Thesis

Joris Z. van den Oever

19th December 2017

Abstract

To be written

Contents

1 Introduction

1.1 Problem Statement

Robotic systems have been getting more mobile and prevalent in recent years, and with that come new possibilities and uses. One project trying to take advantage of this is the TRADR project, Long-Term Human-Robot Teaming for Robot-Assisted Disaster Response[6]. It aims to create teams of robots and humans that work together to provide incidence response. Research into robot team communication is what prompted this research.

Autonomous robots in in disaster areas, including the industrial areas given by the TRADR use cases, need to be able to cooperate amongst themselves and with humans[6]. However because of the mobile nature of such robots this raises new questions about what happens when communication becomes unreliable. Wireless signals may not reach everywhere[8, 3] and wired communication isn't generally feasible when staying mobile.

This requires us to look closer at the effects of having communication fail. While it is easy to presume that this will have undesired effects it is hard to predict what effects that will be. Only once these effects, if they exist, have been measured we can look at potential mitigation strategies. And of course determine their effectiveness.

1.2 Research Questions

To this end we will provide answers to the following questions:

- What are the effects of communication failure of multi-agent team effectiveness at performing multi-stage tasks?
- How can we make multi-agent teams resilient to the negative effects of communication failure on performing tasks?

Here we define a single instance communication failure as the recipient not reading a message that was sent to it by the sender. Irregardless of where in the transmission process the failure occurred, e.g. while sending or receiving. And effectiveness is both the amount of times it successfully completes the task, and the time taken to complete the task.

1.3 Research approach

To gather the data we will create simulations of agent teams enduring communication failure. To run the simulations we will use a well explored domain for multi-agents systems: the Blocks World for Teams (BW4T) GOAL environment. It is a well defined domain for doing research on multi-agent systems.[5] The basic concept is that an agent, or multiple agents, have to bring blocks to a certain place, the dropzone in a specific order. The blocks are distinguished by color only so any block that is the right color will suffice. These blocks are found in rooms that have one entrance from the hallway spaces. While that is

the basic scenario it has built in several option for adjusting various environment variables such as the room placement. How the simulations are set up exactly is described in section 3.

The BW4T environment is very suitable for investigating communication failure while doing multi-stage tasks for several reasons. It implements a simple multi-stage task that can be completed without communication but teams are faster when agents do communicate. The environment only contains elements relevant to the multi-stage task and can be controlled. And GOAL is an open source programming language specially made for multi-agent systems often used with BW4T. So there already exist a lot of agents that can be used, either for the research or as examples. As well as GOAL having a robust agent communication support.

The simulations will be done in two parts. The first part will be exploratory; using pre-existing agent teams made to solve BW4T scenarios. These agents have been created without considering communication failure and will help identify potential stumbling blocks. The second part uses a set of specifically designed agents to control for different solving strategies and more closely evaluate the effects of different kinds of communication.

The differences in effectiveness from the second part will be used to see if the effects differ depending on how the communication fails and how well different agent strategies deal with the failure. The experiments will all be in a simulated environment as that allows for stricter control of the experimental dimensions in addition to making it easier to perform a large amount of experiment runs for each scenario. Afterwards the results of the simulations can be compared to baselines where no communication failure is happening to answer the first research question. Then based on the data we can see where the effects are strongest, and if there are agents that work well in different situations. From these we can identify potential mitigation strategies which can be simulated in the same scenarios as before to compare their effectiveness and answer the second question.

The gathered data will be analysed using Chi square analysis to determine if the results are significantly different from the baseline data. This will give us two analyses using both the one agent team baseline and the no communication team. For this comparison we will only look at the rate of task completion within the timeout period.

If this shows significant differences we can look for the effect the different communication failure scenarios on task completion time. As a first step regression analysis can see if there is an effect using both the type of team, size of the team, and the failure scenarios as independent variables. Pending determination on if they have a signicant effect on the task completion rate. As an additional analysis to if there is a difference in how the different team types are affected each team type will have a separate regression analysis using the team size and failure scenarios. The same analysis will be done using the task completion time as the dependent variable.

1.4 Exploratory research

The exploratory research was the initial research in agent communication failure using already existing components. It used pre-existing BW4T GOAL agents on one of the maps that come with the BW4T server, SuperRandom. These agents were only modified to introduce communication failure. We looked at several failure models to see if there might be significant differences compared to the no communication failure baseline. The full overview of the failure types and the results can be found in Appendix A.3.

The different agents were chosen to look at communication in different ways while still finishing the sequence under ordinary circumstances. The failures in the baseline are due to errors external to the agents. Some of them fixable in the experiment methodology but others resulted in fixes in GOAL to prevent them from happening during the main experiment.

One of the results to note is that the Maximum Coordination agent team differs from the others slighlty when it comes to communication. This agent uses the *SequenceIndex* percept to update how far into the sequence it is. Where the other agents only communicated about delivering blocks and ignored the percept. When looking at the communication failure results there is a significant difference in the amount of success simulation runs between agents using that percept and those that don't. While the results seem to converge when more communication failure occurs it is important to note this result in our more controlled experiments. If the effect is as noticeable there as in the exploratory research then not only could it be a way to mitigate the impact but it might also obscure other effects of communication.

Other than the potential effects of maintaining the task state outside of communication the results do not tell us much about the differences between the different failure models. We also realised that it might be useful to look at data other than task completion and task completion time to understand why the agents perform as they do. For example how many blocks were picked up and dropped into a room other than the dropzone. To get an insight into how much unnecessary work or incorrect actions were taken. Or how often delivery goals were dropped instead of completed.

2 Related Work

Multi-agent systems and how the agents interact is a field that has been explored from many angles. The BW4T testbed focuses on joint activities based on a simple planning problem adapted for multi-agent systems[5]. It applies few restrictions on the agent teams performing in the environment allowing research on various aspects of coordination, planning, and interaction between agents[4, 11, 12, 1]. Communication is important for these problems but much of the research in that direction relates to human-agent interaction[2, 1].

Multi-robot systems have physical contraints that influence how and when messages can be sent. As such there is research done on implicit communication[7]

or how to effectively utilised limited and/or shared communication channels[10]. This is becoming more relevant with the trend of multi-agent systems growing in size and complexity, such as the drone delivery systems currently in development for various companies[14].

These systems are expected to deal with unreliable communication at least some of the time and research is already being done to see how the communication systems can be made more reliable[13, 9]. However this research is mostly done from a perspective of detecting that a fault has happened or making sure that the communication systems work, rather than investigating what happens when communication fails.

3 Experiment Design

The experiment will consist of multiple simulation runs in the BW4T environment as discussed in section1.3. Each simulation instance requires an agent team to move around in a 2D environment to pick up and deliver blocks to a room called the dropzone in a given colour sequence. These blocks are found in other rooms in the 2D environment[5].

In this section we will discuss our choices for the BW4T environment variables, such as topology and block sequences, and agent setups such as the types of communication failure and team composition. The design of the agent programs is discussed in section 4. First the variables will be classified in the different experiment dimensions and then we will discuss how the options will be reduced and why. Finally we give an overview of the actual experiment parameters and how the simulations will be performed.

