Flexible self-Organizing Robotic Modules (FORMbots) Automatic Controller Synthesis

Nialah Wilson

1. INTRODUCTION

Swarm robotics is a growing field, where multiple robotic agents work together to perform a task. Modular robotics combines smaller, units to form one working robot. In nature, individual amoeba in a slime mold practice a technique where they move collectively to create a mobile mass, enabling them to reach new food sources. This project, FORMbots, Flexible self-Organizing Robotic Modules, takes inspiration from this technique.

Our robotic modules are not mobile on their own because they have no moving parts, but in a group, by communicating with their neighbors, they can collectively move to a source by rotating around each other in a planar space. They are flexible, so they can conform to their environment for easier navigation, and they are easy and cheap to replace. FORMbots are advantageous because they are immune to failure of a subset of units. They could work in any hostile environment where it would be dangerous if the malfunction of one part meant the whole robot is immobilized. This is ideal for underwater applications where satellite communication is infeasible or robots with more complex actuation could experience more difficulties withstanding pressure at low depths.

Our goal is to ensure the FORMbots will reach the target destination while avoiding obstacles when navigating through a partially-unsafe environment. The focus of this project is determining the transition relation between a swarm of modular robots, with the physical constraint that they must always be in contact while performing a task. An individual module is immobilized if it becomes detached from the group. Given n modular robots in m cells in a partitioned workspace, a transition system T, and a set of specifications, a controller is synthesized that guarantees the task will always be completed.

The outcomes of this project are an algorithm for determining the transition relation, an automated Büchi automaton representation, an automatically composed product automaton, search, and simulation. The results prove that a controller can be generated for two robotic modules in a workspace of m size with obstacles. The simulations demonstrate various combinations of workspace size, obstacle and goal arrangements, and specifications.

2. SYSTEM DESCRIPTION

A. Hardware

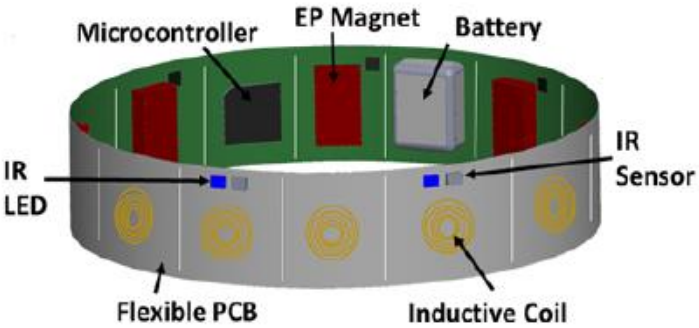


Figure 1: System Diagram of one FORMbot [1]

These modules work by selectively turning on and off electro-permanent magnets located on the circumference of their ring-shaped bodies (Figure 1). These magnets are 6.35x10x4 mm and switch state when an electric current passes through them. [1] This allows the modules to latch and unlatch as necessary, to rotate around their neighbors. Each FORMbot has 8 magnets. Their bodies are 7 cm in diameter and are formed from rolled flexible printed circuit boards. [1] The module is powered by a 3.7V, 450mAh, Lithium Polymer battery. The voltage is boosted to 30V via a boost regulator integrated circuit. [1] IR sensors detect light emitted from a beacon so the overall system knows which direction to travel. IR emitters enable the module itself to become a beacon if stranded. When neighboring robots agree on which module will move, they coordinate which magnets need to be switched in order to travel. The 'mover' robot is the module with the lowest light gradient, as it is farthest away from the beacon. Communication is possible using an ATmega168A microcontroller. Small pulses send bits via electromagnetic interference containing information about the module's magnetic states and desired direction of movement. Figure 2 shows the movement of a FORMbot rotating from the bottom position towards the top of the group. This paper will focus on modeling centralized control of the robots.

