Topological Localization using FSM-Based Testing Sequences
– Technical Report –

Carlos Diego N. Damasceno
damascenodiego@usp.br
USP - ICMC / SSC5888 @ 2016

Contents

1 INTRODUCTION

This technical report presents the final project developed in the course *SSC5888 - Robôs Móveis Autônomos* at the Institute of Mathematical and Computer Sciences (ICMC) of the University of São Paulo (USP). This project was developed by Carlos Diego Nascimento Damasceno, PhD candidate at the ICMC-USP, and the course was taught by Fernando Osório. More information about the course can be found in http://wiki.icmc.usp.br/index.php/SSC-5888-2016(fosorio)

1.1 PROJECT DESCRIPTION

The project developed consists of an experiment investigating the usage of Finite State Machine (FSM) based testing methods [1] for robot localization (Figure 1) in topological maps.

Topological maps describe environments as a graph that connects specific locations in the world, represented by vertices, with edges expressing their accessibility [3].

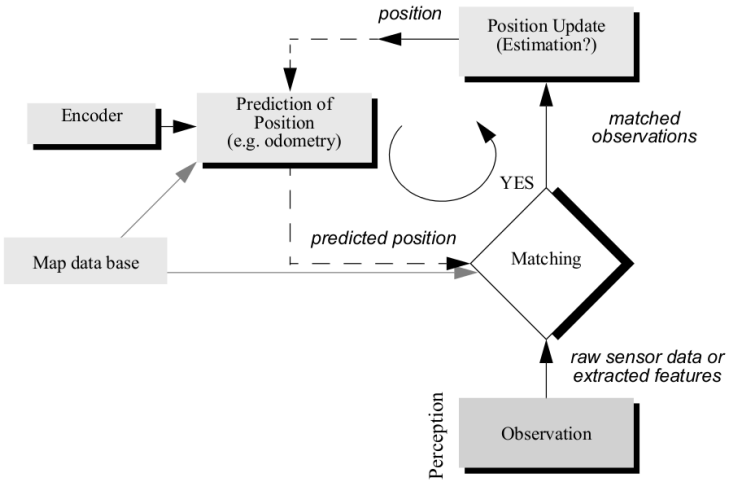


FIGURE 1: GENERAL SCHEMATIC FOR MOBILE ROBOT LOCALIZATION [6]

At the same time, in FSM-Based Testing, the current state uncertainty is a data structure used for solving one of the most basic problems in this domain which consists of identifying the current state of an FSM using input (and possibly output) symbols [1].

In this sense, the robot location problem in topological maps can be modeled as the problem of reducing the current state uncertainty until identifying an unique current state (or position) in this map. Thus, two common solutions for solving the current state uncertainty were applied on topological maps of random mazes specified as FSM models:

- *Synchronizing tree*, and
- *Homing tree*

A tool called *topologicalFsm* was implemented to support this investigation of this approach. The *topologicalFsm* tool was implemented using a software which randomly generates mazes to

simulate a robotic environment. Given a maze, the *topologicalFsm* tool can generate its respective topological map. Afterwards, it can generate synchronizing and homing trees to support the identification of the state actually occupied by a robot by means of reducing the current state uncertainty.

The topological maps are modelled as *mealy machines* and used for generating synchronizing and homing trees for solving the current state uncertainty problem. Predefined sequences of operations can be extracted from these trees to support the localization of a robot randomly placed on mazes.

2 MATERIALS AND METHODS

2.1 FSM-BASED TESTING

A mealy machine, or simply an FSM, is an hypothetical machine M composed by states and transitions [1]. In [4], an FSM is formally defined as a tuple $M =< S, s_0, I, O, D, \delta, \lambda >$ where

- S is a finite set of states,
- $s_0 \in S$ is the initial state,
- I is the finite set of input symbols,
- O is the finite set of output symbols,
- $D \subseteq S \times I$ is the specification domain,
- $\delta : D \rightarrow S$ is the transition function, and
- $\lambda : D \rightarrow O$ is the output function.

An example of FSM is presented in Figure 2.