3.1 Experiment Dimensions

We have categorised the aspects involved in the experimental setup into three dimensions: environment, agents, and communication failure. These dimensions each have several variables as show in Table 1.

Environment BW4T offers several customisation options: topology, the sequence to deliver, and optional agent settings. At the time of writing the optional settings available are agent collisions and human controlled agents.

With topology we refer to the maps used for the simulations in BW4T. The topology specifies where and how many hallways, rooms, blocks, and agents there are. Hallways are spaces through which agents can move. These spaces can contain multiple agents. Rooms are special spaces that have a single door as entrance and only allow one agent inside at a time. However only rooms can contain blocks. Hallways are used to connect the rooms. There is one special type of room called the dropzone, which is where agents have to deliver the blocks of the sequence. These blocks can be a specific amount randomly distributed over the rooms or a block of a given colour can be assigned to a specific room. Any block that is put down in the dropzone or a hallway is

removed from the simulation. Finally we have agents. In this context the agents are the simulated entities the agent programs control. This means that any given map can only accommodate as many agents programs as there are agent entities present in the map definition.

The sequence in a BW4T context refers to a list of colours that the agent teams need to deliver to the dropzone. This list has to be delivered in order. If a block is delivered too early or to late it will be removed from the simulation. The sequence can be set to be random, however a given length or a list of colours can be provided. The two options can even be combined where the set of random blocks is appended to the specified list. It is important to note that when generating a random sequence a set of corresponding blocks is randomly distributed over the rooms. When specifying the sequence the blocks required to finish are not automatically created and distributed.

The first of the optional BW4T settings is agent collision. By default agents do not collide with each other. With the collision option turned on there would be additional complexity to the agents that does not need communication. As such it is not relevant to our research questions and wo do consider it in our experiment. Human controlled agents are similarly excluded because for this research we just look at the communication between agents without human intervention.

Agent Teams GOAL agents are organised in agent teams, the size of which can be adjusted and composed of various agents.

The agent team's size specifies the amount of agents present in the simulation. GOAL itself does not pose limits on the amount of agents; however in practice the available computing power and in the case of BW4T the map capacity does. Though implicit topology limitations, such as rooms allowing no more than one agent in a room, might limit the effectiveness of a team larger than the amount of rooms available or than there are blocks in the sequence.

Agent team composition defines what agent programs are used and if they are linked to the environment. GOAL allows an agent team to use different programs, a heterogeneous agent team as opposed to a homogenous agent team where all agents use the same program. Neither team composition requires an agent to be connected to the environment. This allows agents to potentially specialise or separate out code that does not directly affect the environment. An example of a type of agent that might not be linked to the environment is one that decides on the goals and divides them over the other agents that are. However for the purposes of this experiment disembodied agents add a layer of complexity that is not needed to make the agent teams function. As such we did not consider using disembodied agents as they make analysis more complex through additional layers of indirection.

Communication Failure Communication fails when a receiving agent does not get a message that another agent sent. However the failure can have several causes and have different impacts because of that. Does sending a message fail,

Environment	Topology	Block sequence
Agents	Team size	Hetero-/homogenous teams
Communication Failure	Send/receiving failure	Failure chance

Table 1: Experiment dimensions before reduction

or does receiving? Because if sending fails nobody gets the message, and if receiving fails a subset of agents might still get the message.

Or does communication fail for a fixed percentage of messages? Perhaps the failure rate increases over time? Is the message more likely to fail if the recipient is further away? While different failure models could have varying outcomes we are only looking to determine the effects of the failure and not the impact of communication failure models. Therefore we chose 'fixed percentage communication failure' as the model allows us the most control over the variable it introduces. Agent position and the time spent in a simulation can vary based on random choices the agents make, even if all the variables stay the same, leading to less consistent simulation outcomes.

Aside from the model of communication failure we also need to look at the information the BW4T environment communicates to the agents. GOAL provides this information to the agents as percepts. For this experiment we look at one percept in particular. BW4T provides updates on the sequence progress when a block is delivered correctly. This could effectively be a side-channel for task updates. This would always succeed as it does not go through the normal message system. As such agents should ignore this percept and only use the regular communication channels for this information.

3.2 Experiment reduction

The described experiment dimensions have too many options to realistically test everything. Just the environment can have an infinite amount of configurations by varying the maps and block sequence. However not all posibilities need to be tested, some of the variables only need a subset tested or can be eliminated by only using one of the options. How and why this is done will be discussed per dimension in the subsections below.

Environment

The map topology consists of several components, the size of the map, room placement, block placement, corridor placement, and agent entities.

The size of the map is how big the map is overall. Bigger maps can have more rooms, larger hallways, and can accommodate more agents. Room placement decides how many rooms there are and where on the map they are placed, as well as their size. More rooms means more places that agents need to look in find the blocks they need. Block placement defines the rooms blocks are in and how many blocks there are. If not all blocks in the sequence are present the simulation can not end. Blocks disappear if they are delivered at the wrong

time. As such block placement also determines how many 'spare' blocks there are present to cover for incorrect deliveries. Corridor placement determine the spaces agents have to move around in between rooms. These determine how many routes there are for agents to take.

Map size can be kept consistent if we make it big enough to contain all the variations we need, it does not need to be changed after that. The amount of rooms only influences how many options agents have to look at and how far they have to move to reach the rooms. Varying this does not change what needs to be communicated and as such only one appropriate number needs to be chosen. Corridors just need to connect the rooms that exist with the dropzone. As with the rooms this does not need varying. This means that all maps can be the same aside from the block placement. We start by excluding all block placements that make the sequence impossible to deliver. (Chris identified 12 block placement scenarios and made maps for them. How to correctly refer to that?) These 12 maps are all the same aside from block placement and lets us reduce the topology to 12 options. Each of these maps will have enough agent entities to accommodate the largest agent team that is used for the experiments.

For the sequence options there are two big distinctions for our purposes: all blocks are the same or the colours are mixed. When all blocks are the same it does not matter what order they are delivered, as all blocks of the same colour are equal. Such a sequence means that the current block in the sequence doesn't matter; only if the task is finished or not. This means that the agents will not have to communicate about the intermediate steps. Whereas the other option, a mixed colour sequence, does require communicating intermediate task completion. As such for the experiment we use sequences where the colours vary. The length of these sequences determines how much work the agents have to do but as long as there is more than one block this does not affect the kinds of communication. For the experiment we decided to stick to the default length used for the maps that come with the environment. This allows for comparison with potential other experiments. The sequence colours are dependent on the block placement and tie in to the 12 maps. (refer to the contents of those maps again consistent with the manner above.)

Agents

Outside the agent programming there are the two variables affecting agents we consider, the size of the agent teams and the team composition.

Limiting the sizes used for the agent teams will significantly reduce the experiment space as this is limited only by what the hardware can process. By taking a sampling of the team sizes we can get a sense of the effects, if there are any, communication failure might have. The benefit of using different sizes of teams is that we can evaluate if larger teams experience different effects or effects at different strengths than smaller teams.

The case of having only 1 agent would not need a lot of experimentation. There is nothing to communicate with, however it will still be useful to run the simulations. The single agent would provide a performance baseline for

the agents with communication. Each subsequent agent should reduce the time needed to complete the sequence.

