ARS ADVANCED ROBOTIC SYSTEMS INTERNATIONAL

Localisation and World Modelling: an Architectural Perspective

Daniela Micucci; Domenico G. Sorrenti; Francesco Tisato & Fabio M. Marchese
Università di Milano - Bicocca, D.I.S.Co., Milan, Italy
Corresponding author E-mail: micucci@disco.unimib.it

Abstract: *Autonomous robot world modelling is a "chicken-and-egg" problem: position estimation needs a model of the world, whereas world modelling needs the robot position. Most of the works dealing with this issue propose holistic solutions under an algorithmic perspective by neglecting software architecture issues. This results in huge and monolithic pieces of software where implementation details reify strategic decisions. An architectural approach founded on separation of concerns may help to break the loop. Localisation and modelling, acting on different time scales, are mostly independent of each other. Sometimes synchronisation is required. Whenever needed, an external strategy tunes the relative rates of the two activities. The paper introduces rationale, design, and implementation of such a system which relies on Real-Time Performers, a software architecture providing suitable architectural abstractions to observe and control the system's temporal behaviour.*
Keywords: *mobile robotics, world modelling, modularisation, software architecture, timing*

1. Introduction

A basic activity of the software design process is the partitioning of the software specification into a number of modules that together satisfy the original problem statement (Card, D.N.; Page, G.T. & McGarry, F.E., 1985). Modularisation yields a great number of advantages in building software systems. For example, it allows the handling of the complexity of large systems by splitting loosely coupled parts that can be better dealt with individually; the substitution or the repairing of defective parts of a system without interfering with other parts; the reuse of existing parts in different contexts (Abreu, F.B. & Goulão, M., 2001).

Such a reductionist approach, i.e., thinking in term of components and their compositions, is common practice for a software architect (Shaw, M. & Garlan, D., 1996). In other computer science areas specific problems are often tackled from a holistic perspective turning into large algorithms. This produces huge and monolithic software systems that intermix strategic decisions, specific algorithms, scheduling, and communications. The lack of modularisation and of stratification makes configuration, reuse, and scalability difficult if not impossible. Such systems are research prototypes, which can hardly be extended to large-scale, real-life products.

The above holds for most of the solutions that have been proposed so far to address the SLAM (Simultaneous Localisation And Mapping) problem. The problem can be summarised as follows: an autonomous mobile robot should be provided with methods for both *world modelling* and *self-localising* inside its workspace. Since pose estimation requires a map and modelling requires the pose, a "chicken or the egg" problem arises (Leonard, J.J. & Durrant-Whyte, H.F., 1991). Several approaches address the problem under a holistic perspective by neglecting software architecture issues. This produces the above-mentioned drawbacks and yields unmanageable complexity.

An architectural (i.e., reductionist) approach may provide suggestions for the resolution of the chicken-and-egg problem. *System modularisation* plays a crucial role. CLAM (Concurrent Localisation And Mapping) (Micucci, D., 2004), (Micucci, D.; Marchese, F.M.; Sorrenti, D.G. & Tisato, F., 2004) proposes a solution to the problem by exploiting time-based modularisation criteria. Note that in the SLAM literature a CML (Concurrent Mapping and Localisation) acronym exists, as a synonymous to SLAM.

Time scale is a key concept for modularisation. It breaks the system into two well-distinguished components: localisation and modelling. Their activities rely on timing constraints whose time scales differ by several orders of magnitude. As long as the activities operate on separate information, they do not depend on each other and can be performed *concurrently*.

This holds as long as the information each activity exploits is *reliable*. When this condition is no longer valid, a dependency arises and the two activities must synchronise: whenever needed their relative execution rates may change.

The goal of this paper is not to demonstrate that the proposed approach leads to an optimal solution. This is an exploratory work aimed at showing that good software practices can ease building complex systems.

This is especially true when the overall system behaviour emerges from the individual behaviour of simple components, which loosely interact thanks to their different execution rates.
The paper is organised as follows. Section 2 presents some related works. Section 3 discusses the CLAM approach with static timing constraints, whereas Section 4 introduces timing synchronisation. Section 5 presents some issues related to the adoption of a pre-existent framework for real-time systems. Section 6 deals with conclusions.

