Distributed Systems:
Crossing Intersection with Autonomous Vehicles

Antonio Toncetti
Gabriele Venturato

DMIF, University of Udine, Italy

August 13, 2019

Abstract

The aim of this project is to provide an implemented solution to the problem of autonomous vehicles crossing an intersection. Although the solution relies on some simplifications, it can be further elaborated to work in a real-case scenario.

The solution proposed in this report is meant to be more general and as modular as possible, in order for it to be possibly extended in a concrete situation.

Chapter 1

Introduction

The problem to solve is the one in which some autonomous vehicles have to cross an intersection without being involved in road accidents.

The idea is to solve the problem for a generic intersection. The autonomous vehicles can't rely on a central server, they have to cooperate with each other to cross the intersection by taking decisions which ensure a *fair* and *safe* policy. In particular there are two components in the proposed solution: vehicles and the environment. The latter is necessary in this context in order to simulate sensors that are usually inside autonomous vehicles which allows them to interact with the environment (e.g. proximity sensors, GPS, cameras, etc. . .). The system is *fault tolerant*, but neither byzantine processes nor cybersecurity hazards are taken in consideration, for simplification purpose.

The report starts by analysing the project: Chapter 2 is devoted to functional and non-functional requirements, and system assumptions too. Chapter 3 contains the description of the general architecture, and specific algorithms.

Following chapters aim to describe details of the implementation, and validation w.r.t. requirements.

Chapter 2

Analysis

This chapter describes in detail some fundamental assumptions about the system, as well as functional and non-functional requirements.

2.1 Assumptions

Some general considerations are here presented. First of all it's assumed a situation in which each autonomous vehicle knows its destination and the roads it is going to travel, since we can assume that each autonomous vehicle has a GPS device on board.

Moreover it's assumed, for sake of simplicity, that all vehicles have the same dimension — or better: that each vehicle can fit into a single position of the internal model used to represent the roads. Moreover, common physical quantities (like weight, speed, acceleration, etc...) are omitted. Instead, the autonomous vehicles move in unit steps governed by the internal model.

Further assumptions are:

- *Faults*: at any moment a failure can arise in vehicles — software or mechanical. A mechanical failure doesn't compromise the software abilities, but a software failure implies an hardware failure. Since we can assume that if the software fails, the autonomous vehicle stops and goes in "emergency mode". It's also assumed that if a crash happens in queues, there's enough space for the other vehicles to overtake the faulty one; but if the crash happens inside the crossroads it must be managed by removing the faulty vehicle with a tow truck (or something similar).

- *Moves*: the path that a vehicle can take inside the interception is predefined and known a priori. From each position there's a unique path to each destination, and no one can modify or choose a different path once decided. A visual representation of this assumption can be found in Figure 2.1, in which green dots represent positions occupied while crossing, and light-blue dots are the destinations. Once a vehicle reaches its

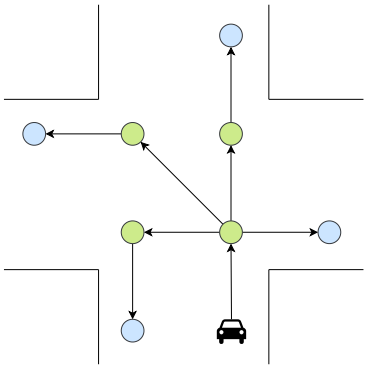

Figure 2.1: Example of predefined movements that a vehicle can follow in crossing an intersection.

destination, it is considered "satisfied", and can not fail anymore, it can only move forward on its own way. Reversing is not allowed.

- *Message Delivery*: there exist an upper bound time in message passing between two nodes. This is justified by the fact that all connection are direct, without routers or broker that can cause bottle-necks or network congestion.

2.2 Functional requirements

The solution provides two main modules: *vehicle* and *environment*.

