Multi Agent Programming Contest 2019

Lessons Learnt From the Applied Artificial Intelligence Project

Abhiraj Bishnoi
Bharathwaj Krishnaswami Sreedhar
Dhananjay Mukhedkar

$21^{st} July, 2019$

Abstract Multi agent systems and decentralized control have been gaining importance in the recent past and they have many widespread applications in various fields including logistics and warehousing, manufacturing, deep sea exploration and computer networking. The Multi-Agent Programming Contest is an annual international programming competition with the goal of stimulating research in the area of multi-agent system development and programming. This paper discusses the design and implementation of a multi agent system based on the Robot Operating System(ROS) and the ROS Hybrid Behaviour Planner(RHBP) framework, towards the fulfillment of goals introduced as a part of the Multi Agent Programming Contest 2019.

Keywords MAPC 2019 · Multi-agent systems · Robot Operating System · Decentralized control, coordination and planning · ROS Hybrid Behaviour Planner(RHBP) · Guided exploration · Landmark based localization · A* search · Behaviour-based Planning Networks,Hybrid Planning, Decision-Making

1 Introduction

The name of the team is Team Emergence with three members,all are students at the Technische Universität Berlin. All three members have a background in computer and electronics engineering and are currently pursuing their Master's degrees in Autonomous Systems at the EIT Digital Master School. All members have completed 'Robotics' course at the TU Berlin,which has helped to give a background behind the essential concepts around the competition. The main contact person for the team is Bharathwaj Krishnaswami Sreedhar (ksb1712@win.tu-berlin.de).

Abhiraj Bishnoi · Bharathwaj Krishnaswami Sreedhar · Dhananjay Mukhedkar ·
DAI Labor, Technische Universität Berlin
Ernst-Reuter-Platz 7
10587 Berlin / Germany
Tel.: +49 30 314 74000
Fax: +49 30 314 74003

The setting behind the Multi Agent Programming Contest 2019 is a galaxy in outer space close to 100 years into the future. The participants have access to a swarm of simulated robot agents that must be controlled and co-coordinated in an intelligent and decentralized manner to complete tasks before a stipulated deadline. A task consists of collecting a specified construction blocks of a specified type from block dispensers located at initially unknown locations in the environment, assembling these blocks into a complex shape and submitting the combined block to a particular goal location. The rules of the contest dictate the starting positions of the agents on the virtual grid based map. At the beginning of the simulation, agents are spawned at random locations all over the virtual environment. Furthermore, the position of each agent is not known relative to a common global reference frame and each agent defines it's environment relative to it's own origin reference frame. At every step of the simulation, each agent has the ability to sense it's environment and perform actions from a predefined list of actions in response to stimuli. The stimuli that are perceivable by agents include detecting other agents, block dispensers, obstacles and goal cells. The ability of each agent to perceive it's environment is limited by it's 'perception field', a parameter that dictates the extent to which an agent has visibility into it's environment. The actions an agent is allowed to perform at each step include move one step in either direction (North, South, East, West), dispense blocks from a dispenser if it is next to one, rotate in any direction, join with another agent to form a more complex structure and submit blocks in a goal cell. Some challenges that have to be overcome include developing strategies for exploring the unknown environment, picking the right tasks to perform, coordinating agents to work together towards a common goal and overcoming the adversarial nature of the agents belonging to the competing team(s). The general setting of the environment is shown in Fig 1.

2 System Analysis and Design

The implementation for this project is based on the Robot Operating System (ROS) and the ROS Hybrid Behaviour Planner (RHBP) framework developed in house at the Technische Universität Berlin[1]. The design and analysis of the system is split into sections as outlined below.

