Highway simulator: Monitoring update

May, 4th 2017

Federico Massa, Adriano Fagiolini, Antonio Bicchi, Lucia Pallottino

Why we needed an update

What we were doing

- We were trying to monitor the behaviour of a vehicle with a discrete states automaton running different controllers.
- The recognition of the discrete state was made via a forward prediction of the vehicle's continuous state, that was compared with the controller model that we supposed to know (first mayor problem)
- The monitoring itself was achieved by evaluating the rules (that we also supposed to know from the constructor, second mayor problem) that determined the transition between discrete states. Uncertain cases were due to the partial vision of the observer with respect to the monitored vehicle's surroundings.

Why we needed an update

What we do now

- We monitor the behaviour of a vehicle based on the abstract features of its movements. This way, we can monitor a vehicle regardless of the dynamic model (it could also be manually-driven)
- Instead of the recognition of the discrete state, we try to recognize the abstract **action** that the vehicle is pursuing.
- The monitoring is now based on the definition of some basic **social rules**, that are defined in an abstract way as the list of conditions that make a certain action forbidden.

Requirements

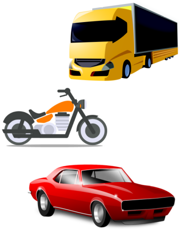
To achieve this, we only need:

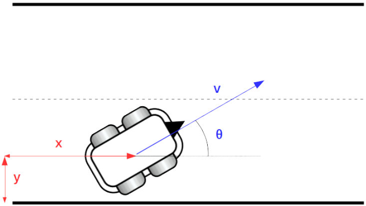
- The sensor data of all the vehicles visible by the observer (x, y on the highway plane, at least);
- Some kind of vehicle ID to follow its trajectory;
- A list of possible actions that the monitored vehicle can do;
- A specification of the social rules that are being monitored.

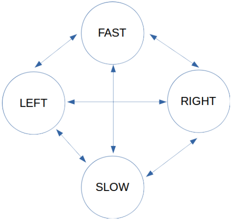
Configuration

In the configuration file it is possible to choose, for every vehicle in the environment:

- Initial continuous and discrete state;
- Geometrical parameters (Type of vehicle);
- Dynamic model (Physical Layer type);
- Automaton type;
- Magnitude of sensor errors.



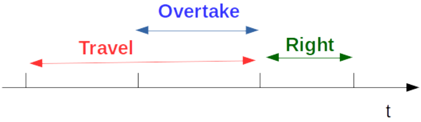




First step: the action recognition

An action is abstractly defined as a vehicle behaviour in a **finite interval of time**. Examples of it include

- *Travel*
- *Left/Right lane change*
- *Left/Right overtake*
- *Brake*



We can also imagine a more sophisticated action recognition system to include more complicated actions, like *slow left lane change* or *abrupt left lane change*.

What the monitor needs to know:

The monitor must have a list of possible actions that the observed vehicle can do. The action recognition system is now **independent** from the monitored vehicle's dynamic model.

How the action recognition works

In order to recognize an action, the monitor needs to:

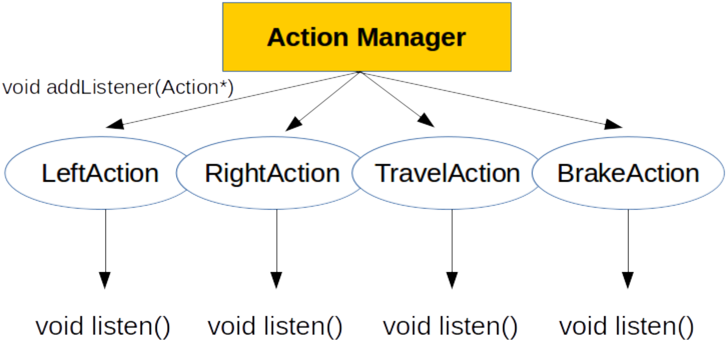
- Identify the observed vehicle;
- Keep track of the visible vehicles' sensor data for a finite interval of time.

Each action is recognized independently $\rightarrow$ concurrent actions are possible.

An **Action Manager** ($\mathbf{\Phi}$) continuously listens to each action (a_1, a_2, ...) and verifies if the trajectory of the monitored vehicle ($q_1, ..., q_N$) is compatible with that action. If it is, the action is *triggered*.

