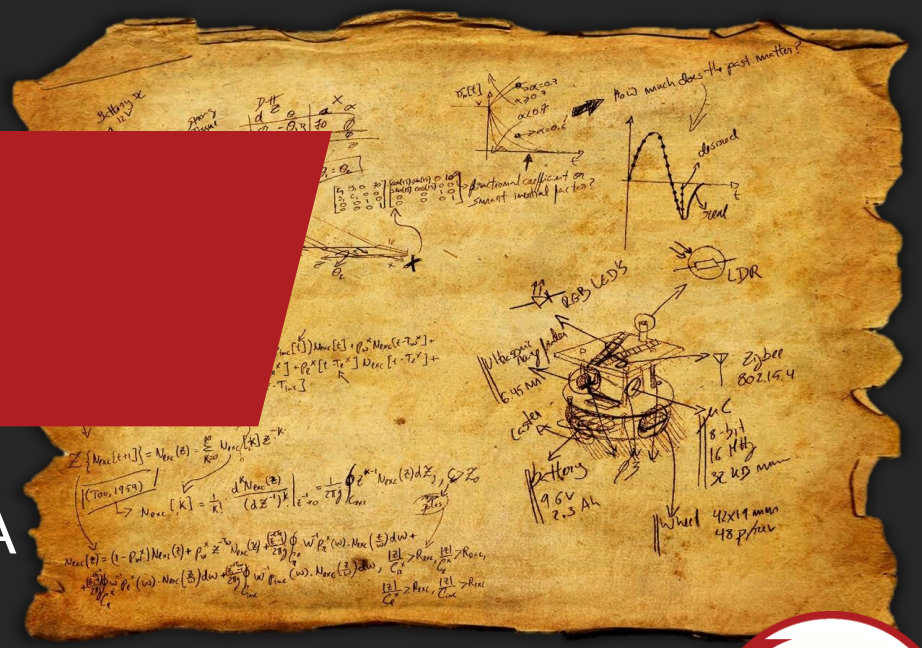
CRAFT #7

Artificial Intelligence



A Bio-inspired perspective with FSM and WPA

Micael S. Couceiro micael@ingeniarius.pt

Beril Yalcinkaya beril@ingeniarius.pt

19/08/2024

OBJECTIVES

- Introduction to Artificial Intelligence
- Finite-state machines
- Robot-Ant: BASIC Maze solver
- Wavefront Planning Algorithm
- Robot-Ant: PRO Maze solver
- **Prepare for competition:** consolidate concepts learned over all crafts and test the mobile robotic platform under the maze scenario

INTRODUCTION

"Artificial intelligence is that field of computer usage which attempts to construct computational mechanisms for activities that are considered to require intelligence when performed by humans." – Derek Partridge

ARTIFICIAL INTELLIGENCE

- Turn computers into more intelligent machines
- Better understand human intelligence
- Four questions:
  - Do we want to reason?
  - Or act?
  - Should we use humans as models to build our intelligent systems?
  - Do we want to establish new rules as a new rational entity should think/act/behave?

ARTIFICIAL INTELLIGENCE

Approaches

- Philosophical
  - Definition of intelligence
- Cognitive
  - Models of human abilities (to better understand humans)
- Engineering
  - Behaviours considered “intelligent” for humans (play chess, recognize a pen, etc.)
  - But why humans?
- Simply: to do the right thing!

ARTIFICIAL INTELLIGENCE

Officially, AI started immediately after WWII, with a paper entitled as “Computing Machinery and Intelligence”, from the English mathematician Alan Turing*, in 1956

- Yet, the initial enthusiasm was between 1952 and 1969:
  - Allegations: Computers can do X
  - General Problem Solver: Newel & Simon developed an approach to solve puzzles in a similar way than humans do
  - …
- Reality check between 1966 and 1974
  - Approaches were not scalable
  - Intractability

*Did you see the movie “*The Imitation Game*”? Pretty interesting if you consider that AI, in a way, had a crucial role in the upshot of WWII.

ARTIFICIAL INTELLIGENCE

Alan Turing, 1950's

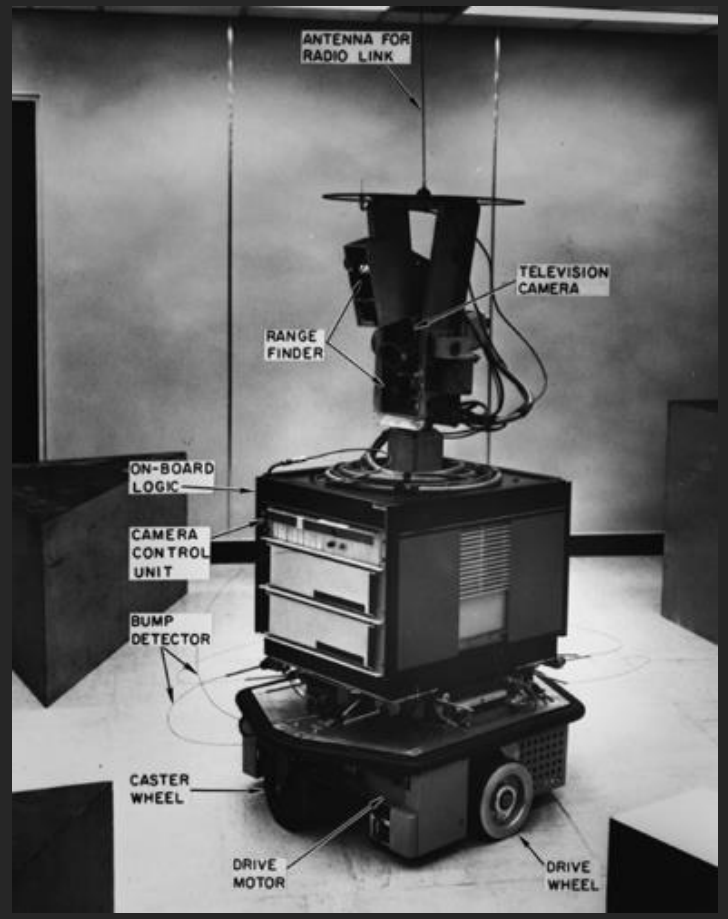
- Turing Machine and the Turing Test

Simon & Newell , 1950's

- General Problem Solver – GPS

Shakey, 1966-72

- First autonomous robot



ARTIFICIAL INTELLIGENCE

Turing Test (1950)*

- Try to assess if whether or not a machine can really **think**
- Assess the ability of a machine to mimic a human being, i.e., can machines communicate in such a natural, indistinguishable and contextualized way as humans do?
- Based in the premise that we can recognize intelligence when we see it, instead of considering some consensual definition

NOTE: The Turing Test requires physical interaction, perception and action.

*Did you see the movie "*Ex Machina*"?

ARTIFICIAL INTELLIGENCE

Can a computer be successful in the Turing Test?

- **Natural language processing**: communicate with the evaluator.
- **Knowledge representation**: store and access information provided before or during the interview.
- **Autonomous reasoning**: use stored information to reply to queries and withdraw new conclusions.
- **Learning**: adapt to new circumstances and detect and extrapolate patterns.
- **Vision**: recognize actions of the evaluator and the several objects presented by him/her.
- **Motor control**: interact with objects as per requested by the evaluator.
- **Other senses**: hear, smell and touch.

ARTIFICIAL INTELLIGENCE

Can a computer be successful in the Turing Test?

ARTIFICIAL INTELLIGENCE

Can ChatGPT be successful in the Turing Test?

- **Architecture**: Uses the Generative Pre-trained Transformer (GPT) architecture
- **Training**:
  - **Two steps**: Pre-training and fine-tuning
  - **Pre-training**: Model learns language by predicting the next word in a sentence
  - **Fine-tuning**: Trained on narrower datasets with specific tasks using human feedback
- **Input Processing**:
  - Text is converted into “tokens” (chunks of words or characters)
  - Tokens are processed by layers of “transformer” blocks
- **Output Generation**:
  - Model estimates the probability of each word/token to follow the given input
  - Selects the most likely next token, word by word, to form a coherent response

ARTIFICIAL INTELLIGENCE

Can ChatGPT be successful in the Turing Test?

- **Strengths**:
  - Handles a vast array of topics
  - Can generate creative and coherent text over long passages

- **Limitations**:
  - Can produce incorrect or nonsensical answers
  - Sensitive to input phrasing
  - Does not have the ability to learn in real-time or remember user-specific data

ARTIFICIAL INTELLIGENCE

Can ChatGPT be successful in the Turing Test?

- The Paper entitled entitled as "*How and Why ChatGPT Failed The Turing Test*", by Robert Hanna (2023), discusses the recent controversy surrounding ChatGPT.
- The paper examines the question of whether ChatGPT can pass the Turing Test, a test developed by Alan Turing to measure whether AI can think and respond like a human being.
- The author argues that ChatGPT failed the Turing Test because it likes to specify that it is a language model in conversation, which is a massive giveaway that the machine is not human.
- However, some sources suggest that this characteristic could be bypassed with a bit of hard wiring on the back end of the model
- Nonetheless, ChatGPT could still not be employed to solve the full version of the Turing Test, wherein perceiving and acting within the environment is key

Hanna, R. (2023). How and Why ChatGPT Failed The Turing Test. *Unpublished MS. Available online at URL: https://www.academia.edu/94870578/How_and_Why_ChatGPT_Failed_The_Turing_Test_January_2023_version_*

ARTIFICIAL INTELLIGENCE

Can ChatGPT be successful in the Turing Test?

But what does ChatGPT “thinks” about this question?

- **Depth and Duration**: While ChatGPT might fool a user in short conversations, longer and more in-depth interactions could reveal its non-human nature due to inconsistencies, lack of genuine understanding, or inability to provide certain kinds of context.
- **Subjectivity**: The Turing Test is somewhat subjective. Different judges or contexts may lead to different conclusions.
- **Purpose**: ChatGPT is designed for a broad range of tasks, not specifically to pass the Turing Test. While it can generate coherent and contextually relevant responses, it does not “understand” in the same way humans do.
- **Errors and Limitations**: ChatGPT can occasionally produce incorrect, nonsensical, or verbose answers, which can give away its machine nature.

ARTIFICIAL INTELLIGENCE

Can ChatGPT be successful in the Turing Test?

ARTIFICIAL INTELLIGENCE

Typical applications:

- Finances
- Medicine
- Games
- Natural language processing
- Planning
- Computer vision
- Intelligent interfaces
- Web
- Big Data
- Robotics

But which AI approaches??

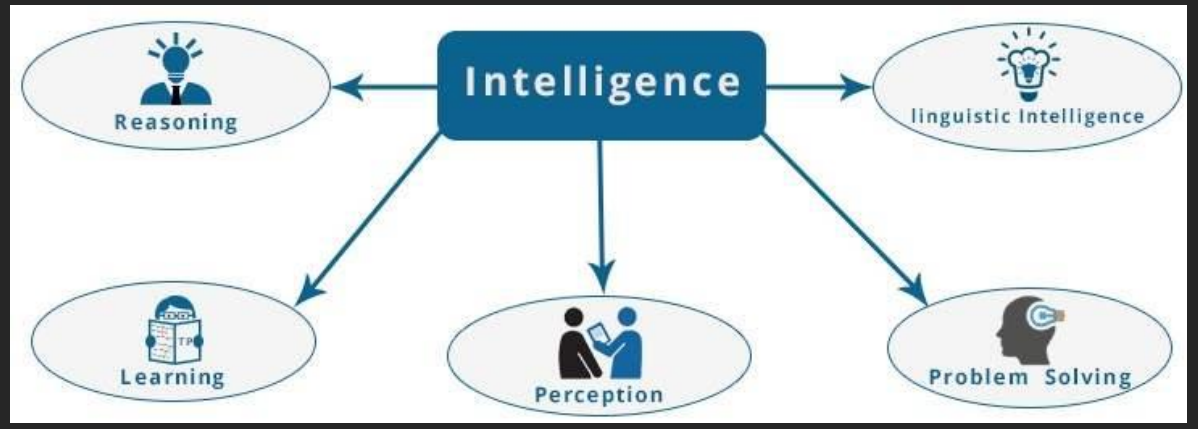
ARTIFICIAL INTELLIGENCE

AI approaches

- Automata theory
- Logics (*e.g.*, fuzzy logic) to represente knowledge and reasoning
- Autonomous agents and multi-agent systems
- Biomimetics and swarm intelligence to solve problems (e.g., optimization problems)
- Classification methods for learning and pattern recognition

ARTIFICIAL INTELLIGENCE

Artificial intelligence, within the context of robotics, often known as **embodied AI**, can generally be defined as a way to endow robots with the capability to perform functions, such as learning, decision-making, or other intelligent behaviours

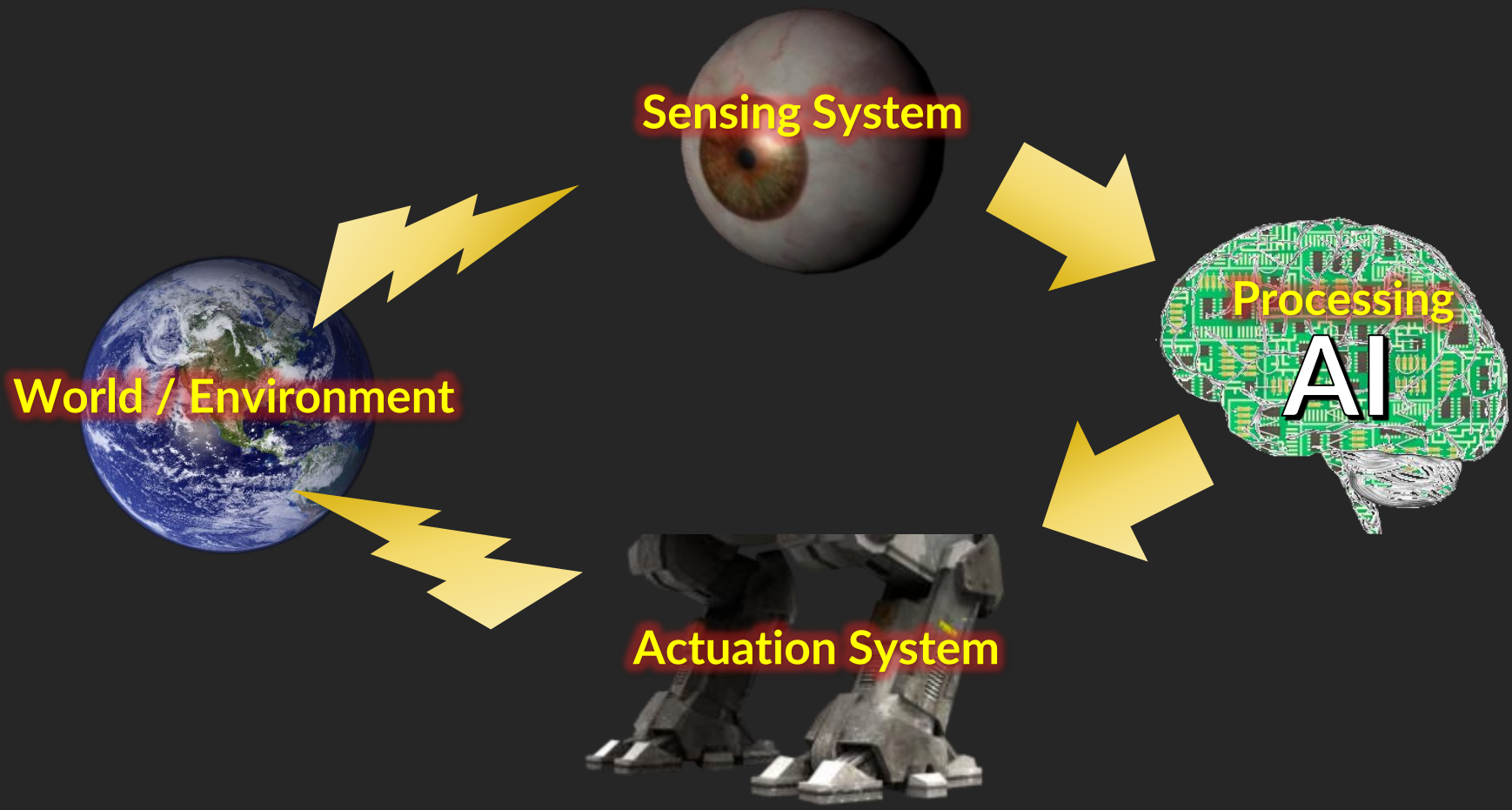


Brady, M. (1985). Artificial intelligence and robotics. Artificial Intelligence, 26(1), 79-121.

Nolfi, S., & Floreano, D. (2000). Evolutionary robotics.

Aylett, R. (2002). Robots: Bringing intelligent machines to life. Barron's Educational Series Inc.

AI & ROBOTICS


Sensing System
Processing
AI
World / Environment
Actuation System

BIO-INSPIRED SYSTEMS

All biological systems result from an evolutionary process

Artificial evolution encompasses a wide range of algorithms inspired in the four principles of natural evolution:

- Survival of the population
- Diversity of mechanisms and adaptability
- Selection mechanisms
- Genetic inheritance

Floreano, D., & Mattiussi, C. (2008). Bio-inspired artificial intelligence: theories, methods, and technologies. MIT press.

BIO-INSPIRED SYSTEMS

Why?

- Find solutions to many optimization problems
- Improve the shape of objects
- Evolve computational routines
- Optimize electrical circuits
- Pattern recognition
- Explore many other fields that are normally associated to human criteria

Floreano, D., & Mattiussi, C. (2008). Bio-inspired artificial intelligence: theories, methods, and technologies. MIT press.

BIO-INSPIRED SYSTEMS

Single-robot

VS

Multi-robot system

BIO-INSPIRED SYSTEMS

BIO-INSPIRED SYSTEMS

SINGLE-ROBOT

The good

- Simpler to deploy and run
- Debugging is an acceptably easy task
- A single robot can perform more complex tasks

The bad

- Hard to implement multi-tasking
- Higher hardware and software complexity
- More expensive than each robot belonging to a multi-robot system
- If the robot fails, there is no turning back: stop everything!

SINGLE-ROBOT

Typical approaches

- Finite-state machine
- Probabilistic finite-state machine
- Fuzzy Logics
- Bayesian Networks

Typical applications

- Surveillance
- Manufacturing
- Transportation
- Mapping
- Indoor & Outdoor maintenance
- Companionship

MULTI-ROBOT SYSTEMS

The good

- Reduced cost for each unit
- Decomposition of tasks
- Heterogeneity may lead to multi-tasking
- Space distribution
- System robustness: a single robot failing may not jeopardize the system

The bad

- Hard to deploy and run in a coordinated manner
- Debugging may be overwhelming

MULTI-ROBOT SYSTEMS

Typical approaches

- Foraging
- Clustering
- Sorting
- Building
- Flocking
- Finite-state machine

Typical applications

- Distributed sensing tasks
- Patrolling
- Search-and-Rescue
- Mining tasks
- Cooperative Mapping
- Mobile Sensor Networks
- Foraging & Optimization

FINITE-STATE MACHINES

FINITE-STATE MACHINE

What is meant by “Finite”?

- Finite means countable.

What is Finite-State Machine?

- A machine that consists of a fixed set of possible states with a set of allowable inputs that change the state and a set of possible outputs.

FINITE-STATE MACHINE

A FSM consists of

- States
- Inputs
- Outputs

The number of states is fixed; when an input is executed, the state is changed and an output is possibly produced.