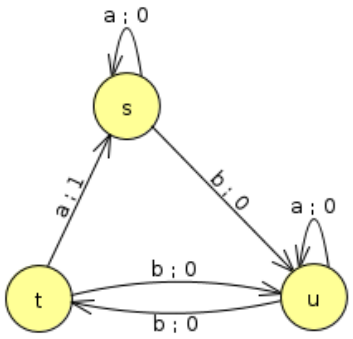


FIGURE 2: EXAMPLE OF FSM

An FSM always has has one single current state $s_i \in S$ which changes to $s_j \in S$ when a defined input symbol $x \in I$ is applied in the transition function, $\delta(s_i, x) = s_j$, and return an

output $y = \lambda(s, x)$ such that $y \in O$. An input x is defined for s if in state s there is a *defined transition* consuming input x. An FSM is said *complete* if all inputs are defined for all the states, otherwise it is named *partial*. A sequence $\alpha = x_1x_2...x_n \in I$ is an input sequence defined for state $s \in S$, if there exist states $s_1, s_2, ..., s_{n+1}$ such that $s = s_1$ and $\delta(s_i, x_i) = s_{i+1}$, for all $1 \leq i \leq n$. A sequence $\alpha = x_1x_2...x_n \in I$ is a transfer sequence from s to s_{n+1} if $\delta(s, \alpha) = s_{n+1}$. We say that s_{n+1} is reachable from s. When every state is reachable from s_0 the FSM is said *initially connected*. When every state is reachable from any of the states of a machine, the FSM is named *strongly connected*.

The concatenation of two sequences α and ω is denoted as $\alpha\omega$. One sequence α is a prefix of another sequence β, if $\beta = \alpha\omega$, for some given sequence ω. An empty sequence is denoted by ϵ such that for a state $s_i \in S$, $\delta(s_i, \epsilon) = s_i$, and $\lambda(s_i, \epsilon) = \epsilon$. Moreover, given a subset Q of the finite set S, expressed as $Q \subseteq S$, and an input x then $\delta(Q, x) = \{\delta(s, x) : s \in Q\}$, which defines the set of states reachable from all states $s \in Q$ by applying the input x.

2.2 HOMING AND SYNCHRONIZING SEQUENCES

In FSM-based testing there is a category of problems where the initial state s_0 is not known [1]. For this problem, there are two types of sequences that can be used as solution: *Synchronizing sequences* and *Homing sequences*. These sequences take an FSM to one unique state regardless its initial position. The homing sequences work based on the output symbols obtained given a sequence, while the synchronizing sequences are independent of outputs. These sequences are briefly described below.

2.2.1 SYCHRONIZING SEQUENCES

A *synchronizing sequence* is a sequence that takes an FSM to a unique final state regardless where it started. Thus, a sequence $x \in I*$ is synchronizing if $|\delta(S, x)| = 1$, which means that x belongs to the set of defined inputs $I*$ and the number of reached states after applying x is equals to 1. In words, regardless the initial state, there is only one state that can be reached after applying the input sequence x. Note that since synchronizing sequences only use the transition function δ, they are independend of outputs.

2.2.2 HOMING SEQUENCES

A *homing sequence* is a sequence such that the final state can be identified by looking at the output symbols obtained. Formally, it means that a sequence $x \in I*$ is homing if for every pair of states $s, t \in S$, if $\delta(s, x) \neq \delta(t, x) \Rightarrow \lambda(s, x) \neq \lambda(t, x)$. In words it means that given a sequence x, if any two different states of an FSM are taken then different output sequences are going to be obtained.

2.3 CURRENT STATE UNCERTAINTY

Given an FSM, if some input sequence is applied, some conclusion about it can be obtained. The *current state uncertainty* describes what is known about the current (or final) state after applying a given sequence $x \in I*$. Formally, the current state uncertainty with respect to a given mealy machine after applying a sequence $x \in I*$ is expressed as $\sigma(x) = \{\delta(B_i, x) : B_i \in \pi(x)\}$ where $\pi(x)$ describes a set of blocks $\{B_1, B_2, ..., B_r\}$ such that two states $s, t \in S$ are in a same block B_i if and only if $\lambda(s, x) = \lambda(t, x)$. In words, two states belong to one same block if a same output is obtained. An example is presented below:

Example: Considering the FSM in Figure 2, the current uncertainty $\sigma(\epsilon) = \{\{s, t, u\}\}$. By applying the input a, the current state uncertainty $\sigma(a) = \{\{s\}_1, \{s, u\}_0\}$, which means that if the output 1 is obtained when a input is applied, there is only one possible state that the FSM is currently on, that is s. Otherwise, if 0 is obtained, the FSM can be on s or u states. The current state uncertainty $\{s, u\}_0\}$ can still be decreased if a is applied again. Then, if 00 is obtained given the input aa, then the state s is reached at the end. If 01 is obtained given aa, then only state u can be reached.

