Thesis

Joris Z. van den Oever

July 2, 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 comes 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[3]. 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[3]. 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[4, 1] 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 undersired 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 occured, 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.[2] 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 because of 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 relevent 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 scenearios. These agents have been 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. This will all be simulated as it allows for stricter control of the experimental dimensions in addition to making it easier to perform a large amount 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 reference data. This will give us two analyses using both the one agent team baseline and the no communication team. This comparison we will only look at the rate of task completion within the timeout period.

If this shows there are indeed 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.

2 Related Work

Talk about BW4T, robots and comunication failure research.

3 Experiment Design

The experiment will consist of multiple simulation runs in the BW4T environment. Each simulation run 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 have to be found in other rooms in the 2D environment.

In this section we will discuss our choices for the environment variables such as topology, block sequences, communication failure, and team composition. The design of the agents themselves is discussed in section 4. First the variables will be classified in the different experiment dimensions and then we will discuss how the variables will be reduced and why.

3.1 Experiment Dimensions

We have categorised the variables, involved in the above mentioned experimental setup into three dimensions, environment, agents, and communication failure. There are several variables in each category as show in Table 1.

Environment is simulation topology, the sequence used, and optional rules the agents function with. Currently the only optional rule available there are agent collisions. By default agents do not collide with each other and with this turned on they would. However it would only result in additional complexity to the agents while it could be solved without communication. As such it is not revelent to our research questions and do not have to consider it in our experiment.

Second are the agents themselves. Aside from the agent programs the team size can be adjusted, as well as the team composition. The former is the amount of agents present in the simulation, limited by the amount of computing power available and map capacity. Though 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. The latter is if the teams are homogenous or not. As more than one agent program can be running in a team.

Finally there is the communication and how it can fail. Does the sending or receiving fail. Because if sending fails nobody gets the message and if receiving fails a subset of agents does not get the message. If it does fail is it a fixed percentage of messages that fails? Or does the failure rate increase over time? is the message more likely to fail if the receipient is further away? However because the intent is to find the effects of the failure and not the communication systems themselves the simplest model shall be used, a fixed percentage communication failure.

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

Table 1: Experiment dimensions before reduction

Aside from the model of communication failure the communication with the BW4T environment should be considered too. As it provides updates on the task progress when a block is delivered correctly. This could effectively be a side-channel for task updates that always succeeeds. As such agents should ignore those task updates and only the normal communcation channels.

3.2 Experiment reduction

The above mentioned experiment dimensions have too many options to realistically test everything. However not everything needs 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, and corridor placement.

The size of the map is how big the map is overal. Bigger maps can have more rooms, larger spaces, and can accomodate 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 for blocks and more places that agents need to look to 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 it 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 as such only one appropriate number needs to be chosen. Corridors just need to connect the rooms that exist with the dropzone. As such a single configuration is needed. This means that all maps can be the same aside from the block placement. We can already exclude all block placements that make the sequence impossible. (Chris identified 6 block placement scenarios and made maps for them. How to correctly refer to that?) These 6 maps are all the same aside from block placement and lets us reduce the topology to 6 options.

For the sequence options there are two big distinctions for our purposes. All blocks are the same or they can differ. When all blocks are the same it does not matter what order they are delivered in as all blocks of the same colour are equal. This 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 does require communicating intermediate task completion. As such sequences where the blocks can differ will be used. However there are enough colours that the sequence can does not have to contain repeating colours. However not all combinations here have to be tried as we just need a sequence where all colours are different and a sequence where there are duplicate blocks. Giving us 2 options for the sequence.

Agents

Outside the agent programming there are two variables affecting agents, 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 failures might have.

The case of having only 1 agent, would not need a lot of experimentation as there is nothing to communicate with, however it will still be useful to do once to get a effectiveness baseline for the agents.

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. Which is needed to see the effects of what happens with the cooperative agents when only some of the agents' responses for task distribution arrives. Ensuring this effect exists for all team sizes.

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. As well as increasing the amount of agents involved in communicating.

The final team size should not go too far, as experience with the environment has shown that 8 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. As such going one agent fewer than the amount of rooms in the chosen maps, 8, has been chosen as the last group size.

