

CLAM: a Time-Sensitive Approach to Mobile Robot Localisation And Mapping

Daniela Micucci - Fabio Marchese - Domenico Sorrenti - Francesco Tisato
D.I.S.Co. - Università degli Studi di Milano-Bicocca
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
(micucci,marchese,sorrenti,tisato)@disco.unimib.it

Abstract

A precondition for a mobile robot to autonomously explore its environment is the ability to self-localise. When a map of the environment is not available, the robot should be able to self-localise and, at the same time, to build a representation of the environment. Here the problem: position estimation needs an environment model, and world modelling needs the robot position. Current research proposes solutions based on the simultaneous execution of the two activities. We propose a novel approach based on their concurrent execution. This approach, titled CLAM (Concurrent Localisation And Mapping), is founded on the conjecture that a proper separation of concerns may help in breaking the loop of the problem. Localisation and Modelling, acting on different time scales, are mostly independent each other. When a synchronisation is needed, it is controlled by an external and suitable strategy. Finally we are testing our approach by exploiting the Real-Time Performers framework based on a new set of architectural abstractions modelling the temporal behaviour of a system.

1. Introduction

The goal of autonomous mobile robotics is to build physical systems that can interact with environments not specifically structured for them. To be autonomous a mobile robot should be able to self-localise inside its workspace. Self-localisation is quite simple to perform, if the robot may exploit an environmental model. When the map is not available, the robot should be able to self-localise, and temporarily to build an incremental representation of its workspace. The problem this issue introduces is well described in [1]: "...to move precisely, a mobile robot must have an accurate environment map; however, to build an accurate map, the mobile robot's sensing locations must be known precisely [...] which came first, the chicken or the egg?"

Current research answers the question by proposing solutions based upon the simultaneity of activities, via the SLAM (Simultaneous Localisation And Mapping) approach [1]. Our proposal, denoted CLAM (Concurrent Localisation And Mapping), is based on the assumption that, in normal conditions, the two activities may be executed concurrently. Basic conjecture of our work is that a proper separation of concerns should help breaking the *chicken-and-egg-loop*. Even if *Localisation* and *Modelling* are related, we claim they can act on different time scales, so that they can be considered as mostly independent activities which sometimes synchronise under the control of a suitable strategy.

A running system based on CLAM ideas has been realised. It has been designed according to Real-Time Performers: a reflective software architecture reifying a set of architectural abstractions that properly capture the temporal behaviour of the system.

2. The CLAM approach

The CLAM approach is founded upon the following principles:

1. *Localisation* and *Modelling*, both relying on *Perception*, are the basic activities performed by a robot when exploring an unknown environment;
2. *Localisation* and *Modelling* operate on *separate information* and are subject to *different timing constraints*. Therefore they can be performed *concurrently* and with *independent timings*;
3. *Localisation* relies on information which loosely depends on the information generated by *Modelling*, and vice-versa. Therefore *Localisation* and *Modelling* must *synchronise* whenever a *criticality* arises, i.e., whenever the information an activity relies on is not reliable;
4. Synchronisation is controlled by a *strategy* which relies on the observation of the *criticalities* and drives the relative rates of the activities.

2.1. CLAM activities

Localisation and *Modelling* are the basic CLAM activities. *Localisation* aim is to make the robot able to self-localise with respect to an environment which is assumed to be known in terms of a (partial) model. *Modelling* aim is the construction of an environment representation, under the assumption that the robot knows its position. Both the activities exploit information provided by *Perception*: the activity that is in charge of getting information from the sensing devices. Although *Localisation* and *Modelling* rely on the same kind (in Object-Oriented parlance, the same classes) of information generated by *Perception*, they do exploit different sets of information and they do that within different timings. This is the basic remark to face the chicken-and-egg problem.

Perception activity generates *perceived located views*, i.e., pairs (*perceived pose*¹, *perceived view*²).