2.3.1 SYNCHRONIZING TREE

To compute the shortest synchronizing sequence, the *synchronizing tree* can be used [1]. A synchronizing tree is a rooted tree where edges are labeled with input symbols and nodes with sets of states, satisfying the following conditions:

1. Each non-leaf has exactly $|I|$ children, and the edges leading to them are labeled with different input symbols.
2. Each node is labeled with the set of states reached by the input sequence occurring as edge labels on the path from the root to the node, $\delta(S, x)$.
3. A node is a leaf iff:
4. (a) either its label is a singleton set,

   (b) or it has the same label as a node of smaller depth in the tree.

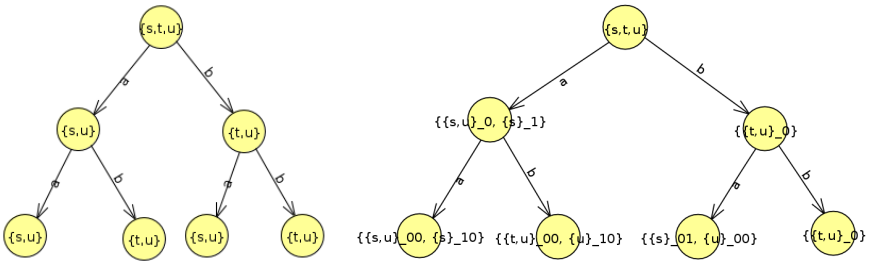
The synchronizing tree of the FSM depicted in Figure 2 is shown in Figure 3(a).

2.3.2 HOMING TREE

To compute the shortest homing sequence, the *homing tree* can be used [1]. A homing tree is a rooted tree where edges are labeled with input symbols and nodes with current state uncertainties, satisfying the following conditions:

1. Each non-leaf has exactly $|I|$ outgoing edges, labeled with different input symbols.
2. Each node is labeled with the current state uncertainty given the input sequence x occurring as edge labels on the path from the root to the node, $\sigma(x)$.
3. A node is a leaf iff:
4. (a) either its label is a singleton set,
   (b) or it has the same label (e.g., set of reached states) as a node of smaller depth in the tree.

The homing tree of the FSM depicted in Figure 2 is shown in Figure 3(b).

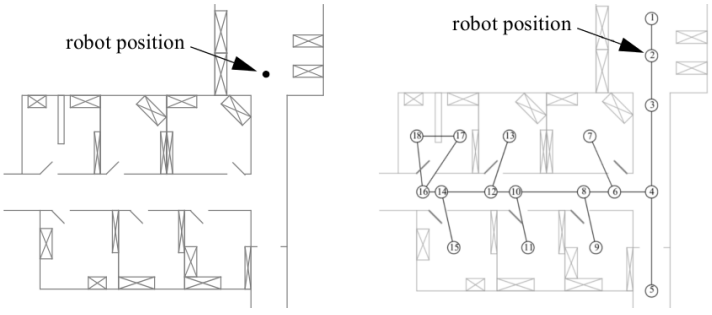


(a) Synchronizing tree (b) Homing tree

FIGURE 3: SYNCHRONIZING AND HOMING TREES

2.4 TOPOLOGICAL MAPS

Localization is the problem of using a map to interpret sensor data to determine the configuration (i.e., position) of the robot [2]. In this experiment, we consider the problem of locating a robot placed in a maze using approaches to solve the *current state uncertainty problem*, such as synchronizing and homing trees, based on topological maps expressed as mealy machines. Figures 4(a) and 4(b) respectively depict the real map of an environment that a robot can move and the topological representation of this same map.



(a) Real Map (b) Topological Map

FIGURE 4: TOPOLOGICAL REPRESENTATION OF A REAL MAP

A topological map consists of an abstraction of an environment where "something distinct", such as intersections and T-junctions, is represented as **v**ertices (nodes) and adjacency relationships between nodes are depicted as **e**dges. These structures are expressed as a graph $G = (V, E)$ which can also have some embedded information or special labels in edges or vertices, e.g., the edges may have length [3].