2. Related works

Even though it seems reasonable to tackle some applications with self-localisation in a known world map, provided obstacle-avoidance capabilities are available, many applications, on the other hand, require a full robot autonomy also in world modelling. This could be due to difficulty in obtaining the known world map or to large discrepancies of the workspace and the available maps, which are often only executive plots. It is therefore necessary to provide the robot with complex world-modelling capabilities.
Past proposals were presented about this complex issue; the most famous contribution on this line was (Kuipers, B., 2000) which addresses the problem at different levels, i.e., geometric, topologic, and control. Current research, on the other hand, has a major focus on the geometric consistency of the modelled world. This research area is currently dealing mostly with static workspaces, with a large amount of contributions focusing on indoor conditions. Folkesson and Christensen (Folkesson, J. & Christensen, H.I., 2004), represent the world as a graph where the nodes carry information about the robot and the pose of each world feature; on the edges they have the relationships between nodes. This approach, because of the large number of DoF (Degree of Freedom) involved (all the features and poses), raises concerns about its computational complexity. Many other approaches, e.g., Constrained Local Submap Filter (Williams, S.B.; Durrant-Whyte, H.F. & Dissanayake, G., 2003), Extended Information Filter (Thrun, S.; Koller, D.; Ghahramani, Z.; Durrant-Whyte, H.F. & Ng, A.Y., 2002), Consistent Pose Estimation (Lu, F. & Milios, E., 1997), Geometric Projection Filter (Newman, P., 1999), are based on very large state EKF (Extended Kalman Filter) for building a geometrically consistent map.
Fastslam (Thrun, S.; Montemerlo, M.; Koller, D.; Wegbreit, B.; Nieto, J. & Nebot, E., 2004) decomposes the problem in two: robot localisation and estimation of the position of the world features, as conditioned by the robot pose considered. As these two sub-problems are relevant for what is proposed in this paper and this decomposition is important for us also, it is necessary to clarify that their algorithmic proposal deals with the multiple hypotheses tracking problem, which is orthogonal to our work.
Besides research on algorithmic solutions to the SLAM problem, large efforts have been made towards the design of architectures for robot control. Preliminary works, based on the SPA (Sense-Plan-Act) paradigm (Murphy, R.R., 2000), conceived the control system for an autonomous mobile robot as composed by functional elements. Afterwards, the research focused on behaviour-based approaches, like Brooks' Subsumption (Brooks, R.A., 1986) and Arkin work (Arkin, R.C., 1998). Finally, beginning from the last decade, robot control systems are mostly designed using hybrid layered architectures. TCA architecture (Simmons, R., 1994), LAAS architecture (Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M. & Ingrand, F., 1998), the architecture used in BIRON robot companion (Haasch, A.; Hohenner, S.; Hüwel, S.; Kleinehagenbrock, M.; Lang, S.; Toptsis, I.; Fink, G.A.; Fritsch, J.; Wrede, B. & Sagerer G., 2004), are only a few examples. An introduction to robot specific architecture may be found in (Murphy, R.R., 2000). In general, robot architectures differ in the number of layers, but they share the concept that each layer is a functional module that implements basic skills. The modularisation criteria adopted may differ, but, to the knowledge of the authors, it is never based on time, like the one we propose.

3. The CLAM approach

3.1. Basic concepts

CLAM approach relies on the following assumptions:

- the robot is equipped with odometric and vision systems providing an error-prone estimate of its position and perceptions about the environment;
- the environment the robot explores is modelled as a collection of objects whose position is defined in a geometrical reference system;
- the robot speed and direction can be controlled;
- the overall environment does not change.

Localisation and *modelling* are the basic CLAM activities. The localisation aim is to make the robot able to self-localise with respect to an environment that is assumed to be known in terms of a (partial) model. Modelling aims at building an environment representation, under the assumption that the robot knows its position. Both activities exploit information provided by the perception system.
Although localisation and modelling rely on the same kind (in Object-Oriented terminology, the same classes) of information generated by perception, they exploit *different sets of information* and they do that within *different timings*. This is the basic idea we use to face the chicken-and-egg problem.
Let us define the *reference pose* (*RP*) as the last best estimation of the robot position and orientation. Under the CLAM assumptions, the perception activity must provide the following information:

- *perceived pose* (*PP*): the robot pose coming from dead-

reckoning inside the (partial) environment. PP is referred to the RP reference frame. In the SLAM community, this pose is best known as "predicted". We prefer the term "perceived" since in a temporal context (like the one we are discussing), "predicted" is used to refer to something that it is expected to happen in the future;

- *perceived view* (*PV*): a representation of the environment made up of *geometrically located objects* (*GLO*s) captured when the robot pose was perceived pose. The reference frame is fixed to the robot.