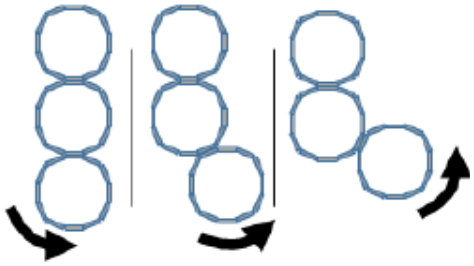


Figure 2: FORMbot movement [1]

B. System Model

To represent the robots' environment, a workspace is partitioned into *m* squares. The current algorithm is configured to work for a two-robot case. The robots are represented by the blue circles in Figure 3. They must always remain in contact and are modeled in adjacent squares. The yellow region is A, the initial state. The red region represents an obstacle and is labeled r1. The green region, represents the goal, and is labeled B. The general specification formulism is written in Linear Temporal Logic (LTL) as follows:

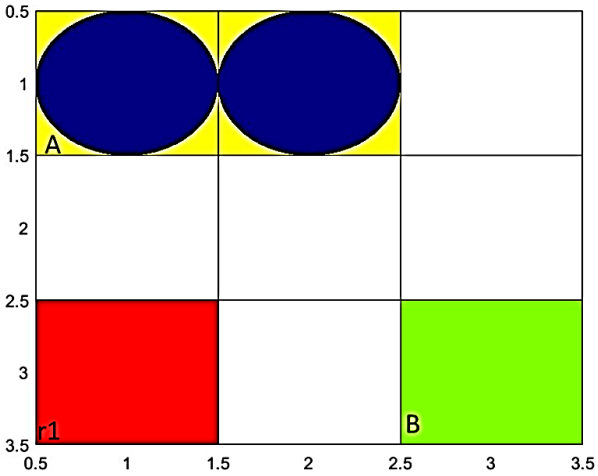


Figure 3: System Representation

$$\varphi = A \wedge \Diamond B \wedge \Box \neg \text{ (r1)}$$

LTL is a way to expresses natural language using temporal and logical operators. The algorithm does not use Full LTL, the next operator is excluded. The specification φ is satisfied if one robot eventually reaches B, and no robot ever reaches r1. Even though there are many modules, the robot collective should be represented as being in a single state. To achieve mutual exclusion, a state is characterized by *n* number of squares. For the remainder of this paper, a state will be assumed to be two squares. For example, in the 3x3 world shown in Figure 3 there are 12 possible states the collective robot can occupy, not 9.

3. METHODS

A. World

The first step in the algorithm is to model the workspace. This is represented by a world with attributes similar to Figure 3. The user inputs the size of the world by specifying a row and column number, the initial region, the goal region, the bad region(s), and any treasure region(s) the user wishes the robots to visit before reaching the goal. The world is drawn separate from the robots. The robots are automatically placed in the initial region at the beginning of the program. Their position is updated as the simulation runs.

Algorithm 1: The Main Structure of the Program

```
1 Input: number of rows, r, columns, c, robots' initial position
   [x1 y1; x2 y2] the goal, bad and treasure positions, Büchi text file
2 Start
3 maze = init(r, c, initial, goal, bad, treasure)
4 robotA.pos = [initial(1,1) initial (1,2)]
5 robotB.pos = [initial(2,1) initial (2,2)]
6 for horizontal transitions
      currentState = [i j i+1 j]
      [states,currentState] = move(currentState,maze,states)
  end
7 for vertical transitions
      currentState = [i j i j+1]
      [states,currentState] = move(currentState,maze,states)
8 if currentState pos1== states pos2 && currentState pos2 == states pos1
      states pos1 = currentState pos1
      states pos2 = currentState pos2
  end
9 for i=1:length(states)
      if parent
            state(end+1) = states(i)
  end
10 ba = buchi(filename)
11 pa = product(states.state,ba)
12 path = search(pa,goal)
13 animate(path)
end
```