2.1 Behaviour and Control Strategy

Different agent actions like explore,attach ,submit,etc. are modelled as behaviours as per the RHBP framework. Every behaviour is executed based on the priority of operation , conditions and agent goals. Our intention was to try to complete as many incoming tasks as possible in a decentralized manner. At this stage dynamically modifying game strategies based on those of opponent teams agents activity is not taken into account. Also we have a narrow implementation scope where we submit a task with a requirement of one block only once. Although we have implemented modules for multi-block tasks and reset of sensors for the next task, we do not include them here. In brief, given below are important logical steps defining agents behaviour or actions:

1. Explore local map targeted towards goal cells first.

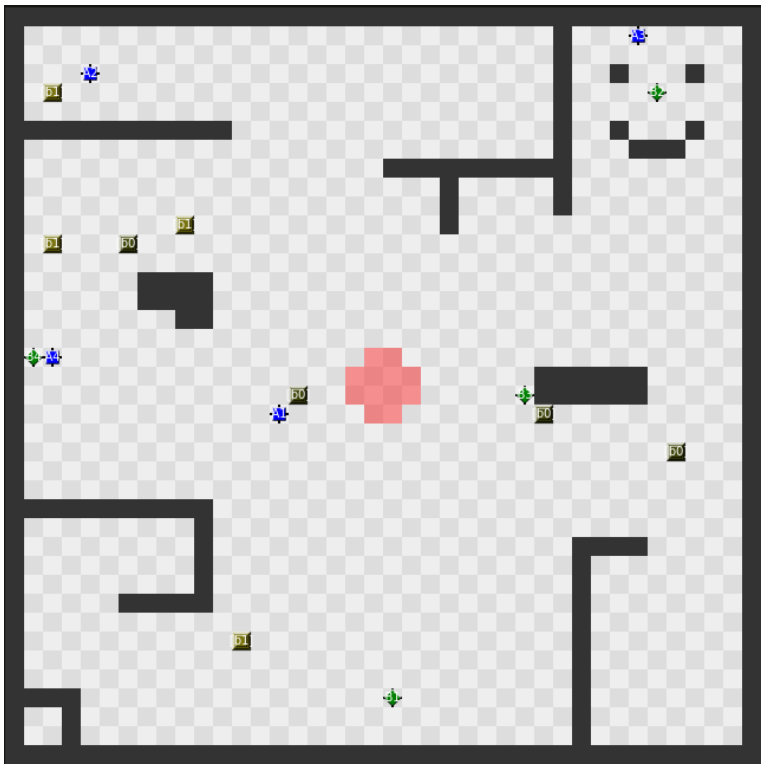
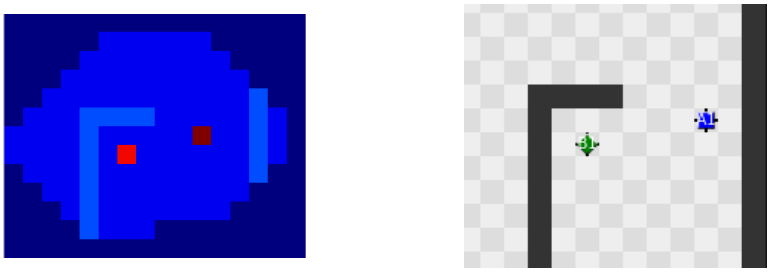


Fig. 1: MAPC 2019 Setting

2. Explore dispensers in local perception
3. Publish local agent and dispenser location after transforming into global locations using goal cell as reference frame
4. Read tasks from perception provider and find required dispenser depending on block type for individual requirements
5. Call path planner to check the optimal path to go dispenser and find the closest agent
6. Move agent to dispenser ,dispense and attach a block
7. Move to pre-calculated goal cell
8. Connect to other agents if applicable
9. If all requirements are fulfilled submit the task

2.2 Local Map Generation

The information about an agent's perception is obtained as a JSON array published on a ROS topic. The ROS topic is configured such that the information is refreshed at every step of the simulation and only the last update is made available to the subscribers of the topic. For the agents to work together and accomplish tasks within the stipulated deadlines, the agents must be able to leverage information discovered by one another and work together towards the fulfillment of



Generated map Actual map

Fig. 3: Comparison between the generated map and actual map

a common goal. In order for the agents to move around the environment more efficiently, it is beneficial to know the location of obstacles, dispensers, goal cells and other agents while planning a path and assigning tasks to each agent. As a preliminary step towards achieving this goal, it is necessary for each agent to record its history of perceptions as it moves about in it's environment. We propose a data structure to keep track of individual perception histories each agents and coin the term 'local map' to refer to an instance of this data structure.