The localisation problem arises since the robot *perceived pose* is subject to cumulative errors. *Localisation* activity aims at correcting the errors. The information it uses is the current *perceived located view* and a *reference located view*, i.e., a pair (*reference pose*, *reference view*) where *reference pose* is a robot pose which is assumed to be correct, and *reference view* is a set of objects as perceived by the robot with pose given by the *reference pose*. At each step, *Localisation* generates an *estimated pose*, i.e., the robot pose after the odometric error has been corrected, and the *estimated located view*, i.e., the pair (*estimated pose*, *perceived view*). *Localisation* activity estimates the robot pose by matching geometrically located objects from the *current perceived view* and the *reference view*.

The *estimated pose* is exploited to tune up the odometric system, and the *estimated located view* is exploited as the *reference view* for the next localisation step.

Of course, there are several problems; some of them are local to the *Localisation* activity:

1. *Normalisation*: to relate the *perceived located view* and the *reference located view* to a common reference frame;
2. *Association*: to identify, for each object in the *perceived view*, a corresponding object in the *reference view*³;
3. *Registration*: to estimate a rototranslation from pairs of corresponding objects in the *perceived view* and the

1 A robot pose provided by specific devices (e.g., the odometric system).

2 A set of geometrically located objects captured by specific devices (e.g., cameras, sonar, and so on) when the robot pose is the *perceived pose*

3 As a matter of fact, Association will be considered to be successful if a reasonable subset of the objects in the *reference view* and the *perceived view* can be associated.

reference view and to generate the *estimated located view*.

Solutions to these local problems turn into steps of the *Localisation* activity whose details are out the scope of our work. The key issue is that if they are performed successfully, the *Localisation* activity can autonomously continue its task.

A global problem arises whenever *Localisation* fails. This means that the *perceived located view* cannot be successfully matched against the *reference located view*. Apart from *Perception* failures (hardware faults, blackouts, image processing uncertainties, and so on) which are out of the scope of this work, *Localisation* fails if the *reference located view* is not a plausible representation of the environment at the moment the *perceived located view* was captured (e.g. the robot turn around a corner, or entered an unexplored part of the environment). This is what we call a *criticality*, which implies (at least) a synchronisation between *Localisation* and *Modelling*, as we shall discuss later.

Modelling activity aims at incrementally building the *world model*, under the assumption that *Localisation* properly works. The *world model* is defined as a collection of geometrically located objects, which rose to the rank of being representative of “real world” objects, whatever they are. *Modelling* increases the *world model* exploiting a *views history*, i.e., a collection of *estimated located view* that are assumed to be (nearly) correct.

Again, the underlying idea is quite simple. If there is a set of views belonging to the *views history*, and a reasonably large set of objects in different *estimated located views* and all the objects exhibit similar geometric features, then a new world model object can be added (if it does not already exist) to the *world model*. As for *Localisation*, there are several local problems:

1. *Association*: to identify sets of corresponding objects in different *estimated located views*⁴;
2. *Fusion*: to merge the features of a set of corresponding objects in order to define the features of a object which is assumed to model a “real world” object with the purpose to update the *world model*;
3. *Integration*: to check whether the world model object already exists in the *world model*, and to possibly increase the *estimated located views*.

Solutions to these local problems turn into the steps of the *Modelling* activity and are out of our scope. What is relevant is that if they are performed successfully, *Modelling* can autonomously continue its task.

A global problem arises whenever *Modelling* fails. This happens whenever sets of corresponding objects cannot be

4 This sub-activity may borrow from the results of the Association sub-activity performed by *Localisation*.

identified in the recent history, i.e., the *view history* is not a plausible representation of a suitable *world model*. Again, this is a *criticality* which implies a synchronisation between *Modelling* and *Localisation*.

2.2. Criticalities and their solutions

As previously pointed out, criticalities occur when:

- the *reference located view* is not a plausible representation of the environment at the moment the *perceived located view* was captured (*Localisation* criticality);
- the *history of views* is not a plausible representation of a suitable *world model* (*Modelling* criticality).