Vehicle

Each vehicle has an internal state at any moment. Moreover he:

- knows its position and what happens around, through the environment (in a real context this is handled by sensors like GPS and proximity sensors): it can pick up if there are other vehicles trying to cross, or if there is a vehicle in front or behind it;
- communicates with its neighbours by sharing: the direction it wants to travel, its internal state, the next position he is about to reach;
- if it's in the queue, it knows nothing other than its internal information and that it is taking part in a queue with someone before and possibly behind, it can communicate only with these two;
- if it's in the head of the queue, it can connect with other vehicles (that are the head of other queues) that are crossing the intersection;

Environment

The environment represent the vehicle's ability to use its sensors. It knows the intersection shape and dimension; it knows vehicles approaching the crossroads and the ones in the queues; it provides vehicles with all the information they need to safely manoeuvre within the environment.

2.3 Non functional requirements

The system doesn't handle byzantine processes nor cybersecurity issues.

- *Safeness*: there can't be more than a vehicle in the same position at the same time;
- *Liveness*: if a vehicle approaches the intersection and is waiting to cross it, eventually it will cross it;
- *Fairness*: if a vehicle approaches the intersection and is waiting to cross it, there exists a bound to the waiting time;
- *Fault Tolerant*: the system is tolerant to hardware and software failures;
- *Distributed*: there is no central server, vehicles have to coordinate each other.

Chapter 3

Project

This chapter is devoted to the description of the general architecture, and specific algorithms.

3.1 Logical architecture

The solution leans on two main modules, and a secondary one for tests.

- *Vehicle*: it is represented as a finite state automaton. A graphical representation can be seen in Figure 3.1. Each state represents a situation in which the vehicle can find itself. A vehicle is in Waiting state when it is in the queue and it's not the first. It's in Ready state when it's the first in its queue. It's in Election when an election is needed but the current leader has not yet finished the crossing. Crossing, Crossed, and Crash states are self-explanatory.
- *Environment*: it represents the physical environment accessible through autonomous vehicle sensors.
- *Random Generator*: it is a module aimed to (pseudo-)randomly generate vehicles that approach the intersection, and it is used to test and validate the solution proposed.

3.2 Protocols and algorithms

This section describes the algorithm to solve the core part of the project. Moreover, some peculiar situation sequence diagrams are presented.

Intersection Crossing Algorithm

Vehicles queue up along the roads leading to an intersection awaiting for their turn to cross. The first vehicle of a queue is the vehicle at the verge of entering

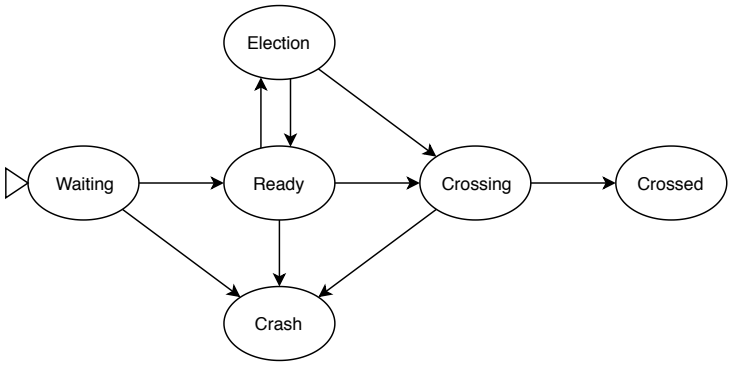

Figure 3.1: Vehicle as a finite state automaton.

the intersection and we will call such vehicle a *participant* of the intersection crossing algorithm.

Note that this algorithm applies to generic intersections. Moreover, it trivially guarantees the *fairness* and *liveness* requirements, thanks to its round robin characteristic.

The algorithm maintains the following invariant through execution:

- one vehicle per time can cross the intersection
- only the leader vehicle can cross the intersection

Algorithm description:

1. The participants (first vehicles in their respective queues) are willing to cross the intersection;
2. They start a slightly modified version of the Bully Algorithm to elect a leader: it terminates with the leader L, and the next leader L', which is the first vehicle after him in a clockwise manner;
3. L begins to cross the intersection;
4. After L has successfully crossed the intersection, the participants are informed;
5. L' identifies the next leader L'' by choosing the first vehicle after him in a clockwise manner;
6. The lead passes to L' informing every participant that it's the new leader;
7. The algorithm repeats from 3, where $L = L'$ and $L' = L''$.

