Emulating Centralized Control in Multi-Agent Pathfinding Using Decentralized Swarm of Reflex-Based Robots

Ján Chudý*, Nestor Popov† and Pavel Surynek‡
Faculty of Information Technology
Czech Technical University in Prague
Prague, Czechia
Email: *chudyja1@fit.cvut.cz, †popovnes@fit.cvut.cz, ‡pavel.surynek@fit.cvut.cz

***Abstract*—Multi-agent pathfinding (MAPF) represents a core problem in robotics. In its abstract form, the task is to navigate agents in an undirected graph to individual goal vertices so that conflicts between agents do not occur. Many algorithms for finding feasible or optimal solutions have been devised. We focus on the execution of MAPF solutions with a swarm of simple physical robots. Such execution is important for understanding how abstract plans can be transferred into reality and vital for educational demonstrations. We show how to use a swarm of reflex-based Ozobot Evo robots for MAPF execution. We emulate centralized control of the robots using their reflex-based behavior by putting them on a screen's surface, where control curves are drawn in real-time during the execution. We identify critical challenges and ways to address them to execute plans successfully with the swarm. The MAPF execution was evaluated experimentally on various benchmarks.**

***Index Terms*—multi-agent pathfinding (MAPF), centralized control, swarm of robots, reflex-based control**

I. Introduction

Multi-agent pathfinding problem (MAPF) is a task of finding paths for multiple agents from their initial positions to their goal positions while ensuring that they do not collide at any point in their path execution [1]–[7]. The abstract form of the MAPF problem assumes an undirected graph $G = (V, E)$ that models the environment. We assume multiple distinguishable agents placed in vertices of G such that at most one agent is placed in each vertex. Agents can be moved between vertices across edges, while problem-specific rules must not be violated. MAPF usually assumes that agents are moved to unoccupied neighbors only. The task of MAPF is to reach a given goal configuration of agents from a given starting configuration using valid movements.

Due to its real-world applications, MAPF has been a deeply researched topic over recent years. New variations of this problem and approaches to solving them, emerged, and new solving algorithms have been designed. Many practical problems from robotics can be interpreted as MAPF. Applications can be found in discrete multi-robot navigation and coordination, automated warehousing [8], transport [9], or coordination and maneuvering of aerial vehicles [10].

However, most of the solutions are tested in the virtual environment. Although this type of simulation is suitable for theoretical research and agile solver benchmarking, it is not sufficient to understand how abstract plans can be transferred into reality. It might also not be the most exciting for students or someone interested in understanding this topic. Moreover, using physical robots usually requires a fundamentally different approach to solving MAPF problems.

The major challenge faced during the execution of abstract plans on real robotic hardware is that most of the contemporary MAPF solvers produce discrete plans, while physical robots act continuously in a continuous environment. The viability of discrete MAPF solvers is that existing solvers for the continuous multi-robot motion planning do not scale for a large number of robots [11], and finding the optimal robot trajectories is often unattainable.

This paper describes a novel approach to simulating the existing MAPF algorithms using a swarm of small mobile robots–Ozobot Evo. This robot is equipped with simple reflex-based control and relatively limited programming capabilities. Precisely, Ozobot can follow a curve drawn on a surface and has its reflex-based behavior modified using a small set of instructions. We used these capabilities to emulate centralized control by placing the swarm on a screen's surface, where control curves are displayed to navigate the robots in real-time. The novel approach aims to:

- Verify if discrete plans can be successfully executed on a swarm of physical robots that move continuously.
- Create a scalable solution for a large number of robotic agents that can utilize contemporary MAPF solvers.
- Provide researchers and educators with an affordable solution for testing and demonstrating their findings in the physical world.

The paper is organized as follows. In Sections II and III, we introduce MAPF formally and recall major solving algorithms. Then Ozobot Evo, a robot we use for execution, is described in Section IV. In Sections V and VI, we introduce the execution of MAPF solution on a swarm of Ozobot Evo robots through reflex-based control. Experimental evaluation of swarm executions on various MAPF benchmarks is presented in Section VII. We conclude with Section VIII.

II. Overview of MAPF

Almost all of the previous MAPF research and proposed solvers were built on top of several assumptions about time and agents:

- Time is not continuous, but rather *discretized* into time steps.
- Agents can move to any direction and their actions take the same amount of time to execute, precisely one time step.
- Agents are of the same size and shape and occupy a single point in the environment representation.

A. Classical Discrete MAPF

The classical MAPF problem [2], [5] is modeled on an undirected graph $G = (V, E)$ with a set of agents $A = \{a_1, a_2, ..., a_k\}$ such that $|A| < |V|$. Each agent is placed in a vertex so that at most one agent occupies each vertex. The configuration of agents is denoted $\alpha : A \to V$. Next, we are given a starting configuration of agents α_0 and a goal configuration α_+.

An agent can either *move* to an adjacent vertex or *wait* in its current vertex at each time step. The task is to find a sequence of move/wait actions for each agent a_i that moves the agent from $\alpha_0(a_i)$ to $\alpha_+(a_i)$ such that agents do not *collide*[1]. So an agent can move into a vertex only if it is unoccupied, and no other agent enters it at the same time. Different constraints might be applied in various MAPF problems as well.