A *Localisation* criticality arises when the objects belonging to current *perceived located view* can not be matched against objects belonging to the *reference located view*. This implies that the *reference located view* must be replaced with another one that might describe the environment (with respect to the current *perceived located view*). To overcome the criticality, a plausible *reference located view* may be derived from the *world model*. Two situations may occur:

1. the *world model* exists and it is a plausible *reference view*;
2. at the moment in which criticality occurs, the *world model* still does not exist.

CLAM successfully faces both the two situations: the first is solved by properly synchronising the information exchange between *Localisation* and *Modelling* activities; the second is solved by also adjusting the relative rates of the activities. In detail, *Localisation* rate must be slowed down, whereas the *Modelling* rate must be speeded up, so that an updated *world model* will be available as soon as possible⁵.

A *Modelling* criticality arises when corresponding objects cannot be identified in the recent *history of view*: i.e., such a history does not provide useful information for *Modelling* purpose. This criticality may be overcome by updating the *history of views*. CLAM approach may solve the criticality by adjusting the relative rates of the activities. Precisely, *Modelling* activity rate must be slowed down, whereas the *Localisation* rate must be speeded up, so that a new *history of views* will be available⁶.

2.3. Timing

The issue concerning timing deserve some more discussion. The timings of *Localisation* and *Modelling* activities

⁵ If the *Localisation* still fails, then a change of high-level strategy is needed, but is out of the scope.

⁶ Even if this “recovery” strategy goes wrong, then a change of high-level strategy is needed, but, likewise *Localisation*, is out of the scope.

are different and independent each other. Their independence is a consequence of concurrence, whereas their difference is due to the different lifetime (i.e., existence period) of the information they produce. In the following we are interested in this peculiarity.

The *estimated robot pose* is an information the lifetime of which is short. As a matter of the fact, the robot, to explore the environment, continuously move inside it. Independently from the dimension of the robot displacement, each movement causes the execution of a new *Localisation* activity (since the robot must understand which its pose is after the movement). Consequently, the new *estimated pose* replaces the previous value. Contrarily, the *world model* is an information the lifetime of which is long. When exploring an unknown environment, the robot moves slowly both to avoid obstacles and to understand its pose with respect to the environment. Small robot displacements let the environment the robot perceives still be quite always the same, i.e., constant. This means that the *world model* changes (better say evolves) very slowly. This consideration is enforced when the environment is structured and immutable (at least for a long term).

The above considerations justify that the activities are characterised by different timings. Precisely, *Localisation* must be executed with a greater rate than the *Modelling* one.

3. Real-Time Performers Framework

Real-Time Performers [2] is a reference architecture for the design of time-sensitive systems. It reifies the following architectural abstractions:

- *RealTimeLine* reifies the “real” time, i.e., a monotonic sequence of time instants characterised by the non-decreasing `now()` value of the current time. The current time splits the timeline into a past and a future timeline;
- *Action* reifies the temporal aspects of a computational operation. It associates a `perform()` operation with a *TimeInterval*;
- A *TimeInterval* defines two `<begin, end>` instant pairs modelling the planned time interval and the actual time interval for the operation execution.

Actual instants of a *TimeInterval* make sense for the past timeline only. They allow the past temporal behaviour of the system to be *recorded and observed*. Both actual and planned instants are immutable for the past timeline. On the other side, *Actions* can be inserted into the future timeline and the planned instants of their *TimeIntervals* can be specified. This allows the global temporal behaviour of the system to be *controlled* by properly planning actions. An execution Engine triggers the execution of a computational operation whenever the corresponding *Action* is enabled, i.e.,

the current time falls inside its planned interval, and sets the actual begin and end of the interval whenever a computational operation is actually started and completed respectively. The execution Engine provides the required reflective causal connection between the actual system temporal behaviour and the architectural abstractions that represents it, i.e., the Actions and the RealTimeLine. Finally, a Strategist is in charge of observing the past behaviour of the system and of planning its future behaviour by observing Actions in the past timeline and by setting Actions in the future timeline according to the application goal(s) and requirements.