2.4.1 TOPOLOGICAL MAPS AS MEALY MACHINE

In this project, the topological maps of mazes are modelled as mealy machines. Figure 5 shows an example of 2×2 maze where a robot, represented as a blue dot, is placed at the $(1, 1)$ position.

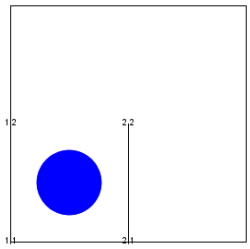


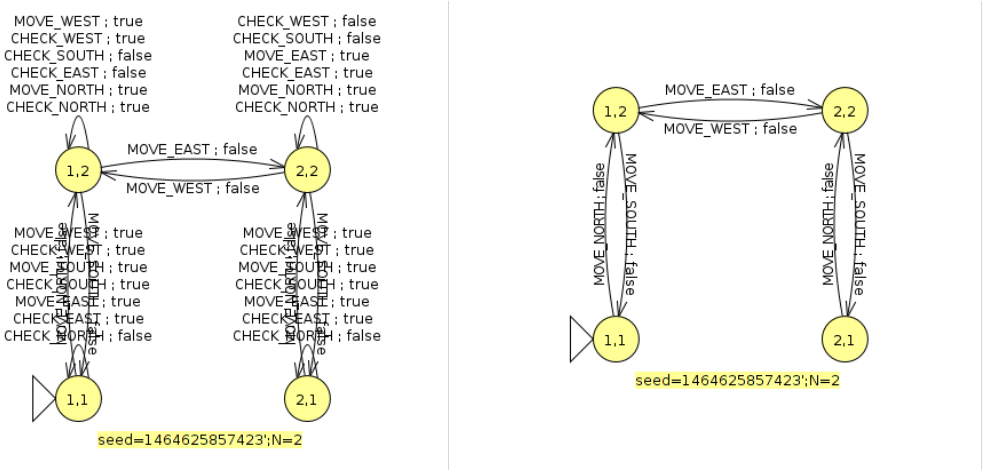
FIGURE 5: EXAMPLE OF MAZE

The topological maps of mazes are modelled as a mealy machine $M =< S, I, O, \delta, \lambda >$. The finite set of states S describes each position at the maze, e.g., there is a state to position $(1, 1)$. The input domain I describes the operations that the robot can perform. The operations considered are listed bellow:

- `CHECK_NORTH`: Verifies if there is an obstacle blocking **north** direction.
- `CHECK_EAST`: Verifies if there is an obstacle blocking **east** direction.
- `CHECK_SOUTH`: Verifies if there is an obstacle blocking **south** direction.
- `CHECK_WEST`: Verifies if there is an obstacle blocking **west** direction.
- `MOVE_NORTH`: If allowed, move the robot to *north* direction.
- `MOVE_EAST`: If allowed, move the robot to *east* direction.
- `MOVE_SOUTH`: If allowed, move the robot to *south* direction.
- `MOVE_WEST`: If allowed, move the robot to *west* direction.

The output domain O contains the symbols `{true,false}`. The output function λ work as follows: `CHECK_XXX` operations return `false` if there is no obstacle blocking the direction `XXX`, otherwise the `true` symbol is returned. The state transition function δ essentially depicts the

robot moving from a given state to a neighbor state where one of the coordinates of the position is incremented. When a `MOVE_XXX` operation is requested, the state transition function returns `false` if a move to the given `XXX` direction allowed. Transitions expressing not allowed movements are depictedc as *self-loops* and return `true`. `CHECK_XXX` operations are also self-loops which return `true` if there is an obstacle, otherwise it returns `false`. Figures 6(a) and 6(b) respectively show an example of topological map with and without self-loop transitions to keep the figure uncluttered.



(a) Topological Map (b) Topological Map without self-loops

FIGURE 6: TOPOLOGICAL REPRESENTATION OF A 2×2 MAZE

The topological maps presented in Figures 6(a) and 6(b) depict the 2×2 maze presented in Figure 5. The map has four states representing each of the positions at the maze. Since the robot is placed at the position $(1, 1)$, then the initial state is marked as the state with same label.