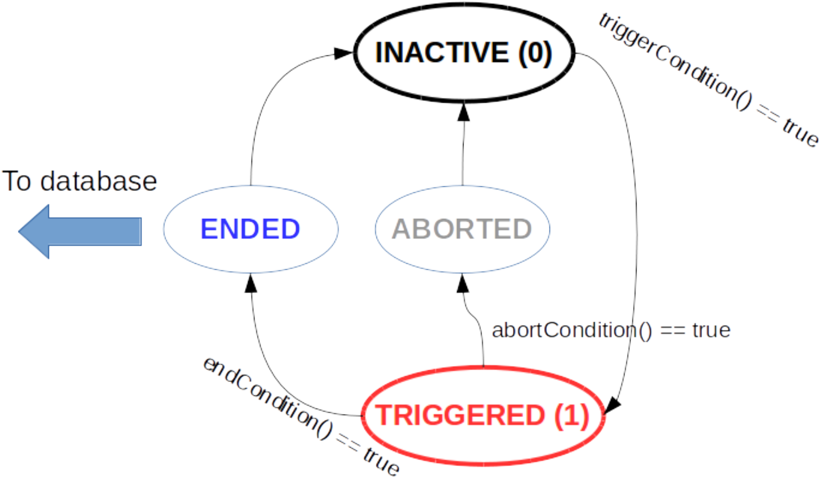
The Action Manager structure

The ActionManager initializes several *listeners*, i.e. actions that the system can recognize.



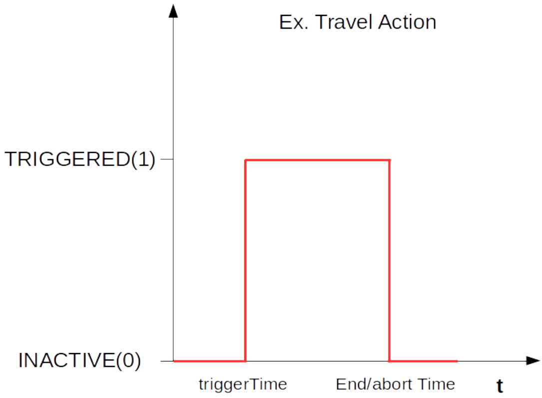
listen() cycle

If an action is registered in the ActionManager listener list, it enters a cycle:



listen() cycle

If an action is registered in the ActionManager listener list, it enters a cycle:



LeftAction example

We use a N-point trajectory $(q_1, ..., q_N)$ and the following parameters:

- Δy: Transversal distance covered in the recorded trajectory $(y_N - y_1)/N$;
- σ_y: Standard deviation of transversal position.
- f: constant between 0 and 1 (needs tuning);
- $\Delta y_{tolerance}$: transversal position tolerance on central position (needs tuning);
- ϵ_y: transverse position sensor error.

triggerCondition() $\Delta y > f \cdot v_{max} \cdot \Delta t_{sim.step}$

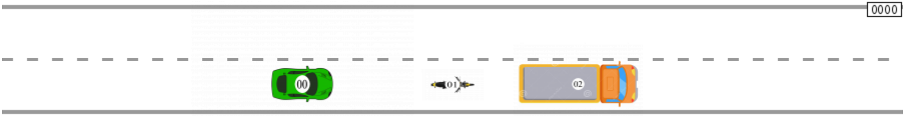
$|y_1 - y_{initLane}| < \Delta y_{tolerance} + 3 \cdot \epsilon_y$

endCondition() $|\mu_y - y_{targetLane}| < \Delta y_{tolerance} + 3 \cdot \epsilon_y / \sqrt{N}$

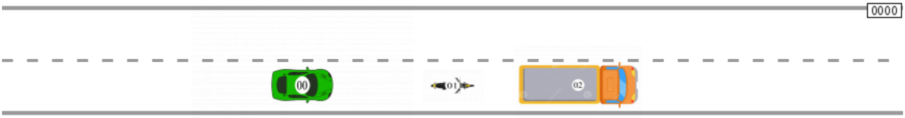
$\sigma_y < \epsilon_y$

abortCondition() After trigger time, the vehicle has disappeared.