Additional details:

- After the leader election, every participant knows who is the current leader, and who is the next one;
- Once the leader starts crossing, the vehicle behind it (if any) becomes a new participant and asks to join the algorithm. Only the leader answers, providing its identity.
- Once the vehicle behind the leader move forward, it's assumed that all the vehicles in the queue move forward too.

A leader election starts only when new participants don't receive an answer when asking to join the algorithm. In particular: when one or more vehicles join an empty intersection, i.e. when they become participants and no prior participants are available, or when the candidate leader fails before becoming the new leader.

Managing abnormal cases

All participants monitor the leader and the candidate leader, if the leader fails, everyone knows who the next leader is (see 2. in the algorithm description).

- If the leader fails, the vehicle in clockwise order after him becomes the new leader — identifying also the next candidate leader — and the algorithm restarts from 3.
- If a (non-crossing) participant fails, the vehicle behind it in the queue becomes immediately a participant, justified by the assumption that in queues there's enough space to overtake the faulty vehicle.
- If the leader fails while crossing, a timeout T is needed before it is removed from the intersection. Meanwhile a new election can start, and the new leader can start crossing, but it has to stop if the position in front of him is still occupied from the previous faulty leader.
- If the candidate leader fails before becoming the new leader, a new election must be started *after* the current leader have completed the crossing.

Sequence Diagrams

This section presents some sequence diagrams of relevant case situations.

Normal execution of Intersection Crossing Algorithm

The diagram in Figure 3.2 represent the normal execution of the algorithm described above. It starts at the end of the Bully Algorithm, with the AV_4 comunication it is the leader and the candidate leader is AV_1. It's assumed the queues are enumerated from 1 to 4 in clockwise order.

The leader crosses the intersection by asking the environment if the position in front of it is free, and continues to cross until it reaches its destination.

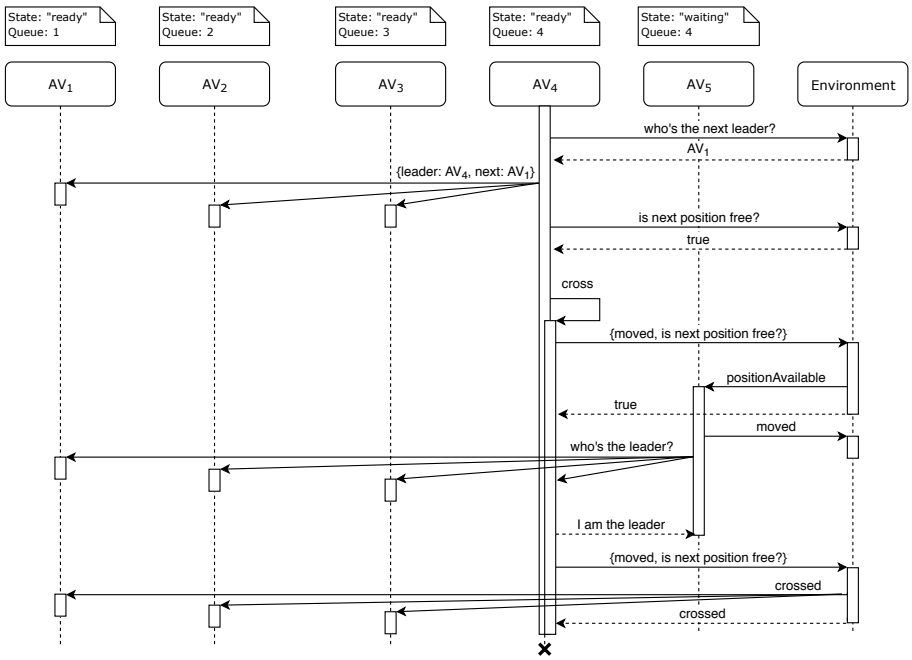

Figure 3.2: Sequence diagram on a normal execution of the Intersection Crossing Algorithm described above.