Below, we define a classical *move-to-unoccupied* variant of MAPF.

Definition 1: **(move-to-unoccupied MAPF).** Configuration α' results from α if and only if the following conditions hold:

(i) $\alpha(a) = \alpha'(a)$ or $\{\alpha(a), \alpha'(a)\} \in E$ for all $a \in A$ (agents move along the edges or wait in the current vertex);

(ii) for all $a \in A$ it holds that if $\alpha(a) \neq \alpha'(a) \Rightarrow \alpha'(a) \neq \alpha(a')$ for all $a' \in A$ (target vertex must be unoccupied);

(iii) and for all $a, a' \in A$ it holds that if $a \neq a' \Rightarrow \alpha'(a) \neq \alpha'(a')$ (no two agents enter the same vertex).

Solving the MAPF instance is to find a sequence of configurations $[\alpha_0, \alpha_1, ..., \alpha_\mu]$ such that α_{i+1} results from α_i for $i = 1, 2, ..., \mu - 1$ using only valid actions, and $\alpha_\mu = \alpha_+$. A *feasible solution* of a solvable MAPF instance can be found in polynomial time [1], [12]. Precisely the worst-case time complexity of most practical algorithms for finding feasible solutions is $\mathcal{O}(|V|^3)$ (asymptotic size of the solution is also $\mathcal{O}(|V|^3)$) [13].

B. Cumulative Objectives in MAPF

Various objectives are used to evaluate the quality of a solution. The most common objectives are *makespan* and *sum-of-costs*. With the *makespan* [14], we need to minimize μ in the solution sequence mentioned after Definition 1. Below, we define the *sum-of-costs* objective [3], [4], [15]:

[1]We understand a *collision* when at least two agents occupy the same vertex simultaneously.

Definition 2: **(sum-of-costs objective).** The sum-of-costs objective is the summation, over all agents, of the number of time steps required to reach the goal vertex. Denoted ξ, where $\xi = \sum_{i=1}^{k} \xi(path(a_i))$, where $\xi(path(a_i))$ is an *individual path cost* of agent a_i connecting $\alpha_0(a_i)$ calculated as the number of edge traversals and wait actions. [2]

Finding an optimal solution with respect to the sum-of-costs objective is NP-hard [16].

C. Real-world Complications

Besides the already mentioned assumptions, the environment is usually modeled as an undirected graph or a tiled grid. Even though this can often be sufficient even for a real-world scenario, the reality is not generally that easy to model. Also, physical agents have their specific form, and they can collide in many different ways–they are geometrical. Different agent representation and collision detection are therefore needed if pursuing a more realistic model. Geometry-aware collision detection for agents of different shapes, speeds, and continuous movements is studied in [17]. Also, a study [18] solely around this concept of geometrical agents was done. In that work, authors refer to such agents as *large agents*, and they formalize and study a *MAPF for large agents*, then propose a new algorithm for this problem.

Actions of physical agents do not usually take the same time, and they do not merely snap between positions in the environment instantaneously, as in the discrete approach. The agents require continuous movement to reposition. Continuous MAPF can adequately accommodate this problem, but also other approaches might sufficiently simulate the fluid transfer of agents.

In the classical MAPF, the agents can move in any direction for every time step of the plan. Even though this might be true for some agents like drones, this is not true for most mobile agents moving on the ground. When an agent wants to change the direction of movement, it must rotate what takes some time and adds to the plan desynchronization. These rotation movements can also be incorporated in the MAPF abstraction, as proposed in [19]. The authors suggest splitting *position vertices* into *directional vertices*, which represent the direction the agent is facing. Edges between these new vertices represent rotation actions and original edges movements between the original vertices. This change also requires a modification in the solver, namely in conflict detection. The study's primary purpose was to test the behavior of several defined MAPF models when executed on physical robots. The experiments concluded that classical MAPF plans are not suitable for such use, and some of the other proposed models yielded better results than the classical one.

III. Optimal MAPF Solvers

We use optimal solutions with respect to the sum-of-costs objective in our prototype of the novel approach. There are

[2]The notation $path(a_i)$ refers to a path in the form of a sequence of vertices and edges connecting $\alpha_0(a_i)$ and $\alpha_+(a_i)$, while ξ assigns the cost to a given path.

several solvers used for fining od optimal MAPF solutions based on different paradigms. There are currently two major streams of optimal solvers for MAPF: **(i) search-based** solvers and **(ii) compilation-based** solvers.

Search-based solvers are often variants of A* search [20], but also other than common search spaces derived from actions are used. The notable examples are ICTS [4] or textsc [3]. These algorithms implement a two-level search. In the ICTS, distributions of the cost to individual agents are searched at the high-level search. The low-level search of the algorithm tries to find non-conflicting paths for individual agents that follow the cost distributions. On the other hand, the ICTS algorithm performs a search across all possible conflicts between agents at the high-level, and at the low-level paths that avoid the conflicts are assigned to individual agents.

Compilation-based solvers reduce the MAPF problem instance to an instance in a different paradigm for which an efficient solver exists. Such target paradigms include Boolean satisfiability (SAT) [21], constraint satisfaction problem (CSP) [22], [23], or answer set programming (ASP) [24]. The major challenge in using compilation-based methods is designing the encoding of MAPF in the target formalism.