The structure of the local map is a 2D array, where each cell stores a value based on the perception of the agent. We fix certain numbers to represent certain objects of interest (eg. 0 - free cell, 1 - obstacles, ..) and the map is updated in real time. In Fig 3 the obstacles, other entities and free cells are represented by different colours in the local map. As the agent moves, the map gets updated, growing in size and updating new cells. Unlike the actual perception from the server, which assumes the current location of the agent as origin, we set the top left corner of the local map as the origin for the agent. When the map grows, the origin might no longer be the intended cell. In this case, it is required to shift the origin back to the top left position. Every time the agent moves, the area within the perception range is updated while rest of the map remains same. It is important to know that the agent is able to "see" through obstacles, which causes the area beyond the boundaries to be also marked as free.

As each agent is independent and has its own origin and history, it is possible to obtain the maps of multiple agents simultaneously. This helps in defining a decentralized approach to solve the parameters of the challenge. Fig 4 represents the generated local maps with different starting positions. It is taken at the same instance of Fig 1 where the agents have been allowed to explore the environment.

2.3 Global Map

The individual local maps can be combined to form a globally unique map, and an instance of a data structure that holds this map is referred to as the 'global map'. The global map can be updated in real time by each agent and shared between agents to aid in decisions involving exploration, task allocation and path planning. The fact that the environment is only partially visible coupled with the absence of a common global reference frame motivated a search into solutions to the problem of decentralized simultaneous localization and mapping that could be applied to

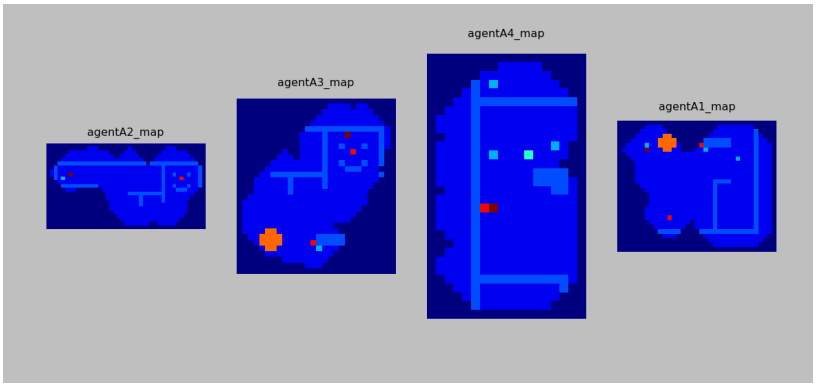


Fig. 4: Local maps of all agents

Fig. 5: Global Origin / Landmark

the MAPC context. We wanted to avoid the traditional "map merge" approach of combining the individual maps to create a central map as it is resource intensive. We wanted to provide a simple approach that allows only meaning full information exchange among the agents in a timely manner. Solutions utilizing map merging based on landmark detection turned out to be promising and cross applicable to the MAPC scenario. The basic tenet of these approaches is to look for a common landmark that is uniquely identifiable and use this information transform the local reference frames of each agent into a global reference frame based on the common landmark.

Initial attempts at using obstacles and dispensers as landmarks turned out to be prone to ambiguity, two agents with different starting points and local maps incorrectly identify two different obstacles / dispensers as being the same globally identifiable landmark. Experience and empirical evidence led to the conclusion that the goal area is unique for a given test run of the simulation and does not change with the passage of time. This feature makes the goal area well suited to be used as a global landmark with respect to which all other points of interest such as dispensers, obstacles and positions of agents can be uniquely defined and identified. Given the plus (+) shaped area defined to be the goal, the cell marked in Fig 5is chosen to be the candidate global reference frame origin and is referred to as "goal cell" or the "global origin", as it is invariant to unique parameters of each agent like starting position, local map and direction of motion.