Next, using a size sample of 3 agents will be useful to test, as this is the minimum amount of agents needed to have a team where it is possible to communicate with more than one agent at the same time. This is required for agents to potentially believe different subsets of the communicated information when communication failures occur.

The third team size would be 5 agents, as this is a larger group, but is also just a bit more than half the amount of rooms available in the default map setup and one less than the amount of blocks needed in the sequence. This means the way tasks get divided amongst agents becomes important as not every agent can continue exploring after just one set of rooms. The same holds for the sequence. Most agents will only be able to deliver one block and there will often not be enough blocks of a given colour to have every agent try deliver a given block in the sequence. Of course this also increases the amount of agents involved in communicating.

The final team size should not be too large, as experience with the environment has shown that 8-10 agents start to tax the hardware available to us and might degrade the simulation. This will also make teams with a size close to the amount of rooms available and it might be interesting to see how that influences the behaviour. Especially as teams would be larger than the amount of blocks in the sequence. A team of 8, one agent fewer than the amount of rooms in the chosen maps, has been chosen as the last group size.

All these agent teams still need a team composition. Team composition refers to the programs the agents follow. While we could go for a heterogenous team, this might introduce higher order effects that would make analysis more complicated, as well as requiring a lot more simulations. While these effects might be interesting, determining the effects, if any, in the simple case takes priority. So in the interest of limiting the time spent with simulations and analyzing, agent teams will be homogenous agent teams.

Communication

There are two variables to reduce regarding the communication: if the sending or receiving fails, and how often the messages fail to arrive.

When it comes to sending failure and receiving failure the difference is as follows: sending failure means that if a message fails no agent receives it, but receiving failure means that some agents in the team may receive it while other agents don't. The latter however increases variability between runs as the failure check is made for each receiving agent instead of just for the one sending agent. Yet we chose to use receiving failure. This is more likely to correspond to real world scenarios where communication is happening wirelessly. Due to various known environmental factors, such as interference and dead zones, one agent might not receive a message while another does.

The receiving failure will use the fixed percentage failure but that could be any amount between 0 and a 100% of the messages failing. Just like with team

sizes a sampling of failure percentages will suffice to identify trends. To limit the amount of runs needed the sampling will use 25% increments. However no messages failing and all messages failing could have different results from only a little or a lot of messages failing. Therefore 5% and 95% will be tested in addition to the 25% increments. Making the complete range of tested percentages $0, 5, 25, 50, 75, 95$, and 100%.

One final thing to take into consideration is that the exploratory data has shown that without intermediate task state correction agent teams can break down fast. If agents can not determine that an intermediate task has already been completed because they missed the message even a 5% failure can in many cases cut the amount of successful simulation runs in half. An overview of these results can be found in table 4. This effect is so strong that it would hide potential other effects that have less impact. As such we introduce another communication variable, whether an agent can update their sequence progress state when in the dropzone or not. When agents do update their state there would still be a cost of time spent to missing the message. Yet it would prevent mistakenly delivered blocks so the simulation does not end up in a state where the sequence can no longer be finished. Taking this into consideration the state correction will be the first failure mitigation strategy to implement and analyse. If necessary the data from these experiment runs will be used for further analysis of the failure effects.

3.3 Experiment Setup

Using the reduced dimensions the experiment will run each simulation with the following team sizes: 3, 5, 8. There will be 12 maps where only the block placement differs and all rooms are equidistant to the dropzone. (Chris' maps) Each agent team will attempt all maps repeatedly. These repeats are grouped by the communication failure and team size. One set of runs for 0, 5, 25, 50, 75, 95, 100 communication failure and the team sizes. See table 2 for an overview of the combinations.

This set of simulation runs needs repeats because of the random elements of the experiment such as the random chance of communication failure. All agent teams have some random aspects to their behaviour as well. When an agent is provided multiple valid options, an agent will make a random choice. As there is no prior knowledge on the effects of the random components, a desired experimental run time was used to determine the amount of repetitions performed. Based on earlier tests and data (including that of Chris how to reference?) a successful run will generally not take more than a minute. When each simulation run has a timeout of one minute that gets us 252 minutes to do everything once as a worst case scenario. If each agent program gets a week of runtime we can repeat the simulation set 40 times over the course of a week.

The experiment itself will be set up under the following conditions:

- All the simulations will be run on identical machines.

	failure %	0%	5%	25%	50%	75%	95%
Agent team	Size						
Baseline Agent	1	12 maps	N/A	N/A	N/A	N/A	N/A
Simple team	3	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps
	5	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps
	8	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps
Rooms and Blocks	3	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps
	5	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps
	8	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps
Proactive action coordination	3	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps
	5	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps
	8	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps
Reactive action coordination	3	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps
	5	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps
	8	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps

Table 2: Overview of experiment setup.

- The simulations will be run on a dedicated machine that only has the operating system running in addition to the simulations.
- Only one simulation is run at the same time.

These conditions exist to prevent the limit on available processing power on the system from influencing simulations, due to the software or capabilities differing between machines. These simulations will be run through a Python script to automate the entire process. It will gather and group the logs from both the agents and the server for each set of simulations with the same variable set for the experiment dimensions. Every simulation run is a new instance of the GOAL agents connected to a new instance of the BW4T server. This GOAL instance will be run with only one MAS file provided and a timeout. Before going to the next set of simulation runs with the same variables it will repeat the set 40 times instead of repeating once all simulations are done once. **(Validate if repeats should be done in GOAL or the Python script.)**

The resulting data will be analysed after being converted into a csv file as described in section 1.3. To reduce the absolute amount of time needed each agent program will use a duplicated instance on Amazon EC2 running in parallel. As the instances are virtualised in the cloud their performance is identical so it will still satisfy the first condition. The instance will be a standard *t2.micro* with only the additional software the programs needed to run the simulations and organise log outputs.

The log output of the BW4T server and the GOAL runtime of each simulation run will be packaged together for later processing once all simulation runs are done. How the logs will be processed is detailed in section 5.

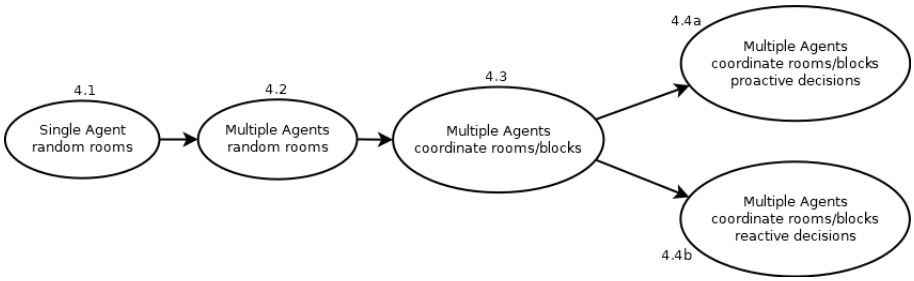

Figure 1: Agent types, each agent team has the capabilities of the previous one in the graph

4 Agent Design

The agent programs are written in an iterative manner as shown in figure 1. There is one base agent team consisting of one agent and the following agent teams build upon that by adding different forms of communication. For communication we need multiple agents so all agent teams except for the base team will have multiple agents. The communication methods are added in 3 iterations. The last iteration implements 2 variants so there are 5 different agent teams an overview of which is given in figure1. The single agent team acting as the baseline for the performance of the other agent teams. Each of the 5 agent programs will be discussed below. The baseline agent defines the default behaviour of the agent and the following sections specify what form of communcation is added and how this affects the behaviour. Because it is a single agent the resulting task completion data can also be used as a baseline for determining how effective the communication is in improving the agents. The full code for the agents can be found in appendix A.