LeftAction video

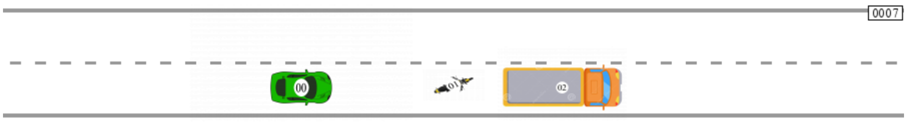


Complete action example



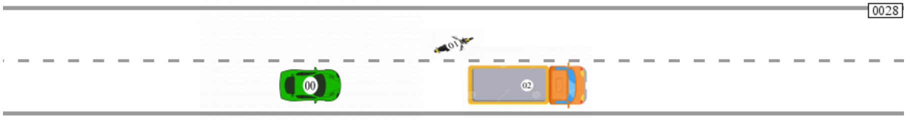

```
TRAVEL        from 149 to -1  TRIGGERED
RIGHT         from 123 to 148 ENDED
TRAVEL        from 90 to 130  ENDED
LEFT_OVERTAKE         from 41 to 91   ABORTED
TRAVEL        from 28 to 91   ABORTED
LEFT from 1 to 37     ENDED
TRAVEL        from 0 to 7     ENDED
```

Complete action example



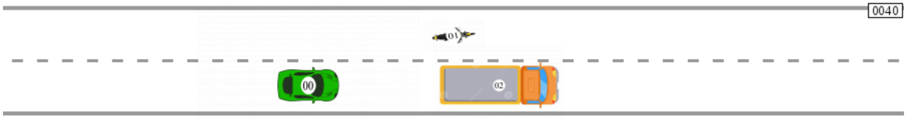

```
TRAVEL         from 149 to -1  TRIGGERED
RIGHT          from 123 to 148 ENDED
TRAVEL         from 90 to 130  ENDED
LEFT_OVERTAKE          from 41 to 91   ABORTED
TRAVEL         from 28 to 91   ABORTED
LEFT from 1 to 37      ENDED
TRAVEL         from 0 to 7     ENDED
```

Complete action example



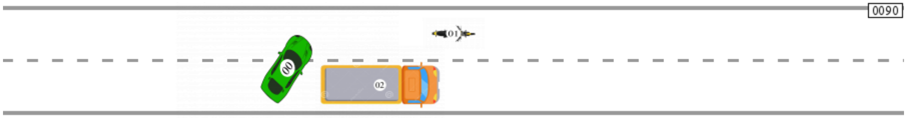

```
TRAVEL        from 149 to -1  TRIGGERED
RIGHT         from 123 to 148 ENDED
TRAVEL        from 90 to 130  ENDED
LEFT_OVERTAKE         from 41 to 91   ABORTED
TRAVEL        from 28 to 91   ABORTED
LEFT from 1 to 37     ENDED
TRAVEL        from 0 to 7     ENDED
```

Complete action example



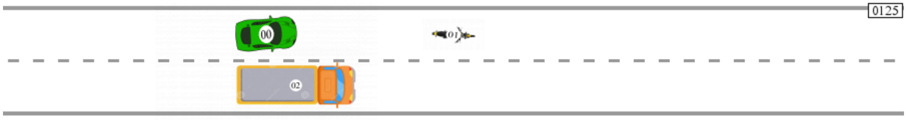

```
TRAVEL        from 149 to -1  TRIGGERED
RIGHT         from 123 to 148 ENDED
TRAVEL        from 90 to 130  ENDED
LEFT_OVERTAKE         from 41 to 91   ABORTED
TRAVEL        from 28 to 91   ABORTED
LEFT from 1 to 37     ENDED
TRAVEL        from 0 to 7     ENDED
```

Complete action example



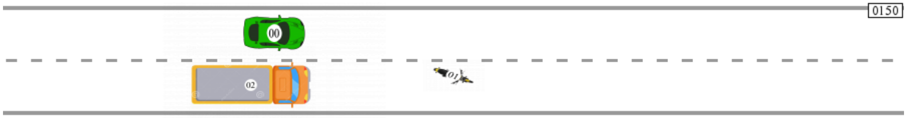

```
TRAVEL        from 149 to -1  TRIGGERED
RIGHT         from 123 to 148 ENDED
TRAVEL        from 90 to 130  ENDED
LEFT_OVERTAKE         from 41 to 91   ABORTED
TRAVEL        from 28 to 91   ABORTED
LEFT from 1 to 37     ENDED
TRAVEL        from 0 to 7     ENDED
```

