Title
Research Project - KIREPRO1PE

Szymon Gałecki - sgal@itu.dk

July 2024

Contents

1 Methods

In the reviewed papers we can find multiple approaches to verification and testing of implemented swarm behaviours. Algorithms for those behaviours were usually well documented alongside the experiments and obtained results. Sometimes there would be an automaton corresponding to the textual description of behaviour or pseudocode. Used logic and tools were also mentioned frequently. However we would not be able to reproduce the experiments because none of them explained in detail how the chosen swarm behaviour was modelled. That is why this section will focus on explaining how algorithm's pseudocode and general assumptions were transformed into a functioning model. The selected algorithm to demonstrate the process of modelling is Alpha algorithm. It was chosen because there were many experiments that addressed Alpha algorithm but omitted the details of what was actually being tested or verified.

1.1 Alpha algorithm

Alpha algorithm was introduced by Julien Nembrini in [1]. It was inspired by Kasper Støy's work [2]. Støy proposed and implemented a simple control system for aggregating robots. Instead of relying on environment and localisation information, it uses physical properties of the signal used for communication. Robot behaviour is solely determined by the change in the number of robots that are in the range of its signal.

Alpha algorithm is an approach to an aggregation task within the category of spatial organisation. It is based on assumption that robots send and receive signals through omnidirectional channels like radio or infrared. Single robots make decisions about their movement based only on the number of connections to other robots. The inter-connectivity of the swarm is controlled by the alpha parameter which is a threshold on the desired number of connections for a single robot. Pseudocode defining the Alpha algorithm can be found in Figure 1.

In order to explain the Alpha algorithm in a greater detail we will divide the most important aspects of the robot behaviour into subsections. First three subsections: Movement, Connection, Initialisation and clocks, will explain the algorithm's general assumptions and design choices. The fourth and last subsection - Automaton, will transform the pseudocode for the Alpha algorithm into automaton while incorporating design choices introduced in previous subsections.

Figure 1: Pseudocode for Alpha algorithm from [1]

```
Create a list of neighbours for robot, Nlist
k = number of neighours in Nlist
i = 0

loop forever {
        i = i modulo cadence

        if (i = 0){
                Send ID message

                Save copy of k in LastK
                k = number of neighbours in Nlist

                if ((k < lastK) and (k < alpha)){
                        turn robot through 180 degrees
                }
                else if (k > LastK) {
                        make random turn
                }
        }

        Steer the robot according to state
        Listen for calls from robots in range
        Grow Nlist with neighbours IDs

        i++
}
```

1.2 Movement

There would be no need for Alpha algorithm without the robot movement so it is important to define how it is modelled. Robot always moves in one of four directions: up, right, down or left. Initial direction is chosen at random and mapped to vertical and horizontal components. Direction will not change unless the robot performs random turn or 180 degree turn. Random turn chooses new direction in the same way that initial direction is determined. This means that random turn may result in maintaining the current direction of the robot with approximately 25% chance.

Robot movement is achieved by incrementing the robot's coordinates with vertical and horizontal direction components. Direction component values are $\in \{-1, 0, 1\}$ with one of them being equal to zero and other being different than zero. This means that robot will move in one of four possible directions with a step size equal to one. Every time the robot moves, it will update its coordinates

in the globally available data structure. Environment in which robot exists is an unbounded grid that can be continuously traversed in four directions.

Robots are not aware of other robots positions. This means that they can occupy exactly the same point of the grid. Additionally, there is no collision avoidance mechanism that would prevent them from crashing into each other while moving. This simplification could be eliminated but at the cost of implementation being further apart from the original idea of what is an Alpha algorithm. We would have to make arbitrary decisions about robot behaviour in the case of collision or inability to move to the occupied point on the grid.

1.3 Connection

Number of robot connections is the main parameter determining the robot behaviour. Therefore it is important to accurately model what it means for robots to be connected. One of the assumptions in Julien Nembrini's [1] is that physical signal is omnidirectional. However, robot movement is modelled to be two dimensional. Consequently we will assume that two robots are connected if their distance is smaller than radius of physical signal that we mimic. Length of the radius will determine how far apart the robots can move before losing connection.