All these agent teams still need a team composition. With this we mean the programs the agents follow. While they could have differences, 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 homogenous agent teams will be used.

Communication

There are two variables to 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 gets it but receiving failure means that some part of 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. As such using just the sending failure provides more consistent data with fewer runs.

The failure will be constant within each simulation run and could be any amount between 0 and a 100% of the messages failing. However just like with team sizes a sampling of failure percentages will suffice to identify trends. To limit the amount of runs needed 25% increments will be used. 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 as well. 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 correction agent teams can break down fast. If agents can not determine that an intermediate task has already been completed but 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 might be difficult to identify other effects that have less impact. As such agents should be written in a way that allows them to update their state when in the dropzone. In this manner agents still have a cost to missing the message without a single mistakenly delivered block potentially making the simulation run impossible to complete. (Discuss if it needs to be done with both options. As it would just be a matter of turning this correction on or off.)

3.3 Experiment Setup

So to summarize, the agents will do each situation 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. Each of these maps will be done multiple times, these repeats are grouped by the communication failure. One set of runs for 0, 5, 25, 50, 75, 95, 100.

Team size\failure chance	0%	5%	25%	50%	75%	95%	100%
3	12 maps	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	12 maps
8	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps
3	12 maps	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	12 maps
8	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps
3	12 maps	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	12 maps
8	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps
3	12 maps	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	12 maps
8	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps	12 maps

Table 2: Overview of experiment setup.

These set of runs have repeats because of the random elements of the experiment. There is the agent that, when provided multiple options, takes a random decision and the random chance of messages not being sent. As there is no prior knowledge on the effects of the random components of the simulations, and the agents itself, 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 succesful run will generally not take more than a minute. Using this one minute gets us 252 minutes to do everything once. If each agent program gets a week of runtime we can repeat the simulations 40 times over the course of a week. (Correct this based on the decision made regarding the intermediate task completion correction.)

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

- All the simulations will be run on identical machines.
- 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 are to prevent the available processing power on the system from influencing simulations between runs. And if there is only the one simulation running on the system other processes can not starve it either.

These simulations will be run through a python script to automate the entire process. It will gather and group the logs of each set of simulations with the same values for the experiment dimensions. Every run is a new instance of the BW4T server to which a new instance of the runtime connects. This runtime will be run with only one mas file provided and a timeout. This is repeated for the repeat amount. The timeout and repeat will be the same between sets. The resulting data will be analysed after being converted into a csv file as described

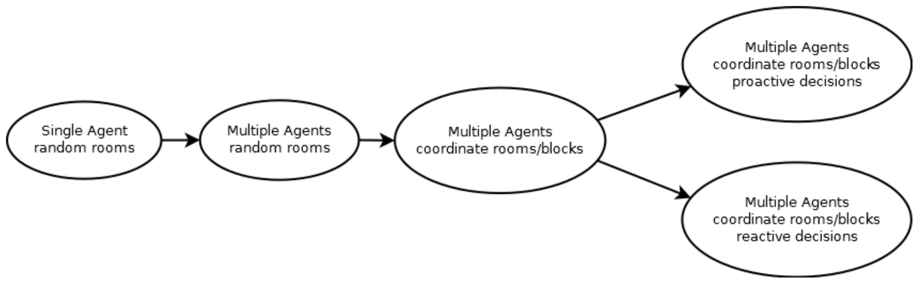

Figure 1: Agent types, each agent team has the capabilities of the previous one in the graph

in section 1.3. To reduce the amount of time needed each agent program will use a duplicated instance on Amazon EC2. 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.

4 Agent Design

The agent programs are written in an interative manner as shown in figure 1. There is one base program and the rest builds upon that with increasing communication. This results in 5 agents, the first of which acts as the baseline agent. This will act as the core of all programs where communication is not used. Each of the agent programs will be discussed below in how they differ from the previous step. Any aspects of the agent program not discussed will be the same as for the previous agent. The full code for the agents can be found in appendix A.

4.1 Baseline Agent

The baseline is a single agent performing a random exploration algorithm. It remembers which rooms are explored and the blocks that are in there. The exploration is interrupted by a greedy delivery algortihm when it finds 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 agents has not found blocks of the right colour yet 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 subsection 3.2.