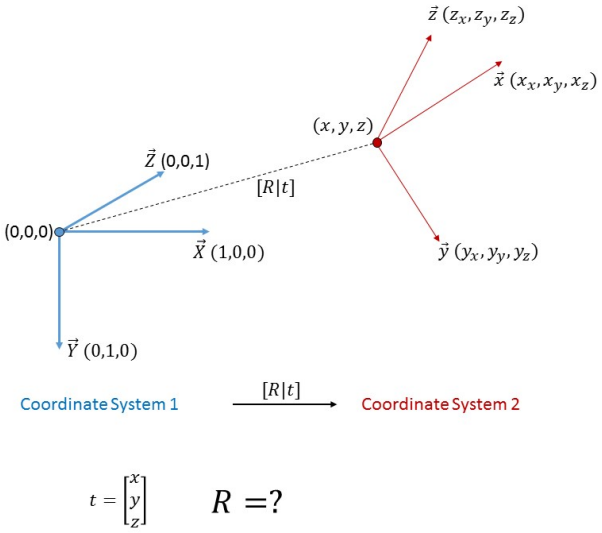


Fig. 6: Transformation Between Coordinate Frames

2.3.1 Avoiding Communication Overheads

Multi-agent systems have a series of advantages over traditional systems but often come with its own set of challenges. One of the major challenges of designing a multi-agent system is designing the control and communication mechanisms. The end goal is almost always to make the system as decentralized as possible, and agent is programmed to individually make its own decisions to mirror this philosophy. However, to reap the benefits of multi-agent systems in terms of computational speed-up and cost, some form of communication between the agents is necessary. In the most simplest form of decentralized communication, a system of N agents requires (N*N-1) update messages during each time step of the simulation. As each agent explores more uncharted territory, the idea of sharing a large matrix representing the state of the global map quickly becomes inefficient and redundant. A more elegant solution is identified and the system minimizes on the communication and processing overheads by aggregating common data and publishing only incremental updates containing changes between states as opposed to the entire state for updating ROS topics involving communication about the state of the environment. To elaborate; as soon as an agent identifies the goal cell chosen to be the origin of the global reference frame, it transforms the coordinates of dispensers and obstacles in its local history of perception fields (local map) and publishes these transformed coordinates (in terms of the global reference frame) to their respective ROS topics if they haven't already been published by another agent. The transformation of coordinates is done using simple coordinate transforms and is illustrated in Fig 3. Once an agent receives these transformed coordinates, it can transform it to its frame of reference with the help of the location of the global origin in its local map.

It is vital that each agent prioritizes in identifying the global origin as soon as possible as the communication framework is based on the assumption that the global origin has been found and all coordinate information is passed as relative transformations of the global origin to provide a common frame of reference for all agents. This calls for a targeted exploration strategy of the unknown environment as explained in Section 2.4

2.4 Exploration

2.4.1 Overview

At the very beginning, all agents start with no knowledge of the environment outside of their own immediate perception fields.Hence it is essential to explore the whole environment and discover all entities within. Ideally, an agent should be able to share the information it obtains with other agents in the system. The agents are programmed to explore the most unexplored regions of their individual local maps until they find the global reference frame origin (goal cell). Once they find the common reference, they use the coordinate transform described in Section 2.3.1 to transform the coordinates of dispensers they have previously encountered in terms of the global origin. These transformed coordinates are published on public topics that can be accessed by other agents which have discovered the global origin. By using this approach the need for map merging from every agent is avoided as agents have access to dispenser locations that they might have not even discovered in their map yet.

The objective of exploration is to find the goal cells and reduce the number of unknown cells in the local map. It is also used to find dispensers that are not yet detected and are required by a task. As exploration involves in moving to a new cell in the map to increase the known environment, there comes a need to select the appropriate cell that increases perception without taking too much time. Also once such a cell is identified, it is advantageous to move to that cell in the shortest time possible. Hence the task of exploration can be broken into two main components; cell identification and path planning.