A. SAT Based Solver

The *SAT-based solver* [25] transforms an instance of a MAPF problem into a *propositional formula* that is satisfiable only if the MAPF problem is solvable. This formula can be consulted with an already existing state-of-the-art SAT solver. When the *satisfying assignment* is found, the solution of the MAPF problem can be reconstructed from this assignment. Therefore, the main challenge is the encoding of the MAPF problem into a propositional formula.

The primary concept, allowing the encoding of a MAPF problem into a propositional formula, is a *time expansion graph* (TEG) of the original graph from the problem instance. The TEG is created by duplicating all vertices from the original graph for all time steps from 0 to a given bound μ. This can be imagined as a layered graph where each layer of vertices represents an individual time step. Then all possible actions are represented by directed edges between consecutive layers. An edge between corresponding vertices in two layers represent a *wait action*, while an edge between neighboring vertices in the layers represent a *move action*. This TEG is created for each agent. In the encoding, a *propositional variable* is introduced for each of the vertices of these new graphs. The variable is *true* if the agent occupies the vertex at the time step that the variable represents. Similarly, each directed edge is encoded, and other constraints are added so that the found satisfying assignments correspond to a valid MAPF solution. However, because of the bound μ of the TEG, the encoded formula can only be satisfied if a solution that takes up to μ time steps exists. Note that this corresponds to the makespan of the solution.

The optimal solver repeatedly asks the SAT solver, if a solution of a certain makespan, which is incremented in a loop, exists. When it finally finds a solution, this solution has the lowest possible makespan and therefore is optimal. First, a lower bound makespan of the MAPF solution is determined by finding the optimal single-agent pathfinding solutions for each agent, which can be done very efficiently. The makespan of the MAPF solution cannot be lower than the longest path of these agents. For each increment of μ, the problem is encoded into a propositional formula $\mathcal{F}(\mu)$, and consulted with the SAT solver. If the formula could not be satisfied, μ incremented, and a new formula is created. If a satisfying assignment has been found, the MAPF solution is extracted from this assignment and returned.

The framework, how it is described, is not complete because if a solution to a given MAPF instance does not exist, the solver would never stop. The existence of a solution is usually checked with another algorithm, for example, Push-and-Rotate [26].

Fig. 1. Ozobot Evo (photo from [27])

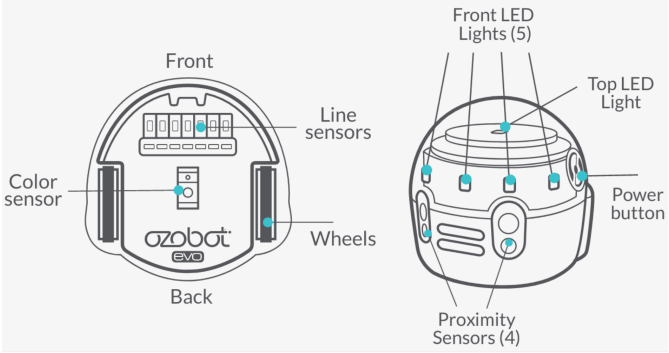


Fig. 2. Sensor layout of Ozobot Evo (photo from [28])

IV. Physical Robot: Ozobot Evo

Ozobot Evo [27] is a small, reflex-based robot without an internal state. Compared to other robots used in research like *e-puck* or *Khepera*, Ozobot is simple and limited, but also affordable. An image of the Ozobot Evo can be found in Figure 1, and its sensor layout is visualized in Figure 2.

The primary capability of Ozobot is to follow lines. Its reflex-based behavior can also be altered, and it can be done in two different ways–*Color Codes* and *OzoBlockly*. Color Codes are instruction markers on lines that can be read by the robot, and OzoBlockly is a visual coding editor providing a set of movement and sensory instructions.

Two main factors were considered when choosing this particular robot for the task:

- We want to test the limits of our proposed approach, and if it is possible to deploy discrete MAPF solutions on such primitive hardware.
- Building large swarms of Ozobot Evo robots is more affordable.

A. Important Hardware

The *motor and wheels* provide the robot with mobility. Ozobot turns by moving the wheels separately at different speeds.

Under the base of the robot are several *line sensors* and an optical *color sensor*. The purpose of these sensors is to detect lines, intersections, and surface color. The color sensor can distinguish eight different colors. Ozobot is highly dependant on this sensor because it allows the robot to read *Color Codes*, and it is also used to load *OzoBlockly* programs into the robot. Four *infrared proximity sensors* can be used to detect an object as far as 10 centimeters from the robot.

B. Movement and Line Following

The essential reflex functionality of Ozobot we used in our approach is following lines drawn on the surface of the physical environment. By default, Ozobot follows lines at a default speed of 30 mm/s. When the line is lost, it stops its movement. At each line intersection, Ozobot chooses one of the possible directions at random, with all the possibilities having the same probability of being chosen. It is important to know that Ozobot does not register a 90-degree turn as an intersection. While following lines, Ozobot will also read and execute *Color Codes* if found.