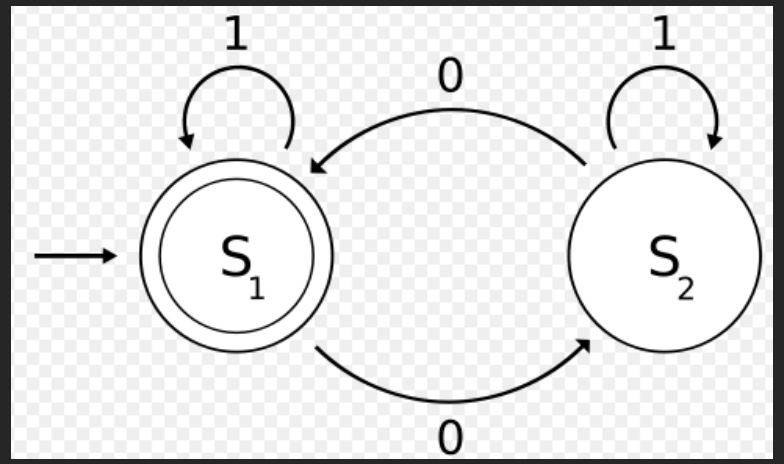
FINITE-STATE MACHINE

Finite-State Machine (FSM)

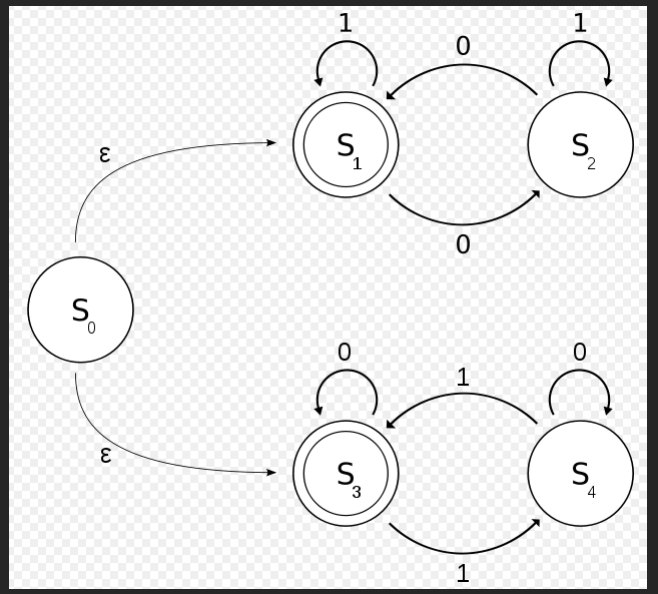
- FSM is a generic model used for sequential design
- A state machine has some number of states, and transitions between those states
- Transitions occur because of inputs
- State machines may produce output (often as a result of transitions)

FINITE-STATE MACHINE

- Deterministic and Non-deterministic
- Deterministic means there is only one outcome for each state transition

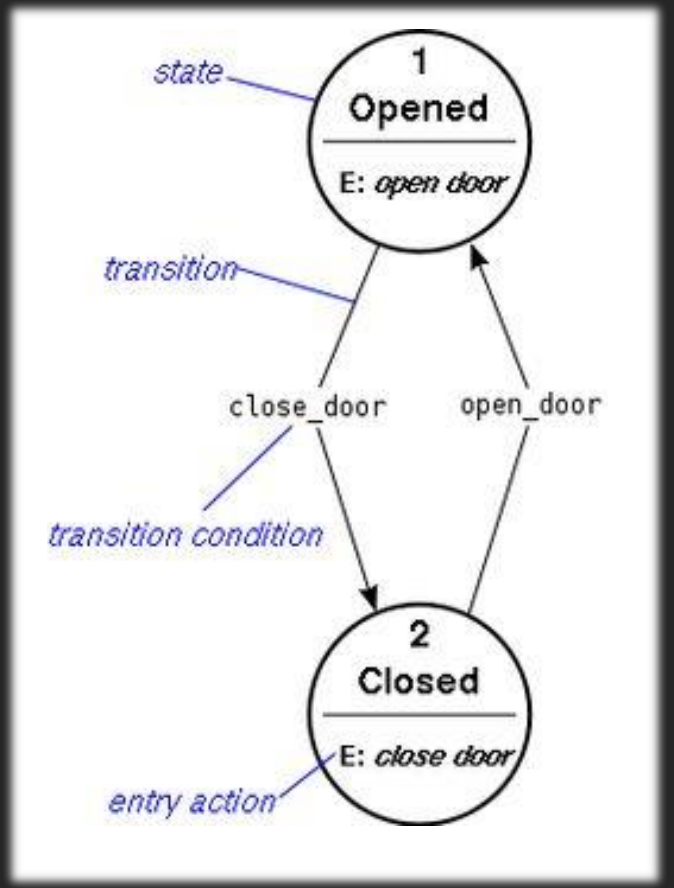


Deterministic



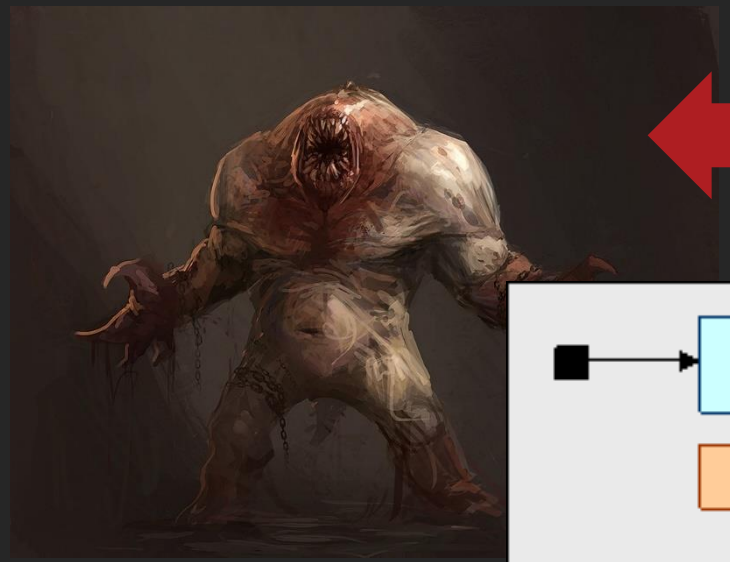
Non-deterministic

FINITE-STATE MACHINE

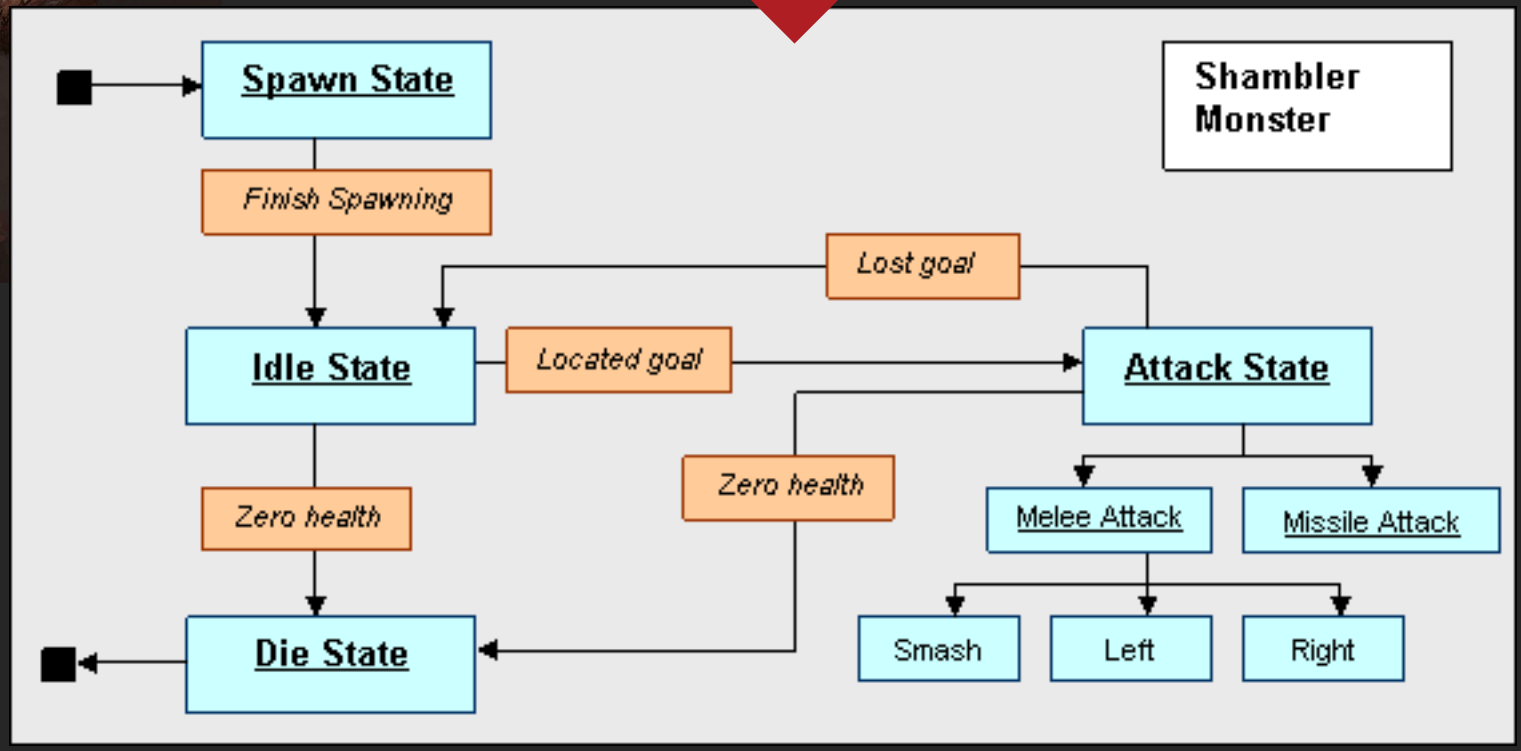


FINITE-STATE MACHINE

FSM ~ *Game AI*



The state-machine that controls Quake's ***Shambler*** monsters…



FINITE-STATE MACHINE

FSM ~ *Game AI*

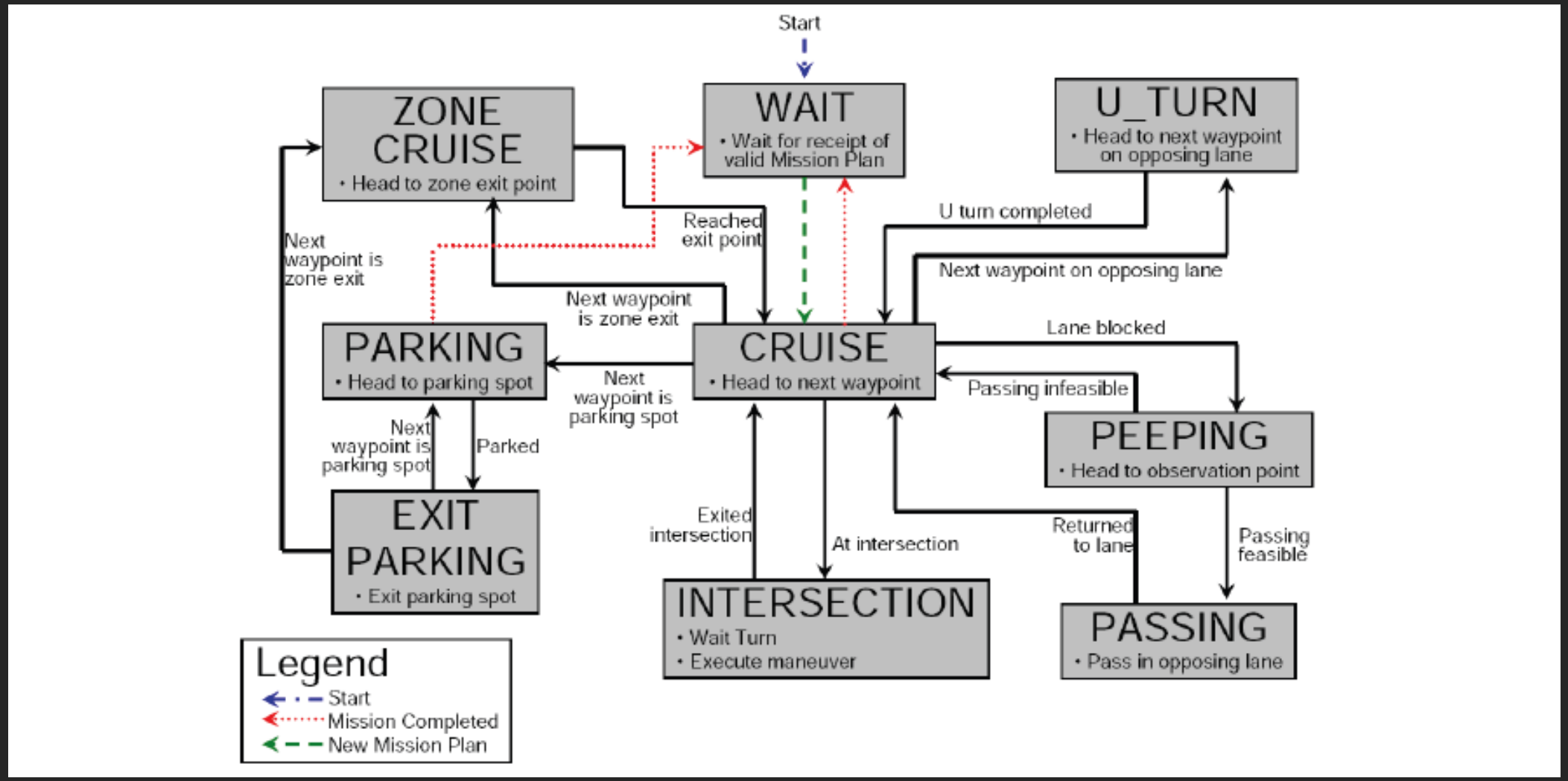
FINITE-STATE MACHINE

So, are FSM really AI? Why are we teaching them?

- **Simple answer**: at its core, AI refers to machines or software that can perform tasks that would typically require human intelligence - under this broad definition, any system that automates decision-making based on inputs, such as FSMs, can be considered a basic form of AI.
- **Practical answer**: designing extremely complex state machines required to fly drones, drive self-driving cars or operate warehouse robots is still one of the most time-consuming and difficult tasks faced by companies.