It's important to note that when the leader starts crossing, the environment notifies AV_5 — which is the one following AV_4 in the queue — that it can move one position forward. Then AV_5 is at the verge of the intersection and can communicate with the other participants in order to compete for the crossing.

Dealing with AV failures

Leader failures don't cause troubles if they are detected by their relative candidates. Once a candidate detects the failure it automatically ascends to leader and notifies all participants. The normal algorithm flow is restored. The situation is represented by the sequence diagram in Figure 3.3

Another delicate situation is the represented in Figure 3.4. The leader fails and before the candidate detects the failure, a new participants asks to join the algorithm. This situation is solved by the assumption that there exist a time bound to message delivery, therefore the new participant knows that in a prefixed amount of time it can receive the message of the new elected leader.

The last situation in Figure 3.5 represents a failure of the candidate leader. The situation is restored when the leader detects the failure and assigns a new candidate leader.

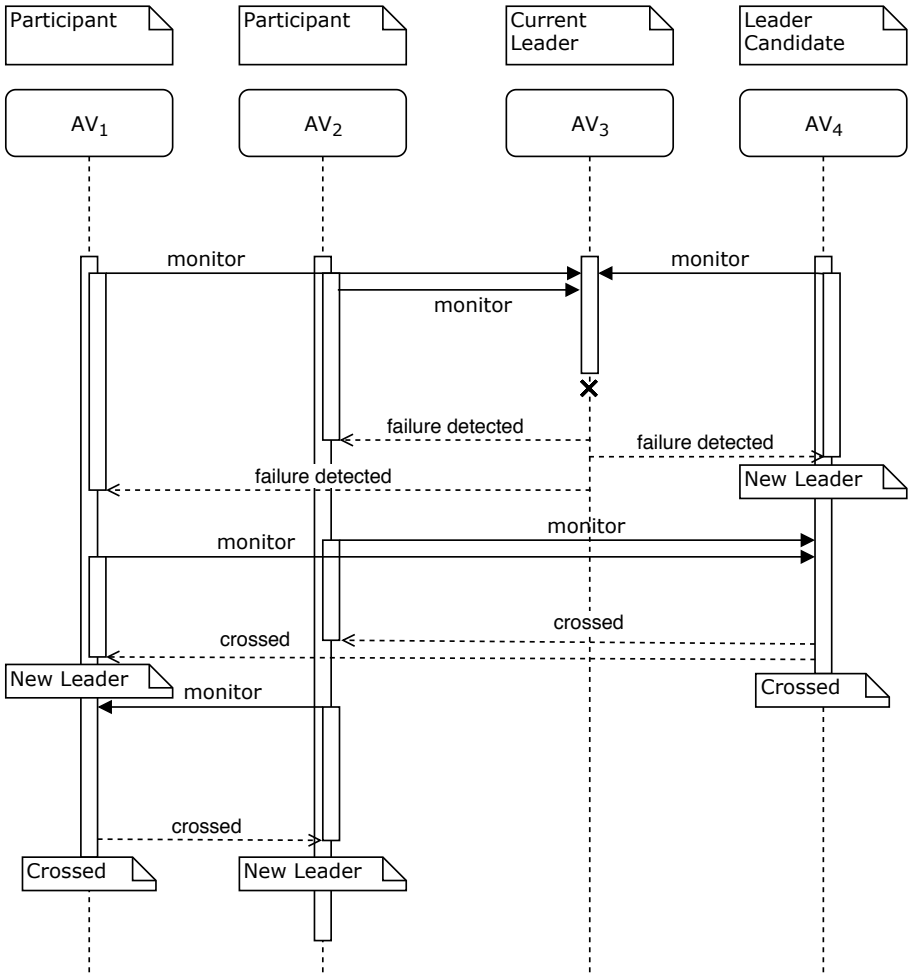

Figure 3.3: Sequence diagram representing a generic leader failure.