The accuracy of the optical sensors is limited, and sensors need to be calibrated to ensure better functionality. This is especially true when there is a change in the display brightness or the surrounding light conditions. Line parameters, like line thickness or angle of turns, also need to be set correctly to ensure correct behavior.

V. Novel approach

Previous solution deploying discrete MAPF solutions on physical robots from [19] is simple and works fine for the comparison of MAPF models. However, because of how that approach utilizes Ozobots, there is not much that can be done to enhance the deployment or expand the usability to other MAPF variations. Moreover, it has several drawbacks that could be solved by using a different strategy. In this section, a novel approach is proposed, compared with the previous solution, and previous drawbacks are explained as well as how the new approach solves them.

A. ESO-Nav: Reflex-Based Navigation

The main idea of this novel MAPF deployment approach is to have robots with *fixed reflex behavior* and an *environment that can output information* for the robots, affecting their behavior and controlling them in effect. The environment is a physical representation of a given MAPF problem instance that can navigate the agents in itself by showing them the planned paths and additional information. The plans are obtained from a *centralized MAPF solver* and then processed for the execution in the environment. Since the less constrained MAPF solvers are more efficient than the sophisticated multi-robot motion planning solvers, we hope to make the planning part of the deployment more efficient and scalable for large number of robots. The computational complexity of the system is, therefore, shifted towards solution postprocessing and execution. We call our approach *Navigation by Environment Surface Outputs* (ESO-Nav).

In our prototype, the environment is represented by the surface of a screen capable of providing outputs for Ozobots. A grid map of a MAPF problem instance is displayed on the screen, and the planned paths are animated for the robots. The behavior of the agents can be determined with a simple *OzoBlockly program* loaded into each robot. This approach provides freedom of creating a more sophisticated and highly customizable MAPF simulation.

B. Improvements from Previous Work

There are two significant improvements compared to the previous approach [19].

(i) Synchronization of simulation start. In the previous solution, the map of the problem instance was constructed from the following lines, which leads Ozobots into movement. The execution had to be initiated manually, so scalability for multiple robots is limited. Since robots in ESO-Nav are regarded as entirely reflex, the previous issues with synchronization of the execution start can be easily solved. Before the start, the environment does not output any paths, and robots can *initiate waiting* in that situation. Additionally, ESO-Nav provides more *flexibility* in environment representation and map design since the planned paths are only a piece of extra information. The novel approach could be used for environments that are not grid-based, and the paths could be continuous curves instead of straight lines between positions.

(ii) No need to memorize the plan in advance. Another factor we consider to be an improvement from the previous approach is that robots entirely modular for all problem instances. In the ESO-Nav approach, the robots can be quickly used on different maps without changing their programming. The paths can even be replanned during the execution without the agents noticing. This allows simulation of sub-optimal MAPF solutions optimized during the agent execution, or replanning can be performed if agents fail to follow their paths. Moreover, different behaviors of agents can be tested and compared with this approach.

C. Expected Problems

Some of the *real-world complications* are still concerning this novel approach, but the complications can be mitigated by correct path processing and outputting. The robot *collisions* can still be an issue, even if a MAPF solver finds a valid solution. The environment needs to be designed so that agents

can move around each other without any contact. Incorrect path processing and outputting can also introduce conflicts that are unanticipated and would interrupt the execution. The biggest challenge remains the fact that different robot movements take different time durations, creating *desynchronization* in the plan execution. However, the ESO-NAV approach with the use of Ozobots provides various ways of solving this desynchronization issue without modifying the MAPF solver nor the problem abstraction.

VI. REALIZATION

In this section, a working prototype of the ESO-NAV approach to MAPF simulation that utilizes Ozobots is presented. The prototype can take a MAPF problem instance in the form of a map and execute the solution on the robots.

A. Overview

The prototype application could be divided into several modules that interact with each other during the execution. Most of these modules should be easy to modify or extend to simulate different MAPF models or situations. The program takes a map file that contains the MAPF problem instance to be executed. The problem is passed to the MAPF solver, which yields a solution. This solution contains paths for all agents that should be outputted to the environment. The main simulator module then takes the obtained solution and ensures correct processing and environment output for the agents. The outputs should make the robots execute their planned paths without any manual interference.