4.1 Baseline Agent

The baseline agent team is a single agent performing a random exploration algorithm. It remembers which rooms are explored and the blocks that present. The exploration is interrupted by a greedy delivery algortihm when it finds a room with a block of the current colour in the sequence. When the block is delivered it will check if it knows where the next colour is located and deliver that too. If there are more blocks of the same colour it will choose one at random. If the agent has not found blocks of the right colour yet it will return to the exploration algorithm. Because it only delivers blocks that are of the current colour in the sequence and is the only agent it is garuanteed to always succeed in the maps defined in the experiment design subsection3.2.

The agent achieves this by performing the following steps:

- First the events module processes all percepts it receives from the environment to update its beliefs.

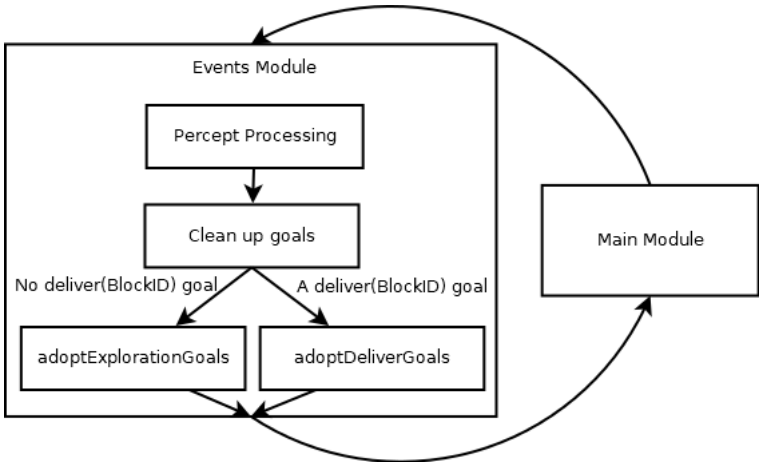

Figure 2: A high level flow diagram of the agent modules.

- With the updated beliefs it determine if the current goals are still possible and relevant. If not the goals are dropped. This will never happen with a single agent as it is the only entity affecting the environment but the step exists so derivative agents can do it when necessary.
- If the agent has a goal to deliver a block it goes into the adoptDeliverGoals module which provides the intermediate goals to get the block delivered to the dropzone.
- If there is no block to be delivered at this moment in time it goes into the adoptExplorationGoals module. This provides the goals to implement the greedy exploration algorithm described earlier in the section and the goal to deliver a block if the right colour is found.
- Then the agent goes into the main module which performs the actions corresponding to the various goals.
- Finally, if the agent has delivered the entire sequence it stops.

The goal modules and the main module will be discussed in more detail below. The percepts will not be discussed as they are available in appendixA.1 and further clarification of the details can be found in the BW4T specification. The evaluation of the goals is skipped because there are no other agents yet that can influence the environment as well.

adoptExplorationGoals

First we look at the adoptExplorationGoals module. The way we indicate that we want to go to a room is with the $in(Place)$ goal. The main module reacts to

this by going to a room with the name *Place*. Once the agent is in that room we get a percept and process it to fulfill the goal. The way we adopt the*in*(*Place*) goal in this module provides the exploration behaviour:

```
1 if bel( room(Place), not(dropZone(Place)), not(visited(
    Place)) ) then adopt( in(Place) ).
```

This rule looks at all rooms that are not the dropzone, this is the location where the blocks have to delivered in BW4T, where the blocks will be delivered. The agent learns of these locations when it is initialised. If those rooms are entered for the first time adds a *visited*(*Place*) belief in the events module. This belief lets the agent exclude visited rooms for future exploration goals.

Without additional configuration this rule would not implement a *random* exploration however. To achieve this the module has to use linear random order to check its rules. This execution order means that the various rules in the module are executed in linear order, stopping after the first rule that succeeds. When evaluating a rule random valid options for the variables are be chosen from all valid options. In the rule above that means that it chooses *Place* at random from all *room*(*Place*) beliefs instead of always choosing the first in the belief base. This differs from the default, linear order, which pick the first valid option in the belief base instead.

The module contains another rule to determine if it has found the next block in the sequence. Because the module uses linear random order this rule has been placed at the start of the module so delivering blocks takes precedence over exploring. The following rule is the end result of the agent team iterations so it is more complex than strictly necessary for the baseline agent team. The reason the rule is in this form is because the agents share code to ensure they are the same aside from the communication.

```
1 if a-goal(sequence(Seq)), bel( agentCount(N),
    nextXColoursInSeq(Colors, N, Seq),
2       findall(ColorID, (delivering(_,ABlock), block(
            ABlock, ColorID, _)), TakenC),
3       single_subtract(Colors, TakenC, UntakenC), member
            (Color, UntakenC),
4       block(BlockID, Color, Place), not(Place = held) )
            then adopt( delivered(BlockID) ).
```

However for the baseline agent this rule can be respresented in a simpler form. Here just the simple version will be explained and the rule will be discussed again in subsection4.3 and subsection 4.4 below to explain the rest. Introducing the more complex aspects as they become relevant for the discussed agent team. At this point in discussing the agents the rule can actually be represented by the following simplified form:

```
1 if a-goal(sequence(Seq)), bel( nextXColoursInSeq([Color],
    1, Seq),
```

```
2           block(BlockID, Color, _)) then adopt(delivered(
                BlockID)).
```

As long as we still have a sequence of colours to be delivered it gets the first colour of the sequence that hasn't been delivered yet. This is done by using the $nextXColoursInSeq(ColourList, X, Seq)$ predicate. $ColourList$ is the list of the first X colours in the sequence excluding the blocks that already have been delivered. As $X = 1$ the predicate always returns the colour that needs to be delivered at this moment.

Then the rule selects a random block from the the ones that were found in all visited rooms with the same color and adopts the goal to have it delivered. This goal does nothing on its own but is used by adoptDeliverGoals module to provide the intermediary goals. The reason that the block chosen is random is that going for the closest block will involve an additional path-finding algorithm in case after delivering there are multiple blocks of the same colour. The agent is purposely kept simple so we chose not to implement that.