2.4.2 SYNCHRONIZING AND HOMING TREES OF TOPOLOGICAL MAPS

Given a topological map like the presented in Figure 6(a), synchronizing and homing trees can be generated to support the identification of the current state (in words, the location) of a robot in a topological map. Figures 7 and 8 respectively show a part of a synchronizing tree and a homing tree obtained from the topological map presented in Figure 6(a).

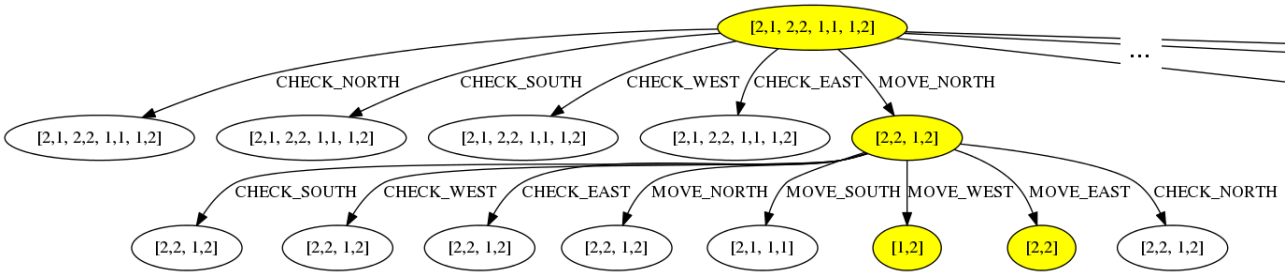


FIGURE 7: EXAMPLE OF SYNCHRONIZING TREE OF A TOPOLOGICAL MAP

In Figure 7, the nodes of the synchronizing tree colored in yellow highlight two sequences of operations that when performed take a robot for two different states, regardless the initial state. The root of the synchronizing tree shows that the current state uncertainty includes all four states as possibly occupied by the robot. After applying the command `MOVE_NORTH`, the robot uncertainty is reduced to two states, $\{(2,2),(1,2)\}$. At this stage, there are two operations that guarantee the localization of the robot: `MOVE_WEST` or `MOVE_EAST`. Both operations reduce the uncertainty to one single possibly current state, called *singleton*. These states are respectively $(1,2)$ and $(2,2)$.

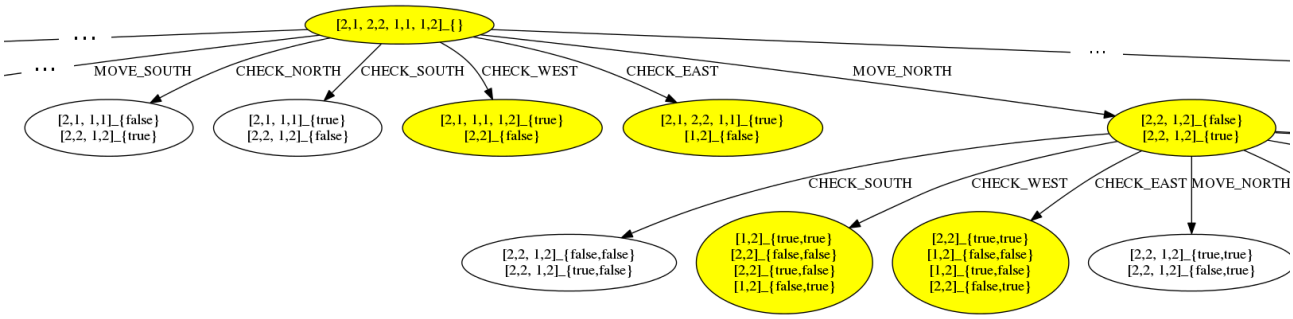


FIGURE 8: EXAMPLE OF HOMING TREE OF A TOPOLOGICAL MAP

In Figure 8, the nodes of the homing tree colored in yellow highlight four sequences of operations that when performed can identify the final state that a robot ocuppies. The root of the synchronizing tree shows that the current state uncertainty includes all four states as possibly occupied by the robot given the empty sequence ϵ. For example, if after applying the command `CHECK_EAST`, the a `false` output symbol is obtained, there will be a single state/position that the robot can be occuppying: $(1,2)$. The command `MOVE_NORTH`, on the other hand does not guarantee the a singleton, but reduces the current state uncertainty to two sets with two states $\{(2,2),(1,2)\}$, both if `true` or `false` are obtained. The inputs `CHECK_EAST` and `CHECK_WEST`, at this point, can distinguish these two same states given one of the sequences of outputs shown.