Complete action example




```
TRAVEL          from 149 to -1  TRIGGERED
RIGHT           from 123 to 148 ENDED
TRAVEL          from 90 to 130  ENDED
LEFT_OVERTAKE           from 41 to 91   ABORTED
TRAVEL          from 28 to 91   ABORTED
LEFT from 1 to 37       ENDED
TRAVEL          from 0 to 7     ENDED
```

Complete action example




```
TRAVEL        from 149 to -1  TRIGGERED
RIGHT         from 123 to 148 ENDED
TRAVEL        from 90 to 130  ENDED
LEFT_OVERTAKE         from 41 to 91   ABORTED
TRAVEL        from 28 to 91   ABORTED
LEFT from 1 to 37     ENDED
TRAVEL        from 0 to 7     ENDED
```

Second step: implementation of social rules

Observation: there are not unique correct behaviours, but there are wrong ones.

Second step: implementation of social rules

Observation: there are not unique correct behaviours, but there are wrong ones.

In practice, a set of social rules is composed by single rules with the following features:

Category A string useful to define groups of rules, such as "Safety rules", "Lane change rules", ...

Second step: implementation of social rules

Observation: there are not unique correct behaviours, but there are wrong ones.

In practice, a set of social rules is composed by single rules with the following features:

Category
: A string useful to define groups of rules, such as "Safety rules", "Lane change rules", ...

Event list
: The rule is activated when at least one event is true. Each event is composed by subevents and is activated when all subevents are true. Each subevent checks a specific logical condition on the monitored vehicle and its neighbors and can have a specific **area** of influence.

Second step: implementation of social rules

Observation: there are not unique correct behaviours, but there are wrong ones.

In practice, a set of social rules is composed by single rules with the following features:

Category
: A string useful to define groups of rules, such as "Safety rules", "Lane change rules", ...

Event list
: The rule is activated when at least one event is true. Each event is composed by subevents and is activated when all subevents are true. Each subevent checks a specific logical condition on the monitored vehicle and its neighbors and can have a specific **area** of influence.

Eval mode
: Specifies when the rule must be evaluated (when the action is triggered or continuously).

Example: Italian-like highway rules

Safety rules

- Safety distance, 1 event: “Someone is in front of the observed vehicle at less than x meters”;
- Maximum speed, 1 event: “The observed vehicle is faster than x km/h”

Left lane change rules

- 1 event: “Nobody is in front of the observed vehicle”;
- 1 event: “Left lane is occupied”;
- 1 event: “The observed vehicle is on the maximum lane”.

Right lane change rules

- 1 event: “Right lane is occupied”;
- 1 event: “The observed vehicle is on the minimum lane”.

Travel rules

- 1 event: “Right lane is free”;

Left overtake allowed, **Right overtake** forbidden.

Action Manager and Rule Monitor

An object called **Rule Monitor** is used to check if the rules are violated, as the Action Manager "listens" to each action.

These two objects are independent, but they can communicate via the **rules categories**.

$\rightarrow$ Each action has a set of rule categories that it must follow.

Ex. LeftAction must follow the "Safety" rules and the "Left Lane Change" rules.

Action Manager and Rule Monitor

An object called **Rule Monitor** is used to check if the rules are violated, as the Action Manager "listens" to each action.
These two objects are independent, but they can communicate via the **rules categories**.
$\rightarrow$ Each action has a set of rule categories that it must follow.
Ex. LeftAction must follow the "Safety" rules and the "Left Lane Change" rules.

This has several pros:

- Avoid the difficult abstraction task that links the specific action to the abstract rule it should follow;
- Keeps the action and the rule systems well separated;
- Different actions can share some rule categories.

Action Manager and Rule Monitor

An object called **Rule Monitor** is used to check if the rules are violated, as the Action Manager "listens" to each action.
These two objects are independent, but they can communicate via the **rules categories**.
→ Each action has a set of rule categories that it must follow.
Ex. LeftAction must follow the "Safety" rules and the "Left Lane Change" rules.

This has several pros:

- Avoid the difficult abstraction task that links the specific action to the abstract rule it should follow;
- Keeps the action and the rule systems well separated;
- Different actions can share some rule categories.

→ This also allows to make complex action managers without changing the rule system.