If there are no blocks of the right colour the rule fails and will return to the random exploration described earlier.

adoptDeliverGoals

The delivery algorithm works in a similar manner. It also contains rules that are only relevant for later agent teams so they will be discussed in the relevant subsections. For the baseline agent we will look at two rules, the first is delivering a block to the dropzone if the agent holds it, and the other is getting to the block that needs to be delivered to pick it up.

```
1 if a-goal(sequence(Seq)), bel(holding(BlockID),
    nextColorInSeq(Color, Seq), block(BlockID, Color, _),
     dropZone(Place)) then adopt(in(Place)).
```

Once again it shares code with later agents so for the baseline agent we examine it in a simpler form:

```
1 if bel(holding(BlockID), dropZone(Place)) then adopt(
    in(Place)).
```

If the agent is holding the block, check which room is the dropzone, adopt the goal to go there. The $BlockID$ is passed to the module as a parameter so this rule will not trigger if the agent is holding a block that it does not need to deliver. The main module is responsible for dropping the block once the agent is in the dropzone.

Getting to the block makes use of the $BlockID$ variable as well to determine the room the block is in. Then adopt the *in* goal so the main module goes there. The main module then makes sure to pick up the block once the agent is in the room.

```
1 if bel( not(holding(BlockID)), block(BlockID, _, Place) )
    then adopt( in(Place) ).
```

main

Where the goal modules are about the agents reasoning, the main module acts on it. The order of the rules in this module matters as it determines the priority of what needs to be done. As such this module performs the rules in linear order.

The first rule is delivering the block the agent holds if it is in the dropzone and still holding the block. Delivering here means putting down the block and going into the updateSequence module. This module updates the *sequence* belief to include the colour of the block just delivered. And it removes the block from the beliefs to prevent the agent from trying to deliver the block again.

Then if the agent is in a room with the block it wants to deliver it should move towards the block and do the goToBlock block action. This will fail if the agent is already at the block. This is followed by a separate rule that attempts to pick up the block if it is not holding anything.

Finally the agent checks if there are any *in*(*Place*) goals and if there are it performs a goto action to go there. If the agent is already moving somewhere the goto action fails.

4.2 Simple agent team

The first step to introduce communication to a single agent would be the same agent but with several in the agent team. With one change to guarantee that they would always be able to finish the task. The agents need to be informed when the current block in the sequence has been delivered. If the agents do not know what the other agents have done they would attempt to do the entire sequence on their own. This can make it impossible to finish instances if there are not enough blocks of each colour to do the sequences in parallel. All blocks delivered to the dropzone disappear from the environment so they can not be reused later. So in case a block is needed more than once in a sequence the agents can never finish if all blocks of a certain colour are delivered before they are needed.

The agents can be informed of what the other agents have delivered in several ways, which have been discussed in subsection3.2 in the part about communication. In our case we decided on having the agent sending a message to all other agents whenever it delivers a block to the dropzone. The communication clause added to the rule is the following:

```
1 allother.send( msg(delivered(BlockID)) )
```

The message is processed by the new comBasic module that is performed before any of the steps the baseline agent takes. The module will then perform the updateSequence module as in the main module but with the BlockID received in the message.

```
forall (Agt).sent(msg(delivered(BlockID))) do
    updateSequence(BlockID).
```

Here the benefit of removing the blocks in the updateSequence module shows itself. The receiving agent might have seen the same block in a room before and still believe it is there. If this belief is not updated it will get stuck trying to deliver a block that no longer exists.

When an agent receives $msg(deliveryDone(BlockID))$ there are four situations it could be in;

- The agent is holding a block and its colour is the next colour in the sequence. It will continue trying to deliver the block.
- The agent is not holding anything and trying to pick up a block. If the next colour in the sequence is the same as before the agent will continue.
- The agent is holding a block and its colour is not the next colour in the sequence. In that case the agent follows the baseline exploration algorithm as described in section 4.1 with one difference. The first time it enters a room it will drop the block it is holding. This is to prevent the risk of losing a block that is needed later because all blocks dropped outside of a room will disappear from the environment.
- The agent is not holding anything and the conditions above do not hold. In that case the agent continues to follow the baseline exploration algorithm as described in section4.1.

There is one situation where this can cause a deadlock. When the agent is without options while in the dropzone. It has already explored all rooms but due to the moving around of blocks by other agents it is not aware where the next needed block might be. So it will not go anywhere. There is nothing it can do but at least one of the other agents does know. The agent that moved the relevant block. That agent can not deliver it however because only one agent can be in a room at any given time. So the delivering agent has to wait for the dropzone to be vacated but that will never happen. To prevent this an additional line as been added at the end of the adoptGoals module:

```
if bel( dropZone(Place), in(Place) ) then adopt( at('
    FrontDropZone') ).
```

With these changes agents should now always complete the scenario. After all, the dropzone will always become available for the next delivery, and there is always at least one agent that knows the location of any given block if a room has been visited. And no blocks will be incorrectly delivered as the agents share their progress. From here the agents follow the same algorithm as the baseline agent so there is always one agent that can deliver the next block. However the deadlock we discussed shows that there is still room for performance improvement.

4.3 Room and block coordination agent team

Like the deadlock described above it can also happen that an agent will enter a room expecting a block to be there and not find it. This will update its knowledge on what is present in the room but it won't know where to find the block. If the agent doesn't know of any other blocks of the desired colour it will return to the exploration algorithm. If all rooms have already been visited the agent has no way to find where the block ended up. This is not a deadlock as one agent will still know the block location but through communication the additional wait can be avoided.

The agent team does this by introducing the comBlocks module. This module sends a message for each block it sees for which it gets new information and also processes those messages it gets from other agents. This happens before the events module. This means that not only do the agents send updates when a block is put down in a different room but also that a message is sent the first time a block is seen. This is not something we want to avoid as it can be used to further optimize the agent. The agent receives the information on blocks others have found and can use that to determine if there is a block it can deliver.

The code below ensures that all agents in the team listen for updates on block locations:

```
1 forall (Agt).sent( msg(block(BlockID, ColorID, Place)) ),
      bel( block(BlockID, ColorID, OtherPlace),
2         Place \= OtherPlace ) do delete( block(BlockID,
              ColorID, OtherPlace) ) +
3         insert( block(BlockID, ColorID, Place) ).
4
5 forall (Agt).sent( msg(block(BlockID, ColorID, Place)) ),
      bel( not(block(BlockID, ColorID, Place)) )
6         do insert( block(BlockID, ColorID, Place) ).
```

These two lines process *block* messages for two possible cases, the first updates block location information in the case an agent already knows of a block with the same $BlockID$. The second line inserts beliefs for blocks the agent was unaware of. This pattern prevents duplicate blocks where the location differs.

The agents also share if they are holding a block to prevent other agents from trying to pick up a block that was already grabbed. It updates the $Place$ for the $block(BlockID, ColorID, Place)$ predicate to 'held' once an agent picks up the block. Once the agent drops the block in the dropzone the block belief is removed by $BlockID$ through the updateSequence module by all agents. If an agent drops a block in a different room, it sees that the block is now present in the room and the location is updated as a percept by the events module. Seeing the block in the room updates the location again as described above. Agents only see blocks that are put down in a room. Below is the code showing how the $heldblock(BlockID)$ message that the agents use to communicate this is processed:

```
1 forall(Agt).sent(msg(heldblock(BlockID))), bel(block(
    BlockID, ColorID, Place))
2                 do delete(block(BlockID, ColorID, Place)
                      ) + insert(block(BlockID, ColorID,
                     held)).
```

One thing to note is that the adoptExplorationGoals module has an addition because of this change. A condition is added to the rule that adopts the *delivered* goals to make sure the block the agent considers for delivery is not already held by another agent. However the block can still be picked up between adopting the goal and getting to the block. This will cause the agent to try and deliver a block it can not locate so we added a rule to drop the *delivered* goal when that occurs.