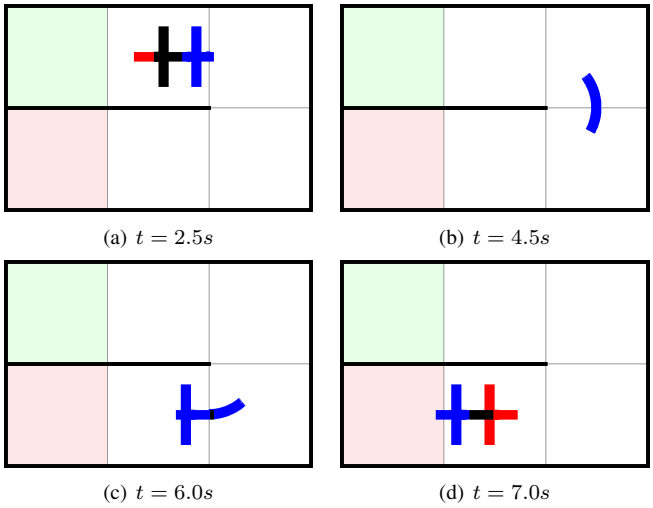
B. Environment

During the simulation, the *abstract environment* representation needs to be transformed into the *physical environment*. This is done by displaying the map on the screen on which the Ozobots can move. For this prototype, *grid-based maps* were chosen to be used and can be displayed as tiles. If there is not an edge between two neighboring vertices in the original graph, a *wall* is displayed on the map between the corresponding tiles. If a particular tile is a *start* or *goal* position of any agent, the tile is colored green or red. If there is a start and also a goal position on a single tile, the tile is filled with both colors using a checker pattern. Note that the colors need to have very *low opacity*, so as the Ozobots do not register them as following lines.

Before the simulation, Ozobots are placed on all green tiles, and at the end of the simulation, they should stand on the red tiles. When all bots are in place, the *following lines* are displayed on the map as well by the simulation module.

C. MAPF Solver

The prototype uses an already existing program that implements the SMT-CBS algorithm [29], which combines the SAT-based solving principle and the CBS algorithm. All algorithms in this program are implemented under the *sum-of-costs* objective function.



(a) $t = 2.5s$ (b) $t = 4.5s$

(c) $t = 6.0s$ (d) $t = 7.0s$

Fig. 3. Colored path segments displayed in the map after t seconds from the start of the execution

D. Simulation via Path Animation

The main and most sophisticated module of the application is the *simulator*. It takes the loaded map of the problem and an obtained solution from the solver module and is responsible for animating the paths for Ozobots. It is responsible for transforming the discrete solution obtained from the solver module into continuous following lines for Ozobots, and displaying them for the bots into their physical environment.

The simplest solution for the path outputting would be to display the whole plan at once. With this outputting method, the robots cannot perform some of their essential actions like wait or turn around at a specific position. They even choose a random direction at each intersection.

These problems have been addressed through animating paths, as shown in Figure 3. The screen outputs animated path segments around the map, on which the Ozobots are moving. The segments are divided into three colored parts, so the robot can reflexively change its movement speed based on the line color. This dynamic speed adaptation provides an active correction of execution desynchronization. However, to make an Ozobot read the line color and change its movement speed, *artificial intersections* had to be displayed on the paths.

If the agent has to stop at a specific position, the path segment stops its animation at the position. When the robot has to stop on a curved turn, a guiding line is displayed to bring the Ozobot into a correct orientation. This ensures that the bot can continue on its path when the path segment reappears under it. In some scenarios, the robot is required to make a U-turn. For this functionality, a special color code supported by the Ozobot is displayed on the screen. The robot can read the code and perform the turn.

An illustration of Ozobots navigating on the path segments is shown in Figure 4.

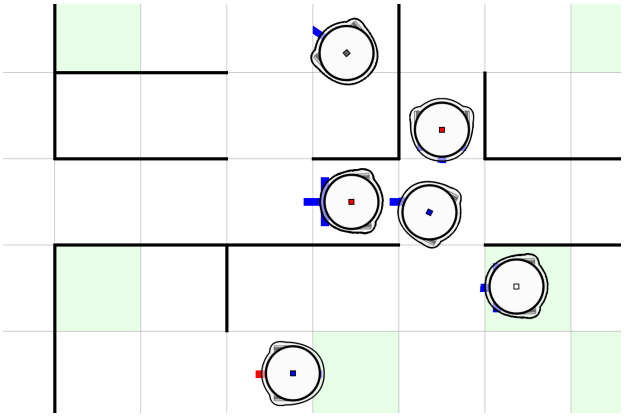


Fig. 4. Illustration of Ozobots following colored path segments on a map

E. Modifying Ozobot Behavior

Ozobots need to follow the lines outputted by the environment and change their movement speed according to the color of the lines. This modified behavior can be written as an OzoBlockly editor and loaded to all robots.

The OzoBlockly program can be found in Algorithm 1. The main loop runs until the program is manually terminated. As can be seen on line 7 of the code, the robot moves on the path segments between the artificial intersections or until it loses the following line. When an intersection or a line end is encountered, the line-following speed is updated on line 2. Because the Ozobot naturally wants to choose a random direction at any intersection, line 4 makes it always go straight. However, if the movement was interrupted and there is no line under the robot, line 6 stops it from moving.

Algorithm 1: Ozobot behavior program

```
1 while true do
2     set line-following speed: getSpeedFromLineColor()
        mm/s;
3     if there is way straight then
4         pick direction: straight;
5     else
6         stop motors;
7     follow line to next intersection or line end;
```

The function `getSpeedFromLineColor` from line 2 is shown in Algorithm 2. First, the function reads the surface color from the optical color sensor on line 2. On lines 3–10, the speed is chosen according to the color, and on line 11, it is returned. The speeds on each of the segments were chosen based on the path segment animation speed.

VII. Experimental Evaluation

We tested the prototype with various scenarios where the focus was on success of execution and problematic maneuvers. Each of the problem instances, represented by a map, was solved with both the move-to-unoccupied variant (denoted *m2u*) and the *standard* MAPF solver[3]. For some maps, both variants yield the same plan[4].