3 EXPERIMENT

An experiment was performed for measuring the length of sequences of operations necessary for locating a simulated robot only using the synchronizing and homing trees.

The experiment essentially was designed with the following four steps:

(i) Generation of random mazes

(ii) Topological map extraction

(iii) Generation of the *Synchronizing* and *Homing* trees

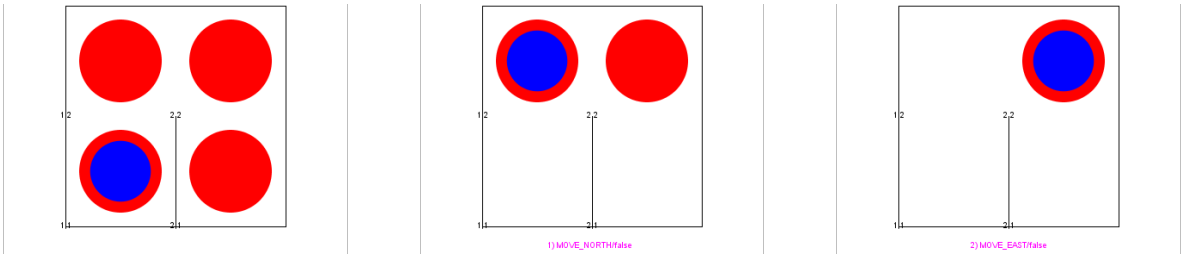
(iv) Verification of the length of the sequences for reaching singleton nodes

At the step (i), random mazes were generated using three configurations for the dimensions: $2 \times 2, 3 \times 3$, and 4×4. Afterwards, in step (ii), the random mazes were traversed using a depth-first search (DFS) algorithm and each of the coordinates at the map were analyzed in order to generate and connect nodes of the topological map. The synchronizing and homing trees of each topological map were generated and four values were extracted from them: (i) The length of the *shortest sequence* which *can reach* one singleton; (ii) The length of the *shortest sequence* which *will reach* one singleton; (iii) The length of the *longest sequence* which *can reach* one singleton; and (iv) The length of the *longest sequence* which *will reach* one singleton.

The difference between a sequence that *can reach* to another that *will reach* a singleton is the following: In homing trees, there may exist some states where only one output guarantees that an unique state is reached, thus only this specific output guarantees that singleton. For example, in Figure 8, a singleton *can be reached* if a `CHECK_EAST` operation is performed and `false` is returned. If a `true` symbol is obtained as output, only one of four coordinates is eliminated. On the other hand, if the sequence of operations `MOVE_NORTH`,`CHECK_EAST` is performed and one of the outputs presented in the state is obtained, there is the guarantee that an unique position ($(1, 2)$ or $(2, 2)$) is going to be reached. If synchronizing trees are used, there is no such distinction since output symbols are not considered. The presented approach was implemented in a tool named *topologicalFsm*, presented in the next section.

3.1 TOPOLOGICALFSM TOOL

The *topologicalFsm* tool was implemented using the java programming language and its source code can be found at https://github.com/damascenodiego/topologicalFsm. The tool was designed based on the maze generator introduced by [5] which can generate random mazes. A value n defines the $n \times n$ area of the random mazes. Topological maps are also generated from these maps and used to obtain synchronizing or homing trees. A simulated robot can be placed in any valid coordinate for investigating the localization problem using the generated tree. A sequence of snapshots showing the current state uncertainty throughout the execution of the sequence `MOVE_NORTH`,`MOVE_EAST` is presented in Figures 9(a), 9(b) and 9(c).



(a) Any position is possible (b) Uncertainty after `MOVE_NORTH` (c) `MOVE_EAST` and singleton

FIGURE 9: EXAMPLE OF ROBOT LOCALIZATION IN A 4×4 MAZE

At first, any of the four states are possibly occupied by the robot. After the `MOVE_NORTH` operation, the uncertainty is reduced from four to two. At the end, the `MOVE_EAST` operation reduces the uncertainty to a singleton.