The perception activity generates *perceived located views* (*PLV*), i.e., pairs (PP, PV).

3.2. Localisation

The localisation problem arises since the robot perceived pose PP is subject to cumulative errors. Localisation activity aims at correcting the errors by exploiting the current perceived located view PLV and a *reference located view* (*RLV*), i.e., a pair (RP, RV) where the *reference view RV* is a set of GLOs as perceived by the robot at the reference pose RP. At each step, localisation generates an *estimated pose* (*EP*), i.e., the robot pose after the odometric error has been visually corrected, and the *estimated located view* (*ELV*), i.e., the pair (EP, PV).

The idea behind localisation is quite simple. Localisation activity estimates the robot pose by matching geometrically located objects from the current perceived view against geometrically located objects from the reference view.

Let us assume that the reference located view RLV is correct, i.e., the robot at pose RP observed the GLOs belonging to the reference view RV. Let P_{RT} be the transform that leads the robot from the reference pose to the perceived pose PP. If PP has been properly perceived, by applying P_{RT} to the GLOs in the perceived view PV should produce a set $PV^1 = P_{RT} \bullet PV$ such that $PV^1 = RV$, i.e., each GLO in PV^1 is similar (i.e., it exhibits similar intrinsic geometrical characteristics) to the corresponding GLO in RV. If this does not hold, a D_{RT} must be identified, such that $D_{RT} \bullet PV^1 = RV$. D_{RT} is due to the perception error of the odometric system. Therefore $EP = D_{RT} \bullet PP$ provides the estimated pose EP and the pair (EP, PV) specifies the estimated located view ELV. Fig. 1 sketches a step of localisation putting in action the above concepts.

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

The localisation activity includes several sub-activities:

1. *normalisation*: to roughly relate PV and RV to a common reference frame;
2. *association*: to identify, for each GLO in PV, a corresponding GLO in RV. The association will be considered successful if a reasonable subset of the GLOs in RV and PV can be associated;
3. *registration*: to estimate a transform from pairs of corresponding GLOs in PV and RV and to generate the estimated located view ELV.

Solutions to these local problems turn into the steps of the localisation activity whose details are out the scope of this paper. If they are performed successfully, the localisation activity can autonomously continue its task.

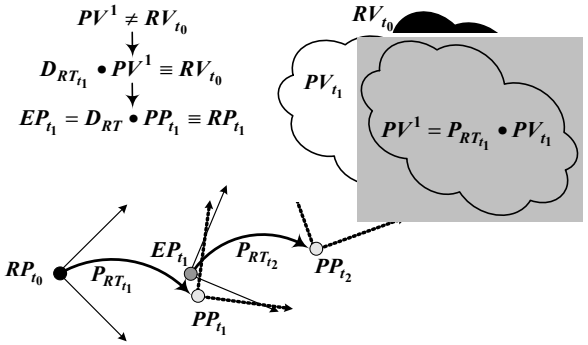

Fig. 1. A localisation step

3.3. Modelling

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 GLOs, which represent "real world" objects, referred to an absolute external reference system. Modelling augments the world model by exploiting a *views history*, i.e., a collection of estimated located views ELVs that are assumed to be correct.

Again, the underlying idea is quite simple. If there is a set of estimated located views belonging to the views history, and a reasonably large set of GLOs in different ELVs exhibit similar geometric features, then a new object can be added (if it does not already exist) to the world model.

As for localisation, there are several local problems:

1. *normalisation*: to refer ELVs in the same reference frame of the world model;
2. *association*: to identify sets of corresponding GLOs in different ELVs. This may borrow from the results of the association sub-activity performed by localisation;
3. *fusion*: to merge the set of corresponding GLOs, i.e., the different measures of the same "real world" object. The purpose is to update the world model;
4. *integration*: to check whether the GLO already exists in the world model, and to possibly augment the world model.

Solutions to these local problems turn into the steps of the modelling activity and are out of the scope of this paper. If they are performed successfully, modelling can autonomously continue its task.

3.4. Discussion

Key remarks are that localisation and modelling exploit separate information and that they are subject to different timing constraints.