Algorithm 2: Function reading line color and returning speed

```
1  Function getSpeedFromLineColor()
2      color ← get surface color;
3      if color = surface color red then
4          speed ← 37;
5      else if color = surface color black then
6          speed ← 30;
7      else if color = surface color blue then
8          speed ← 23;
9      else
10         speed ← 21;
11     return speed
```

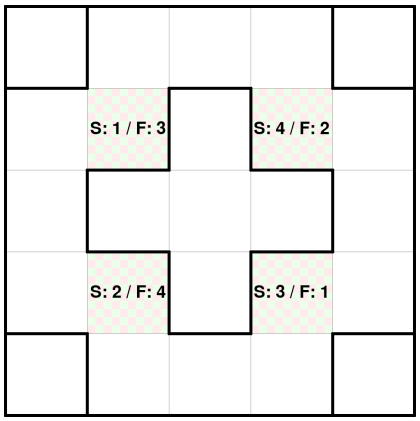


Fig. 5. Experiment map: The rotation

A. Maps

For the experiments, six maps were used. Some of the maps aim to test a specific feature or a critical maneuver of the system, others provide a balanced scenario for the execution. All of the maps are listed in Table I. For each map, its width, height, and number of agents are provided in the table.

In Table II, all plans for these maps are listed. For each plan, some maneuvers that could be problematic for the robots are counted. Namely, the number of turns without waiting, the number of *Color Codes* displayed (CC), and the number of *wait on a turn* positions (WoT).

B. Results

Every plan was executed 32 times with the implemented prototype. The execution is marked as *successful* if all Ozobots reach their goal positions. If at least one loses the following line and does not reach the goal, the execution is marked as a *failure*. The results of the experiments are summarized in Table III. During the testing, five different reasons for

[3]Which allows an agent to moved to an occupied position if the occupant will leave the position at the same time step.

[4]For example, the map *rotation* from our experiments.

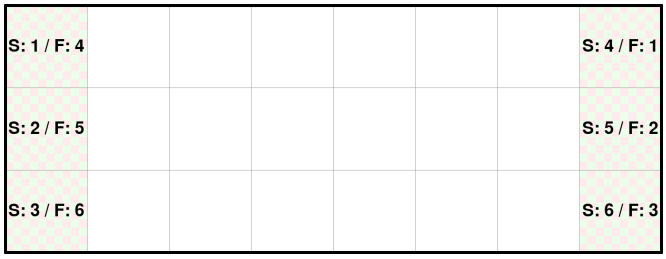


Fig. 6. Experiment map: The swap

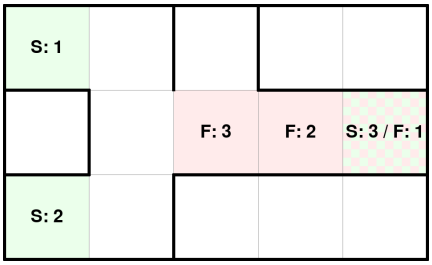


Fig. 7. Experiment map: The ordering

execution failure were recorded. The occurrence of these failures was counted and is presented in the table of results.

Sometimes during the execution, a *severe collision* (SC) can occur, where the robots push against each other or lift their wheels from the surface. This collision results in an execution failure because the robots lose their following lines and are unable to continue. This problem does not occur very often and is non-existent in the move-to-unoccupied plans, , which are generally safer for physical execution.

On the other hand, more frequent was the failure due to *missed intersection* (MI). Sometimes, an Ozobot failed to update its speed at an intersection because it was not detected by its sensors. This almost always resulted in an execution failure if the agent did not slow down before a turn. The first experiments showed that the success rate of an Ozobot to detect an intersection correctly fluctuates with changing light conditions and performing calibrations. However, results from the plan executions on the *roundabout* map suggest that the complexity of the paths might also affect the frequency of these mistakes.

The maneuvers as *CC* and *WoT* also caused a few execution failures. To fail to execute the *U-turn*, the robot can either arrive at the tile too soon or too late. If it arrives too soon, it fails to read the whole code and does not rotate at all. If it arrives too late, it reads the code twice and makes two rotations ending up in the original orientation. Both of these

TABLE I
MAPS CREATED FOR EXPERIMENTS

Map name	Width	Height	Agents	Image of the map
snake	10	2	6	-
rotation	5	5	4	In Figure 5
swap	8	3	6	In Figure 6
ordering	5	3	3	In Figure 7
evacuation	10	5	6	In Figure 8
roundabout	9	5	6	In Figure 9