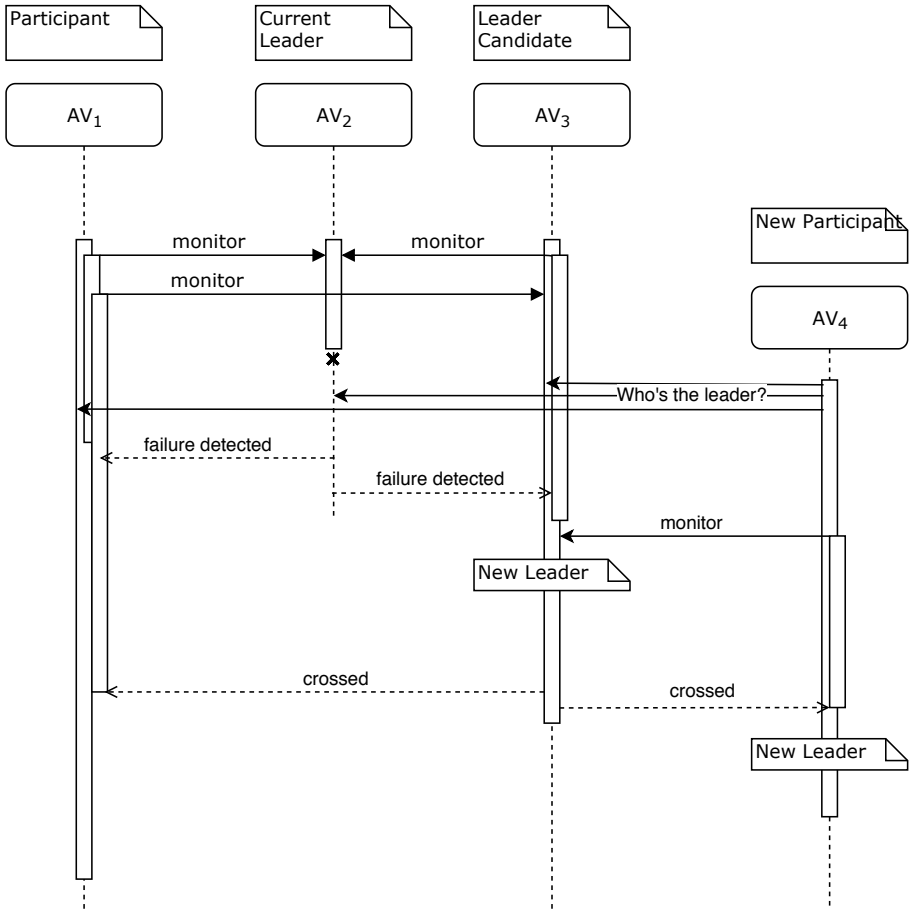

Figure 3.4: Sequence diagram of a leader failure happening at the time a new participant joins.

3.3 Physical architecture and deployment

The architecture is quite simple, a graphical representation is displayed in Figure 3.6. Each Autonomous Vehicle (AV) is a physical node. The Environment represents the world sensed by AVs, so it's not actually a physical node. AVs can ask the environment for information, and they can communicate with their neighbours.

3.4 Development plan

The proposed solution focuses on a very simplified version of what a real solution could be. In particular the aim is to produce a core solution which satisfies