Localisation exploits RLV, i.e., the most recent ELV, apart

from criticalities discussed in Section 4. It is bound to strong real-time constraints depending on the physical speed of the robot and on the requirements of the motion control system. Moreover, localisation sub-activities are simplified if there are small variations between PLV and RLV, i.e., localisation is as fast as possible. In practice, the localisation rate should be bounded by the acquisition rate of the cameras – say some tenths of milliseconds.
Modelling operates on the views history, i.e., a set of *past* ELVs generated by a sequence of previous localisation steps. The history must be as large as possible in order to build an effective world model. Therefore modelling is delayed with respect to localisation and is not subject to hard real-time constraints, apart from critical situations discussed in Section 4. The time scale of the modelling – say, at least, some seconds – is two or more orders of magnitude larger than the time scale of the localisation.
This justifies partitioning the system into two distinct components. Since timings and exploited information are different, the two activities may be performed *concurrently*. As in classical layered systems, the activities of each layer are characterised by execution rates that decrease from the lower levels (localisation) to the upper ones (modelling). Putting together activities characterised by different timing requirements is an architectural mistake. Moreover, the individual activities are simpler from the algorithmic point of view.

4. Criticalities and timing

4.1. Localisation criticalities

A *localisation criticality* arises when the association sub-activity of the localisation activity fails, i.e., GLOs in PLV cannot be successfully matched against GLOs in RLV. Apart from perception failures, which fall out of the scope of this work, this happens if RLV is not a plausible representation of the environment when PLV is captured (e.g., the robot entered a new room).
In this case RLV must be replaced by a plausible RLV. If the world model for the newly entered environment (e.g., the new room) already exists, a new RLV is generated on the basis of the current robot location and the localisation activity goes on. Otherwise, the localisation activity must be slowed down (or totally stopped) and the modelling activity must be speeded up until the world model has been properly updated.

4.2. Modelling criticalities

A *modelling criticality* arises when the association sub-activity of the modelling activity fails, i.e., sets of corresponding GLOs in different ELVs belonging to the views history cannot be recognised.
In this case the views history must be filled up with a plausible set of new ELVs. Therefore, the localisation activity must be speeded up in order to enrich the views history as soon as possible with new ELVs.
Note that, if a localisation criticality arises before the modelling criticality has been recovered, we fall again into the chicken-and-egg problem. A solution is discussed below.

4.3. Discussion

Localisation and modelling must synchronise whenever a criticality arises, i.e., whenever the information an activity relies on is not trustworthy. Relative timings have to be adjusted so that the system may return to its "normal" condition, where activities concurrently run without synchronisation.
In case of localisation criticality, slowing down the localisation implies that its timing constraints must be relaxed. Slowing down the localisation rate accordingly implies reducing the robot speed, which determines the timing constraints.
In case of modelling criticality, several localisation steps must be performed at a high rate without producing localisation criticalities. Again, this can be ensured by reducing the robot speed, so that there are small variations between the PLVs of subsequent steps.
Ultimately, any kind of criticality implies to reduce the physical robot speed, and this looks very reasonable.
The software architecture must provide mechanisms allowing the relative rates of the activities to be properly tuned, as we shall discuss in Section 5. Synchronisation between activities must be controlled by an external entity, which observes the criticalities and drives the relative rates of the activities. In any modularised system, the existence of an external component (i.e., a strategist) in charge of deciding when and how some operations are performed is a common practice. The advantage in doing so is that it keeps out any strategic issues from the components, which turn out to be purely functional. The advantages are at least their reusability and ease of replacement.

5. CLAM at work

We implemented a preliminary system based on CLAM concepts. It is based on a robot endowed with an odometric system and a 3D reconstruction system.
The dynamicity of timing constraints is a key concept in CLAM. Therefore, we need a software architecture that provides a set of architectural abstractions modelling time related issues. In a previous work we developed a software architecture for the design of real-time systems that raises the control of both the internal architecture and the temporal behaviour at the application level (Micucci, D.; Ruocco, S.; Tisato, F. & Trentini, A., 2004). It is called Real-Time Performers (RTP) and it has been exploited for our work.

5.1. Real-Time Performers

According to the definitions in (Eden, A.H. & Kazman, R., 2003) and in (Shaw, M. & Garlan, D., 1996), RTP is defined in terms of components and connectors. RTP