2.4.2 Exploration Cell Identification

To identify the "target cell" for exploration, we look for cells that are near unexplored regions (-1 in local map). Specifically we count the number of unexplored cells in the neighbourhood of each free cell and choose the cell that has the maximum value. But such a method might cause the agent to travel back and forth as it would try to uniformly explore in all areas. To avoid this problem, we divide number of unexplored cells by the Manhattan distance between the agent location and the respective cell.

$$D_{A,B} = |A.x - B.x| + |A.y - B.y| \tag{1}$$

$$Cell_score_{cell} = \frac{count(neighbourhood(cell))}{D_{A,cell}} \tag{2}$$

2.4.3 Path Planning

The path planning algorithm is a derivative of the A^* algorithm [2] which tries to identify the path with the lowest cost. Here cost is based on heuristic measures between the target cell and the current cell. We use the Manhattan distance as the heuristic. We also extended the planner for a system of an agent and block together. Although this is not necessary for exploration, it is needed for submitting a block. Since the algorithm is computationally expensive (limited server time), we avoid finding all the intermediate cells and stop the planner after a certain count. Occasionally this causes the agent to be stuck in a certain area over a long period of time. To reduce this, we added a history of all cells the agent has chosen and avoid repeating cells. We consider locations of unexplored cells, obstacles, blocks and other agents (team and opponent) as unreachable cells to avoid collisions.

2.5 Tasks and Agent Selection

Agent selection about which tasks and requirements to work on are decentralized, meaning every agent reads incoming tasks from perception provider and loops through into its requirements. Based on the block type of each requirement of a task, an agent closest to a dispenser of that particular required block type is selected using path planning.

The functions of task allocation module is to find a dispenser that an agent must go to based on task. Find a "submit origin" for the agent to submit the task, where the block locations are relative to this cell. It must also decide on possible locations for the agent to connect its block with blocks of other agent without colliding. The "submit origin" varies with task and is selected as the cell in the goal area with a free neighbourhood.

A submit cell(a cell inside the goal area) is also pre-calculated and set as agent destination for agent to connect or submit the task. Thus,the need for dynamic shape formation depending on which agent submits is avoided and every agent works only on one unique requirement of a task avoiding two agents working on same requirement without a centralized node controlling task allocation among the agents. When a appropriate dispenser is selected, the agent travels to it via A^* algorithm and dispenses a block. Then it attaches the block to itself and navigates to the required submit location and orientation of block. Once it reaches the location, it tries to submit the task. In the current state of implementation,tasks with only one requirements are supported and selected for processing.

An alternate approach for task allocation was also developed under the class *TaskBreakdown* .When a task is read from the perception provider,first the closest cell position which qualifies as a submitter cell is found using the Euclidean distance between a zero position and the individual requirement position.In short,first the requirement closest from a predefined goal cell is found and agent which qualifies to submit this requirement is calculated.Once an agent picks up a requirement to work on ,it publishes the other pending requirements of a task on a ROS topic *activeTask* ,which are picked up by other agents as a priority over task from perception.This approach can support any number of tasks requirements but is limited by the fact that a predefined goal cell to submit has to be identified and has to be

free and an agent should already be attached to a block,hence currently not being used.

2.6 Communications and Coordination Strategy

All communications and agent coordination is carried out as a decentralized solution. Communication between agents primarily happens through different ROS Topics. For communication and coordination between agents the prerequisite for any agent publishing to a topic is that the agent has already found a dispenser and a goal cell. Different ROS Topics that used for different roles are :

Topic Name	**Message Type**	**Description**
dispenser_loc	DLoc	Tranformed dispenser location
target_dispenser	DLoc	Selected dispenser for agent
target_submit	DLoc	Selected dispenser for agent
target_dispenser	DLoc	Location for connect/submit
submit_origin	Position	Identify cell for task submit
task_selected	Task	Selected task
reset_val	int	Reset all sensors
activeTask	Task	pending requirements of task already picked

Table 1: ROS Topics

2.7 Manual Player for Testing

For testing and exploring the rules of the competition environment, a manual player interface was developed to enable independent control of agents through user inputs . Using python libraries user input is taken developed to be commands and corresponding actions in the game are invoked.This provides as an additional layer for quick testing and for proof of concepts.