FINITE-STATE MACHINE

FSM ~ *ITS*

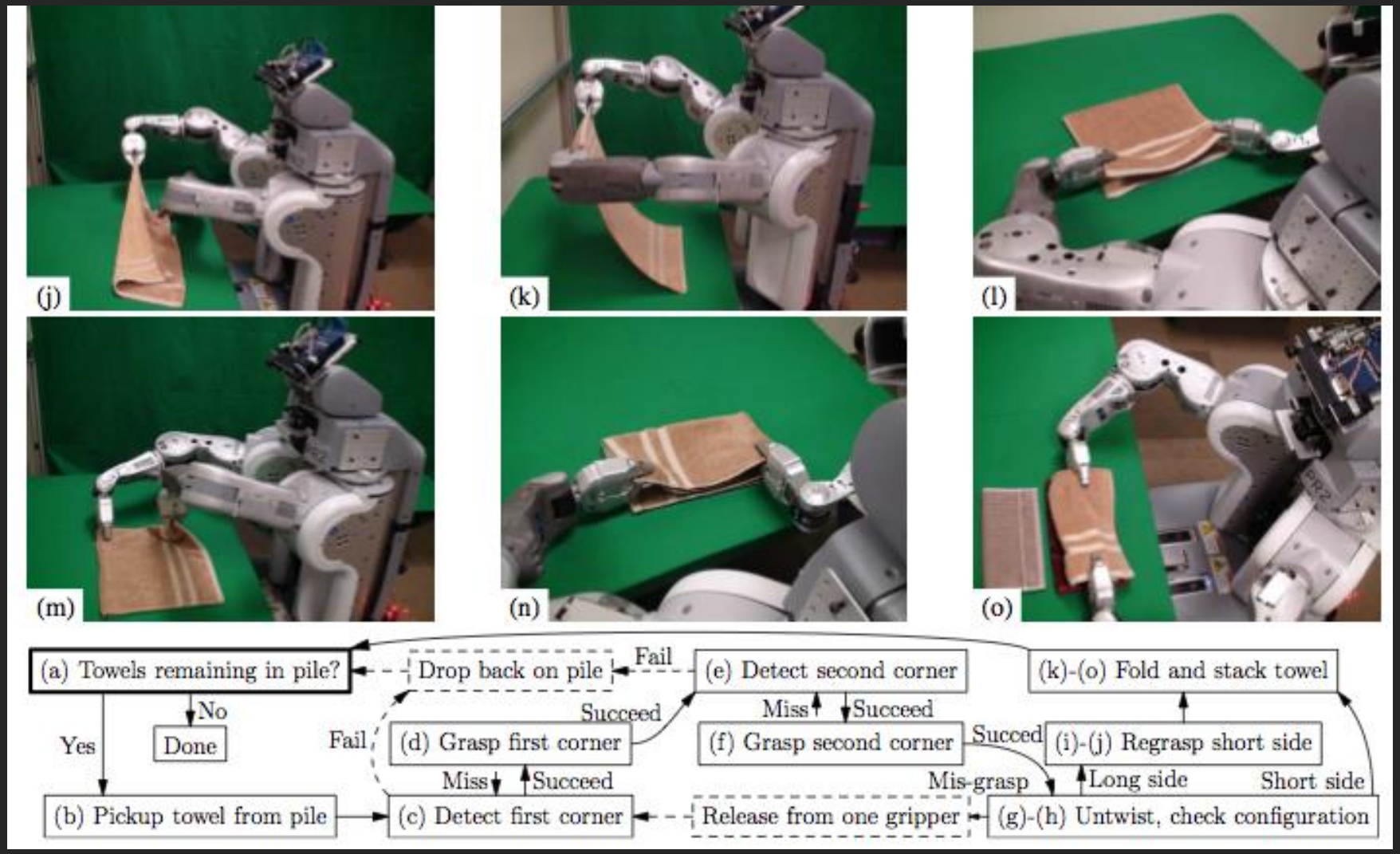


FINITE-STATE MACHINE

FSM ~ *ITS*

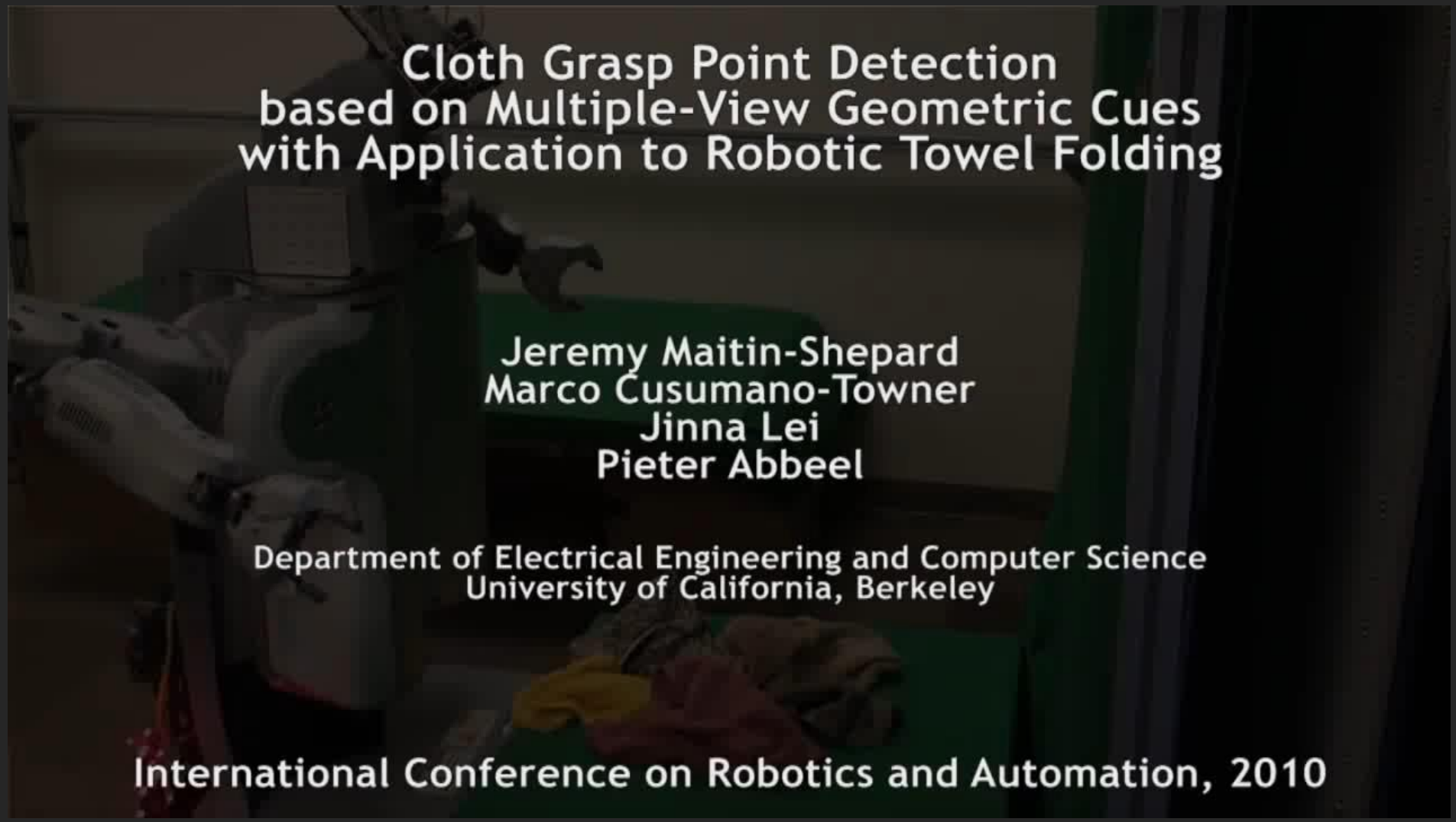
FINITE-STATE MACHINE

FSM ~ Clothes *folding robot*



FINITE-STATE MACHINE

FSM ~ Clothes *folding robot*



FINITE-STATE MACHINE

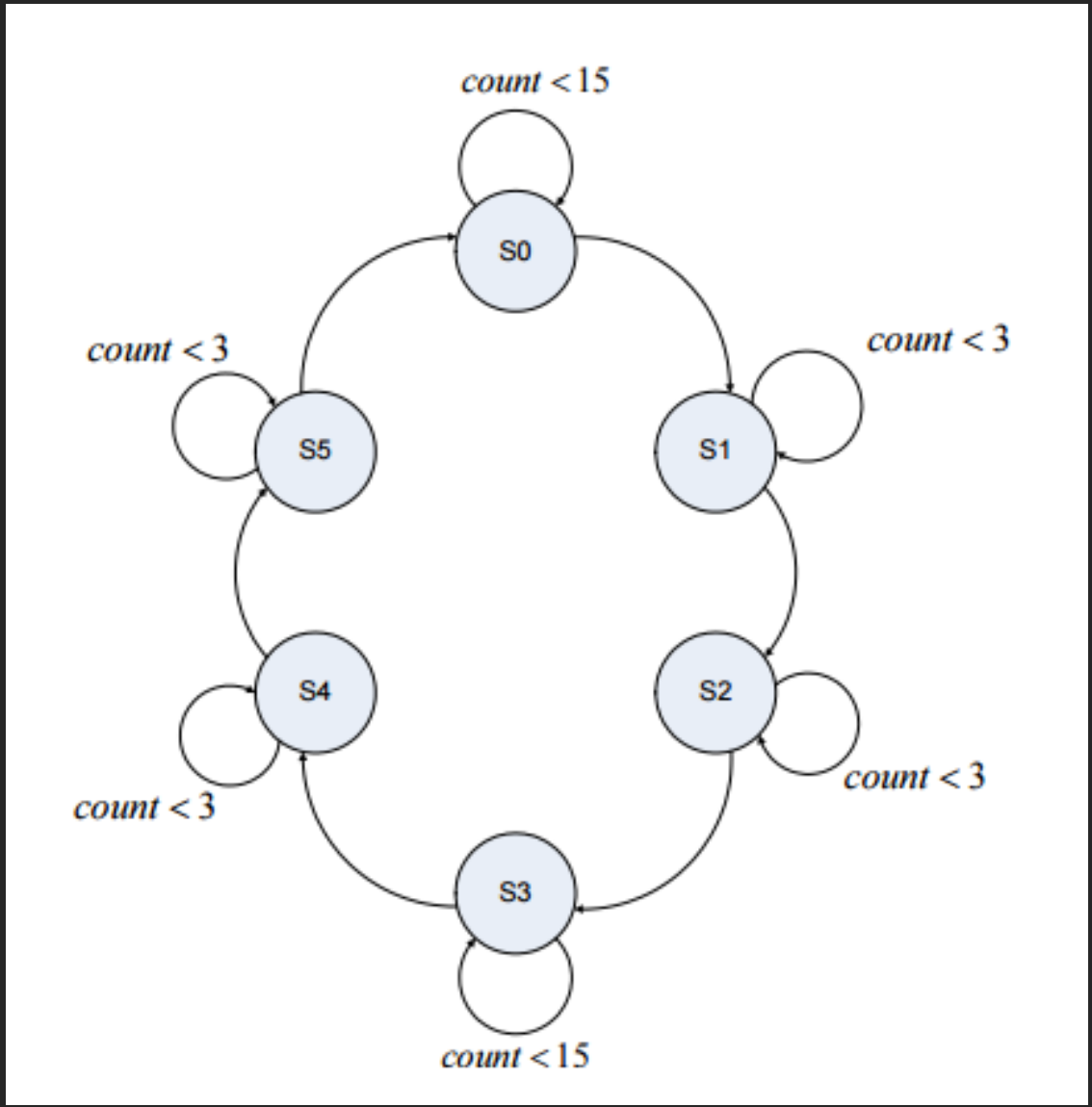
LET’S GET PRACTICAL

FINITE-STATE MACHINE

- Exercise: Traffic Light

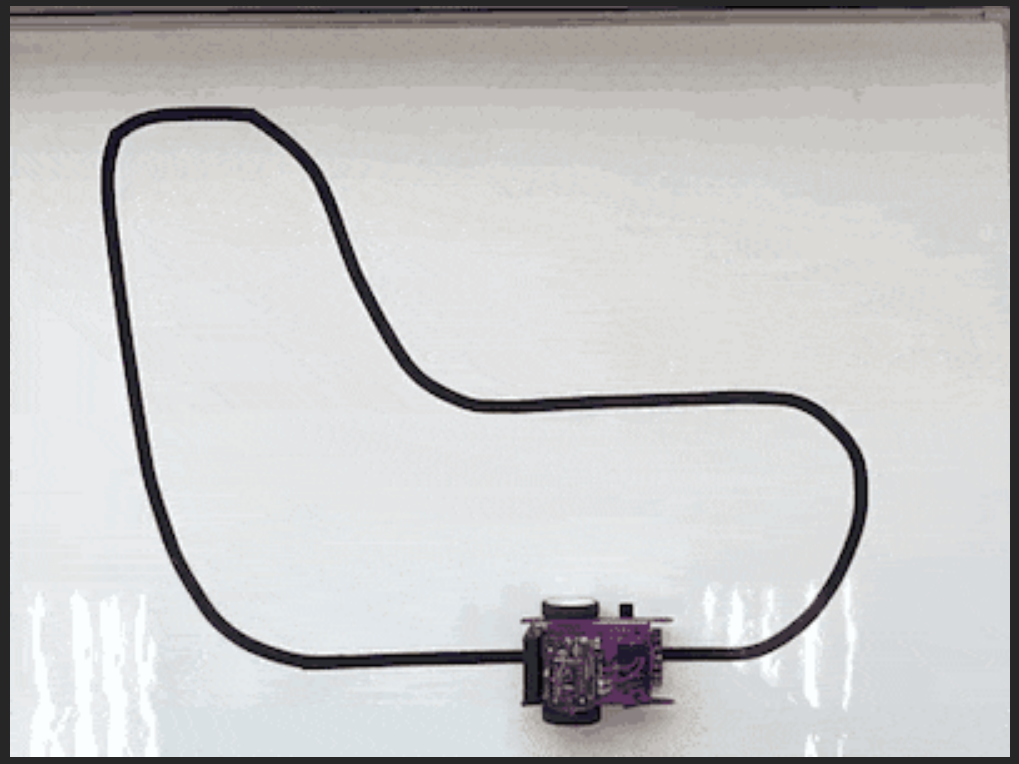
State	North-South	East-West	Delay (seconds)
0	Green	Red	15
1	Yellow	Red	3
2	Red	Red	3
3	Red	Green	15
4	Red	Yellow	3
5	Red	Red	3

FINITE-STATE MACHINE

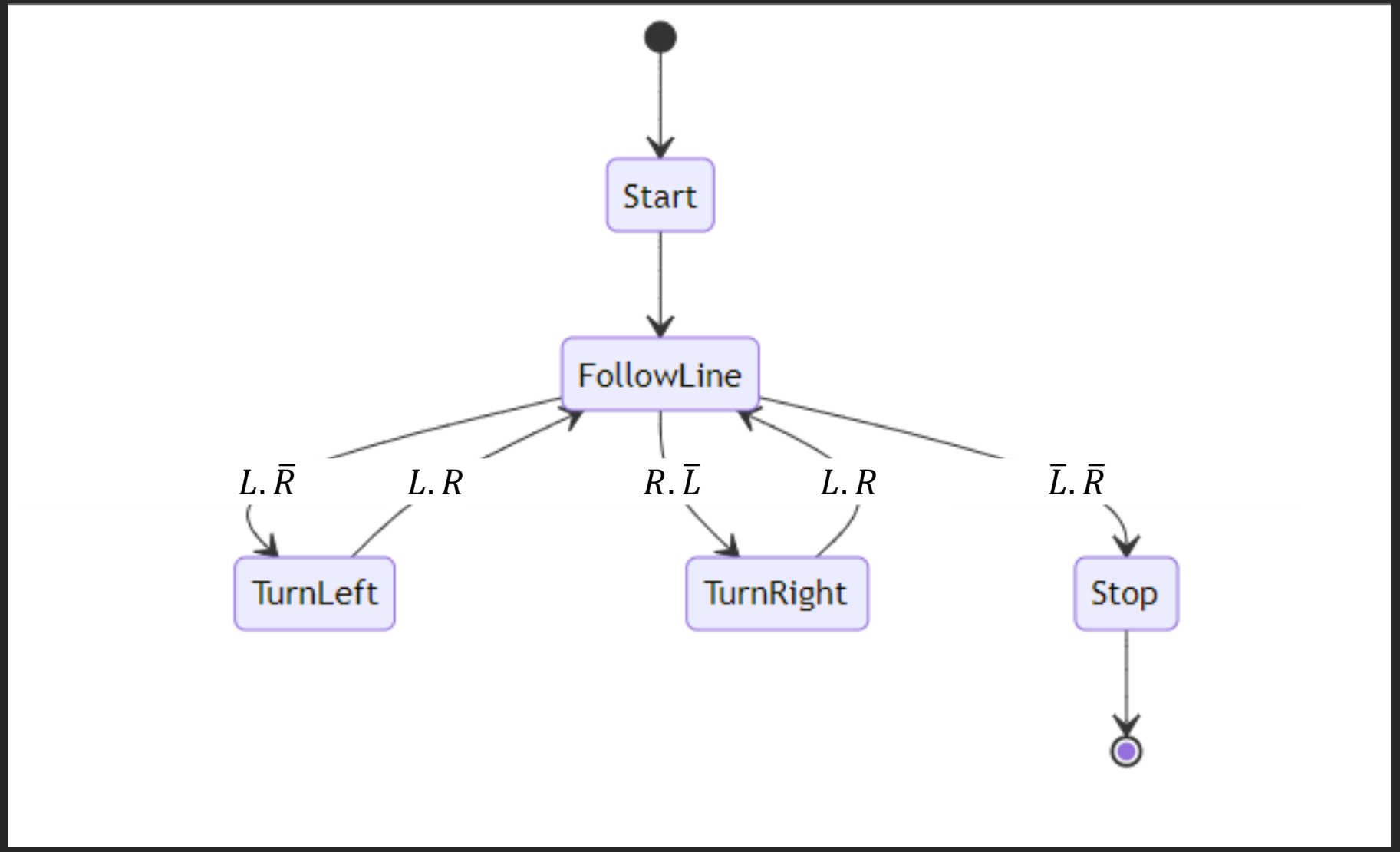


FINITE-STATE MACHINE

- Exercise: Line-following robot
  - The robot should start on top of the line and use the left and right sensors to stay on it
  - If none of the sensors detect the line, then the robot should simply stop



FINITE-STATE MACHINE



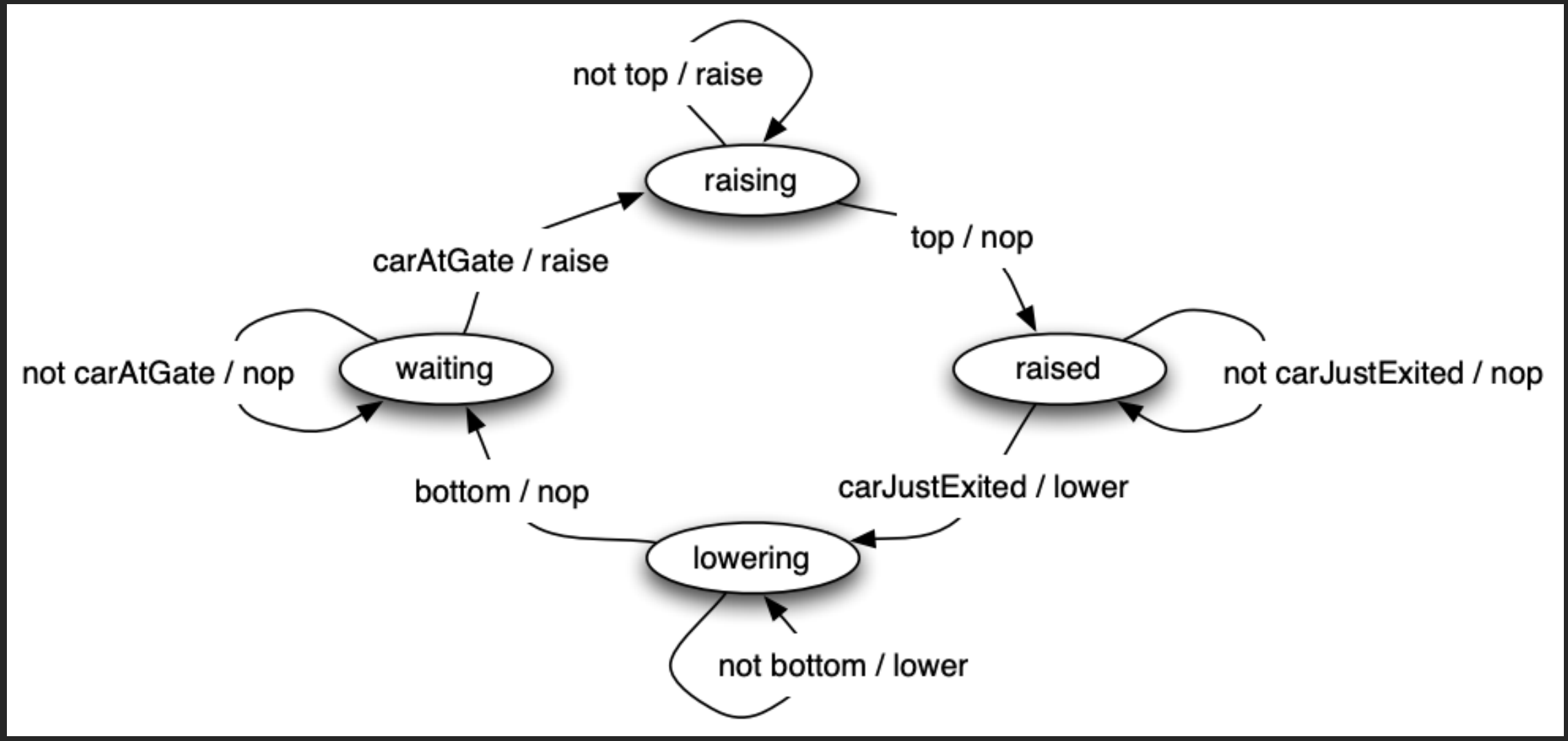
FINITE-STATE MACHINE

- Exercise: Parking gate control
  - The gate can be in one of three positions: ‘top’, ‘middle’ or ‘bottom’
  - A sensor tells the gate if a car is waiting in front of it and if a car has just passed through it
  - The gate can take the following actions: raise the gate, lower the gate, no operation (nop).

We want the following behaviour:

- If a car wants to come through, the gate raises the arm to ‘top’ position
- The gate has to stay there until the car has driven though the gate
- The gate has to go back down after the car has gone through

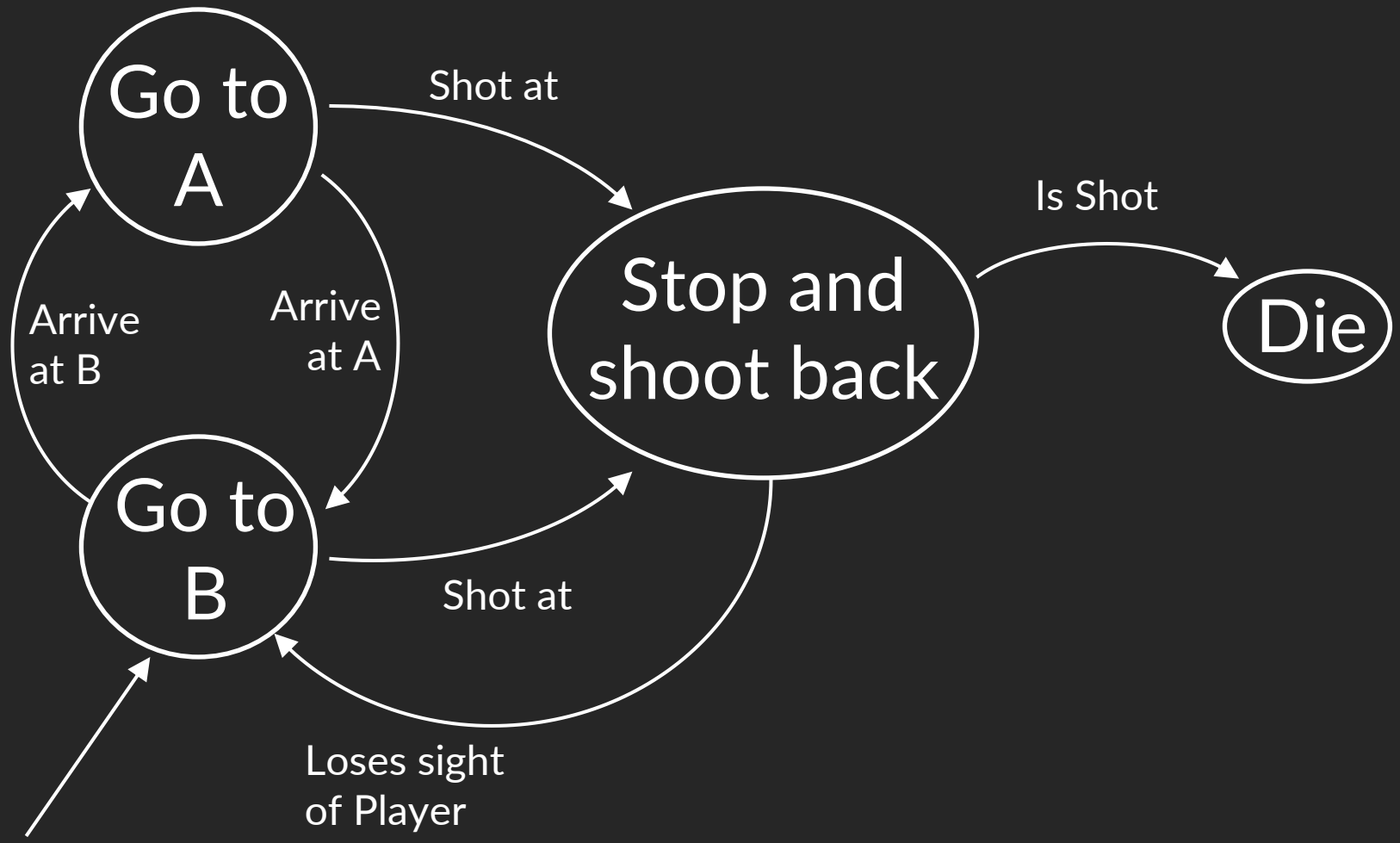
FINITE-STATE MACHINE



FINITE-STATE MACHINE

- Exercise: NPC Guard
  - The NPC guard patrols between point A and point B
  - If shot at, the guard will stop, engage with the player and return fire
  - If he loses sight of the player, the guard will return to patrolling
  - If hit, the guard will fall onto the ground and die

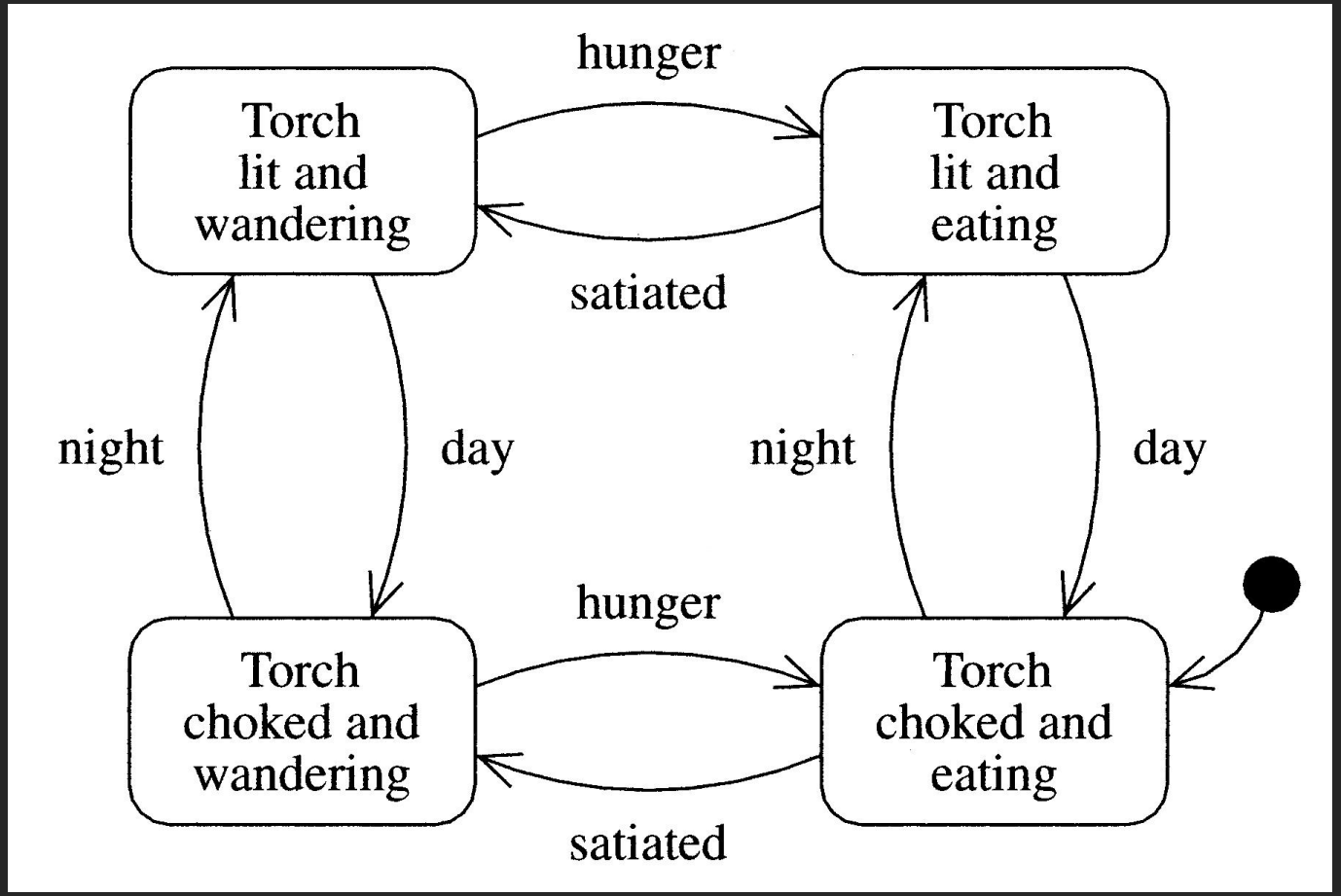
FINITE-STATE MACHINE



FINITE-STATE MACHINE

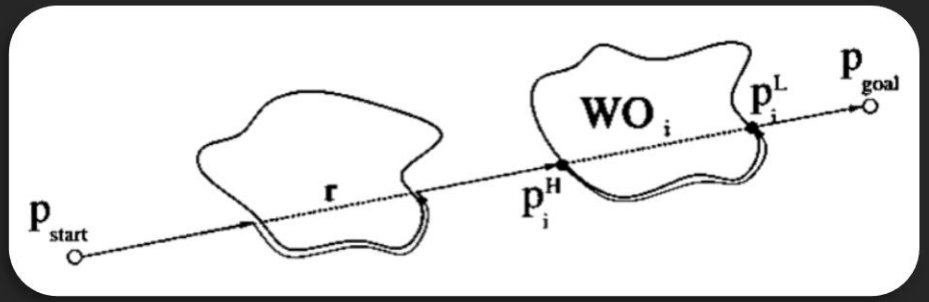
- Exercise: Hunter
  - A hunter wanders the wilderness
  - If she is hungry, she eats
  - If it is night, she lights a torch to be able to see

FINITE-STATE MACHINE



FINITE-STATE MACHINE

- Exercise: Robot-Bug



  - The robot should move towards the goal until an obstacle is encountered
  - The robot should circumvent the obstacle until the *r* straight line is encountered, *i.e.*, the line connecting the starting point and the goal
  - At that point, the robot should return to its motion-to-goal behaviour along the *r* straight line

FINITE-STATE MACHINE

- Exercise: Robot-Bug

Assumptions:

- Each point belongs to a finite set W
- Function isonR returns true if the robot is on the line *r*

Sensing capabilities:

- Touch: digital sensor that returns the logic value '1' whenever touching the wall
- Pos: variable in W, updated by a position estimator (e.g., GPS)

Actuation capabilities:

- go: Robot moves along the straight line in front of it
- turnToGoal: robot rotates instantaneously to the goal
- coastObs: robot proceeds coasting the obstacle
- stop: robot stops

