontology that improves understanding of the world. The peer models enable the robots to take their peers' capabilities and goals into account in their reasoning, and the ontology can be used as a shared basis for communication and forming cooperation plans. The presented work is yet in its early stage, but we have already provided encouraging results and will continue the work.

Acknowledgments

The work was supported by the Academy of Finland (project 328729).

References

[1] J. Berrocal, J. Garcia-Alonso, J. Galán-Jiménez, J. M. Murillo, N. Mäkitalo, T. Mikkonen, C. Canal, Situational context in the programmable world, in: 2017 IEEE SmartWorld, 2017, pp. 1–8.

[2] M. Beetz, D. Beßler, A. Haidu, M. Pomarlan, A. K. Bozcuoğlu, G. Bartels, Know rob 2.0 — a 2nd generation knowledge processing framework for cognition-enabled robotic agents, in: 2018 IEEE International Conference on Robotics and Automation (ICRA), 2018, pp. 512–519.

[3] L. Chaimowicz, M. Campos, V. Kumar, Simulating loosely and tightly coupled multi-robot cooperation (2001).

[4] C. Castelfranchi, Modelling social action for AI agents, Artificial Intelligence 103 (1998) 157–182.

[5] A. Edwardes, D. Burghardt, M. Neun, Fipa communicative act library specification. foundation for intelligent physical agents, in: University of Maine: Orono, John Wiley Sons, 2000, pp. 377–387.

[6] L. Bayındır, A review of swarm robotics tasks, Neurocomputing 172 (2016) 292–321.

[7] S. El Zaatari, M. Marei, W. Li, Z. Usman, Cobot programming for collaborative industrial tasks: An overview, Robotics and Autonomous Systems 116 (2019) 162–180.

[8] N. Mathew, S. L. Smith, S. L. Waslander, Planning paths for package delivery in heterogeneous multirobot teams, IEEE Transactions on Automation Science and Engineering 12 (2015) 1298–1308.

[9] L. Chaimowicz, T. Sugar, V. Kumar, M. Campos, An architecture for tightly coupled multi-robot cooperation, in: Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164), volume 3, 2001, pp. 2992–2997 vol.3.

[10] T. Zhang, G. Liu, Design of formation control architecture based on leader-following approach, in: 2015 IEEE International Conference on Mechatronics and Automation (ICMA), 2015, pp. 893–898.

[11] S. V. Manko, S. A. K. Diane, A. E. Krivoshatskiy, I. D. Margolin, E. A. Slepynina, Adaptive control of a multi-robot system for transportation of large-sized objects based on reinforcement learning, in: 2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), 2018, pp. 923–927.

[12] A. Olivares-Alarcos, D. Beßler, A. Khamis, P. Goncalves, M. K. Habib, J. Bermejo-Alonso, M. Barreto, M. Diab, J. Rosell, J. Quintas, et al., A review and comparison of ontology-based approaches to robot autonomy, The Knowledge Engineering Review 34 (2019) e29.

[13] M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-lopez, Roboearth: a world wide web for robots, IEEE Transactions on Robotics and Automation 6 (2011) 69–82.

[14] Z. Yan, N. Jouandeau, A. A. Cherif, A survey and analysis of multi-robot coordination, International Journal of Advanced Robotic Systems 10 (2013) 399.