3.1.1 COMMAND LINE INTERFACE

The operation of the *topologicalFsm* tool is made via command line interface (CLI). Essentially, there are four main parameters:

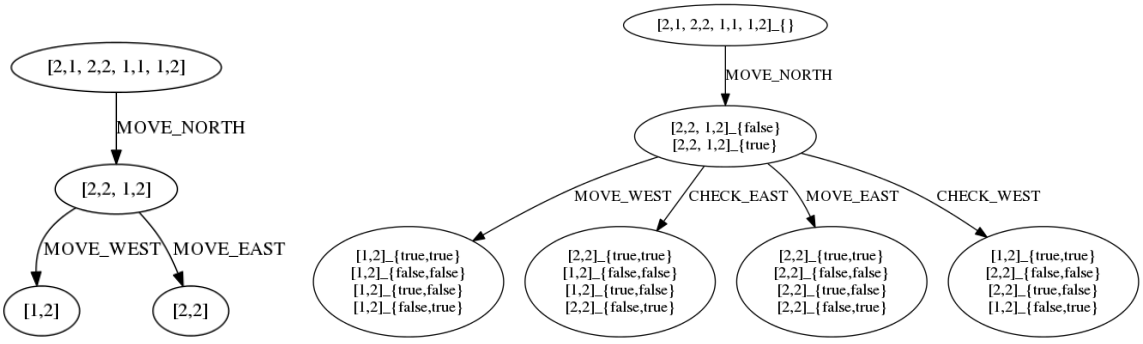
`-n`: It defines the n parameter used to define the dimensions of the $n \times n$ maze.

`-synchronizing`: Generates a synchronizing tree from a given maze.

`-homing`: Generates the homing tree from a given maze.

`-save`: It saves the topological map[1] the maze[2], and the tree [3] generated.

By default, the tool generates synchronizing and homing trees with leaf nodes only containing singleton sets was also defined. Thus, any path from the root to leaf nodes guarantees that a singleton is reached. Figures 10(a) and 10(b) respectively show examples of synchronizing and homing trees with leaf nodes only containing singletons. To generate the complete tree the parameter `-complete` must be used. The `-window` parameter shows the visual presentation of the maze. The `-save` parameter without `-complete` generates a sequence of frames showing the current state uncertainty until reaching a singleton. The `-pos` parameter setup the position of the robot in the map (e.g., `-pos 1,1`). The `-seed` and `-seedp` parameters can be used to control the random generation of mazes and the sequences of operations, respectively.



(a) Synchronizing tree only with singletons

(b) Homing tree only with singletons

FIGURE 10: SYNCHRONIZING AND HOMING TREES ONLY WITH SINGLETON LEAVES

[1]Topological maps are saved in *.jff* format. See JFLAP: `http://www.jflap.org/`

[2]Mazes are saved as *.png* and text file

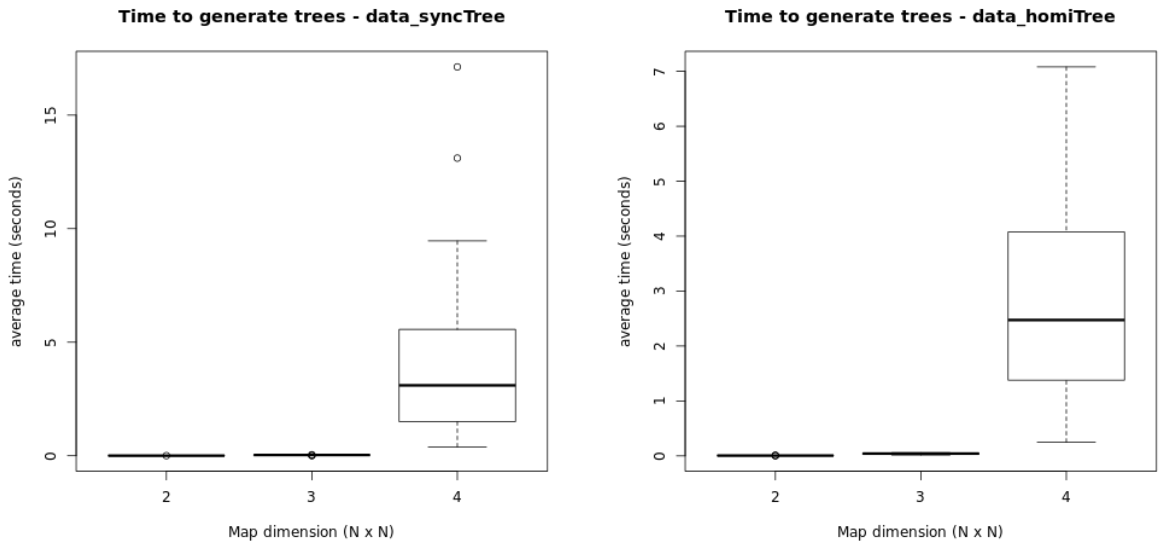
[3]Homing and synchronizing trees are saved as both *.jff* and *.dot* files.