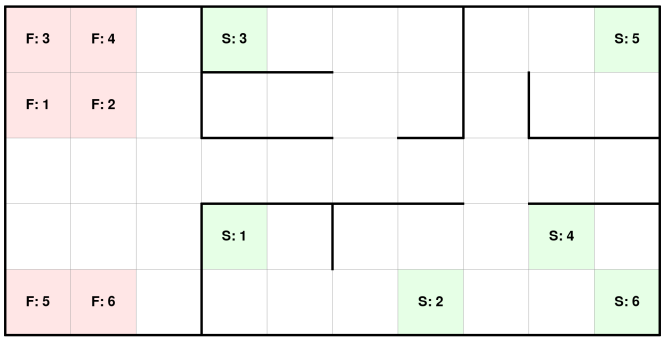


Fig. 8. Experiment map: The evacuation

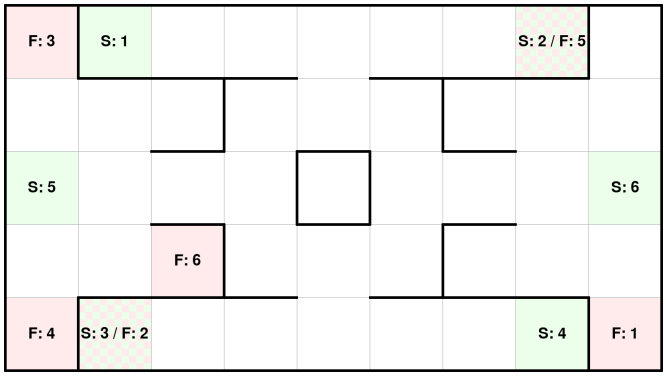


Fig. 9. Experiment map: The roundabout

TABLE II
PLANS EXECUTED WITH THE PROTOTYPE

Plan name	Map	Turns	CC	WoT
snake	snake	12	0	0
snake_m2u	snake	12	0	0
rotation	rotation	20	0	0
swap	swap	10	0	1
swap_m2u	swap	9	0	3
ordering	ordering	5	1	0
ordering_m2u	ordering	3	1	2
evacuation	evacuation	26	0	1
evacuation_m2u	evacuation	25	0	1
roundabout	roundabout	33	0	0
roundabout_m2u	roundabout	30	0	1

TABLE III
RESULTS OF THE EXPERIMENTS

Plan name	Success	Fail	SC	MI	CC	WoT	EC
snake	32	0	0	0	0	0	0
snake_m2u	32	0	0	0	0	0	0
rotation	29	3	0	3	0	0	0
swap	30	2	0	1	0	1	0
swap_m2u	30	2	1	0	1	0	0
ordering	30	2	1	0	1	0	0
ordering_m2u	24	8	0	0	4	3	1
evacuation	28	4	2	0	0	2	0
evacuation_m2u	30	2	0	2	0	0	0
roundabout	23	9	0	9	0	0	0
roundabout_m2u	19	13	0	8	0	3	2

scenarios ensure the failure of plan execution. Performing the *WoT* maneuver, the robot sometimes makes a U-turn at the end of the following guiding line. This behavior is most likely triggered when Ozobot loses the following line without detecting a line end.

The last problem noticed during the experiments was Ozobot failing to *exit a curve* (EC). Even though this was a rare occurrence, sometimes, when the bot passed through a curved turn, it was unable to detect the following line correctly and lost the path.

VIII. Conclusion

This work has introduced a novel approach to executing discrete MAPF algorithms on a swarm of physical robots called ESO-Nav. The approach uses environment outputs and reflex-based robotic agents to emulate centralized control and shifts the computational complexity of the system from planning towards execution. This should make the system more scalable for a large number of robots. A swarm of Ozobot Evo robots has been used for the prototype.

Despite the limitation of these robots, the tests showed that the ESO-Nav approach can deploy various MAPF scenarios on physical agents using existing MAPF algorithms that use classical discrete models. We also identified various problems that need to be overcome to carry out the execution successfully. Most of them are dependent on the capabilities of robots being used. Unlike previous works dealing with robotic agents deployment, we did not need to augment the classical MAPF model in any sense, and an off-the-shelf MAPF solver was used. Using reflex control of robots through path animation on the surface of the screen makes our approach more flexible and easy to use.

The secondary contribution of our simulation approach is the prototype that can be used for demonstration in research or academics, as well as in testing of real-world applications. We believe the novel approach could be used in applications like intelligent evacuation systems or indoor transporter navigation.

Acknowledgements

This work has been supported by GAČR - the Czech Science Foundation, grant registration number 19-17966S.

References

[1] D. Kornhauser, G. L. Miller, and P. G. Spirakis, "Coordinating pebble motion on graphs, the diameter of permutation groups, and applications," in *Proc. 25th Annu. Symp. Foundations of Computer Science (FOCS)*, 1984, pp. 241–250.

[2] M. R. K. Ryan, "Exploiting subgraph structure in multi-robot path planning," *J. Artif. Intell. Res. (JAIR)*, vol. 31, pp. 497–542, 2008.

[3] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artif. Intell. (AIJ)*, vol. 219, pp. 40–66, 2015.

[4] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding," *Artif. Intell. (AIJ)*, vol. 195, pp. 470–495, 2013.

[5] D. Silver, "Cooperative pathfinding," in *Proc. 1st Artif. Intell. and Interactive Digital Entertainment Conf. (AIIDE)*, 2005, pp. 117–122.

[6] P. Surynek, "A novel approach to path planning for multiple robots in bi-connected graphs," in *2009 IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2009, pp. 3613–3619.

[7] K. C. Wang and A. Botea, "MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees," *J. Artif. Intell. Res. (JAIR)*, vol. 42, pp. 55–90, 2011.