The random exploration and the delivery algorithms are performed by the same main module for executing the goals. Starting with the random exploration we only have to look at the *in*(*Place*) goal:

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

The main module performs the actions needed to go to the room with the name *Place* and once in there the environment provides a percept that fulfills the goal. The code snippet above looks at all rooms that are not the dropzone where the blocks will be delivered. If those rooms are entered for the first time adds a *visited*(*Place*) belief. So these are excluded for future exploration goals. The adoptGoals module uses linear random order to evaluate its goals. Meaning that a random valid evaluation for *Place* will be chosen from all possible evaluations. This is how the random aspect of the exploration is achieved.

The delivery algorithm works in a similar manner. Because of the linear random order lines earlier in the module get evaluated first. So the following is a simplified version of what is placed above the random exploration code:

```
if a-goal(sequence(Seq)), bel( nextXColoursInSeq([Color],
      1, Seq),
                    block(BlockID, Color, Place), not(Place =
                         held) ) then adopt( delivered(BlockID
                        ) ).
```

The actual code is more complex because of the shared codebase between all iterations of agents but this is what it evaluates to for all agents that don't have the action coordination. It gets the first colour of the sequence that hasn't been delivered yet. Then selects a random block it knows of with the same color and adopts the goal to have it *delivered*, This goal does nothing on its own but there is a submodule of the adoptGoals module that handles this. If the agent is not holding the block, adopt *in*(*Place*) where *Place* is the room that the block is in. If the agent is holding the block *Place* should be the dropzone.

Having the above described baseline program intended for running as a single agent team will let us evaluate how teams perform compared to a single agent, even with the communication failure. The program also provides a base for all of the other agents so that the degree of communication is the only thing that changes between the different programs.

4.2 Simple agent team

The smallest step from a single agent would be using the same agent but multiples. However to garuantee that they would always be able to finish the task they need to communicate one thing. When the current block in the sequence has been delivered the agent will notify the others that the block has been delivered. This is to prevent each agent doing the entire sequence which can lead to impossible to finish instances when there are less blocks of a colour than agents and colours need to be delivered more than once. As blocks delivered out of order disappear. If an agent gets the message to move on to the next colour in the sequence there are four things that can happen.

- The agent is not holding anything and trying to pick up a block. When that block is still the first colour in the sequence the agent will continue trying to pick up deliver the block.
- The agent is not holding anything and the conditions above do not hold. In that case the agent follows the baseline agents algorithm as described in section 4.1.
- The agent is holding a block and it is the next colour in the sequence. It will continue trying to deliver the block.
- The agent is holding a block and it is not the next colour in the sequence. In that case the agent follows the baseline agents 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. When a block is dropped outside of a room it will disappear.

The communication is a single message that is sent when a block is dropped in the dropzone. This is done by adding the following to the main module line that drops the block in the dropzone:

```
allother.send( msg(deliveryDone(BlockID)) )
```

When other agents receive this message it triggers the same module as the agent use to update the sequence when they drop the block. The *BlockID* is included in the message so that block that is delivered can be removed from the belief bases. This is to prevent agents attempting to deliver already delivered blocks.

These agents will always be able to complete the scenario there is still room for improvement. Every agent can pick up blocks, and while agents know which blocks have been delivered, they don't get informed when a block ends up in a different room. So it can 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 it 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 it has no way to find where the block ended up. However there is always one agent that knows where the block is.

There is one situation where this can cause a deadlock. When the agent is without options while in the dropzone. The same restrictions apply so this prevents blocks being delivered and getting out of the situation. As such an additional line as been added at the end of the adoptGoals module:

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

With this addition the dropzone is always available for deliveries. There will still be delays because of the above described issube but never a failure in the provided maps.