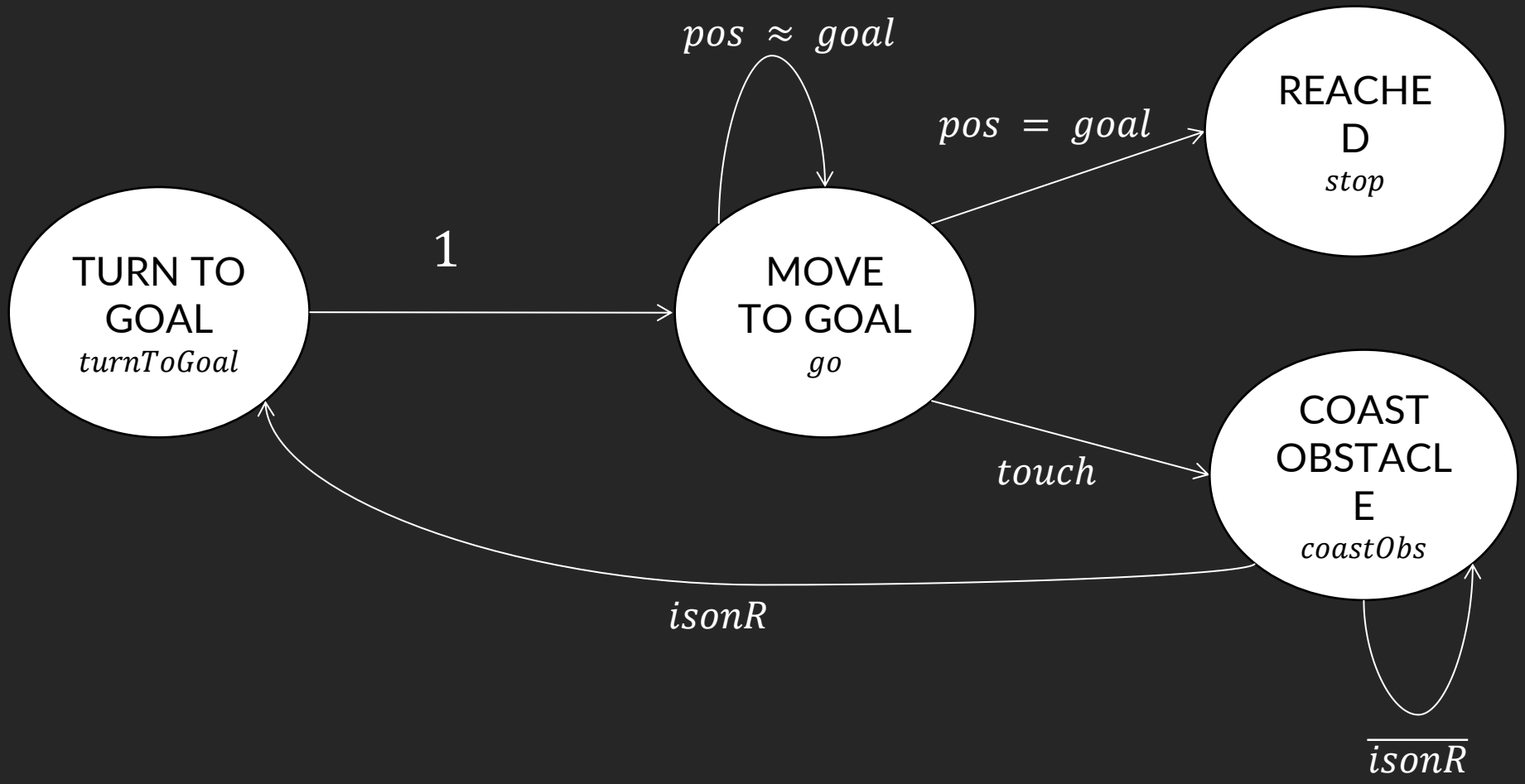
Strategy:

- Rotate to goal and move straight
- If an obstacle is found, coast it until *r* is found again

Objective:

- Provide enough intelligence to the robot-bug so it can reach the goal and avoid obstacles in-between

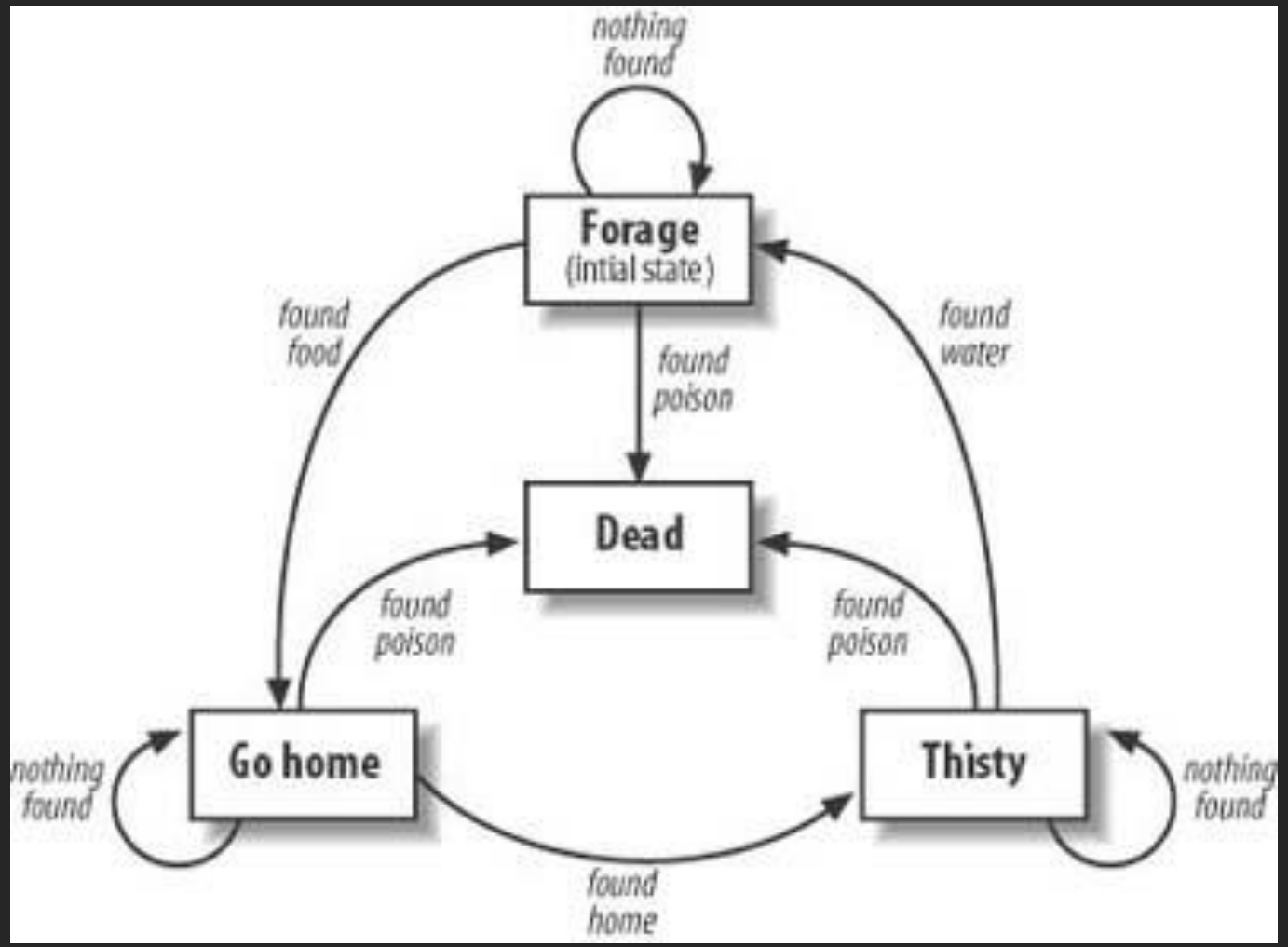
FINITE-STATE MACHINE



FINITE-STATE MACHINE

- Exercise: Ant looking for food
  - First, the ants will move randomly in their environment in an attempt to locate a piece of food.
  - Once an ant finds a piece of food, it will return to its home position.
  - When it arrives home, it will drop its food and then start a new search for water rather than food.
  - The thirsty ants will roam randomly in search of water.
  - Once an ant finds water, it will resume its search for more food.
  - Returning food to the home position also will result in a new ant emerging from the home position.
  - The ant population will continue to grow so long as more food is returned to the home position.
  - Of course, the ants will encounter obstacles along the way.
  - In addition to the randomly placed food will be randomly placed poison.
  - Naturally, the poison has a fatal effect on the ants.

FINITE-STATE MACHINE



FINITE-STATE MACHINE

LET’S WRAP IT UP

WEAKNESSES OF FSMS

1. Rigid modeling of behaviors
   - Each model can only be in one state at a time – straightforward but limited
2. Complex transition conditions can affect efficiency
   - …if many conditions need to be checked
3. Behaviors are deterministic
   - Designer pre-determines the actions and behaviors of the character, unlikely for it to respond outside what it was designed to

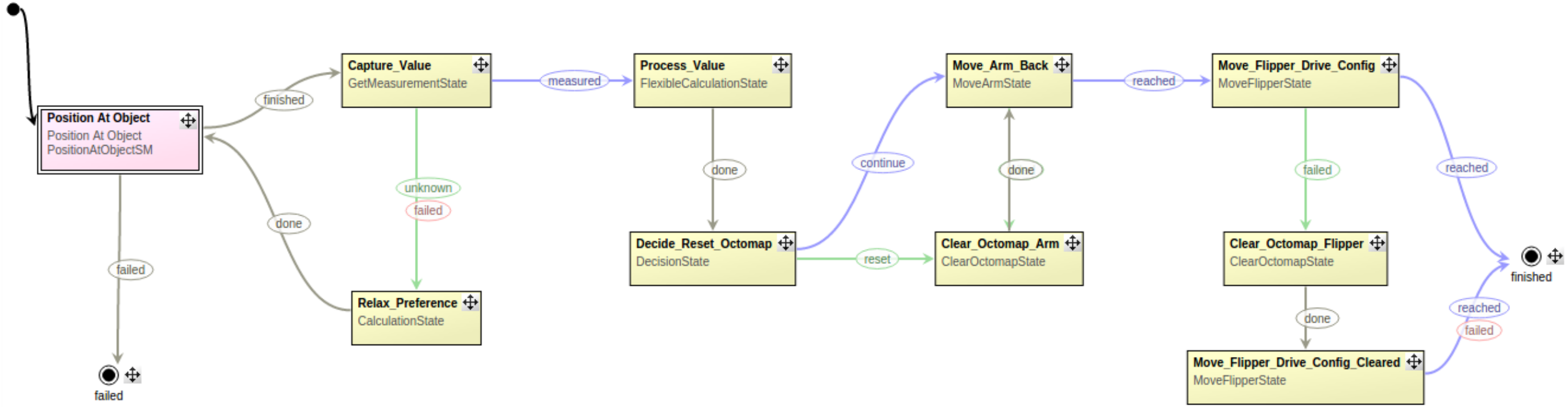
WEAKNESSES OF FSMS

1. Rigid modeling of behaviors
   - Modeling multiple states in a nested hierarchy might help to construct more complex behaviors
2. Complex transition conditions can affect efficiency
   - Use decision trees in the transitions
3. Behaviors are deterministic
   - Apply fuzzy logic or probabilistic models to the transitions

FSM FRAMEWORK

ROS

FlexBE (http://philserver.bplaced.net/fbe/)

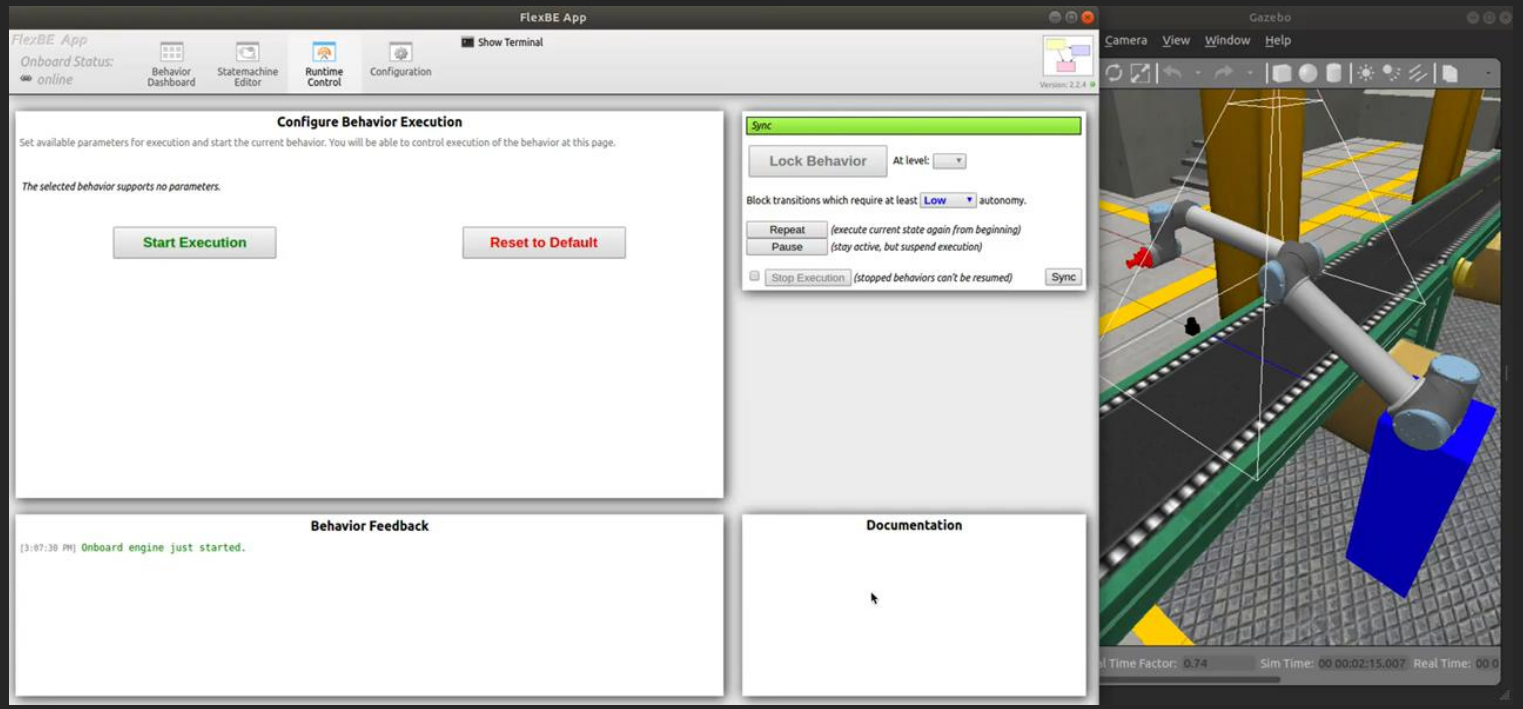


FSM FRAMEWORK

ROS

FlexBE (http://philserver.bplaced.net/fbe/)

- Drag & drop behavior creation
- Monitor behavior execution
- Runtime-modifications
- Automated code generation
- Adjustable level of autonomy



FlexBE (compared to Groot)

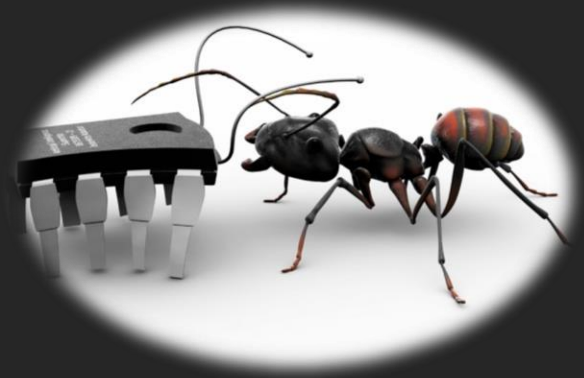
The good

- Hierarchical state machines
- Autonomy levels
- Dynamic behaviour adaptationPython-friendly
- Python-friendly
- Integrated monitoring

The bad

- Complexity
- Less modularity
- Performance
- No shared blackboard

LET'S GET (MORE) PRACTICAL…



ROBOT-ANT: BASIC MAZE SOLVER

ROBOT-ANT BASIC

Sensing capabilities:

- Left (L) and right (R) antennas as digital sensors that return the logic value ‘1’ whenever touching the wall

Actuation capabilities:

- Move forward (F), turn left (TL) and right (TR)

Objective:

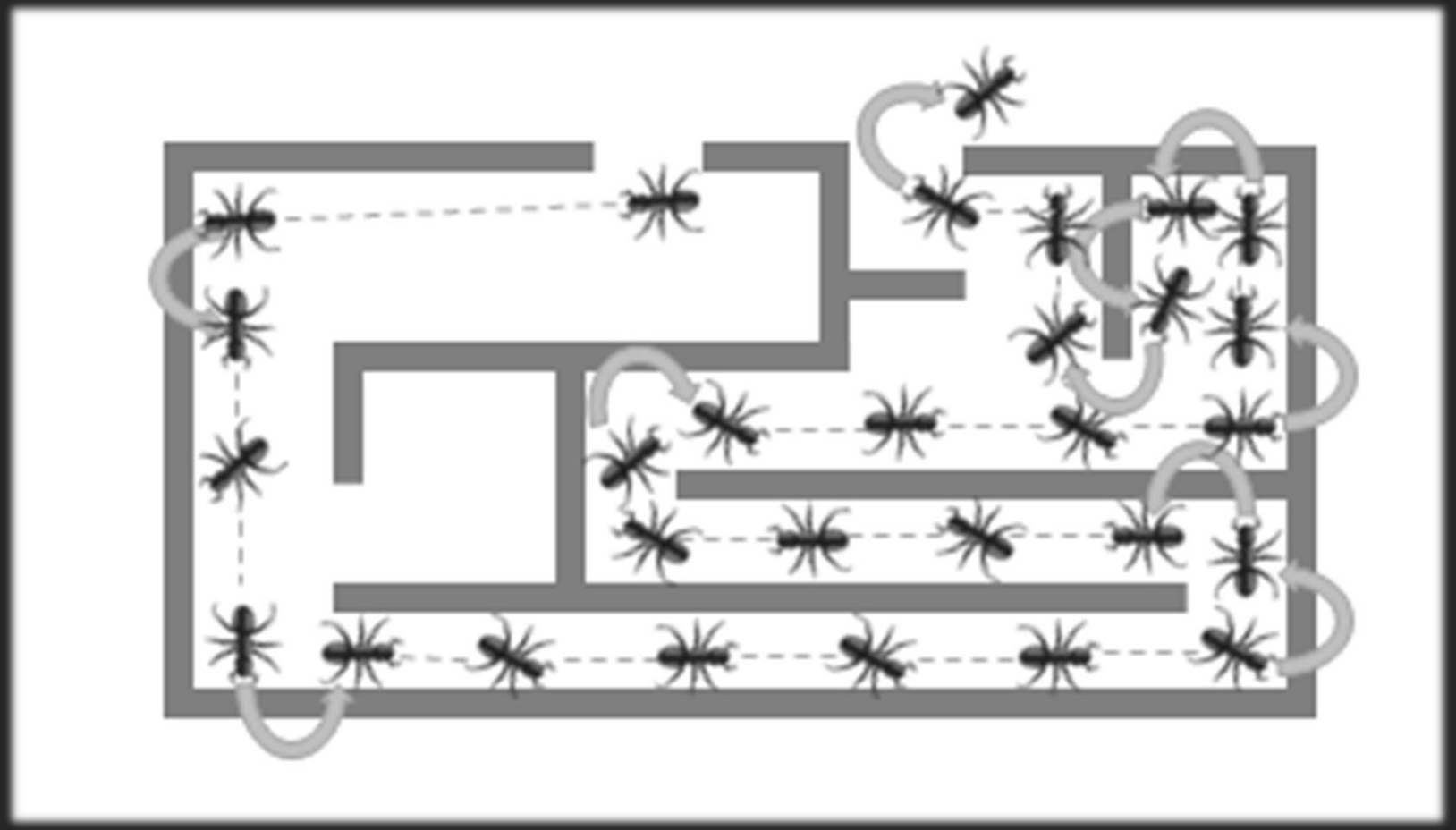
- Provide enough intelligence to the robot-ant so it can get out of the maze

Strategy:

- Keep the right antenna facing the wall

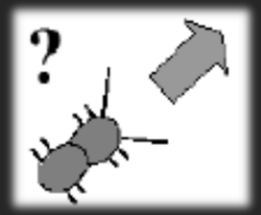
ROBOT-ANT BASIC

Illustrative example



ROBOT-ANT BASIC

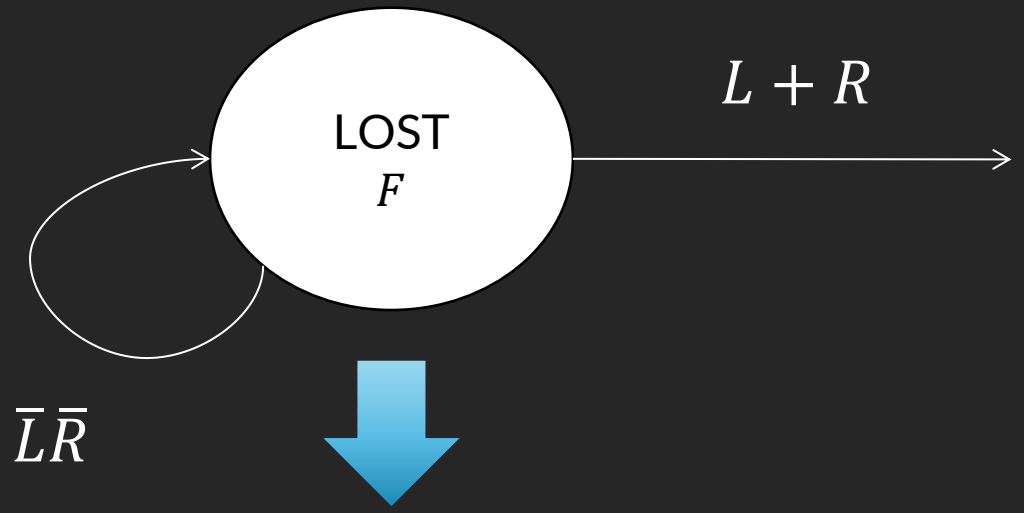
Expected behaviour



1. The robot-ant is lost, so move forward (F) until, at least, one of the antennas touches the wall ($L + R$)

Try to solve the finite-state machine (FSM) for this action

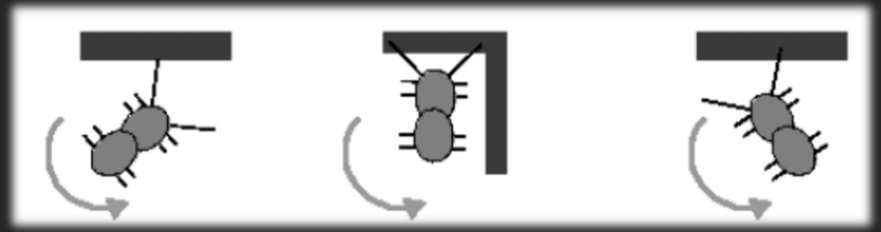
ROBOT-ANT BASIC



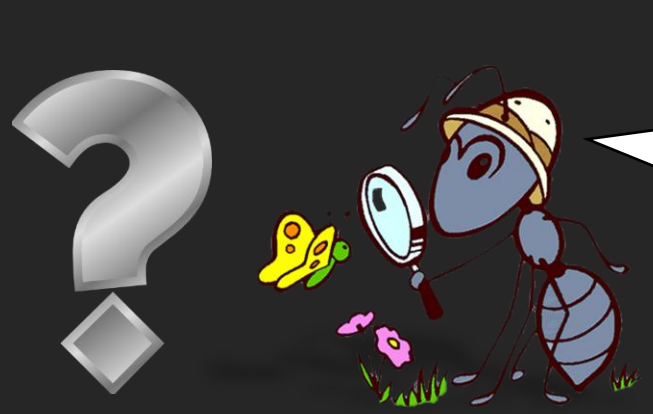
LOST state: the robot-ant is not touching any wall