components (*performers*) are passive entities in charge of computing information. RTP connectors (*projectors*) are passive entities also in charge of aligning information between performers. *Strategist* is the RTP component that controls the temporal behaviour of the system by activating both performers and projectors. Under this perspective, the strategist resembles the supervision layer/module of many robot control architectures (e.g., (Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M. & Ingrand, F., 1998)). The strategist decides why some action will be executed. A *timed trace* determines when actions must be performed, i.e., it reifies the real-time. A timed trace is a partially ordered set of *timed requests*. A timed request models an action: it is defined as (*recipient*, *command*, *planned_time*), where *recipient* is a performer or a projector, *command* is the command that the recipient has to execute, and *planned_time* specifies the time interval in which the command must be executed. For each timed trace there is a *virtual clock*, an active component that is in charge of advancing the current time. The "now" value of the current time splits a timed trace into past and future. The strategist plans the future behaviour by inserting/modifying/deleting timed requests in the future portion of a timed trace. It decides why some actions have to be performed by observing the status of the components and the past system behaviour (past timed trace). Moreover, a strategist may accelerate or decelerate activities by tuning their virtual clocks. Fig. 2 sketches the system temporal behaviour controlled by a strategist as defined by RTP.

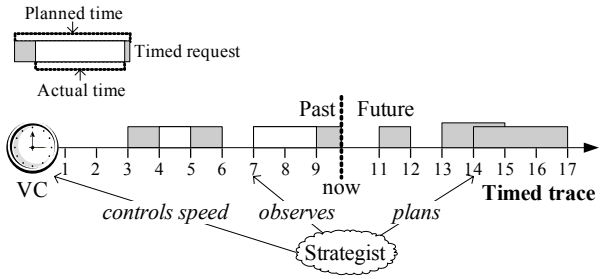

Fig. 2. RTP temporal behaviour

5.2. CLAM and RTP

CLAM sub-activities turn into RTP performers. The configuration of a CLAM system involves the definition of the system topology, i.e., the connection of performers via projectors, and the definition of timed traces and virtual clocks.

The localisation and modelling activities are driven by independent timings modelled by two different RTP timed traces. The time scale of the localisation virtual clock is more fine-grained than the modelling one, due to the difference of the time scales. Fig. 3 sketches the initial situation.

When criticalities arise, the strategist has to synchronise the two activities.

Localisation criticality implies the insertion into the localisation timed trace of an immediate request to substitute the current reference view according to the world model. If the world model is not available, the strategist speeds up the modelling activity and slows down the localisation one by acting on the proper virtual clocks.

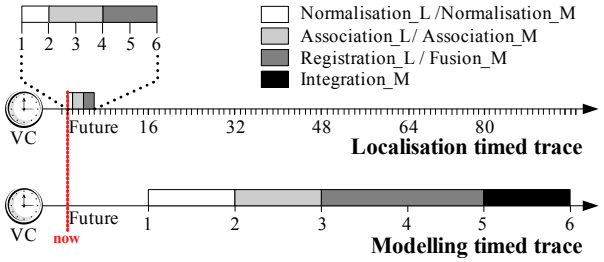

Fig. 3. The system start-up

Modelling criticality implies the adjustment of the relative speeds of the activities (Fig. 4). The strategist slows down the virtual clock of the modelling activity, speeds up the virtual clock of the localisation activity, and adjusts the modelling timed trace by both inserting requests to create a new views history and deferring the requests dealing with world model updating.

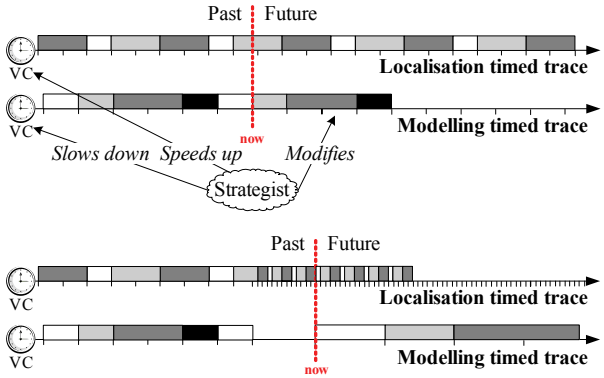

Fig. 4. Criticality management

As pointed out in Section 4.3, criticalities may imply that the robot speed must be reduced. This can be achieved by tuning the virtual clock of a third timed trace, which controls the physical motion commands.

6. Conclusions and future works

This paper describes how we faced the SLAM problem from an architectural perspective. The result of our work is a new approach, denoted CLAM, founded on the following key concepts:

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 they 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 no longer trustworthy;

4. synchronisation is controlled by a strategy which relies on the observation of the criticalities and drives the relative rates of the activities.