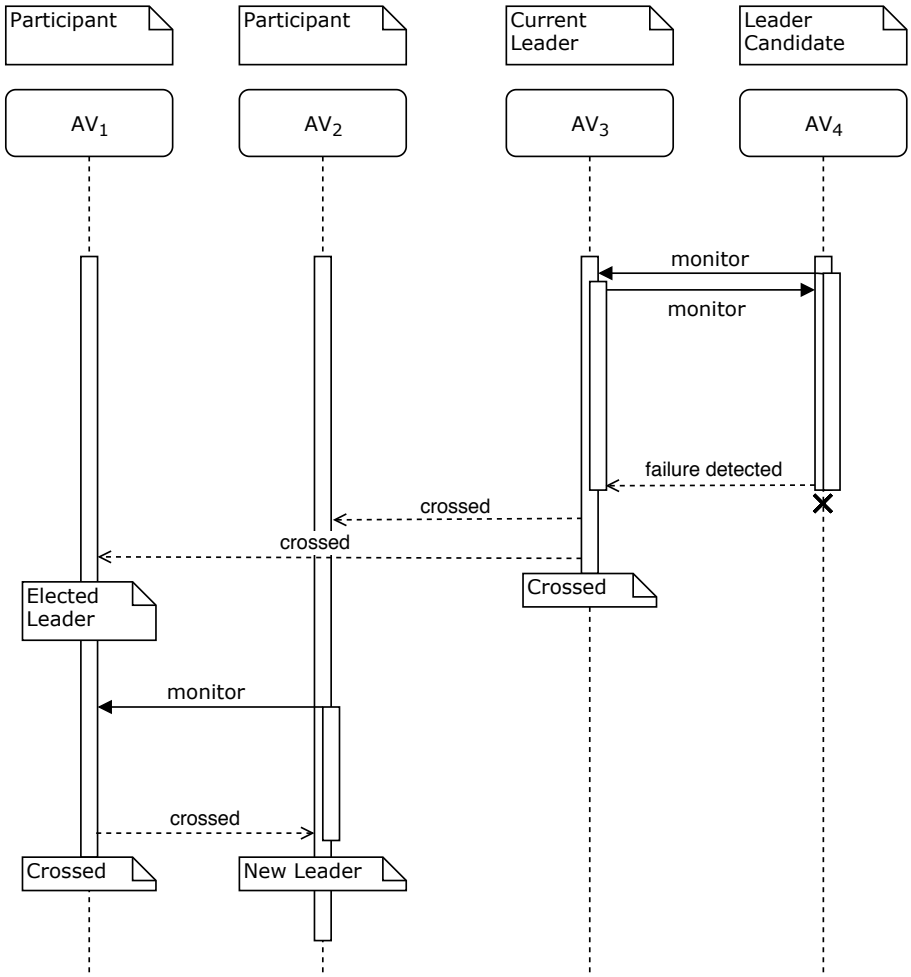

Figure 3.5: Sequence diagram of a candidate leader failure.

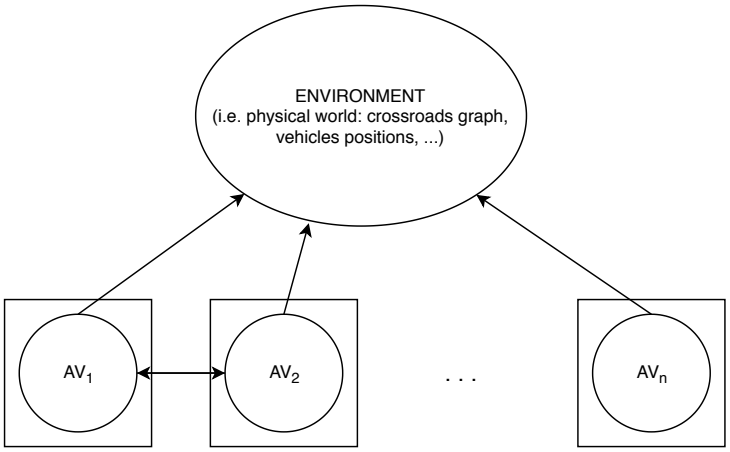

Figure 3.6: Physical architecture: Environment and Autonomous Vehicles (AV).

the requirements but that also doesn't contain more than necessary assumptions, and in particular that doesn't rely on non realistic ideas that can lead future development to a blind spot.

If the core simplified part will be easily developed and validated, then some advancement can be pursued, for example:

- Add vehicles with priorities, e.g. emergency vehicles;
- Dynamically modify paths inside the crossroad if some position becomes unavailable;
- Let more than one vehicle per time cross the intersection, maybe with an interface to a planner in order to find the best route (exploiting epistemic planning could be an interesting option).