[8] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Mag.*, vol. 29, no. 1, pp. 9–20, 2008.

[9] R. Morris, C. S. Pasareanu, K. S. Luckow, W. Malik, H. Ma, T. K. S. Kumar, and S. Koenig, "Planning, scheduling and monitoring for airport surface operations," in *Planning for Hybrid Systems, Papers from the 2016 AAAI Workshop, Phoenix, Arizona, USA, February 13, 2016*, ser. AAAI Workshops, vol. WS-16-12, 2016.

[10] D. Zhou and M. Schwager, "Virtual rigid bodies for coordinated agile maneuvering of teams of micro aerial vehicles," in *IEEE Int. Conf. on Robotics and Automation (ICRA) 2015, Seattle, WA, USA, 26-30 May*, 2015, pp. 1737–1742.

[11] R. Shome, K. Solovey, A. Dobson, D. Halperin, and K. E. Bekris, "drrt*: Scalable and informed asymptotically-optimal multi-robot motion planning," *Autonomous Robots*, vol. 44, no. 3-4, pp. 443–467, 2020.

[12] R. M. Wilson, "Graph puzzles, homotopy, and the alternating group," *Journal of Combinatorial Theory, Series B*, vol. 16, no. 1, pp. 86–96, 1974.

[13] R. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *Proc. 22nd Int. Joint Conf. on Artif. Intell. (IJCAI), 2011*, 2011, pp. 294–300.

[14] P. Surynek, "Time-expanded graph-based propositional encodings for makespan-optimal solving of cooperative path finding problems," *Ann. Math. Artif. Intell.*, vol. 81, no. 3-4, pp. 329–375, 2017.

[15] K. M. Dresner and P. Stone, "A multiagent approach to autonomous intersection management," *J. Artif. Intell. Res. (JAIR)*, vol. 31, pp. 591–656, 2008.

[16] D. Ratner and M. K. Warmuth, "Finding a shortest solution for the N × N extension of the 15-puzzle is intractable," in *Proc. 5th Nat. Conf. on Artif. Intell., Volume 1: Science*, 1986, pp. 168–172.

[17] A. Andreychuk, K. S. Yakovlev, D. Atzmon, and R. Stern, "Multi-agent pathfinding with continuous time," in *Proc. 28th Int. Joint Conf. on Artif. Intell. (IJCAI), 2019*, 2019, pp. 39–45.

[18] J. Li, P. Surynek, A. Felner, H. Ma, T. K. S. Kumar, and S. Koenig, "Multi-agent path finding for large agents," in *33rd AAAI Conf. on Artif. Intell., 31st Innovative Appl. of Artif. Intell. Conf., 9th AAAI Symp. on Educational Advances in Artif. Intell., Honolulu, Hawaii, USA, Jan./Feb., 2019*, 2019, pp. 7627–7634.

[19] R. Barták, J. Svancara, V. Skopková, and D. Nohejl, "Multi-agent path finding on real robots: First experience with ozobots," in *Advances in Artif. Intell. - IBERAMIA 2018 - 16th Ibero-American Conf. on AI, Trujillo, Peru, November 13-16, 2018, Proc.*, ser. Lecture Notes in Computer Science, vol. 11238, 2018, pp. 290–301.

[20] T. S. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *Proc. 24th AAAI Conf. on Artif. Intell.*, 2010.

[21] P. Surynek, "Towards optimal cooperative path planning in hard setups through satisfiability solving," in *Proc. 12th Pacific Rim Int. Conf. on Artif. Intell. 2012.*, ser. Lecture Notes in Computer Science, vol. 7458, 2012, pp. 564–576.

[22] M. Ryan, "Constraint-based multi-robot path planning," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2010, pp. 922–928.

[23] G. Gange, D. Harabor, and P. J. Stuckey, "Lazy CBS: implicit conflict-based search using lazy clause generation," in *Proc. 29th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2019, pp. 155–162.

[24] Y. Izmirlioglu, B. A. Pehlivan, M. Turp, and E. Erdem, "A general formal framework for multi-agent meeting problems," in *2017 IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2017, pp. 1299–1306.

[25] P. Surynek, A. Felner, R. Stern, and E. Boyarski, "Efficient SAT approach to multi-agent path finding under the sum of costs objective," in *22nd European Conf. on Artif. Intell. (ECAI)*, ser. Frontiers in Artificial Intelligence and Applications, vol. 285, 2016, pp. 810–818.

[26] B. de Wilde, A. ter Mors, and C. Witteveen, "Push and rotate: a complete multi-agent pathfinding algorithm," *J. Artif. Intell. Res. (JAIR)*, vol. 51, pp. 443–492, 2014.

[27] I. Evollve. (2020) Ozobot. [Online]. Available: https://ozobot.com/

[28] ——, *Ozobot sensor layout images*, 2020. [Online]. Available: https://files.ozobot.com/classroom/2019-Educator-Guide.pdf

[29] P. Surynek, "Unifying search-based and compilation-based approaches to multi-agent path finding through satisfiability modulo theories," in *Proc. 28th Int. Joint Conf. on Artif. Intell. (IJCAI), Macao, China, August 10-16*, 2019, pp. 1177–1183.