Finally because the agents already share the information on blocks the agents do not need to visit rooms that other agents have visited. They already know which blocks are in there. Therefore the comVisited module was introduced to stop agents from visting rooms they do not need to. This module sends a message when an agent visits a room for the first time. While this information could be derived from block communication this provides more clarity in what the communication is intended for. The message sent is identical to the *visited*(*Place*) belief used by the baseline exploration algorithm and treated the same by the receiving agents.

4.4 Proactive/reactive action coordination agent teams

The communication added in section4.3 leaves one situation where an agent would do nothing. When all rooms have been visited and there is only one block of the current colour in the sequence. If another agent has already picked up the block it will wait until it can do something again. This is because all agents are trying to go for the same block in the sequence instead of dividing the work.

The final two agent teams implement nearly identical planning algorithms to divide the work. The reason there are two teams is because we have identified two ways the agent teams can communicate to coordinate their actions. These two methods should have similar results when the communication has no failures but could have different performances once failures are introduced. The full details on how the communication is done for each team is discussed below. Before that the parts of the algorithm that are the same are discussed.

The teams both communicate which block they intend to deliver to the other agents. The communication they share is part of the lookahead module. This is where the agents communicate the intent to deliver a block and also if the agent no longer intends to deliver a block for any reason.

The adoptExplorationGoals module has a rule to adopt the *deliver*(*BlockID*) goal and follows the following steps: if a different agent is already attempting to deliver a block of the current colour in the sequence, the agent will behave as if that colour has already been delivered, and so on for each colour that an agent

is delivering. The only deviation from the baseline behaviour is that when an agent has been working ahead it will wait with the block in front of the dropzone until it should be delivered.

To understand how this is achieved we look again at the rule where the agent adopts *delivered* goals we also showed in subsection4.1.

```
1 if a-goal(sequence(Seq)), bel( agentCount(N),
    nextXColoursInSeq(Colors, N, Seq),
2       findall(ColorID, (delivering(_,ABlock), block(
            ABlock, ColorID, _)), TakenC),
3       single_subtract(Colors, TakenC, UntakenC), member
            (Color, UntakenC),
4       block(BlockID, Color, Place), not(Place = held) )
            then adopt( delivered(BlockID) ).
```

First we look at $agentCount(N)$. This predicate only returns the amount of agents if on initialization the agent added the *lookahead* belief. Otherwise it returns 1. That's why the previous agents only attempt to deliver the current colour in the sequence.

Then $NextXColoursInSeq(Colors, N, Seq)$ gives us list of the colours as $Colors$ that still need to be delivered to a maximum of N colours. It could be less in case there are less than N colours to be delivered. This is combined with the list $TakenC$ which is the list of all colours we know another agent is trying to deliver. The list is created by finding all $delivering(Agent, BlockID)$ beliefs and getting the corresponding colours.

The list $Colors$ then has the list $TakenC$ subtracted from it, this list is called $UntakenC$, leaving only the colours no other agent is delivering. Each colour already being delivered is only subtracted once in case there are duplicate colours in the sequence. The agents should not skip a colour entirely because one block of the relevant colour is already being delivered.

The agent then picks a random element from $UntakenC$ which is the colour it will try to deliver. The rule then follows the same principles as with the baseline agent with the addition for held blocks.

4.4a Reactive action coordination agent team This team is called reactive because the agent sends a message broadcasting which block it wants to deliver but does not act on it. Then the agent reacts to the other agents in the team telling it what to do. Only once all agents give it the okay will the agent start on the common algorithm described above.

If another agent planned the same goal, it resolves the conflict using the following method: compare the agent names assigned by the BW4T environment through string comparison. If the other agent's name is larger that agent gets to deliver the block. The agent doing the check informs the original agent by sending a dismissal. The original agent will in turn confirm the plan of the agent with the larger name as its name is smaller.

Success Guaranteed	Could fail
-	sequenceUpdate broadcast fails: wrong blocks de
room/block information broadcast fails: slows down	-
Agent Intention Broadcast: slows down	-
-	Agent plan negotiation message failure: Might wait fo

Table 3: Table of the effects communication failure might have. On the left success is guaranteed if given time. On the right situations may arise where success is no longer possible.

The actions above are done after the adoptGoals steps in the baseline agent because it needs the latest information to make the decision.

4.4b Proactive action coordination agent team Where the reactive agent team waits confirmation or dismissal, the proactive agent team acts first and verifies later. The team proactively adopts the goal first and then verifies that nobody else is doing the same. The agents use the same conflict resolution method as the reactive team but instead of sending a message on if it approves or disapproves it drops its own *delivered* goal if it has the smaller name in string comparison. Then it communicates that it has dropped its goal since it could infer that someone else is doing the same and will continue to do so.

The above steps are done after the adoptGoals steps in the baseline agent. Because it needs the latest information to make the decision.

5 Analysis

Data points to analyse: amount of succesful runs, time of successful runs (less reliable since we expect to have less successes), amount of blocks returned to rooms (incorrect/unnecessary blocks), difference between the dropzone correction and without.

Table 3 gives us an overview of what was expected to happen in the various scenarios. While even the guaranteed successful scenarios might not finish within the timeframe given for the experiments they should more often do so than in scenarios where success might not always be possible. In these cases different strategies can be compared for how much they slow down compared to the reference and what the major causes for slowdown are.

The types of communication that can cause failure do this for different reasons. The first does this by incorrect information, the place in the sequence, leading to wasting resources and making the task impossible. The second one does it by having the agent logic deadlock and cause them to not do anything. It should be noted that in the second case it is theoretically possible to still finish as opposed to the first situation. {{Does this mean anything? Like make it more important to fix as it is only limited by programming?

6 Mitigation Strategies

7 Future Work

References

[1] Maaike Harbers, Jeffrey M Bradshaw, Matthew Johnson, Paul Feltovich, Karel van den Bosch, and John-Jules Meyer. Explanation and coordination in human-agent teams: A study in the bw4t testbed. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2011 IEEE/WIC/ACM International Conference on*, volume 3, pages 17–20. IEEE, 2011.

[2] Maaike Harbers, Catholijn Jonker, and Birna Van Riemsdijk. Enhancing team performance through effective communication. In *Proceedings of the 4th Annual Human-Agent-Robot Teamwork Workshop*, pages 1–2, 2012.

[3] Cisco Systems Inc. 20 Myths of Wi-Fi Interference: Dispel Myths to Gain High-Performing and Reliable Wireless , 2007. http://www.cisco.com/c/en/us/products/collateral/wireless/spectrum-expert-wi-fi/prod_white_paper0900aecd807395a9.pdf [Online; accessed 10-Februari-2016].