ROBOT-ANT BASIC

Expected behaviour

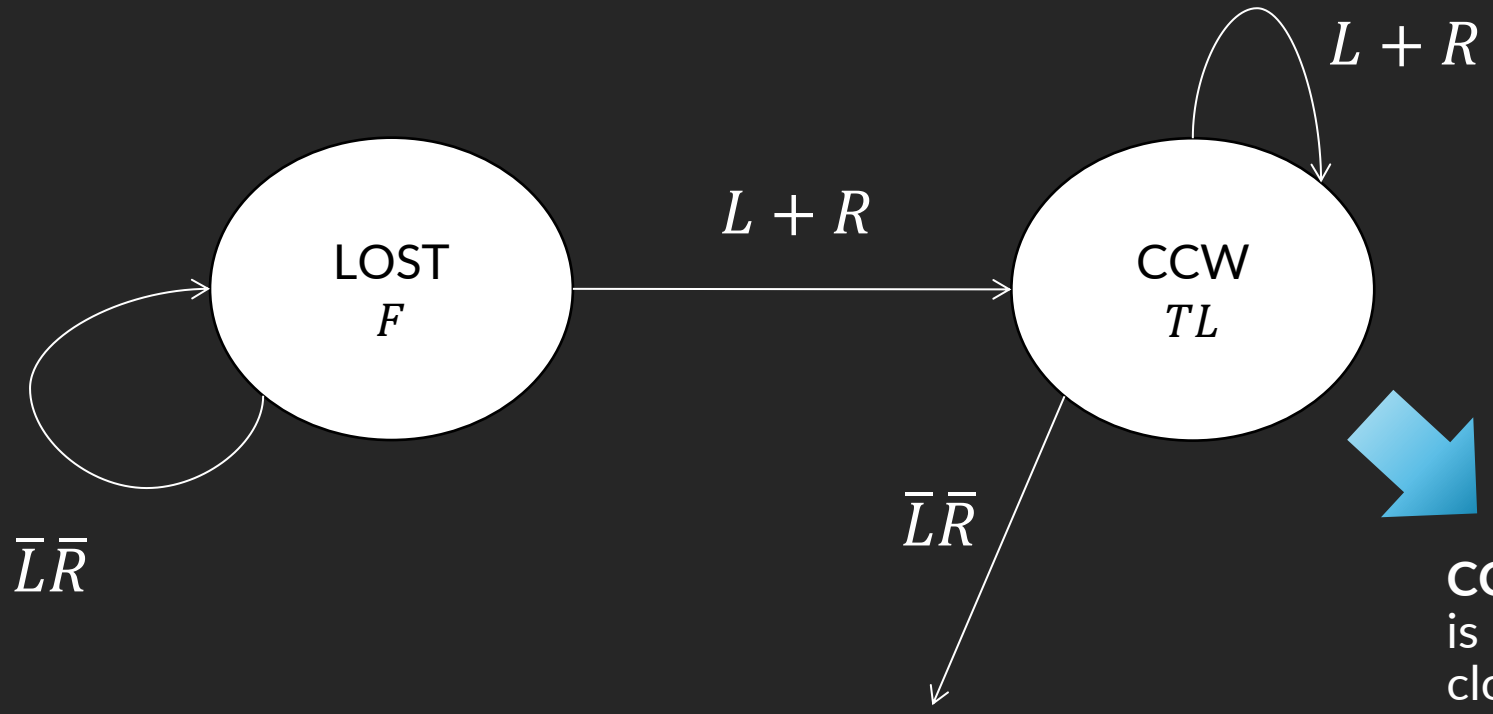


2. Turn left (TL) until the robot-ant stops touching the wall ($\bar{L}\bar{R}$)



Try to solve the finite-state machine (FSM) for this action

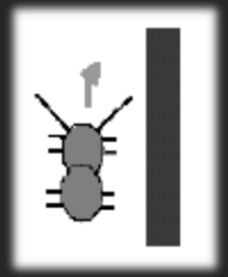
ROBOT-ANT BASIC



CCW state: the robot-ant is rotating in the counter-clockwise direction

ROBOT-ANT BASIC

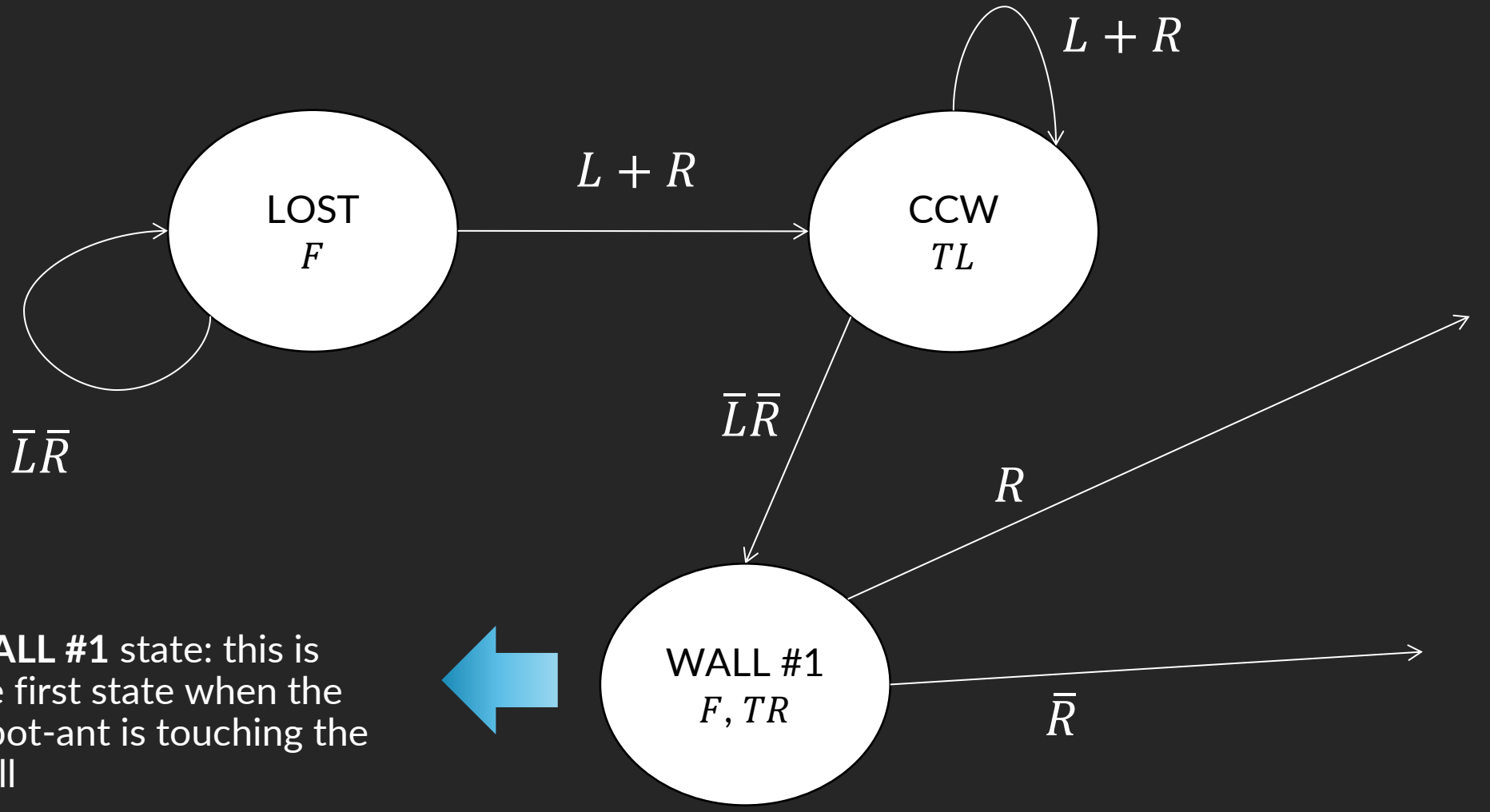
Expected behaviour



3. Move one-step forward (F) and then right (TR), looking for a wall (R)

Try to solve the finite-state machine (FSM) for this action

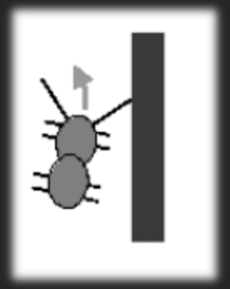
ROBOT-ANT BASIC



WALL #1 state: this is the first state when the robot-ant is touching the wall

ROBOT-ANT BASIC

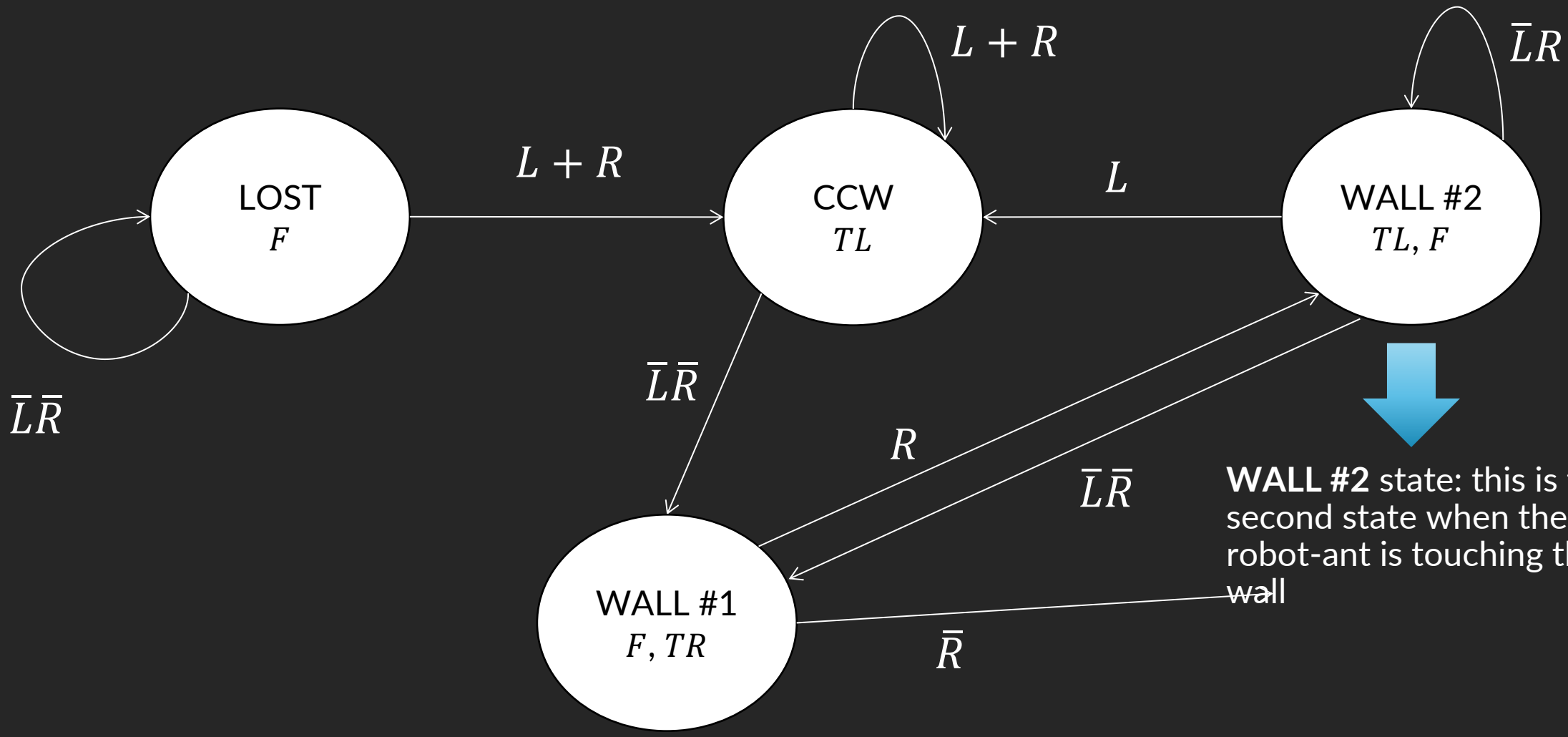
Expected behaviour



4. Move left (L) and then one-step forward (F), until the robot-ant stops touching the wall ($\bar{L}\bar{R}$)

Try to solve the finite-state machine (FSM) for this action

ROBOT-ANT BASIC



WALL #2 state: this is the second state when the robot-ant is touching the wall

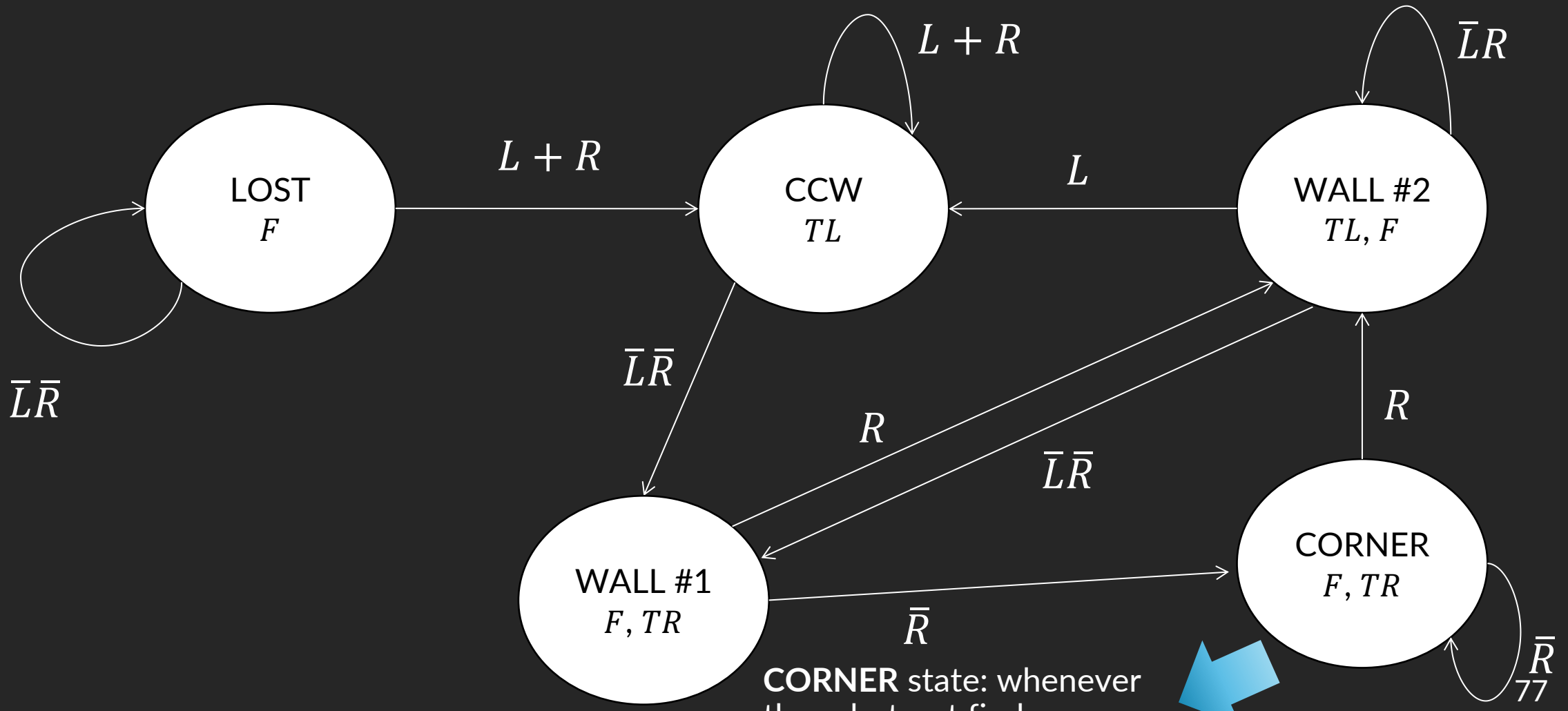
ROBOT-ANT BASIC

Expected behaviour

5. Move one-step forward (F) and then right (TR), looking for a wall (R)

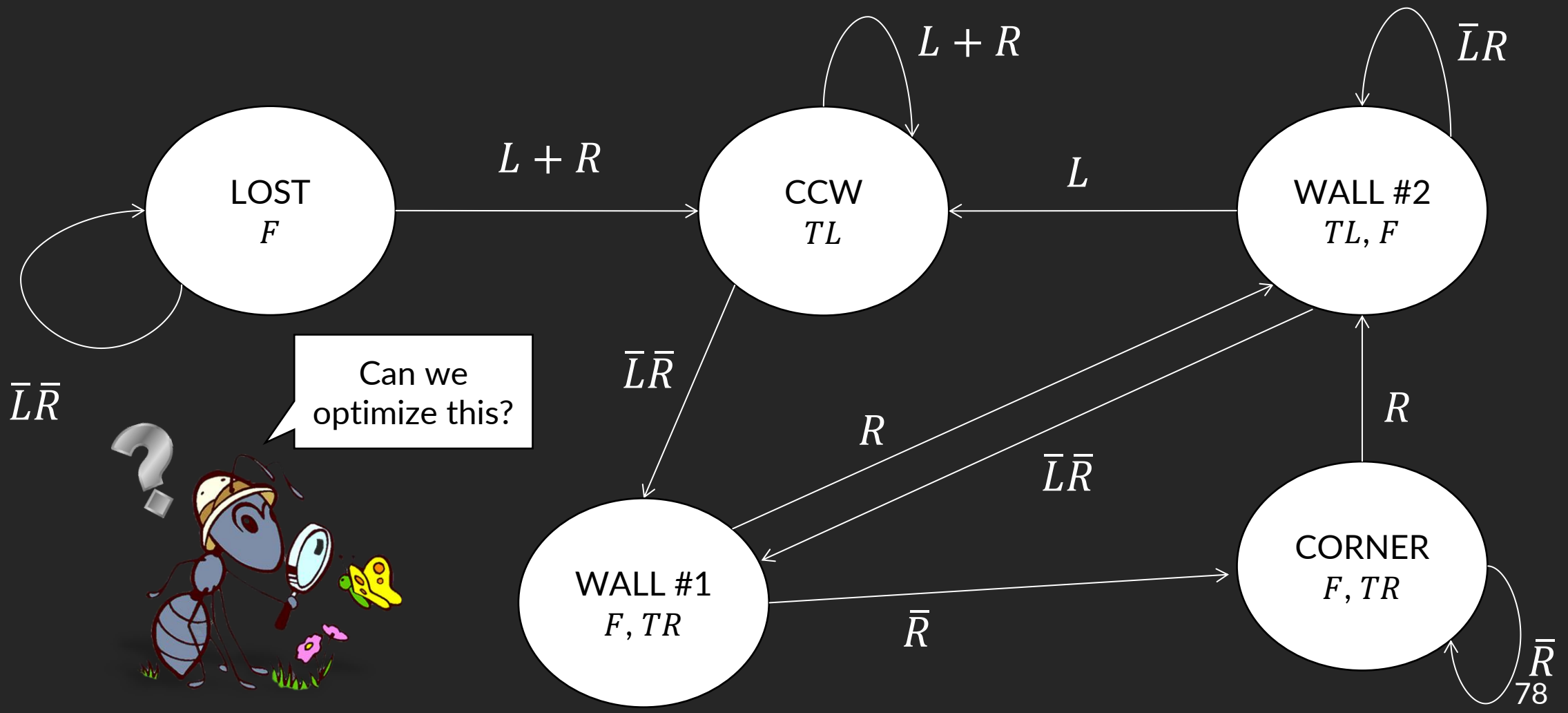
Try to solve the finite-state machine (FSM) for this action