2.8 RHBP Extensions Experimentation

As a parallel exercise, simple tests were carried out with the RHBP Self-Organization packages, in particular with repulsion and flocking behaviours to work on given 2-D map space .
The explanatory examples mentioned in [1] using the SO packages were modified to utilize transformed coordinates between the two environments. The examples were built using the ROS turtle simulator and programmed to perform behaviours in terms of the coordinates of the turtle simulator and differential drive the uses differential drive frames which had to be transformed into 2-D frame.For this a simple polar transform was used on the output of base SO behaviours,which is available as a utility function in code-base .Based on the transformed output an agent was successfully moved to a desired cell.
This study showed that integration of based SO packages running on differential

drive frames and using it to execute different behaviours in our two dimensional grid map is definitely possible ,which could be potentially used for up-scaling the application.Hence future development using SO packages is possible.Further investigation and tests in this direction were aborted due to time constraints and the code for this exploration is not included in the code-base of the current submission. However, the results were promising and may be useful in future developments.

3 Software Architecture

We used the ROS Hybrid Behaviour Planner framework developed in house at the Distributed Artifcial Intelligence (DAI) Laboratory to model the behaviour of the system in response to its environment. The RHBP framework provides the ability to the system as a network of behaviours that are able to influence one another response to changing stimuli, in combination with a goal directed symbolic planner. This approach simultaneously combines the best of both approaches, having a static goal oriented planner capable of intelligently reacting to dynamic changes in the environment. The Robot Operating System is used as a platform for distributed multi agent system development due it's inherently distributed design and support for distributed programming paradigms. Python is the programming language of choice due to it's simplicity, versatility and compatibility with ROS. In particular ROS Kinetic and Python 3 was used for development and Pycharm was used as IDE for development and debugging.Using debugging for Pycharm helped in quick and efficient debugging.

4 Evaluation and Conclusion

The scenario for this year's Multi Agent Programming Contest was different from the previous few years and this meant that our solution had to be designed and implemented from the ground up.The only component that could be easily adapted to the new scenario was the interface between the simulation server and ROS.

Nevertheless, the contest was helpful for the team in getting introduced to multi-agent and decentralized systems paradigm .It helped to gain hands on experience on fundamental challenges of any multi-agent systems like exploration,localization ,co-ordination and decision making. RHBP framework was very helpful in understanding how different coordinated actions and decisions can be modelled as behaviour mechanism and executed as decentralized.
Some of the strengths of approach implemented is that there is no explicit map merging involved ,hence it reduces computational complexity with almost no lag between time steps .Also frequent calls to path planner are reduced which can be computationally heavy otherwise.
During exploration phase within a local map,goal cells where explored in 98 time steps on a average,where 300 time steps as the longest ,for trial of 10 consecutive runs.

On the drawbacks however, a single global map is not available which can lead to sub-optimal global path planning and agent co ordination as obstacle information is not shared between agents.
Opponent teams agents are not taken into account so aggressive behaviours on other team agents is not possible in current state.
The exploration strategy is not optimal and tends to stay in certain locations for a long period of time.

RHBP framework supports dynamic behaviour modelling and decentralised decision making and executions.This is very well suited for the challenge at hand.With modelling the correct behaviour and conditions, potentially any desired operation or behaviour can be implemented into the system which all can run in a decentralised manner. Using ROS helped in agent communications and coordination which can all run without the need for a single centralised server. Python made the programming easy and scalable, however object referencing can be a problem sometimes.

For further development , the task handling between multiple agents and coordination steps for connecting complex requirements can be improved,which in current state is a limited.Also ,with the experiments carried with SO packages,integrating suitable SO mechanisms would be a next step towards incorporating additional layer of intelligence.

References

1. Christopher-Eyk Hrabia and Tanja Katharina Kaiser and Sahin Albayrak *Combining Self-Organisation with Decision-Making and Planning.* EUMAS/AT,2017
2. P.EHart, N.J.Nilsson and B.Raphael journal=IEEE Transactions on Systems Science and Cybernetics, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* IEEE Transactions on System Science and Cybernetics, 1968