To mimic the physical signal we need information about robot coordinates, mutual distances, previous and current number of neighbours. This allows for evaluating whether robots are in the signal distance and therefore connected or not. Every time robot moves it will update its own coordinates, previous number of neighbours for itself, current number of neighbours for all robots and mutual distances for all robots. It may seem like the robot is using more information than it is assumed in the Alpha algorithm. Information that it passes is used to mimic the physical signal of the real robot. Although, it updates its coordinates, robot is not aware of the position of its neighbours. Its movement is derived solely from information about previous and current number of neighbours.

The desired number of connections is set by the alpha parameter. If the number of connections falls below alpha, robot will react and turn 180 degrees in order to try to reconnect with other robots. Alpha parameter influences connectivity of the swarm. If the parameter is set to higher values the robots will try to maintain more connections and the resulting swarm will be more compact. If the parameter is set to lower values, robots will be able to move further apart as they will need to maintain less connections. What is low and high value for alpha parameter is subjective to the size of the swarm.

1.4 Initialisation and clocks

This subsection explains the initial state of the robot and the influence that the time has on its behaviour. Every robot gets initialised with the same set of parameters apart from its ID. In the beginning all robots are placed at exactly the same point. They have no direction until transitioning to the next state and will not move forward until transitioning further. In the initial state robots

can't access information about number of current neighbours and number of last neighbours. The initial state is not reused within the automaton, once a robot leaves that state, it can never reach it again.

Each robot has its own clock that controls how long it can remain in the current state before being forced to transition. After each transition to the next state the robot will reset its own clock. Maximum time for the robot to remain in state can influence the behaviour of the whole system. If we decrease the maximum time of inactivity, we will obtain a system that is on average more reactive and even. With increasing the maximum time of inactivity we increase the chances of robots operating in different pace. As we are modelling swarm we should aim for uniform operation and therefore lower maximum times for inactivity.

1.5 Automaton

In order to implement the Alpha algorithm presented in [1], we utilised the pseudocode from Figure 1, with design choices described in the previous subsections. The implementation is a timed automaton - Figure 2.

We will describe how pseudocode was mapped into the timed automaton. Pseudocode starts with creating and initialising variables that are needed for algorithm execution. This part is not explicitly visible in the automaton but achieved during initialisation of the model. The state of the algorithm before entering the infinite loop is represented by the INITIAL state. Transitioning from the INITIAL state is equivalent to entering the infinite loop defined in the pseudocode. The entry point for the loop is a RANDOM TURN state where the robot obtains its initial random direction. Infinite loop defines that robot will send its ID and make a decision about turning only every 'cadence' loop iterations. To mimic this behaviour cadence was implemented as the maximum time of inactivity T. In the pseudocode decision about turning is realised with 'if' and 'if else' statements. Statements and their corresponding conditions were transformed to IF and IF ELSE states with conditions guarding subsequent transitions. Guarded transitions utilise original conditions and their negations. In this way we can guarantee that there is always only one transition possible at any point. Variables used in the conditions correspond to the current number of neighbours, previous number of neighbours and desired number of neighbours. Those variables control the behaviour of a single robot alongside the maximum time of inactivity T which acts as a cadence parameter of the loop. Composition of many of such robots into a single system is an implementation of Alpha algorithm for the swarm of robots.