ROBOT-ANT BASIC

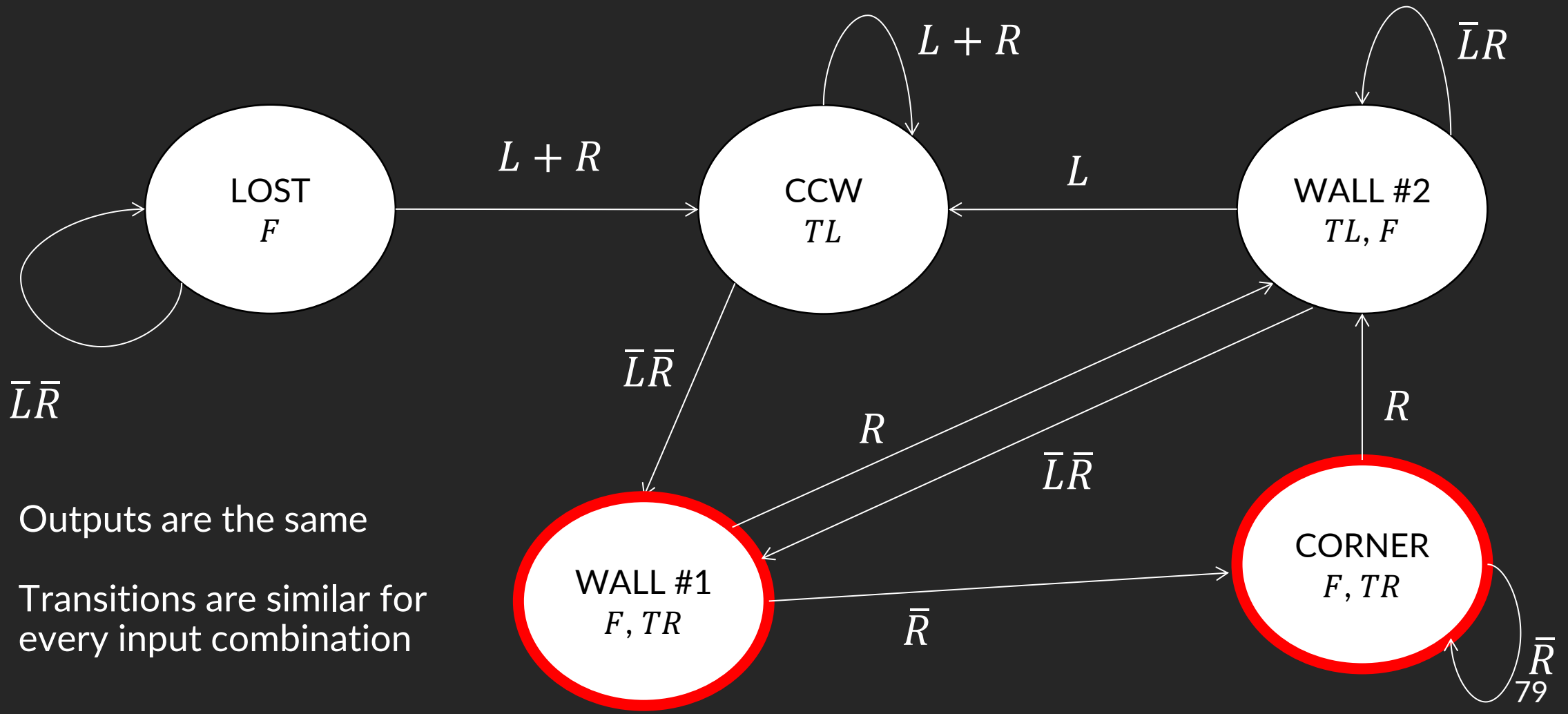


CORNER state: whenever the robot-ant finds a corner

ROBOT-ANT BASIC

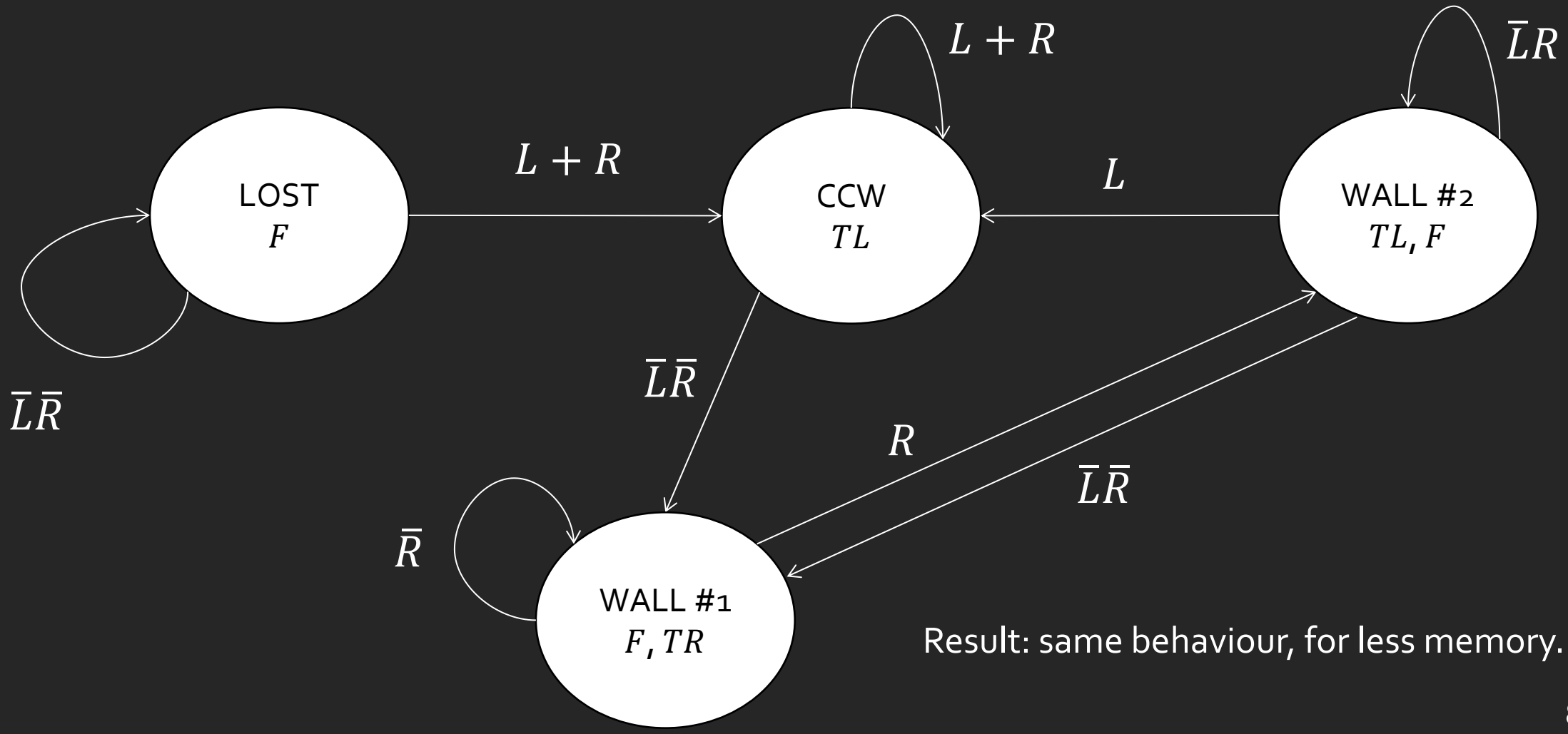


ROBOT-ANT BASIC



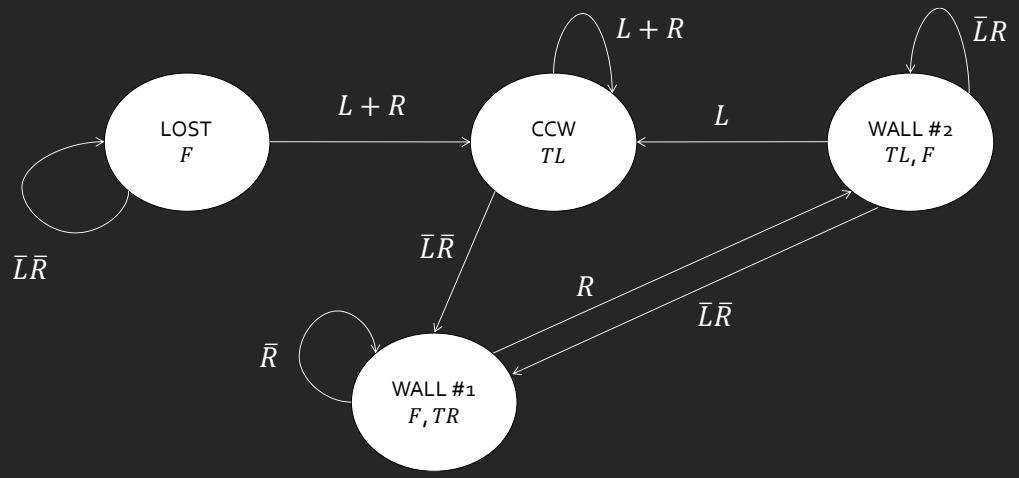
- Outputs are the same
- Transitions are similar for every input combination

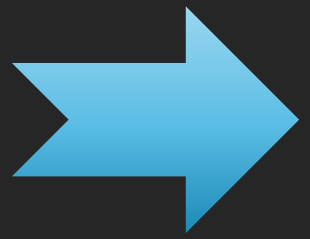
ROBOT-ANT BASIC



Result: same behaviour, for less memory.

ROBOT-ANT BASIC





ARDUINO™

Can you translate this into Arduino language?

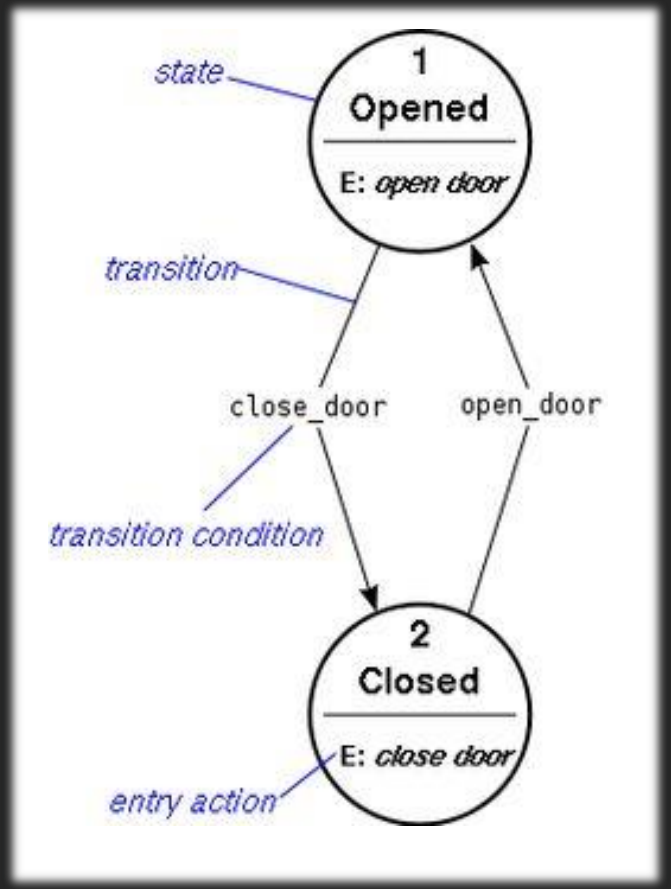
ROBOT-ANT BASIC

Basic rules to implement a FSM in Arduino

1. Each state should be implemented within a given function, called state function
2. The logic where transitions are processed should take place within the state function
3. The output should be triggered within the state function as well
4. The return of the state function should identify the triggered state, which can be of the type int, uint8_t or similar (i.e. coded as 0, 1, 2, …)
5. A switch-case should be used to manage states accordingly, being called at a specific rate (e.g. within a timer function or under any other periodically-constrained function)

ROBOT-ANT BASIC

Basic rules to implement a FSM in Arduino (example)




```
1   boolean close_door_in(){ return digitalRead(2);}
2   boolean open_door_in(){ return digitalRead(3); }
3   void open_door_out(){ Serial.println("E: Open door"); }
4   void close_door_out(){ Serial.println("E: Close door"); }
5
6   void setup() {
7     Serial.begin(9600); // for debugging
8     pinMode(2, OUTPUT);
9     pinMode(3, OUTPUT);
10  }
11
12  void loop() {
13    uint8_t state = 0;
14    while(1){
15      switch(state){
16        case 0:
17          Serial.println("Closed");
18          state = CLOSED();
19          break;
20        case 1:
21          Serial.println("Opened");
22          state = OPENED();
23          break;
24      }
25    }
26    delay(1000);
27  }
28
```

ARDUINO

```
28
29  uint8_t CLOSED(){
30    if (open_door_in())
31      open_door_out();
32    else
33      return 0;
34    return 1;
35  }
36
37  uint8_t OPENED(){
38    if (close_door_in())
39      close_door_out();
40    else
41      return 1;
42    return 0;
43  }
```

ROBOT-ANT BASIC

```
// let us define generic functions; no need to deal with real sensors and motors at this point.
// moveF(), moveTR() and moveTL() corresponds, respectively, to the robot-ant moving one-step forward,
// rotating 10 degrees to the right, and another 10 degrees to the left
// sensR() and sensL() corresponds, respectively, to the sensors' output, returning a boolean value
// (logic level '1' when it is touching the wall)
// LOST(), CCW(), WALL1() and WALL2() corresponds, respectively, to the states defined in the FSM

boolean sensR(){ return digitalRead(2); }
boolean sensL(){ return digitalRead(3); }
void moveF(){ Serial.println("Moving forward"); }
void moveTR(){ Serial.println("Rotating right"); }
void moveTL(){ Serial.println("Rotating left"); }


void setup() {
  Serial.begin(9600); // for debugging
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
}
```

ARDUINO

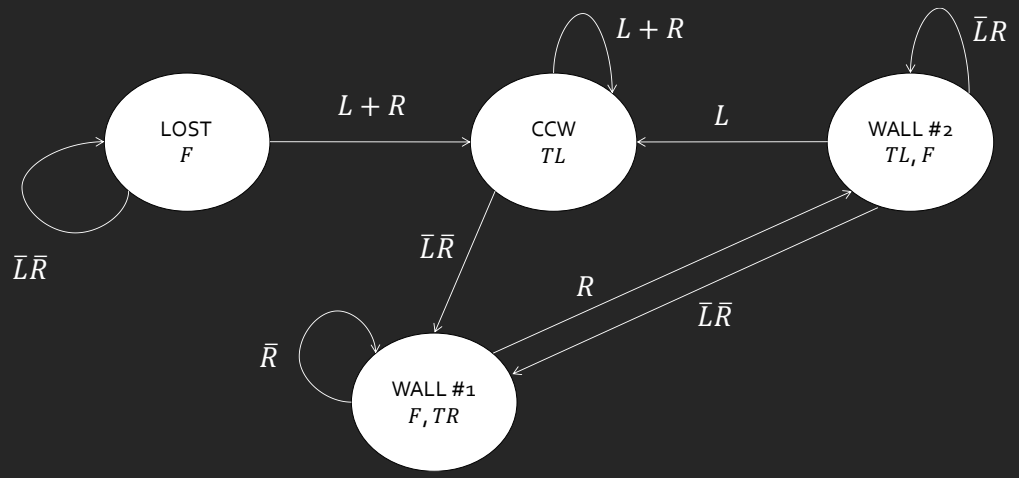
ROBOT-ANT BASIC

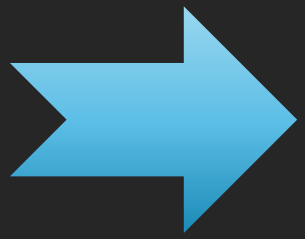
```
void loop() {
  int state = 0;
  while(1){
    switch(state){
      case 0:
        Serial.println("LOST");
        state = LOST();
        break;
      case 1:
        Serial.println("CCW");
        state = CCW();
        break;
      case 2:
        Serial.println("WALL1");
        state = WALL1();
        break;
      case 3:
        Serial.println("WALL2");
        state = WALL2();
        break;
    }
    delay(1000);
  }
```

```
int LOST(){
  if (!sensL() && !sensR())
    moveF();
  else
    return 1;
  return 0;
}

int CCW(){
  if (sensL() || sensR())
    moveTL();
  else
    return 2;
  return 1;
}
```

```
int WALL1(){
  if (!sensR()){
    moveF();
    moveTR();
  }
  else
    return 3;
  return 2;
}

int WALL2(){
  if (!sensL())
    if (sensR()){
      moveTL();
      moveF();
    }
    else
      return 2;
  else
      return 1;
  return 3;
}
```

ROBOT-ANT BASIC





ROS
Robot Operating System

Can you translate this into ROS language?

ROBOT-ANT BASIC

ROS

Basic rules to implement a FSM in ROS

1. Even though specific frameworks could be used to implement FSMs, such as FlexBE, we will implement it from scratch

2. In this case, all the previous tips for Arduino applies, bearing in mind that:
   - Inputs are expected to come in the form of subscribers
   - Outputs are expected to come in the form of publishers

ROBOT-ANT BASIC

```
// let us define generic functions/variables; no need to deal with real sensors and motors at this point.
// move(), turn() correspond, respectively, to the robot-ant moving forward or rotating (right/left)
// sensR and sensL correspond, respectively, to the sensors' output, having a boolean value
// (logic level '1' when it is close to a wall)
// lost(), ccw(), wall1(), wall2() correspond to the states defined in the FSM.

#include "ros/ros.h"
#include "sensor_msgs/Range.h"
#include "geometry_msgs/Twist.h"

#define STATE_LOST   0
#define STATE_CCW    1
#define STATE_WALL1  2
#define STATE_WALL2  3

bool sensR, sensL;
ros::Publisher cmd_vel_pub;

void sensR_Callback (const sensor_msgs::Range::ConstPtr& msg){
    if (msg->range < msg->max_range){
        //detecting a wall (e.g. range below max_range of the sensor).
        sensR = true; //You can use any other rule that you prefer
    }else{
        sensR = false;
    }
}

void sensL_Callback (const sensor_msgs::Range::ConstPtr& msg){
    if (msg->range < msg->max_range){
        sensL = true; //detecting a wall
    }else{
        sensL = false;
    }
}
```

ROS
Robot Operating System

ROBOT-ANT BASIC

```
void move (double linear_speed){ //move the robot forward with a given linear_speed (m/s)
    geometry_msgs::Twist cmd_msg;
    cmd_msg.linear.x = linear_speed;  //make sure linear_speed > 0 to move forward
    cmd_vel_pub.publish(cmd_msg);
}

void turn (double angular_speed){ //turn the robot with a given angular speed (rad/s)
    geometry_msgs::Twist cmd_msg;
    cmd_msg.angular.z = angular_speed; //make sure angular_speed > 0 to turn left, and <0 to turn right
    cmd_vel_pub.publish(cmd_msg);
}

int lost(){
    if (!sensR && !sensL){
        move(linear_speed); //use linear_speed>0 (move forward)
    }else{
        return STATE_CCW;
    }
    return STATE_LOST;
}

int ccw(){
    if (sensR || sensL){
        turn(angular_speed); //use angular_speed>0 (turn left)
    }else{
        return STATE_WALL1;
    }
    return STATE_CCW;
}
```

```
int wall1(){
    if (!sensR){
        move(linear_speed);  //use linear_speed>0 (move forward)
        turn(angular_speed); //use angular_speed<0 (turn right)
    }else{
        return STATE_WALL2;
    }
    return STATE_WALL1;
}

int wall2(){
    if (!sensL){
        if (sensR){
            turn(angular_speed); //use angular_speed>0 (turn left)
            move(linear_speed);  //use linear_speed>0 (move forward)
        }else{
            return STATE_WALL1;
        }
    }else{
        return STATE_CCW;
    }
    return STATE_WALL2;
}
```

ROS
Robot Operating System

ROBOT-ANT BASIC

```
int main(int argc, char **argv) {

    ros::init(argc, argv, "ant_fsm");
    ros::NodeHandle n;
    ros::Rate loop_rate(10); //10Hz (loop at 10 times/sec)

    //Publisher for /cmd_vel
    cmd_vel_pub = n.advertise<geometry_msgs::Twist>("cmd_vel", 100);
    //Subscriber for sensR:
    ros::Subscriber sensR_sub = n.subscribe("sensR_topic_name", 100, sensR_Callback);
    //Subscriber for sensL:
    ros::Subscriber sensL_sub = n.subscribe("sensL_topic_name", 100, sensL_Callback);

    int state = STATE_LOST; //initial state

    while (ros::ok()){
```

```
    while (ros::ok()){

        switch(state) {

            case STATE_LOST :
                ROS_INFO("Lost");
                state = lost();
                break;

            case STATE_CCW :
                ROS_INFO("CCW");
                state = ccw();
                break;

            case STATE_WALL1 :
                ROS_INFO("Wall1");
                state = wall1();
                break;

            case STATE_WALL2 :
                ROS_INFO("Wall2");
                state = wall2();
                break;

            default :
                ROS_ERROR("Invalid State");
                state = STATE_LOST;
        }

        ros::spinOnce();
        loop_rate.sleep();
    }

    return 0;
}
```

ROS
Robot Operating System

WAVEFRONT PLANNING ALGORITHM

PATH PLANNING

Path planning, namely within the ROS community, is broken down into two main categories:

1. **Global path planning**: Concerned with finding a path from start to goal considering the entire known map, without immediate regard for dynamic obstacles.
2. **Local path planning**: Focuses on real-time navigation and reacting to unforeseen or moving obstacles using sensor data.

GLOBAL PATH PLANNING

Relevance in Robotics

- **Navigation**: Essential for robots to autonomously navigate in their environment, be it a warehouse, home, outdoor terrain, or industrial setting.
- **Optimization**: Ensures efficient utilization of battery life and time, by finding the shortest or least costly path.
- **Safety**: Avoids known obstacles and unsafe areas, ensuring the robot and its surroundings are undamaged.

GLOBAL PATH PLANNING

A global path planner should (ideally)…

- **Be complete:** Always be able to find a path if one exists.
- **Be optimal:** Provide be the best path according to some criteria, such as shortest distance or least energy consumption.
- **Be deterministic:** Produce consistent results given the same environment and start/goal points.