4.3 Room and block coordination agent team

The delays discussed in the previous section are solved by communicating what the agents know about blocks. This updates the location of blocks that got picked up and dropped in a different room.

```
forall (Agt).sent( msg(block(BlockID, ColorID, Place)) ),
      bel( block(BlockID, ColorID, OtherPlace),
              Place \= OtherPlace ) do delete( block(BlockID
    , ColorID, OtherPlace) ) +
                    insert( block(BlockID, ColorID, Place) ).
forall (Agt).sent( msg(block(BlockID, ColorID, Place)) ),
      bel( not(block(BlockID, ColorID, Place)) )
              do insert( block(BlockID, ColorID, Place) ).
```

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

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

The agents also share if they are holding a block to prevent agents from trying to pick up a block that was already grabbed. By making the location *held* it is not present in any of the rooms. There is no need to worry about conflicting percepts here as block location percepts are only provided in rooms and a room can contain only one agent at a time.

Finally the agents also send a message when they visit a room for the first time. This is to speed up the exploration part of the baseline agent. Since the block info is already passed on the agents don't have to visit it themselves. 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 belief used by the exploration algorithm and directly inserted into the beliefbase when recieved.

The described improvements leave only one situation where an agent could be standing still. 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.

4.4 Proactive/reactive action coordination agent teams

The problem discussed in subsection 4.3 is a symptom of a larger optimisation possiblity. All agents are still trying to deliver the same colour, they do not work ahead. The final two agent programs implement nearly identical planning algorithms. They communicate their intentions to the other agents when they go to deliver blocks. If the current colour is already being delivered the agent will behave as if that colour has already been delivered, and so on for each colour that an agent is delivering. But instead of delivering the block right away it will wait in front of the dropzone until it is time to deliver the block.

The agents communicate which blocks they intend to deliver by sending on the $delivered(BlockID)$ goal as a message to all other agents. Who can then respond to this intention.

The one aspect in which the two agents programs differ is their conflict resolution:

- The reactive team first asks the other agents for confirmation on their action. When all other agents confirm they aren't planning on doing the same the goal gets adopted. If another agent planned the same goal, the agent names are compared through string comparison. If the other agent's name is larger they get to do it. This is achieved by having that agent send a dismissal. The original agent will in turn confirm the plan of the agent with the larger name as its name is smaller.

- The proactive team adopts the goal right away and then verifies that nobody else is doing it. It uses the same check as for a reactive team but it drops the goal once it is informed another agent declined its plan. This needs less information as an agent only needs to be informed when another agent declines the goal.

{{Figured out better way to implement this that is a lot simpler than the manner this was implemented. Improving readability and reliability. Will add codesnippets of the two conflict resultion methods after implementing this the upcoming week.}}

5 Analysis

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.

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 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] 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].

[2] 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.

[3] 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.

[4] 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.

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.

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

        %drop blocks in rooms when they are not needed
            anymore
        if bel(in(Loc),Loc\='DropZone', not(
            holdingNextBlock)) then putDown.

        %go to goal places
        if a-goal(in(Place)) then goTo(Place).

        %go to goal blocks
        if a-goal(holding(BlockID)), bel(in(Place), block
            (BlockID,_,Place), not(atBlock(BlockID))) then
             goToBlock(BlockID).

        %pickup goal blocks
        if a-goal(holding(BlockID)), bel(atBlock(BlockID)
            ) then pickUp.
}
```

robotEvents.mod2g

```
use robot as knowledge.
use robotGoals as module.
use updateSequence as module.

module robotEvents {
```

```
%———communication updates & conclusions about
    other agents———
forall bel( send('allother', Y) ) do allother.
    send(msg(Y)) + delete(send('allother', Y)).

% deduce information based on deliveries of other
    agents:
forall (Agt).sent(msg('deliveryDone')) do
    updateSequence.

% Update the agent's state of movement.
forall bel( state(State)), percept(state(NewState
    )) do delete( state(State) ) + insert( state(
    NewState) ).

% Record when we are entering or leaving a room.
forall percept(in(Place)) do insert( in(Place) ).
forall percept(in(Place)), bel( not(visited(Place
    )), room(Place) ) do insert( visited(Place) ).
forall percept(not(in(Place))) do delete( in(
    Place)).

% Discover new blocks
forall percept(color(BlockID, ColorID)), bel( in(
    Place), not(block(BlockID, ColorID, Place)) )
    do insert( block(BlockID, ColorID, Place) ).