[4] Matthew Johnson, Jeffrey M Bradshaw, Paul Feltovich, Catholijn Jonker, Birna van Riemsdijk, and Maarten Sierhuis. Autonomy and interdependence in human-agent-robot teams. *IEEE Intelligent Systems*, 27(2):43–51, 2012.

[5] Matthew Johnson, Catholijn Jonker, Birna Van Riemsdijk, Paul J Feltovich, and Jeffrey M Bradshaw. Joint activity testbed: Blocks world for teams (bw4t). In *International Workshop on Engineering Societies in the Agents World*, pages 254–256. Springer, 2009.

[6] Ivana Kruijff-Korbayová, Francis Colas, Mario Gianni, Fiora Pirri, Joachim de Greeff, Koen Hindriks, Mark Neerincx, Petter Ögren, Tomáš Svoboda, and Rainer Worst. Tradr project: Long-term human-robot teaming for robot assisted disaster response. *KI-Künstliche Intelligenz*, 29(2):193–201, 2015.

[7] C Ronald Kube and Hong Zhang. Collective robotics: From social insects to robots. *Adaptive behavior*, 2(2):189–218, 1993.

[8] D Micheli, A Delfini, F Santoni, F Volpini, and M Marchetti. Measurement of electromagnetic field attenuation by building walls in the mobile phone and satellite navigation frequency bands. *Antennas and Wireless Propagation Letters, IEEE*, 14:698–702, 2015.

[9] Lynne E Parker. Alliance: An architecture for fault tolerant multirobot cooperation. *IEEE transactions on robotics and automation*, 14(2):220–240, 1998.

[10] Paul E Rybski, Sascha A Stoeter, Maria Gini, Dean F Hougen, and Nikolaos P Papanikolopoulos. Performance of a distributed robotic system using shared communications channels. *IEEE transactions on Robotics and Automation*, 18(5):713–727, 2002.

[11] Changyun Wei, Koen V Hindriks, and Catholijn M Jonker. Multi-robot cooperative pathfinding: A decentralized approach. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 21–31. Springer, 2014.

[12] Changyun Wei, Koen V Hindriks, and Catholijn M Jonker. The role of communication in coordination protocols for cooperative robot teams. In *ICAART (2)*, pages 28–39, 2014.

[13] Alan FT Winfield. Distributed sensing and data collection via broken ad hoc wireless connected networks of mobile robots. In *Distributed Autonomous Robotic Systems 4*, pages 273–282. Springer, 2000.

[14] Tiger TG Zhou, Dylan TX Zhou, and Andrew HB Zhou. Unmanned drone, robot system for delivering mail, goods, humanoid security, crisis negotiation, mobile payments, smart humanoid mailbox and wearable personal exoskeleton heavy load flying machine, May 23 2014. US Patent App. 14/285,659.

A Goal Agents Source

The agent teams each have their own main module. However a lot of the other modules are shared as they share capabilities. First the different agents are detailed. Then the shared modules and knowledge.

A.1 No Communication Agent

This agent does not communicate and serves as the basis for both the single agent team and the no communication multiple agents team. The only difference between the two is the amount of agents instantiated by the .mas file.

```
1 use robot as knowledge.
2 use robot as actionspec.
3 use updateSequence as module.
4 exit=nogoals.
5 module main {
6         %drop blocks in the dropzone when they are needed
              , and communicate this.
7         if bel(in('DropZone'), holdingNextBlock) then
              putDown + updateSequence.
8
```

```
9           %drop blocks in rooms when they are not needed
                anymore
10          if bel(in(Loc),Loc\='DropZone', not(
                holdingNextBlock)) then putDown.
11
12          %go to goal places
13          if a-goal(in(Place)) then goTo(Place).
14
15          %go to goal blocks
16          if a-goal(holding(BlockID)), bel(in(Place), block
                (BlockID,_,Place), not(atBlock(BlockID))) then
                 goToBlock(BlockID).
17
18          %pickup goal blocks
19          if a-goal(holding(BlockID)), bel(atBlock(BlockID)
                ) then pickUp.
20  }
```

events.mod2g

```
1   use robot as knowledge.
2   use robotGoals as module.
3   use updateSequence as module.
4
5   module robotEvents {
6           %-------communication updates & conclusions about
                 other agents-------
7           forall bel( send('allother', Y) ) do allother.
                send(msg(Y)) + delete(send('allother', Y)).
8
9           % deduce information based on deliveries of other
                 agents:
10          forall (Agt).sent(msg('deliveryDone')) do
                updateSequence.
11
12          % Update the agent's state of movement.
13          forall bel( state(State)), percept(state(NewState
                )) do delete( state(State) ) + insert( state(
                NewState) ).
14
15          % Record when we are entering or leaving a room.
16          forall percept(in(Place)) do insert( in(Place) ).
17          forall percept(in(Place)), bel( not(visited(Place
                )), room(Place) ) do insert( visited(Place) ).
18          forall percept(not(in(Place))) do delete( in(
                Place)).
```

```
19
20          % Discover new blocks
21          forall percept(color(BlockID, ColorID)), bel( in(
                Place), not(block(BlockID, ColorID, Place)) )
                do insert( block(BlockID, ColorID, Place) ).
22
23          % Record atblock location of agent
24          forall percept(atBlock(BlockID)) do insert(
                atBlock(BlockID)).
25          forall percept(not(atBlock(BlockID))) do delete(
                atBlock(BlockID)).
26
27          % Record if a block is being held
28          forall percept(holding(BlockID)) do insert(
                holding(BlockID)).
29          forall percept(not(holding(BlockID))) do delete(
                holding(BlockID)).
30
31          %remove blocks that are not held or in the room
                anymore
32          forall bel(in(Place), block(BlockID, ColorID,
                Place), not(holding(BlockID))), not(percept(
                color(BlockID, ColorID))) do delete(block(
                BlockID, ColorID, Place)).
33
34          %remove blocks that are not held or in the room
                anymore
35          forall bel(in(Place), block(BlockID, ColorID,
                Place), not(holding(BlockID))), not(percept(
                color(BlockID, ColorID))) do delete(block(
                BlockID, ColorID, Place)).
36
37          % Update sequence when in dropzone.
38          if percept(sequenceIndex(X)), bel(sequenceIndex(
                OldX)) then delete(sequenceIndex(OldX)) +
                insert(sequenceIndex(X)).
39          if bel( in('DropZone'), seqDone(Seq), length(Seq,
                 N), sequenceIndex(X), N < X , sequence(
                NewSeq), length(NewSeq, X), append(NewSeq, _,
                Full) ) then delete( seqDone(Seq) ) + insert(
                seqDone(NewSeq) ).
40
41          %remove obsolete goals
42          if goal(holding(BlockID)), bel(not(block(BlockID,
                 ColorID, Place))) then drop(holding(BlockID))
                .
```

```
43
44          %adopt new goals (and stop traveling)
45          if not(goal(in(Place))), goal(seqDone(_)) then
                adoptgoals.
46  }
```