Action and rules

Action	Rule categories
LeftAction	Safety Left Lane Change
RightAction	Safety Right Lane Change
TravelAction	Safety Cruise
LeftOvertakeAction	Safety Cruise
RightOvertakeAction	Safety Right Overtake

The Rule Monitor structure

Just alike the Action Manager, the Rule Monitor object is used to initialize the rule system and verify that the right rules are checked. At each instant:

1. Ask the Action Manager the list of triggered actions;

The Rule Monitor structure

Just alike the Action Manager, the Rule Monitor object is used to initialize the rule system and verify that the right rules are checked. At each instant:

1. Ask the Action Manager the list of triggered actions;
2. Extract the list of rule categories to be checked by merging the list of rule categories of each action;

The Rule Monitor structure

Just alike the Action Manager, the Rule Monitor object is used to initialize the rule system and verify that the right rules are checked. At each instant:

1. Ask the Action Manager the list of triggered actions;
2. Extract the list of rule categories to be checked by merging the list of rule categories of each action;
3. For each rule category, take each rule and consider its "Evaluation Mode";

The Rule Monitor structure

Just alike the Action Manager, the Rule Monitor object is used to initialize the rule system and verify that the right rules are checked. At each instant:

1. Ask the Action Manager the list of triggered actions;
2. Extract the list of rule categories to be checked by merging the list of rule categories of each action;
3. For each rule category, take each rule and consider its "Evaluation Mode";
4. If the Evaluation Mode is on TRIGGER, the rule is checked only at trigger time. If it is on CONTINUOUS, the rule is checked every time until the action ends (e.g. Safety rules are generally in CONTINUOUS mode);

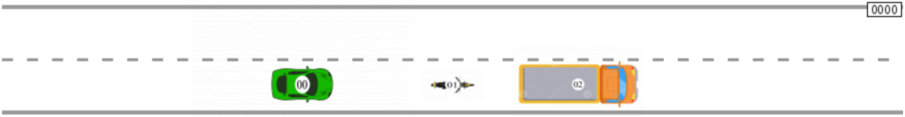
The Rule Monitor structure

Just alike the Action Manager, the Rule Monitor object is used to initialize the rule system and verify that the right rules are checked. At each instant:

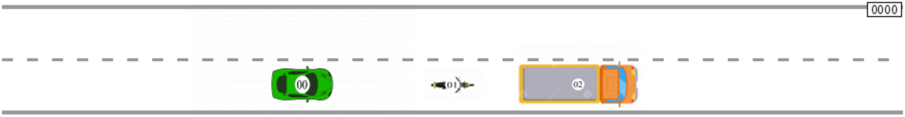
1. Ask the Action Manager the list of triggered actions;
2. Extract the list of rule categories to be checked by merging the list of rule categories of each action;
3. For each rule category, take each rule and consider its "Evaluation Mode";
4. If the Evaluation Mode is on TRIGGER, the rule is checked only at trigger time. If it is on CONTINUOUS, the rule is checked every time until the action ends (e.g. Safety rules are generally in CONTINUOUS mode);
5. A rule is **True** if a wrong behaviour was spotted (at least one of the events is true);
   Uncertain if the observer cannot tell if a rule is verified or not due to hidden areas (no events are true, at least one uncertain);
   False if the behaviour is correct (all events are false).

LeftAction example

Without sensor errors:



With sensor errors:



Remarks on the C++ code

The C++ code is mostly written using polymorphism, so that it is very easy to make a more complex system without having to understand the underlying code:

- Specific actions inherit from the Action class. To add a new action, it is only necessary to manually write the trigger, end and abort conditions.
- Specific set of rules inherit from the SocialRules class. After manually writing a new set of rules it is possible to change the monitored rules by changing a single line of code.

A similar concept applies when wanting to change the vehicle geometry, dynamics or automaton. The code is written keeping flexibility in high regard.

Summing up

The simulator should now be able to:

- Recognize and record list of actions done by the monitored vehicles;
- Verify, if possible, if the monitored vehicle is following the rules;
- Do this no matter the vehicle size, dynamic model, state transition rules (could also be manually driven in theory).

Critical point:

Action parameters tuning

Further studies:

- Use of consensus to improve the observer dataset (both to reduce sensor errors and to reveal the content of the hidden areas);
- Reputation system: how to deal with infractions?
- Roomba?
- Possible application in Roborace?
- Besides highways?