B. Transition Relation

The transition system, T, is generated by following a few rules. From any state, a robot can only transition either clockwise (CW) or counter clockwise (CCW) to reach a square adjacent to its neighbor. For the two-robot case, there are a maximum of 4 possible states that can be reached from a given state. The number of states that can be reached is reduced if the state is adjacent to an obstacle, or at the edge of the world. While the square model is a reasonable abstraction, of the robots and their environments, it is not perfect. It is necessary to check that the adjacent squares are not obstacles, and also any diagonal squares the robot must pass through to reach its destination. For example, in the 3x3 world, if one robot is located in (1,2) and the second in

(2,2), robot one is not allowed to move CCW to (2,3) because it must travel through (1,3). In the algorithm, states are represented numerically by the robots' coordinates. The robots in Figure 3 are in state 1121, the pairs of numbers being the x and y-coordinates of the first and second robot respectively. A list of states is generated and stored in a struct, and their respective transitions are listed subsequently. A visualization of the transition relation is shown in Figure 4.

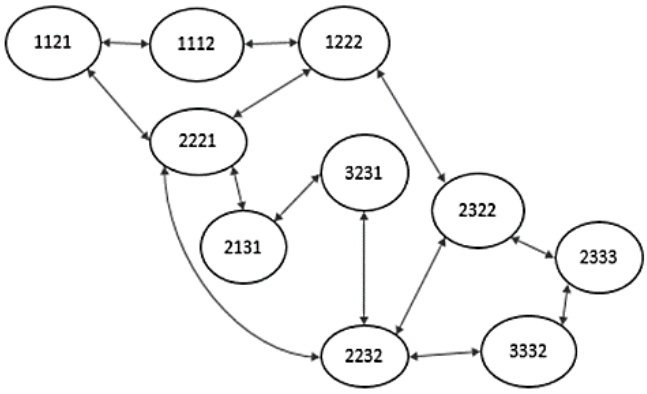


Figure 4: 3x3 world transition relation

C. LTL2BA

The next step in the algorithm converts an externally generated Büchi automaton into a format the program can understand. A Büchi automaton is a transition relation that guarantees all specifications are satisfied once a run reaches an accepting state. Oddoux and Gastin developed an automatic LTL to Büchi Automaton generator and published their work in a 2001 paper. [2] The software is available for public use on their website. [3] Their software outputs a graphical and verbose description of the Büchi automaton. The verbose description is saved as a text file and this is what is analyzed by the algorithm. The states and transitions of the Büchi automaton are extracted, and stored in a matrix. The LTL formula used to produce the Büchi automaton for this example is:

$$b1121 \wedge \Diamond (b2333 \vee b3332) \wedge \Box \neg (b1213 \vee b1323)$$

This is because there are two possible goal states and two possible obstacle states. With reference to φ, b1121 is A, b1213 and b1323 correspond to r1, and b2333 and b b3332 correspond to B.

```
Buchi automaton after simplification
state init
b1121 & {3,4} -> T0_2
b1121 & b2333 & {3,4} -> accept_3
b1121 & b3332 & {3,4} -> accept_3
state T0_2
{3,4} -> T0_2
b2333 & {3,4} -> accept_3
b3332 & {3,4} -> accept_3
state accept_3
{3,4} -> accept_3
```

Figure 5: Verbose mode Büchi representation [3]

Figure 5 is the output of the LTL2BA software in verbose mode. Here the accepting state is denoted "accept_3". On the right side of the arrows are transitions to states which have satisfied the specification on the left. A matrix of 0.1s the size of the number of Büchi states is created. The program reads the text file line by line and stores the number at the end of line to a column in the matrix. When "state" is read, numbers are added in the next row. For example, the matrix for this automaton is:

2	3	3
3	3	0.1
3	0.1	0.1

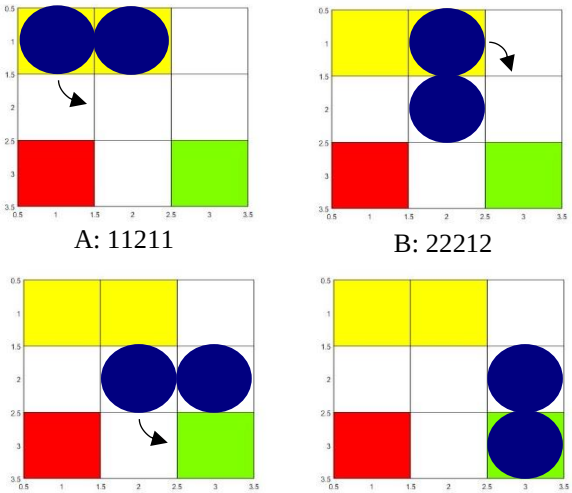
D. Product Automaton

The Product Automaton is created from taking the cross product of the transition relation, T and the Büchi Automaton. A comprehensive list of combined states is generated. From these new states, the allowed transitions are determined using the restrictions from the Büchi automaton and the transition relation. The generated list is in the same form as the transition relation. The combined state is represented the same as the transition relation states, with the addition of the Büchi state at the end. For example, the initial state crossed with the first Büchi state is represented as: 11211.

E. Search

A depth-first search is implemented to find a path to the goal region. A run of the search over the product automaton transitions produces the path shown in Figure 6.

The goal region, B is in grid space (3,3). The robots start in the initial state. Robot one, has reached B and is in accepting state, 3. Neither robot entered r1, grid space (1,3), therefore φ is satisfied.



A: 11211 B: 22212

C: 22322 D: 33323

Figure 6: Result of depth-first search

F. Proof

To verify the transition relation, Büchi converter, and product automaton algorithms are working correctly the expected number of states and transitions, without obstacles, that should be produced were determined analytically, then checked with various numerical cases. The number of states produced are in Table 1. For *T*, the number of states is determined by the world size:

$$\#states = (\#columns - 1)\ x\ \#rows + (\#rows - 1)\ x\ \#columns$$

Table 1: Validation of *T*, Büchi, and Product Automaton Algorithms

World Size	# States	# Büchi States	# Product States
3x3	12	3	36
4x4	24	3	72
5x5	40	3	120
10x10	180	3	540
15x15	420	3	1260
20x20	760	3	2280
50x50	4900	3	14700

4. Analysis

A. Results

The example 3x3 case produced a path that did not violate φ and produced the shortest path to the goal. There is only one obstacle for the robots to contend with. In Figure 7, there are two obstacles, and their positioning does not allow the robots to complete the task. In this case, the program stops running once it enters the search function. Transitions and the product automaton will still be produced. The algorithm also works well when the size of the world is expanded. In the 10x10 world in Figure 8, φ is still satisfied. The search algorithm does not always produce the most optimal path—Dijkstra or A* algorithms could be implemented. In the 10x10 world scenario in Figure 8, the path is shown in blue. The robots avoid all obstacles and reach the goal. The path produced explores a large portion of the world before reaching the goal instead of first traveling down towards the goal, as is the case in Figure 9. This is because the obstacle at (4,2) prevents the robots from traveling right. Videos of these runs are available in [4].