A.2 Common Code

robot.act2g:

```
1   use robot as knowledge.
2
3   % The goTo action makes the agent move to a place (
        location) in the BW4T environment.
4   % As long as the agent has not arrived at the place it is
        going to, it will be in "traveling" mode.
5   define goTo(Location) with
6           pre { not(state(traveling)) }
7           post { true }
8
9   % goToBlock only when not traveling and in a room
10  define goToBlock(BlockID) with
11      pre {in(_), not(state(traveling)) }
12          post { true }
13
14  % pickUp can only be performed when not holding a block
15  define pickUp with
16          pre{not(state(traveling)), not(holding(_))}
17          post{ true }
18
19  % putDown can only be performed when holding a block
20  define putDown with
21          pre{not(state(traveling)), holding(_)}
22          post{ true }
```

UpdateSequence.mod2g

```
1   use robot as knowledge. order=linearall.
2
3   module updateSequence{
4           %update sequence
5           if bel(seqDone(SDone), nextNeededColor(ColorID),
                append(SDone,[ColorID],NewSDone) ) then delete
                (seqDone(SDone)) + insert(seqDone(NewSDone)).
6
7           %remove beliefs about the delivered block (if
                this agent was delivering it)
```

```
8           if bel(in('DropZone'), holding(BlockID), block(
                BlockID, ColorID, Place)) then delete(block(
                BlockID, ColorID, Place)).
9   }
```

robotGoals.mod2g

```
1   use robot as knowledge. order=linear.
2
3   module adoptgoals{
4           %If holding the next needed block go to the
                dropzone.
5           if bel( holding(BlockID), nextNeededColor(ColorID
                ), block(BlockID, ColorID, _)) then adopt(in('
                DropZone')).
6
7           %Otherswise, If the next needed block is known
                then adopt a goal to go there and hold it.
8           if bel( nextNeededColor(ColorID), block(BlockID,
                ColorID, Place)) then adopt(in(Place),holding(
                BlockID)).
9
10          %Otherwise go to a random room we haven't seen
                yet'.
11          if bel( not(finished), bagof(Place, (room(Place),
                 not(dropZone(Place))), Places), random_member
                (Dest, Places) ) then adopt(in(Dest)).
12  }
```

robotInit.mod2g

```
1   use robot as knowledge.
2
3   module robotInit {
4           % Store map information, i.e., navigation points
                in the agent's belief base.
5           forall percept(zone(ID, Name, X, Y, Neighbours))
                do insert( zone(ID, Name, X, Y, Neighbours) ).
6
7           % Record the initial state of movement in belief
                base.
8           if percept(state(State)) then insert( state(State
                ) ).
9
10          % Record goal sequence
11          if percept( sequence(Seq) ) then insert(sequence(
                Seq), seqDone([])) + adopt(seqDone(Seq)).
```

Agents\chance of failure	reference	communication cuttoff	5%	10%	25%	50%	95%	50% send
Blockbuster Mincom	193	101	109	98	74	64*	63	45'
Blockbuster dzone only	199	11	98	50	15	5	16	
Blockbuster all room	190	18	85	33	13	52	38	60'
Maximum Coordination	188	168	182	182	162	138	64	

Table 4: The amount of successes over 200 runs per agent per failure mode. All agent and failure mode combinations resulted in a chi square $p < 0.01$ except for Coordination at 5%($p \approx 0.07$) The entries marked with * have been performed with fewer than 200 runs and scaled up to the same amount so less accurate.

```
12          if percept( sequenceIndex(X) ) then insert(
                sequenceIndex(X) ).
13
14          % Adopt initial goal going to a random place
15          if bel(room(PlaceID), PlaceID\='DropZone') then
                adopt(in(PlaceID)).
16  }
```

A.3 Exploratory research data

References

[1] Maaike Harbers, Jeffrey M Bradshaw, Matthew Johnson, Paul Feltovich, Karel van den Bosch, and John-Jules Meyer. Explanation and coordination in human-agent teams: A study in the bw4t testbed. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2011 IEEE/WIC/ACM International Conference on*, volume 3, pages 17–20. IEEE, 2011.

[2] Maaike Harbers, Catholijn Jonker, and Birna Van Riemsdijk. Enhancing team performance through effective communication. In *Proceedings of the 4th Annual Human-Agent-Robot Teamwork Workshop*, pages 1–2, 2012.

[3] Cisco Systems Inc. 20 Myths of Wi-Fi Interference: Dispel Myths to Gain High-Performing and Reliable Wireless , 2007. http://www.cisco.com/c/en/us/products/collateral/wireless/spectrum-expert-wi-fi/prod_white_paper0900aecd807395a9.pdf [Online; accessed 10-Februari-2016].

[4] Matthew Johnson, Jeffrey M Bradshaw, Paul Feltovich, Catholijn Jonker, Birna van Riemsdijk, and Maarten Sierhuis. Autonomy and interdependence in human-agent-robot teams. *IEEE Intelligent Systems*, 27(2):43–51, 2012.

[5] Matthew Johnson, Catholijn Jonker, Birna Van Riemsdijk, Paul J Feltovich, and Jeffrey M Bradshaw. Joint activity testbed: Blocks world for

teams (bw4t). In *International Workshop on Engineering Societies in the Agents World*, pages 254–256. Springer, 2009.

[6] Ivana Kruijff-Korbayová, Francis Colas, Mario Gianni, Fiora Pirri, Joachim de Greeff, Koen Hindriks, Mark Neerincx, Petter Ögren, Tomáš Svoboda, and Rainer Worst. Tradr project: Long-term human-robot teaming for robot assisted disaster response. *KI-Künstliche Intelligenz*, 29(2):193–201, 2015.

[7] C Ronald Kube and Hong Zhang. Collective robotics: From social insects to robots. *Adaptive behavior*, 2(2):189–218, 1993.

[8] D Micheli, A Delfini, F Santoni, F Volpini, and M Marchetti. Measurement of electromagnetic field attenuation by building walls in the mobile phone and satellite navigation frequency bands. *Antennas and Wireless Propagation Letters, IEEE*, 14:698–702, 2015.

[9] Lynne E Parker. Alliance: An architecture for fault tolerant multirobot cooperation. *IEEE transactions on robotics and automation*, 14(2):220–240, 1998.

[10] Paul E Rybski, Sascha A Stoeter, Maria Gini, Dean F Hougen, and Nikolaos P Papanikolopoulos. Performance of a distributed robotic system using shared communications channels. *IEEE transactions on Robotics and Automation*, 18(5):713–727, 2002.

[11] Changyun Wei, Koen V Hindriks, and Catholijn M Jonker. Multi-robot cooperative pathfinding: A decentralized approach. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 21–31. Springer, 2014.

[12] Changyun Wei, Koen V Hindriks, and Catholijn M Jonker. The role of communication in coordination protocols for cooperative robot teams. In *ICAART (2)*, pages 28–39, 2014.

[13] Alan FT Winfield. Distributed sensing and data collection via broken ad hoc wireless connected networks of mobile robots. In *Distributed Autonomous Robotic Systems 4*, pages 273–282. Springer, 2000.

[14] Tiger TG Zhou, Dylan TX Zhou, and Andrew HB Zhou. Unmanned drone, robot system for delivering mail, goods, humanoid security, crisis negotiation, mobile payments, smart humanoid mailbox and wearable personal exoskeleton heavy load flying machine, May 23 2014. US Patent App. 14/285,659.