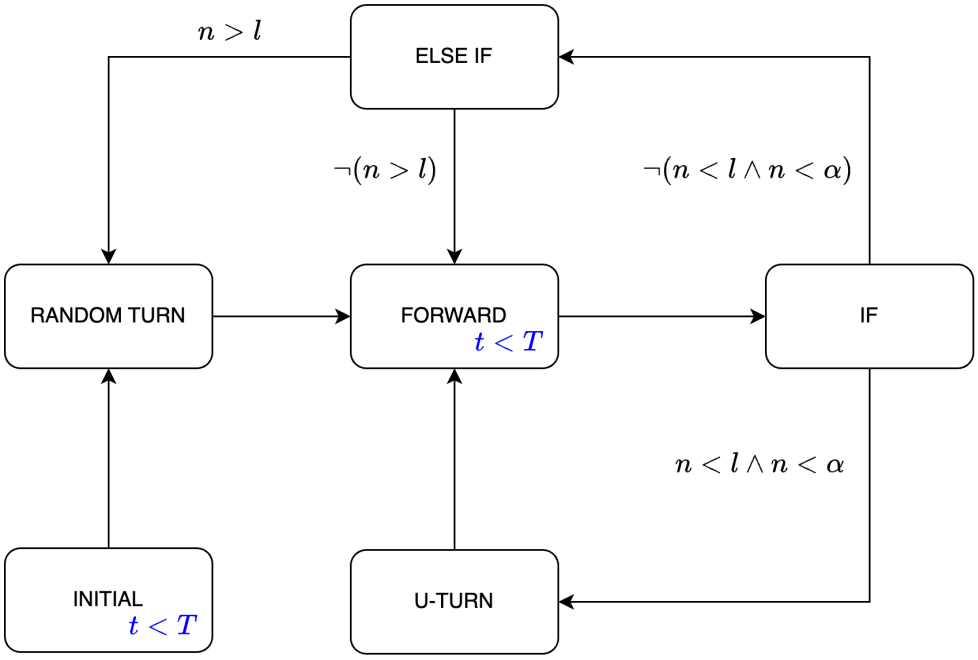
State descriptions:
- INITIAL, where the robot is not moving and has no direction;
- RANDOM TURN, where the robot randomly chooses one of four directions;
- U-TURN, where the robot changes direction by 180 degrees;
- FORWARD, where the robot moves forward in the set direction;
- IF, where the robot checks if it is disconnected from neighbours;

- ELSE IF, where the robot checks if it is connected to more than desired number of neighbours;

States are not enough to define the behaviour if the robot is not forced to transition. Each robot has its own clock that controls how much time it can remain in a given state. Time - t a robot can remain in states FORWARD and INITIAL is limited by the maximum time T. Robot has to transition before time t is equal to T. After leaving mentioned states, the robot resets its clock. In remaining states the time doesn't pass and therefore time t is not incremented. However other transitions are guarder by associated conditions. Transition from one state into another can only happen if the corresponding condition is satisfied. If there is no condition over the edge, transition can always be taken.

Conditions controlling the robot are associated with the following variables:
n - current number of neighbours;
l - previous number of neighbours;
α - Alpha parameter, desired number of neighbours;

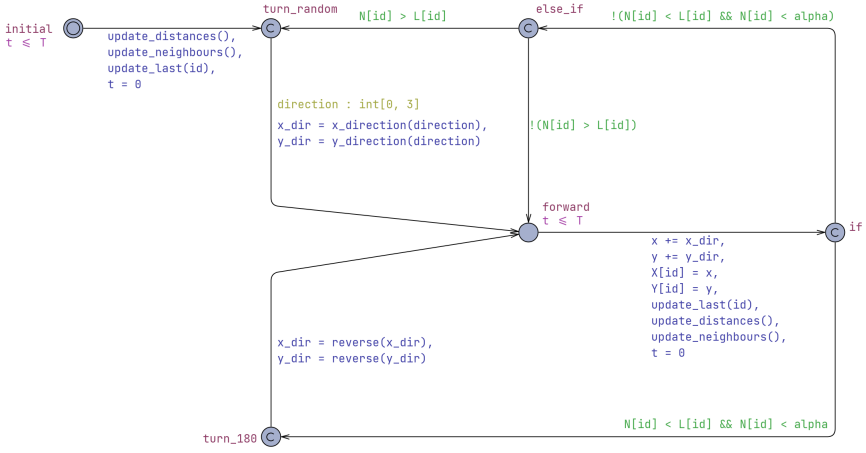
Figure 2: Timed automaton for the Alpha algorithm



2 Implementation

In the previous section we introduced the Alpha algorithm and taken assumptions which were transformed into a timed automaton. The former description should be complete enough to implement Alpha algorithm with any suitable tool of choice. In this section we will focus on implementation with regards to the specific tool of our choice, UPPAAL. As the previous section finished on timed automaton, we will start by presenting timed automaton implemented in UPPAAL. We will draw connections between them as presented in Figure 2 and Figure 3. Then, we will explain global functions and state.