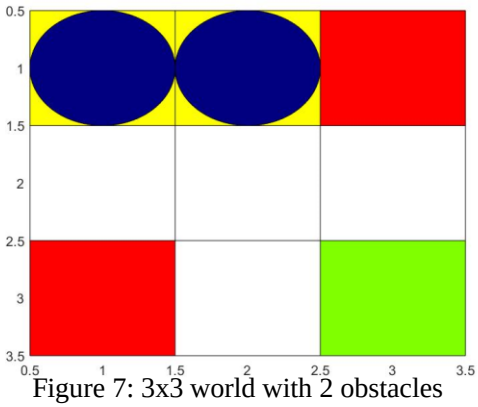


Figure 7: 3x3 world with 2 obstacles

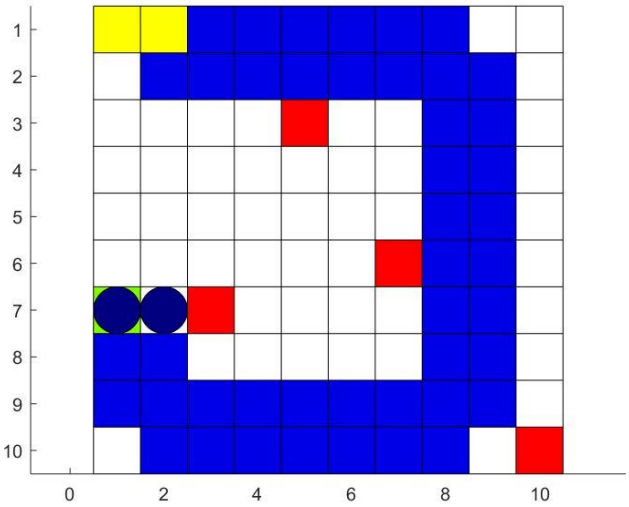


Figure 8: 10x10 world with 4 obstacles

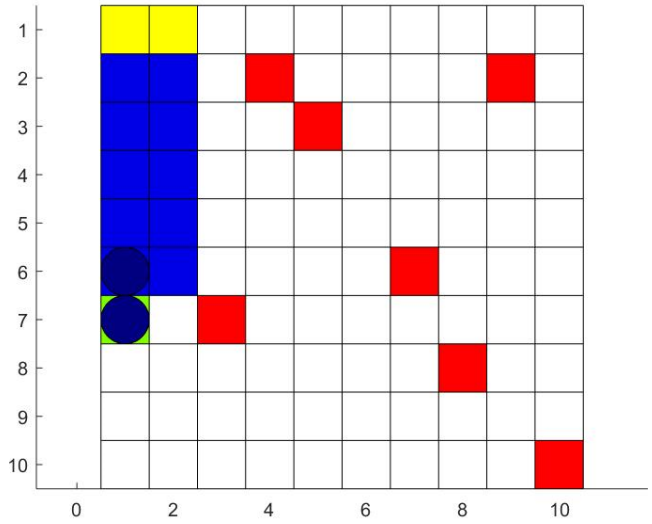


Figure 9: 10x10 world with 7 obstacles

B. Program Performance

The simulation results are produced in MATLAB R2017a. The program is run on an Intel(R) Core(TM) i7-7500U CPU at 2.70GHz. The runtime for the simple 3x3 world is 1.720 s. The 10x10 world with 4 obstacles took 47.892 s, and the 10x10 world with 7 obstacles took 7.763 s. These times include pauses for the animation. The actual time to generate the paths are 0.254 s, 1.474 s, and 0.782 s for the three worlds respectively. Computation time to compute all states, transitions, and automaton for the 50x50 world is 6.285 min. A search was not done in this size world.

C. Discussion

The strength of this method is that it produces a safe path regardless of the size of the world, the location of the obstacles, or the goal. The techniques used to produce this result are common in the verification community, but this application is unique to the FORMbot project and will aid in developing navigation algorithms for the modular robot. The current version of the program can be further optimized. An error message is not returned when a safe path does not exist to the goal, the program simply stops running. The search algorithm is not optimized, and there are ways the program can be further optimized to decrease the runtime for larger worlds. One possibility is to not generate a full transition map in the product automaton, but checking allowable transitions during the search.

Future work will include implementing the A* algorithm for an optimized search. Expanding the program from two robots to *n* robots will allow the user to better represent and make guarantees for a large number of FORMbots. Lastly, adding decentralized control to the algorithm will incorporate the communication aspect of the actual FORMbots.

6. References

[1] Liu, Jingyang, O'Brien, Kevin, and Ren, Jingyao. "Flexible self-Organizing Robotic Modules (FORMbots)." (2017), Final Report, ECE6970 Multi Agent Systems Class at Cornell University, NY.

[2] Gastin, Paul, and Oddoux, Denis. Fast LTL to Büchi Automata Translation. In Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01), Paris, France, July 2001, LNCS 2102, pages 53-65. Springer.

[3] http://www.lsv.fr/~gastin/ltl2ba/index.php

[4] https://www.youtube.com/watch?v=Whik3b-l-9s&feature=youtu.be