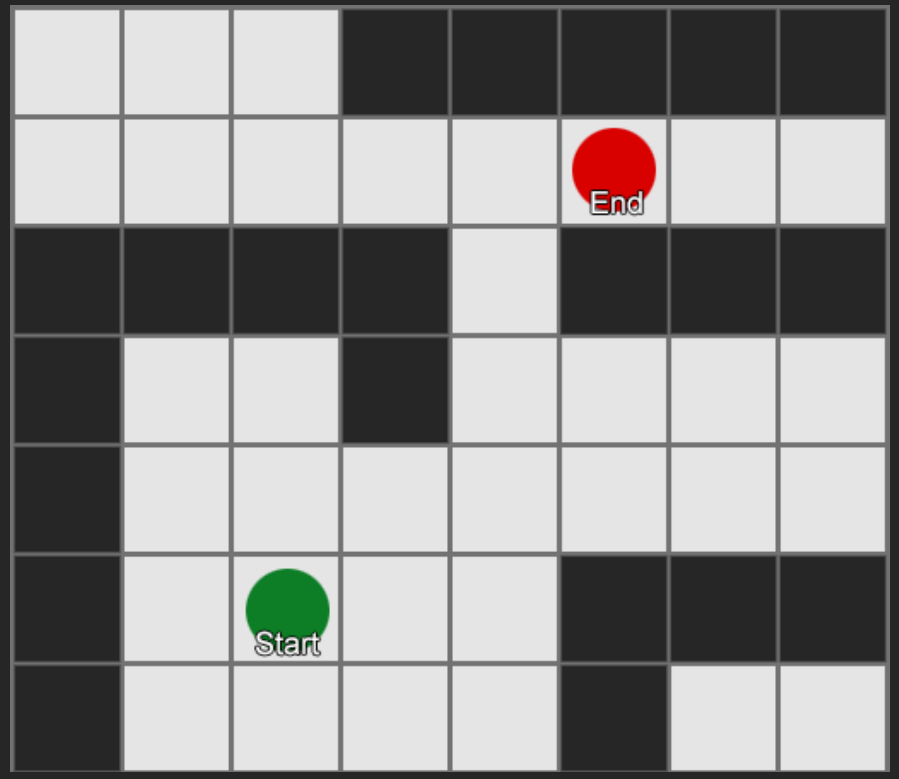
GLOBAL PATH PLANNING

Challenges in global path planning:

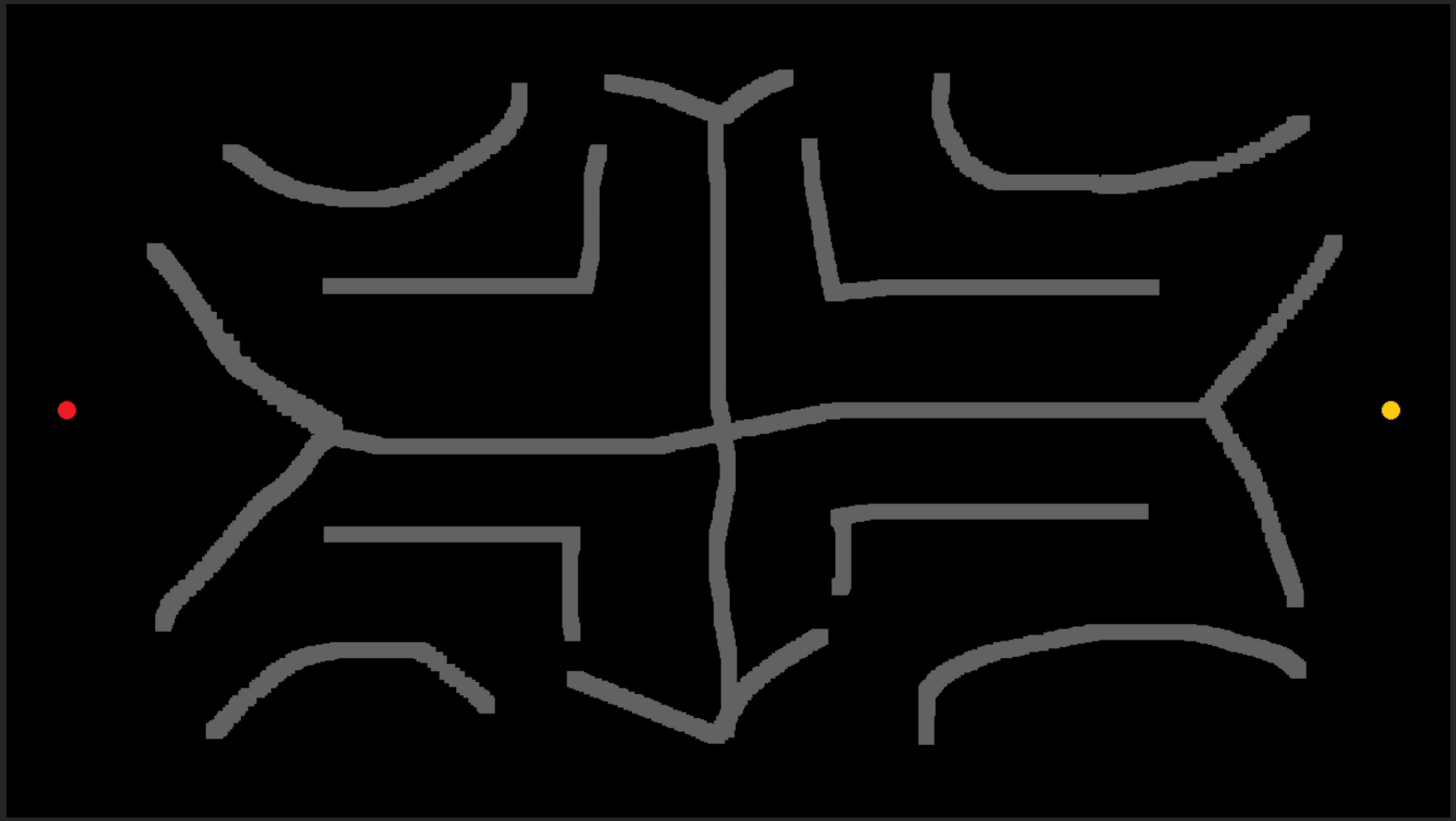
- **Dynamic Environments:** Environments that change over time can invalidate a pre-planned path.
- **Computational Complexity:** Larger and more intricate environments increase the computation time and complexity of the path planning.
- **Unknown Environments:** Robots might not always have a complete map, necessitating exploration and on-the-fly planning.

WAVEFRONT PLANNING ALGORITHM

- Wavefront Planning Algorithm (WPA) is a grid-based approach



WAVEFRONT PLANNING ALGORITHM



WAVEFRONT PLANNING ALGORITHM

WPA basic steps

1. Create discretized map of the environment as a X-Y grid (matrix) identifying empty spaces and obstacles
2. Identify robot position (start) and target position (end) within such X-Y grid
3. Fill in the wavefront
4. Choose one of the many available paths

WAVEFRONT PLANNING ALGORITHM

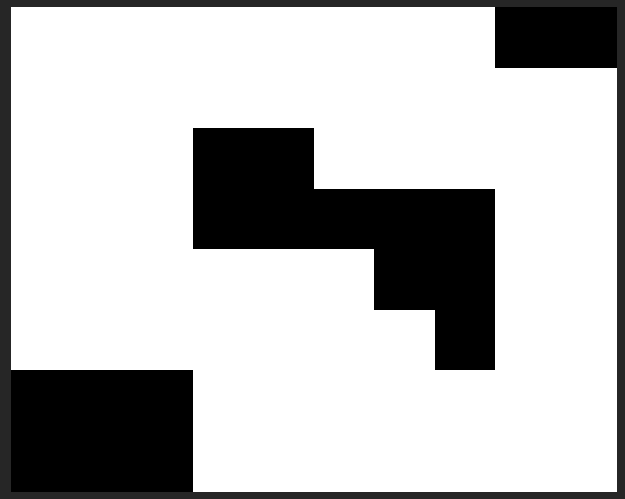
WPA basic steps

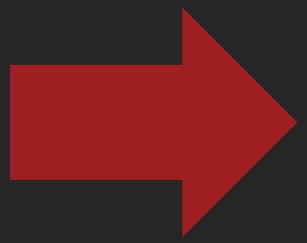
1. Create discretized map of the environment as a X-Y grid (matrix) identifying empty spaces and obstacles
2. Identify robot position (start) and target position (end) within such X-Y grid
3. Fill in the wavefront
4. Choose one of the many available paths

WAVEFRONT PLANNING ALGORITHM

1. **IF you already know the real map**

- Discretize the map as a **X-Y grid matrix** called M
- A rule-of-thumb is to choose the cell size as the dimension of the smallest object in the scene – in many cases, **the size of the robot** will be just fine
- Identify **empty cells as '0' and non-empty ones as '1'**



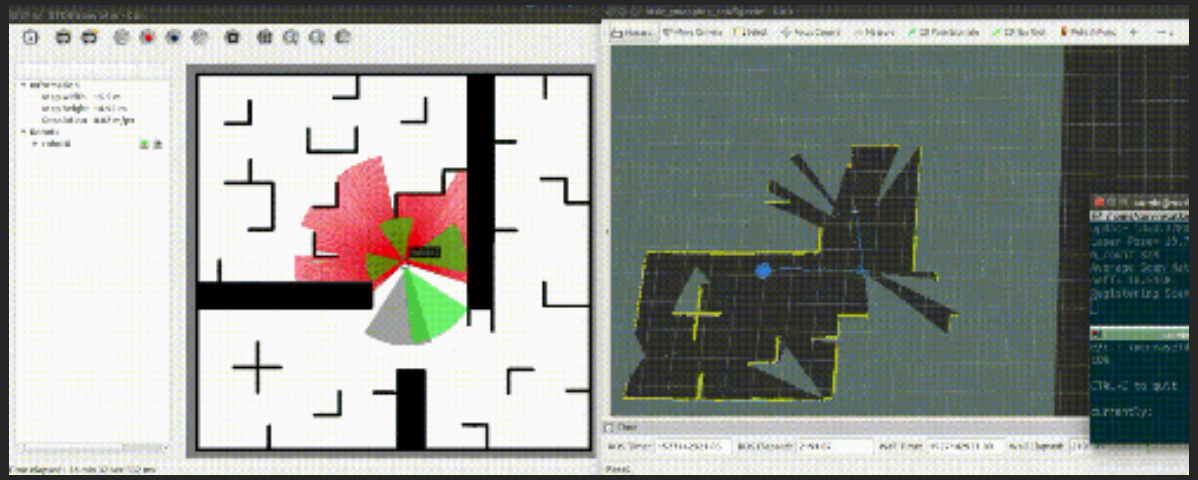


0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0
0	0	0	1	1	1	1	1	0	0
0	0	0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0

WAVEFRONT PLANNING ALGORITHM

2. **IF you DO NOT know the real map**

- **Map** the scenario using one of the many existing ROS methods
  - **Gmapping**: one of the most widely-used SLAM ROS packages to create a 2D occupancy grid map out of **laser and pose data**
  - **Hectormap**: also broadly used, relying entirely on laser data, **without odometry**
- Reshape the obtained grid map (nav_msgs/OccupancyGrid topic) repeating the **same steps as for the known-map situation**



WAVEFRONT PLANNING ALGORITHM

WPA basic steps

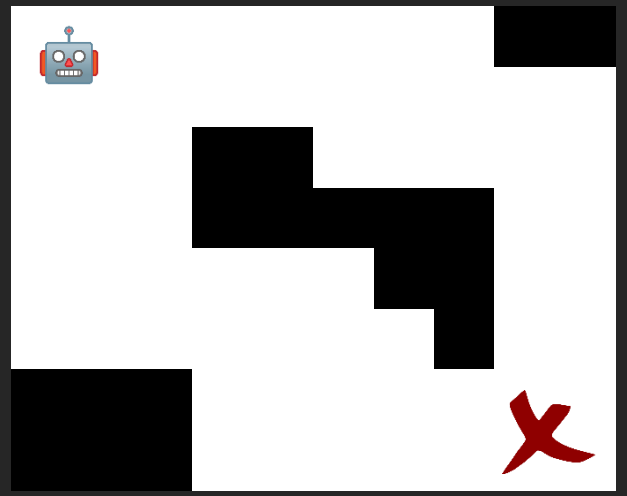
1. Create discretized map of the environment as a X-Y grid (matrix) identifying empty spaces and obstacles
2. Identify robot position (start) and target position (end) within such X-Y grid
3. Fill in the wavefront
4. Choose one of the many available paths

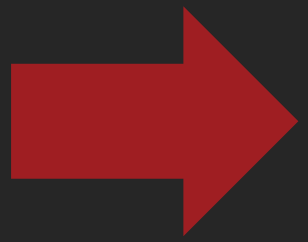
WAVEFRONT PLANNING ALGORITHM

- Calculate the **minimum X and Y distances** between the X and Y grid and both robot position and target position

$$(i_{target}, j_{target}) = \left(\underset{i=1,\dots,n}{\operatorname{argmin}}|x_i - x_{target}|, \underset{i=1,\dots,m}{\operatorname{argmin}}|y_i - y_{target}|\right)$$

$$(i_{robot}, j_{robot}) = \left(\underset{i=1,\dots,n}{\operatorname{argmin}}|x_i - x_{robot}|, \underset{i=1,\dots,m}{\operatorname{argmin}}|y_i - y_{robot}|\right)$$





x \ y	1	2	3	4	5	6	7	8	9	10
1	R	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	1	0	0	0	0	0
4	0	0	0	1	1	1	1	1	0	0
5	0	0	0	0	0	0	1	1	0	0
6	0	0	0	0	0	0	0	1	0	0
7	1	1	1	0	0	0	0	0	0	0
8	1	1	1	0	0	0	0	0	0	T

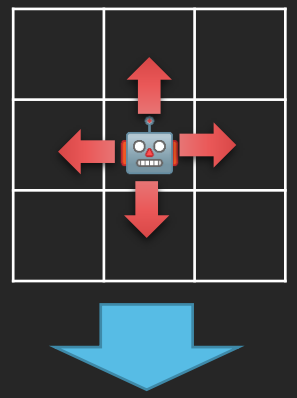
WAVEFRONT PLANNING ALGORITHM

WPA basic steps

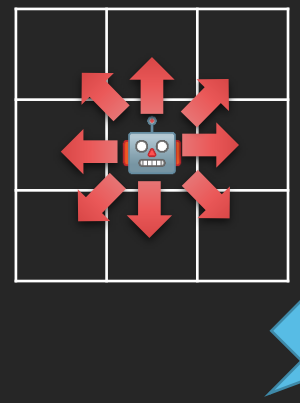
1. Create discretized map of the environment as a X-Y grid (matrix) identifying empty spaces and obstacles
2. Identify robot position (start) and target position (target) within such X-Y grid
3. Fill in the wavefront
4. Choose one of the many available paths

WAVEFRONT PLANNING ALGORITHM

- Create a $c \times 2$ matrix called A identifying the immediate adjacent cells the robot should be allowed to move



$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}$$

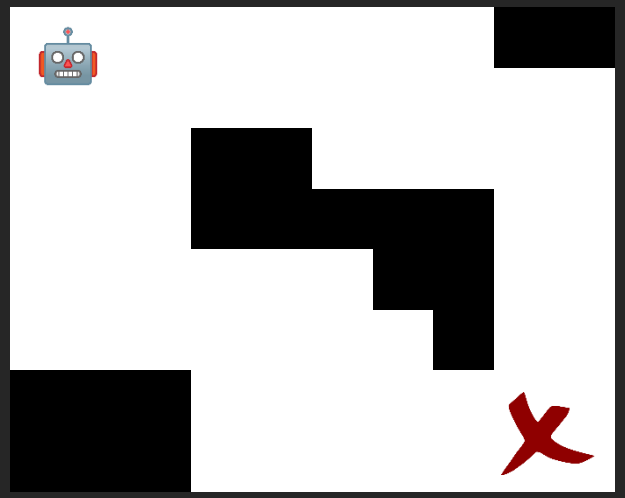


$$A = \begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ -1 & 1 \\ -1 & 0 \\ -1 & -1 \\ 0 & -1 \end{bmatrix}$$

WAVEFRONT PLANNING ALGORITHM

- Create a zero-matrix called C with the same size $n \times m$ as M and initialize the cell at indices (i_{target}, j_{target}) with '1'

$$C = \mathbf{0}_{n \times m}$$
$$C(i_{target}, j_{target}) = 1$$



Matrix C

x \ y	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	1

WAVEFRONT PLANNING ALGORITHM

- Create a matrix called O that will be initialized with the target indices (i_{target}, j_{target}) as a 1×2 array and will be iteratively manipulated, adding and removing new rows accordingly with cells where cost is still not calculated and cells where cost is calculated, respectively

$$O = [i_{target}, j_{target}]$$

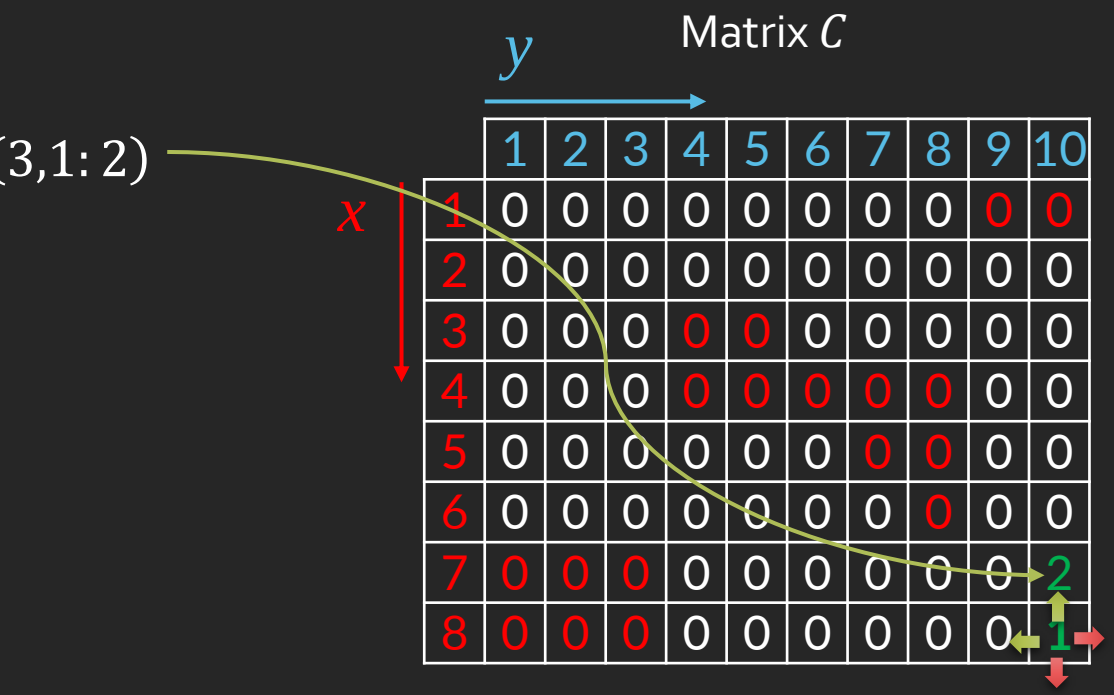
Matrix C

x \ y	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	1

WAVEFRONT PLANNING ALGORITHM

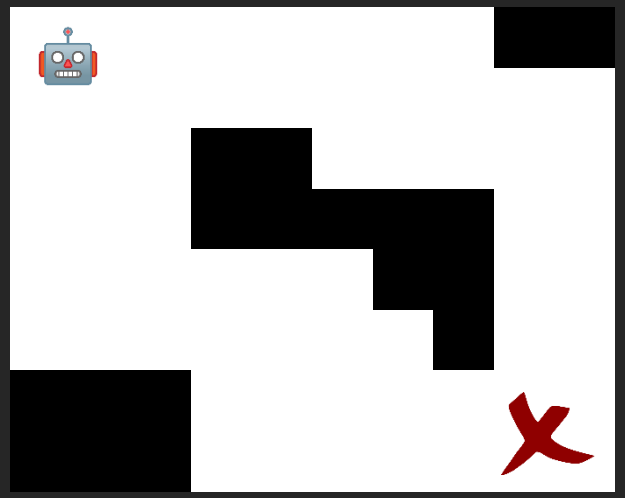
- At each iteration, the idea is to check cell by cell, starting at the target indices, which cell it is next to, according to matrix A (adjacent cells)

$$O(2,1:2) = O(1,1:2) + A(3,1:2)$$



WAVEFRONT PLANNING ALGORITHM

- In the process, one should ignore walls and points outside the scenario, look at cells around the cell previously checked, then count up (+1)

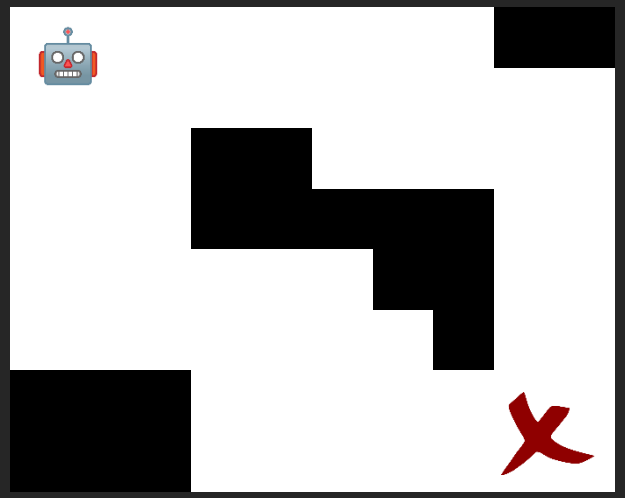


Matrix C

x \ y	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	8	7	6
4	0	0	0	0	0	0	0	0	6	5
5	0	0	0	0	0	0	0	0	5	4
6	0	0	0	0	0	7	6	0	4	3
7	0	0	0	0	0	6	5	4	3	2
8	0	0	0	0	0	5	4	3	2	1

WAVEFRONT PLANNING ALGORITHM

- The main loop that allows to fill in the wavefront will stop when all cells have an associated cost, *i.e.*, when **matrix C is completely filled in and $\mathbf{dim}(O) = \mathbf{0}$**



Matrix C

x \ y	1	2	3	4	5	6	7	8	9	10
1	17	16	15	14	13	12	11	10	0	0
2	16	15	14	13	12	11	10	9	8	7
3	15	14	13	0	0	10	9	8	7	6
4	14	13	12	0	0	0	0	0	6	5
5	13	12	11	10	9	8	0	0	5	4
6	12	11	10	9	8	7	6	0	4	3
7	0	0	0	8	7	6	5	4	3	2
8	0	0	0	7	6	5	4	3	2	**1**

WAVEFRONT PLANNING ALGORITHM

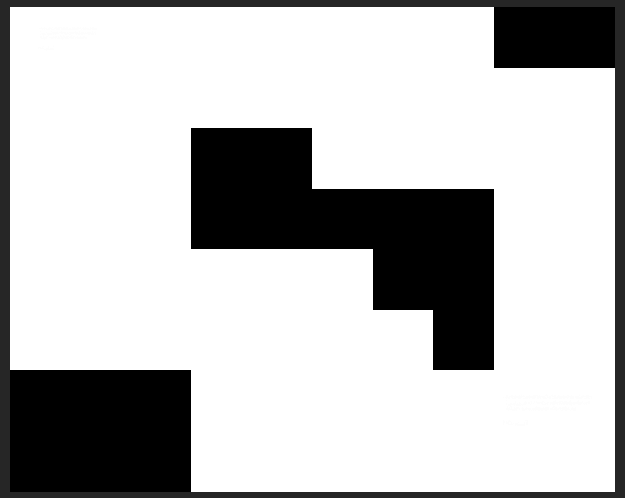
WPA basic steps

1. Create discretized map of the environment as a X-Y grid (matrix) identifying empty spaces and obstacles
2. Identify robot position (start) and target position (end) within such X-Y grid
3. Fill in the wavefront
4. Choose one of the many available paths