4. RTP exploitation for CLAM

In a CLAM system two timelines are needed: one modelling the temporal behaviour of *Localisation*, the other of *Modelling*. Since the execution rate of *Localisation* is faster than *Modelling*, the time scale of *Localisation* timeline is finer than the *Modelling* one.

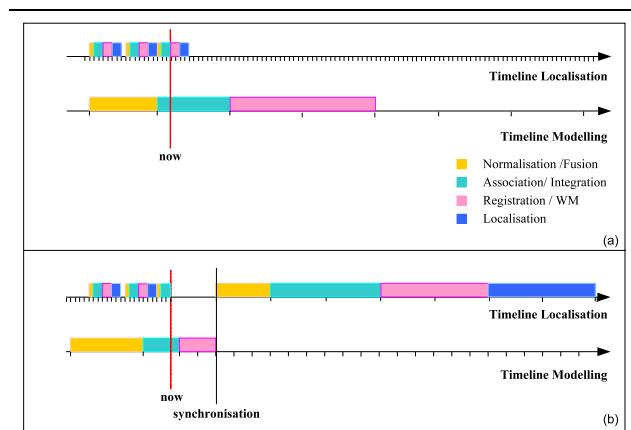


Figure 1. The activities timing

By opportunely enlarging or reducing the time scale of each timelines, it is possible to control the execution speed of the CLAM activities. This leads to face the criticalities. Figure 1 shows how the criticality concerning *Localisation* may be faced: if the *world model* does not exists, or it is not a plausible representation of the environment, then, the time scale of *Localisation* is enlarged, whereas the time scale of *Modelling* is reduced. Modifying the time scale leads to modify the execution speed of the activities in order to synchronise them. Finally, it is interesting to note the effects produced by playing with different execution rates, e.g., the enlargement of the time scale of *Localisation* may cause the robot to slow down.

5. Conclusions and Future Developments

We propose a novel approach time-based (denoted CLAM) to solve the problem of mobile robot autonomous navigation inside unknown environments. CLAM assumes that even if *Localisation* and *Modelling* are related, they act on different time scales, so that they can be considered as mostly independent activities which sometimes synchronise under the control of a suitable strategy. Synchronization may be realised by modifying opportunely the relative speed of the activities.

A concrete CLAM implementation needs an underlying software architecture capable of capturing the temporal aspects characterising such a system. From CLAM key concepts, it follows that a CLAM system needs to execute different activities with different, dynamic, and inter-dependent temporal requirements. Moreover a CLAM system needs to dynamically change the activities temporal constraints. All the above requirements are completely fulfilled by Real-Time Performers: a framework exploiting reflective mechanisms to allow the monitoring and control of time-sensitive system.

A CLAM implementation has been developed exploiting RTP framework. Preliminary qualitative testing has been done to verify that the management of CLAM criticalities works as expected. The implemented system actually succeeds in changing strategy and timings when needed. Currently, every test of our system is done in batch, offline with respect to the robot itself: we capture sensors data and feed them into CLAM. CLAM then works on robot movements in a “stored” reality. Our next step is the complete integration into the robot guidance system. Thereafter, we will validate the whole architecture. We are setting up qualitative and quantitative test sets to measure localisation error, model quality (shape, dimensions, etc.), correctness of criticality identification, RTP time constants correctness (and their tuning) with respect to the robot-world constants, and resonances/loops. Finally, since CLAM is completely independent from the algorithms used for *Localisation* and *Modelling*, we would like to experiment with new algorithms to let new criticalities arise and try to manage them.

References

- [1] J. J. Leonard and H. F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Proceedings of IEEE/RSJ IROS '91*, volume 3, pages 1442–1447, 1991.
- [2] D. Micucci, A. Trentini, and F. Tisato. A connector-based approach for controlled data distribution in rtp architecture. In *Proceedings of 23rd ICDCS - DDRTS 2003 Workshop*. IEEE Computer Society Press, May 2003.