4 RESULTS AND DISCUSSION

In this section the results of the experiment are shown. The duration of the generation of the homing and synchronizing trees and the average length of the shortest and the longest paths that can and will reach singletons are presented.

4.1 TIME TO GENERATE TREES

Figure 11(a) and 11(b) show the time necessary to generate the synchronizing and homing trees, respectively. The boxplots show that the time to generate homing trees was shorter compared to the time to generate synchronizing trees. Since homing sequences are usually shorter, homing trees also tend to often present a smaller height compared to synchronizing trees. Thus, it can be expected that homing trees may need a shorter time to be generated compared to synchronizing trees.



(a) Synchronizing trees

(b) Homing trees

FIGURE 11: AVERAGE TIME TO GENERATE TREES (SECONDS)

4.2 NUMBER OF OPERATIONS FOR LOCALIZATION

Tables 1 and 2 respectively show the average length of paths to singletons on synchronizing and homing trees. This average length essentially depicts the number of operations necessary for performing the location of a robot in a maze.

The column named *"Can reach a singleton"* presents the average length of the shortest and longest paths which reach a node with *at least one singleton*. In words, paths that can reach a singleton point out a sequence of operations that, in case of conformance between obtained outputs and these specified on the singleton nodes, describe an single known position in the

maze. On the other hand, the column named *"Will reach a singleton"* shows the average length of the shortest and the longest paths *able to reach a singleton*. Thus, these other paths regardless the obtained ouputs, the position of the robot will become known.

TABLE 1: PATHS TO SINGLETONS ON SYNCHRONIZING TREES (AVG. LENGTH)

N	Can reach a singleton		Will reach a singleton	
	Shortest	Longest	Shortest	Longest
2	2	2	2	2
3	5.45	8.6	5.45	8.6
4	10.18	15.75	10.18	15.75

In homing trees, the average length of the paths to singletons is significantly reduced. The shortest paths which can reach singletons decreased around three times. For mazes with 4×4, there was a significant reduction on the shortest paths to singletons. It is important to highlight that the two columns *Can reach a singleton* and *Will reach a singleton* do not change on *synchronizing trees* since reaching singletons strictly depend on the input symbols. On the other hand, there may exist input sequences that can reach tree nodes where some specific output sequences do not point to singleton sets (e.g., in Figure 8, the `CHECK_WEST` operation, when applied at the root node, returns a `false` output symbol and a singleton can be reached. However, in case of `true`, no singleton can be reached).

TABLE 2: PATHS TO SINGLETONS ON HOMING TREES (AVG. LENGTH)

N	Can reach a singleton		Will reach a singleton	
	Shortest	Longest	Shortest	Longest
2	1	2	2	2
3	2	7.5	4	7.2
4	2.4	19.1	6.45	18.8

5 FINAL REMARKS

The data structure used to specify the current state uncertainty can be helpful for localization in topological maps. Synchronizing and homing trees generated from topological maps specified as mealy machines enabled to identify paths on the maze with half to twice the dimension of the mazes considered in this experiment. Homing trees also enabled to identify paths shorter than using synchronizing trees. As extensions for this experiment, optimizations on the topological mapping can be used (e.g., only corner nodes are modelled or `CHECK` operation evaluating all directions at once – 360^o sensors) can be considered. All the data generated, source-code and scripts are available at `https://github.com/damascenodiego/topologicalFsm`.

References

[1] Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner. *Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[2] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, June 2005.

[3] Gregory Dudek and Michael Jenkin. *Computational Principles of Mobile Robotics*. Cambridge University Press, New York, NY, USA, 2000.

[4] Arthur Gill. *Introduction to the Theory of Finite State Machines*. McGraw-Hill, New York, 1962.

[5] Robert Sedgewick and Kevin Wayne. *Algorithms, 4th Edition*. Addison-Wesley, 2011.

[6] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Company, Scituate, MA, USA, 2004.