% Record atblock location of agent
forall percept(atBlock(BlockID)) do insert(
    atBlock(BlockID)).
forall percept(not(atBlock(BlockID))) do delete(
    atBlock(BlockID)).

% Record if a block is being held
forall percept(holding(BlockID)) do insert(
    holding(BlockID)).
forall percept(not(holding(BlockID))) do delete(
    holding(BlockID)).

%remove blocks that are not held or in the room
    anymore
forall bel(in(Place), block(BlockID, ColorID,
    Place), not(holding(BlockID))), not(percept(
    color(BlockID, ColorID))) do delete(block(
    BlockID, ColorID, Place)).

```

```
        %remove blocks that are not held or in the room
            anymore
        forall bel(in(Place), block(BlockID, ColorID,
            Place), not(holding(BlockID))),  not(percept(
            color(BlockID, ColorID))) do delete(block(
            BlockID, ColorID, Place)).

        % Update sequence when in dropzone.
        if percept(sequenceIndex(X)), bel(sequenceIndex(
            OldX)) then delete(sequenceIndex(OldX)) +
            insert(sequenceIndex(X)).
        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) ).

        %remove obsolete goals
        if goal(holding(BlockID)), bel(not(block(BlockID,
             ColorID, Place))) then drop(holding(BlockID))
            .

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

A.2 Common Code

robot.act2g:

```
use robot as knowledge.

% The goTo action makes the agent move to a place (
    location) in the BW4T environment.
% As long as the agent has not arrived at the place it is
     going to, it will be in "traveling" mode.
define goTo(Location) with
        pre { not(state(traveling)) }
        post { true }

% goToBlock only when not traveling and in a room
define goToBlock(BlockID) with
    pre {in(_), not(state(traveling)) }
        post { true }

```

```
% pickUp can only be performed when not holding a block
define pickUp with
        pre{not(state(traveling)), not(holding(_))}
        post{ true }

% putDown can only be performed when holding a block
define putDown with
        pre{not(state(traveling)), holding(_)}
        post{ true }
```

UpdateSequence.mod2g

```
use robot as knowledge. order=linearall.

module updateSequence{
        %update sequence
        if bel(seqDone(SDone), nextNeededColor(ColorID),
            append(SDone,[ColorID],NewSDone) ) then delete
            (seqDone(SDone)) + insert(seqDone(NewSDone)).

        %remove beliefs about the delivered block (if
            this agent was delivering it)
        if bel(in('DropZone'), holding(BlockID), block(
            BlockID, ColorID, Place)) then delete(block(
            BlockID, ColorID, Place)).
}
```

robotGoals.mod2g

```
use robot as knowledge. order=linear.

module adoptgoals{
        %If holding the next needed block go to the
            dropzone.
        if bel( holding(BlockID), nextNeededColor(ColorID
            ), block(BlockID, ColorID, _)) then adopt(in('
            DropZone')).

        %Otherswise, If the next needed block is known
            then adopt a goal to go there and hold it.
        if bel( nextNeededColor(ColorID), block(BlockID,
            ColorID, Place)) then adopt(in(Place),holding(
            BlockID)).

        %Otherwise go to a random room we haven't seen
            yet'.
```

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.

```
        if bel( not(finished), bagof(Place, (room(Place), not(dropZone(Place))), Places), random_member(Dest, Places) ) then adopt(in(Dest)).
}
```

robotInit.mod2g

```
use robot as knowledge.

module robotInit {
        % Store map information, i.e., navigation points in the agent's belief base.
        forall percept(zone(ID, Name, X, Y, Neighbours)) do insert( zone(ID, Name, X, Y, Neighbours) ).

        % Record the initial state of movement in belief base.
        if percept(state(State)) then insert( state(State) ).

        % Record goal sequence
        if percept( sequence(Seq) ) then insert(sequence(Seq), seqDone([])) +  adopt(seqDone(Seq)).
        if percept( sequenceIndex(X) ) then insert( sequenceIndex(X) ).

        % Adopt initial goal going to a random place
        if bel(room(PlaceID), PlaceID\='DropZone') then adopt(in(PlaceID)).
}
```

A.3 Exploratory research data