WAVEFRONT PLANNING ALGORITHM

- Create a matrix called P that will be initialized with the robot indices (i_{robot}, j_{robot}) as a 1×2 array and will be iteratively manipulated, adding new rows accordingly with cells that reduce cost

$$P = [i_{robot}, j_{robot}]$$



y

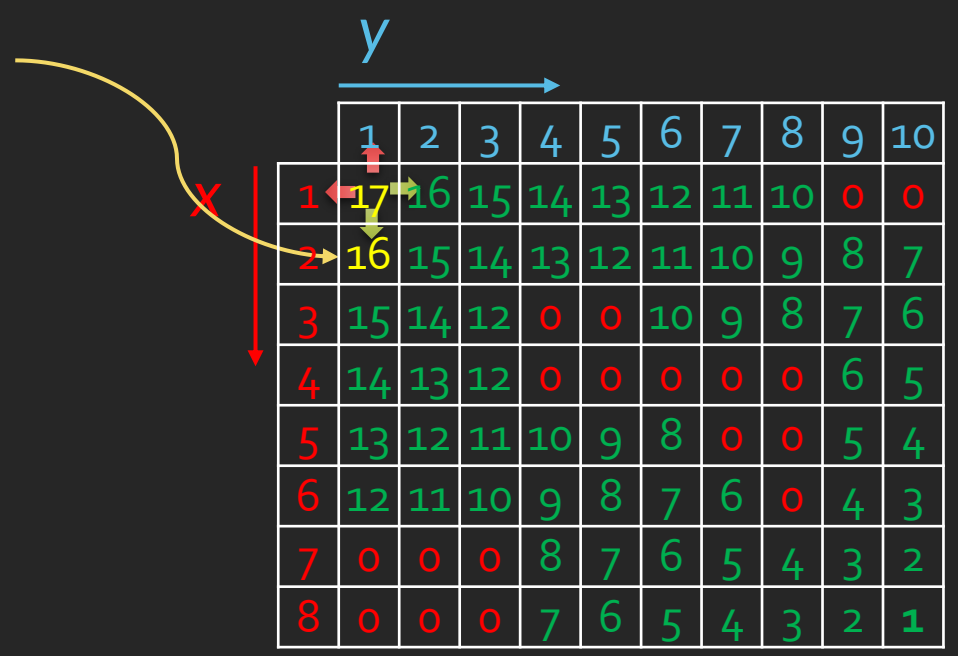
x

	1	2	3	4	5	6	7	8	9	10
1	17	16	15	14	13	12	11	10	0	0
2	16	15	14	13	12	11	10	9	8	7
3	15	14	12	0	0	10	9	8	7	6
4	14	13	12	0	0	0	0	0	6	5
5	13	12	11	10	9	8	0	0	5	4
6	12	11	10	9	8	7	6	0	4	3
7	0	0	0	8	7	6	5	4	3	2
8	0	0	0	7	6	5	4	3	2	1

WAVEFRONT PLANNING ALGORITHM

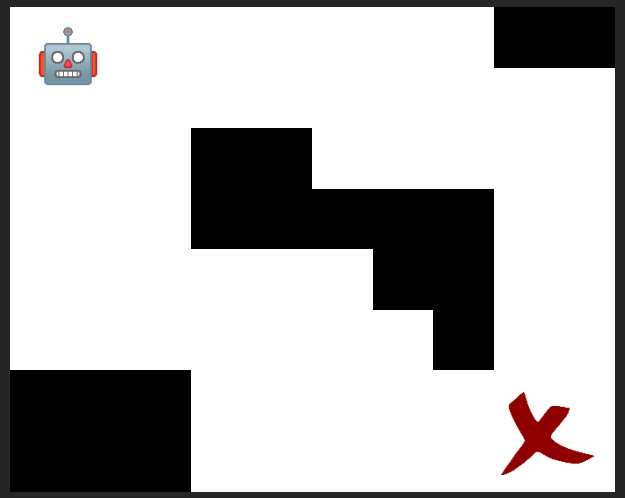
- At each iteration, the idea is to check cell by cell, starting at the robot indices, which cell it is next to, according to matrix A (adjacent cells)
- As before, one should ignore walls and points outside the scenario

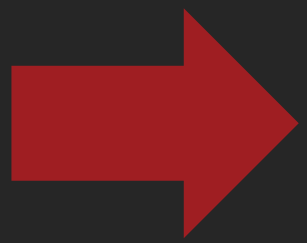
$$P(2,1:2) = P(1,1:2) + A(1,1:2)$$



WAVEFRONT PLANNING ALGORITHM

- The main loop that allows to find the path to the target position will stop when the last row of the matrix *P* is equal to the target indices





y

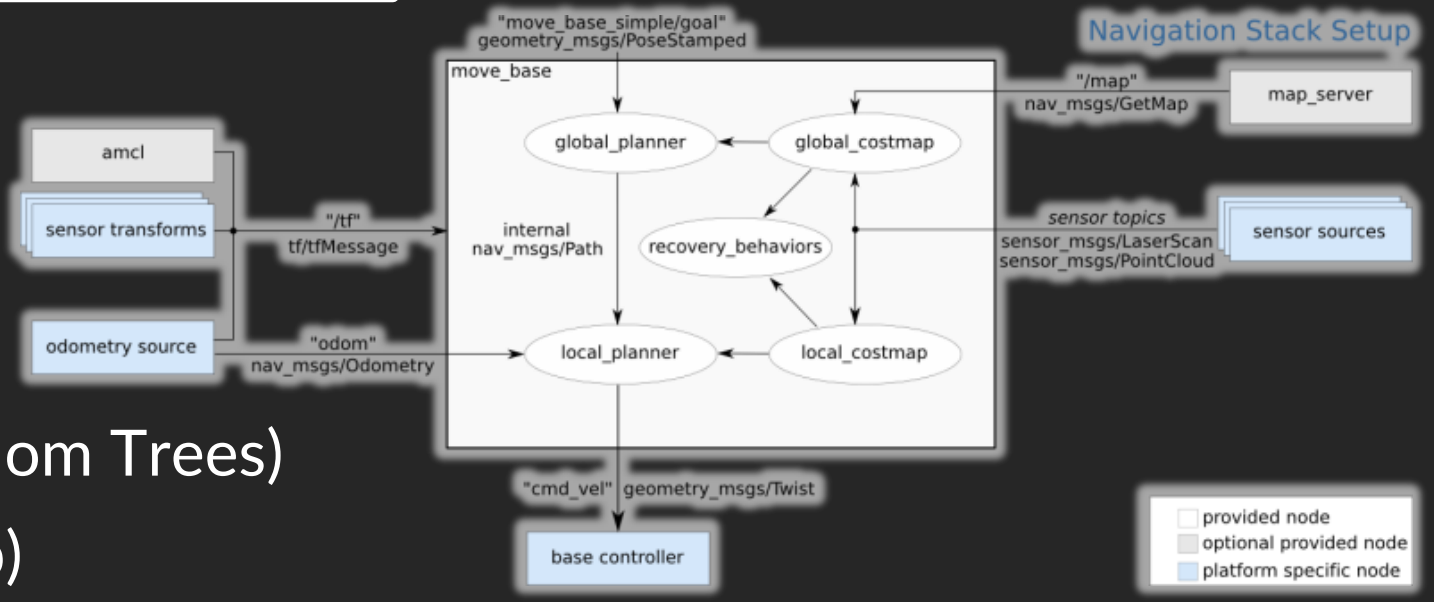
x

	1	2	3	4	5	6	7	8	9	10
1	17	16	15	14	13	12	11	10	0	0
2	16	15	14	13	12	11	10	9	8	7
3	15	14	12	0	0	10	9	8	7	6
4	14	13	12	0	0	0	0	0	6	5
5	13	12	11	10	9	8	0	0	5	4
6	12	11	10	9	8	7	6	0	4	3
7	0	0	0	8	7	6	5	4	3	2
8	0	0	0	7	6	5	4	3	2	**1**

WAVEFRONT PLANNING ALGORITHM

Alternatives to WPA:

- A* (A-star)
- Dijkstra's Algorithm
- RRT (Rapidly-exploring Random Trees)
- PRM (Probabilistic RoadMap)



ROS Navigation Stack: http://wiki.ros.org/move_base

ROS Flexible Navigation Stack: http://wiki.ros.org/move_base_flex

WPA (compared to others)

The good

- Simplicity
- Complete
- No priority queue
- Uniform cost
- Easy path extraction

The bad

- No heuristics
- Space intensive
- Not optimal
- Slower for large maps

ROBOT-ANT: PRO MAZE SOLVER

ROBOT-ANT PRO

Pseudocode

Inputs:

- Robot start position (x_{start}, y_{start})
- Maze exit position (x_{target}, y_{target})
- Robot position (x_{robot}, y_{robot})
- Occupancy grid map M
- Occupancy grid X-Y axis (x_{axis}, y_{axis})

Outputs:

- Path matrix P
- Cost matrix C

ROBOT-ANT PRO

Pseudocode

1. $(i_{target}, j_{target}) = \left(\underset{i=1,\dots,n}{\operatorname{argmin}} |x_{axis}(i) - x_{target}|, \underset{i=1,\dots,m}{\operatorname{argmin}} |y_{axis}(i) - y_{target}|\right)$
2. $(i_{start}, j_{start}) = \left(\underset{i=1,\dots,n}{\operatorname{argmin}} |x_{axis}(i) - x_{start}|, \underset{i=1,\dots,m}{\operatorname{argmin}} |y_{axis}(i) - y_{start}|\right)$
3. $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}$
4. $C = \mathbf{0}_{n \times m}$
5. $C(i_{target}, j_{target}) = 1$
6. $O = [i_{target}, j_{target}]$

ROBOT-ANT PRO

Pseudocode

7. $while\ (\dim(O) > 0)$
8. $\quad for\ each\ r\ row\ of\ A$
9. $\quad\quad aux(1,1:2) = O(1,1:2) + A(r,1:2)$
10. $\quad\quad if\ 1 \leq aux(1,1) \leq \dim(x_{axis})\ and\ 1 \leq aux(1,2) \leq \dim(y_{axis})\ and\ M(aux) = 0\ and\ C(aux) = 0$
11. $\quad\quad\quad C(aux,1:2) = C\big(O(1,1:2)\big) + 1$
12. $\quad\quad\quad O(\dim(O) + 1,1:2) = aux$
13. $\quad O = O(2:\dim(O),1:2)$

ROBOT-ANT PRO

Pseudocode

14. $P = [i_{start}, j_{start}]$
15. $while\ (P(\dim(P), 1{:}2) \neq [i_{target}, j_{target}])$
16. $\quad aux\_c = C(P(\dim(P), 1{:}2))$
17. $\quad for\ each\ r\ row\ of\ A$
18. $\quad\quad aux(1,1{:}2) = P(\dim(P), 1{:}2) + A(r, 1{:}2)$
19. $\quad\quad if\ 1 \leq aux(1,1) \leq \dim(x_{axis})\ and\ 1 \leq aux(1,2) \leq \dim(y_{axis})\ and\ C(aux) \neq 0$
20. $\quad\quad\quad if\ C(aux) < aux\_c$
21. $\quad\quad\quad\quad P(\dim(P) + 1, 1{:}2) = aux$
22. $\quad\quad\quad\quad break$

ROBOT-ANT PRO

Pseudocode

23. $for\ each\ t\ row\ of\ P$
24. $\quad do$
25. $\quad\quad GoTo\left(x_{axis}(P(t,1)),\ y_{axis}(P(t,2))\right)$
26. $\quad\quad (i_{robot}, j_{robot}) = \left(\underset{i=1,\ldots,n}{\operatorname{argmin}}|x_{axis}(i) - x_{robot}|, \underset{i=1,\ldots,m}{\operatorname{argmin}}|y_{axis}(i) - y_{robot}|\right)$
27. $\quad while\ ([i_{robot}, j_{robot}] \neq [P(t,1),\ P(t,2)])$

#7 TASK

#7 TASK

- Download the Stage/ROS *robotcraft_maze* framework to your PC, and create a "maze_ BASICsolver.cpp" (or "maze_BASICsolver.py") and "maze_ PROsolver.cpp" (or "maze_PROsolver.py") codes to send velocity commands to your robot and solve the maze without hitting any walls. **At this stage, assume that you have a map!**

- Two behaviors are expected:

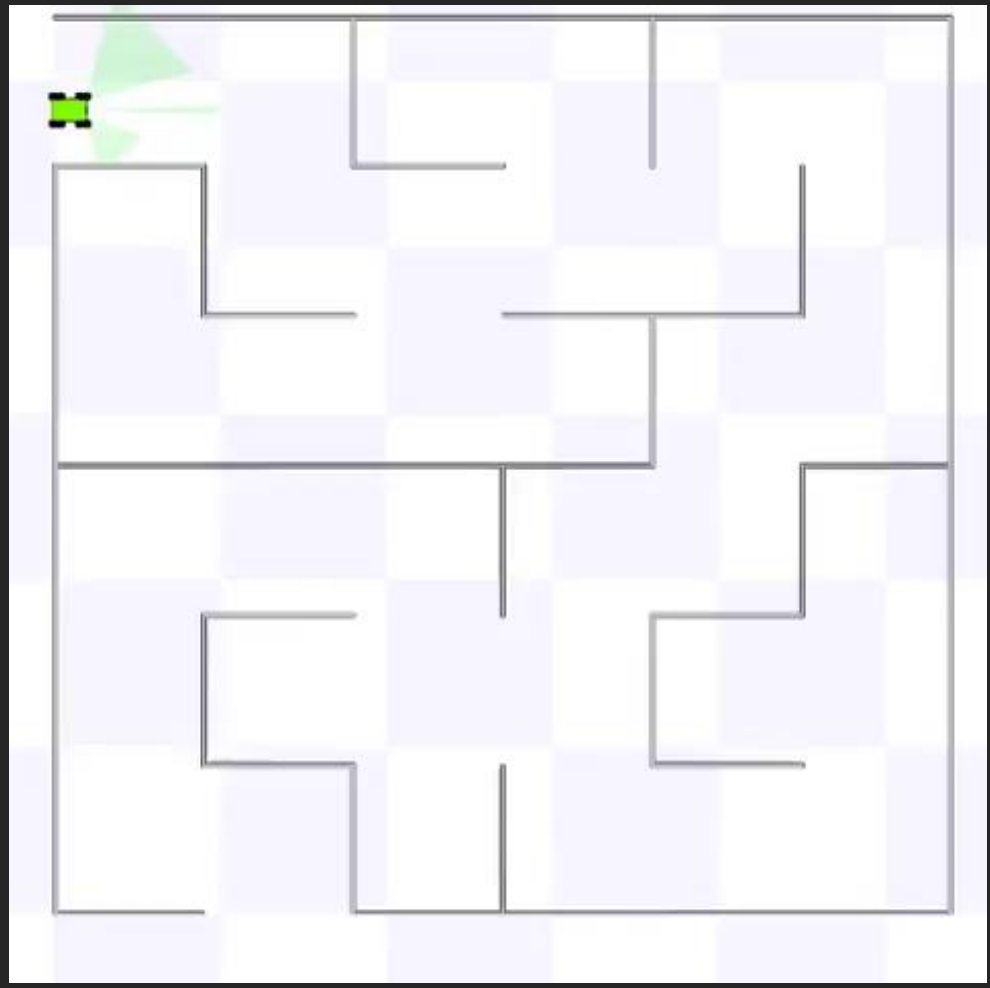
  - maze_BASICsolver – Follow the FSM previously presented and solve the maze using a wall-follower approach. In this scenario, only infrared sensors can be used for navigation - **the LiDAR cannot be used for navigation** (nor positioning). Nevertheless, **you should use the LiDAR to map the environment for the PRO maze solver**.

  - maze_PROsolver – Follow the WPA previously described and solve the maze by choosing the optimal path. In this scenario, the map retrieved from the BASIC maze should be used. **The LiDAR should be employed for navigation (and positioning)**.

#7 TASK

- **Video (from 2017)**: https://www.youtube.com/watch?v=2B_4M2gSJ1M

- **Evaluation**:
  - Solve the maze BASIC (35%)
  - Solve the maze PRO (35%)
  - Create launch files (15%)
  - Create proof videos (15%)



#7 TASK

Instructions:

1. To make it easier, download the *robotcraft_maze* framework into your Workspace:

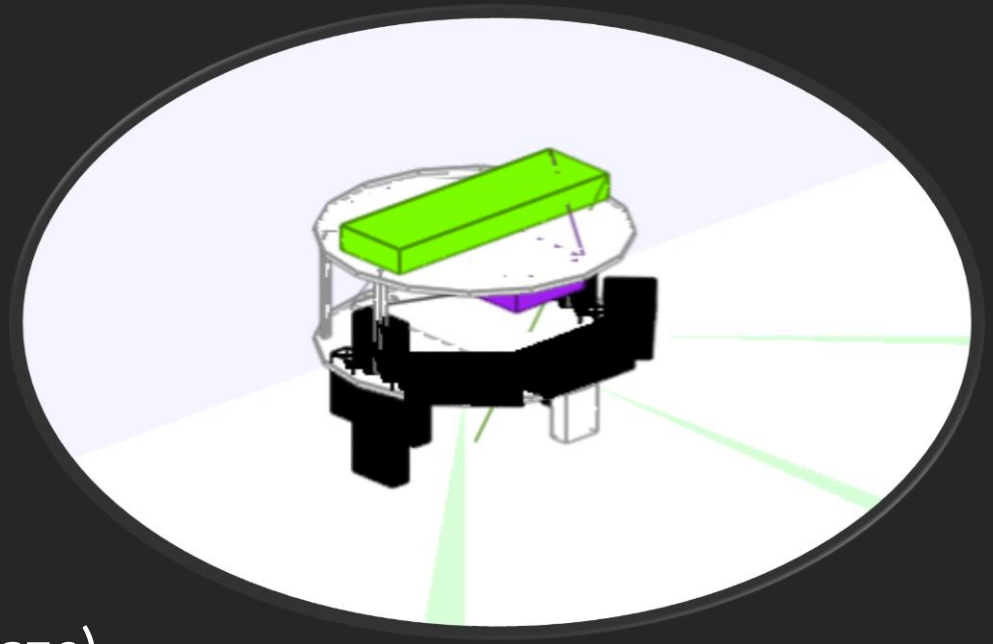
   ```
   $   roscd; cd src;
   $   git clone https://github.com/ingeniarius-ltd/robotcraft_maze
   ```

2. Create the *maze_BASICsolver and maze_PROsolver* (.cpp or .py) in the “src” folder to solve the maze.

3. Compile, and include your nodes in the *maze.launch* file (inside the “launch” folder). You can create a launcher for each node or call both nodes simultaneously.

4. Start the launch file (launching stage + your maze_solver nodes).

#7 TASK



Submission:

- Deadline: **26th August, until 23:59**
- Submission file type: .zip
- Submission file contents:
  - ✓ Full ROS package (folder *robotcraft_maze*)
  - ✓ Video of the robot solving the maze (.ogv or .mp4)

FINAL TASK

After solving the maze in simulation, your group will then move on to solve the maze in the **real world** using the RobotCraft robot developed during the summer course. A report containing all the work you did should be submitted until the 29th August, at 23h59.

The maze solving strategy should be **tuned for your own robot** and will be tested in the final competition, on the 30th August!

Remember:

You should only move on to the real deal when you prove to be capable of solving the maze in simulation.

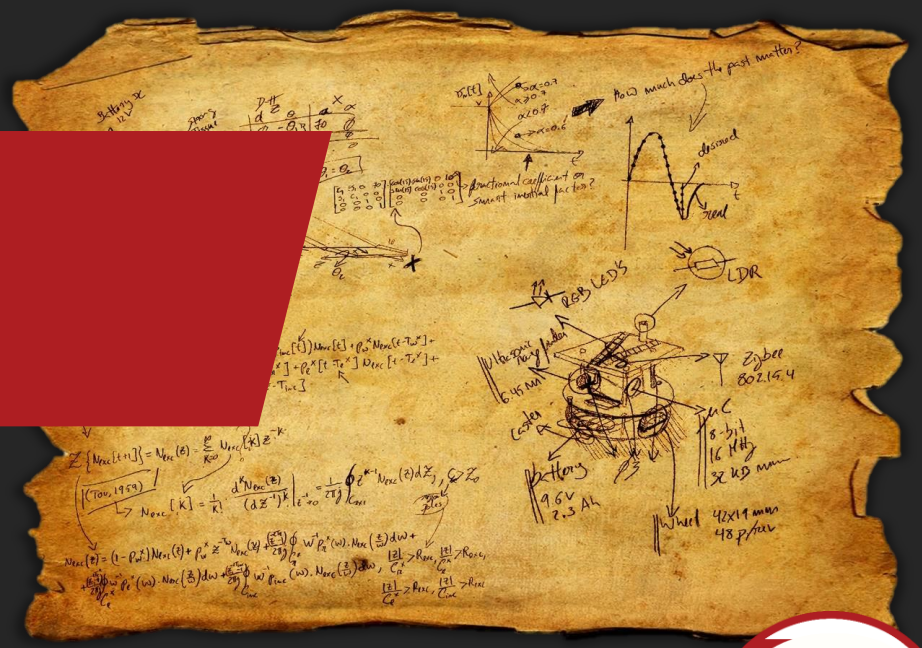
CONCLUSIONS

- A brief introduction to Artificial Intelligence was provided, focusing on biologically-inspired models in robotics and, ultimately, finite-state machines

- Two different approaches, namely FSM-based and WPA-based approaches, were presented and preliminarily delineated

CRAFT #7

THANK YOU



Micael S. Couceiro micael@ingeniarius.pt

Beril Yalcinkaya beril@ingeniarius.pt

19/08/2024