Figure 3: UPPAAL implementation of the automaton



To start with comparison between general automaton and its specific implementation in UPPAAL, let's notice that both automata have the same set of states and transitions. Guarded transitions use the same conditions. Each robot accesses its own set of condition variables by providing **id** to the global arrays. Array **N** holds the number of neighbours for each robot. Array **L** holds the last number of neighbours for each robot. Those variables are held in the global arrays so that a single robot could update the number of neighbours for all of the other robots every time it moves. It may seem counter intuitive that a single robot updates the variables of other robots. However, the execution of the system composed from multiple robots is based on interleavings. At any given time a single robot transitions between states. This means that only the moving robot is guaranteed to have true information about number connections. Global updates of the number of neighbours is a way to mimic physical, continuous signal that would determine whether robots are connected or not. It eliminates the situation where robot would make a decision on already outdated information. The implementation assumes that each time a robot moves it will update its coordinates, update the last number of neighbours for itself

and update the current number of neighbours for all robots. It achieves the last two actions by calling **update_last(id)** and **update_neighbours()**. Figure 4 shows the process of updating the number of connections for all robots.

Figure 4: Implementation of **update_neighbours()** and **neighbours()**

```
int neighbours(int id){
    // returns the number of connections for a robot with a given id
    int x = X[id];  // x coordinate
    int y = Y[id];  // y coordinate
    int k = 0;      // neighbour counter k
    int i = 0;
    for (i = 0; i < n; i++){
        double d = distance(x, y, X[i], Y[i]);
        bool c = connected(d);
        // is robot connected to another robot
        if(i != id && c){
            k++;
        }
    }
    return k;
}

void update_neighbours(){
    // updates the array that stores number of neighbours for each robot
    int i;
    for (i = 0; i < n; i++){
        N[i] = neighbours(i);
    }
}
```

The number of connections will then determine the next transitions. Transition that is forced by the clock in the **forward** state will always result in the robot moving by a unit step in its current direction. Then based, on logical value of the conditions guarding transitions, it can also change its direction. The direction will change after robot reaches and transitions from states **turn_random** or **turn_180**. The random turn is realised as a random choice between four directions: up, right, down, up and translating it into x, y coordinates. The 180 degree turn is realised as choosing the opposite direction to the current one. Change of direction is achieved by functions **x_direction(direction)**, **y_direction(direction)**, **reverse(direction)**. Figure 5 show the process of changing direction.

Figure 5: Implementation of **x_direction(direction)** and **reverse(direction)**

```
int x_direction(int direction){
    // maps direction to x axis
    if (direction == 1){
        // right
        return 1;
    }
    if (direction == 3){
        // left
        return -1;
    }
    // neither
    return 0;
}

int reverse(int direction){
    // reverses direction
    return direction * -1;
}
```

The last element of the implementation is time. Each robot has its own clock that limits how long it can stay in the states with defined invariants: **initial**, **forward**. After transitioning from those states, a robot will reset its clock. Only in those states, the time passes. Other states are so called committed locations. In committed locations time doesn't pass which means that individual robot clocks do not progress. As a consequence the time progresses during the forward motion of the robot but decision to turn happens instantaneously.

We started from general timed automaton implementing the Alpha algorithm. We implemented it using UPPAAL and recreated states, transitions and conditions. We explained the reasoning behind using the global state for condition variables and the mechanism of updating them. We showed how the robot movement and changes of directions are realised and when it happens. Finally we described the timing aspect and its practical implications on the behaviour of the model.

References

[1] Julien Nembrini, Alan Winfield, and Chris Melhuish. Minimalist coherent swarming of wireless networked autonomous mobile robots. 09 2002.

[2] Kasper Stoy. Using situated communication in distributed autonomous mobile robotics. pages 44–52, 01 2001.