Note that, even if the "real" world does not evolve (this is an assumption for CLAM), the robot perceived environment evolves, but its evolution rate can be controlled. For instance, when localisation fails, the localisation activity is slowed down. Obviously, localisation is strictly related to robot motion, i.e., its timing constraints depend on the robot speed. If strategies relax the robot motion timing, then the robot perceived environment evolves with a more relaxed rate. This is noticeable because in other application domains the variability of the environment has timing constraints that are not controllable. Imagine a video-surveillance system monitoring a highway: when a car passes it modifies the perceived environment, but the car is not under the application control and its speed cannot be decreased with the aim, for instance, of acquiring more frames.

The design of a system reifying CLAM concepts needs an underlying software architecture capable of capturing the temporal aspects of such a system. Real-Time Performers fulfils CLAM requirements since it allows system temporal behaviour to be observed and controlled at the application level.

Preliminary qualitative testing has been performed to verify that the management of CLAM criticalities works as expected. The implemented system actually succeeds in changing strategy and timings when needed. Currently, testing of our system is done offline with respect to the robot itself: we capture data from sensors and feed them into CLAM. CLAM then works on robot motion in a "stored" reality. Our next step is the complete integration into the robot guidance system. 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, thanks to a good design of the system, we could easily experiment new algorithms (new components substituting the implemented one) to let new criticalities arise and try to manage them.

7. References

Abreu, F.B. & Goulão, M. (2001). A Merit Factor Driven Approach to the Modularization of Object-Oriented Systems, *L'Objet*, Vol. 7, No. 4

Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M. & Ingrand, F. (1998). An architecture for autonomy, *International Journal of Robotics Research*, Vol. 17, No. 4, pp. 315-337

Arkin, R.C. (1998). *Behavior-based robotics*, MIT press

Brooks, R.A. (1986). A Robust Layered Control System for a Mobile Robot, *IEEE Journal of Robotics and Automation*, Vol. 2, No. 1, pp. 14-23

Card, D.N.; Page, G.T. & McGarry, F.E. (1985). Criteria for software modularization, Proc. of ICSE, pp. 372-377, IEEE

Eden, A.H. & Kazman, R. (2003). Architecture, design, implementation, Proc. of ICSE, pp. 149-159, IEEE

Folkesson, J. & Christensen, H.I. (2004). Robust Slam, Proc. of IFAC Symposium on IAV, Elsevier

Haasch, A.; Hohenner, S.; Hüwel, S.; Kleinehagenbrock, M.; Lang, S.; Toptsis, I.; Fink, G.A.; Fritsch, J.; Wrede, B. & Sagerer G. (2004). BIRON - The Bielefeld Robot Companion, Proc. of ASER, pp. 27-32, IEEE

Kuipers, B. (2000). The Spatial Semantic Hierarchy, *Artificial Intelligence*, Vol. 119, No. 1-2, pp. 191-233

Leonard, J.J. & Durrant-Whyte, H.F. (1991). Simultaneous map building and localization for an autonomous mobile robot, Proc. of IEEE/RSJ IROS, Vol. 3, pp. 1442-1447

Lu, F. & Milios, E. (1997). Globally consistent range scan alignment for environment mapping, *Autonomous Robots*, Vol. 4, No. 4, pp. 333-349

Micucci, D. (2004). *Mobile robot localisation and world modelling in a real-time software architecture*, Ph.D. thesis, University of Milan, Italy

Micucci, D.; Marchese, F.M.; Sorrenti, D.G. & Tisato, F. (2004). A time-sensitive approach to mobile robot autonomous navigation, Proc. of CCCT, pp. 377-382, IIIS

Micucci, D.; Ruocco, S.; Tisato, F. & Trentini, A. (2004). Time Sensitive Architectures: a Reflective Approach, Proc. of ISORC, pp.189-196, IEEE

Murphy, R.R. (2000). *Introduction to AI Robotics*, MIT press

Newman, P. (1999). *On the structure and solution of the simultaneous localisation and map building problem*, Ph.D. Thesis, University of Sydney

Shaw, M. & Garlan, D. (1996). *Software Architecture. Perspective on an Emerging Discipline*, Prentice Hall

Simmons, R. (1994). Structured control for autonomous robots, *IEEE Transaction on Robotics and Automation*, Vol. 10, No. 1, pp. 34-43

Thrun, S.; Koller, D.; Ghahramani, Z.; Durrant-Whyte, H.F. & Ng, A.Y. (2002). Simultaneous mapping and localization with sparse extended information filters, Proc. of WAFR

Thrun, S.; Montemerlo, M.; Koller, D.; Wegbreit, B.; Nieto, J. & Nebot, E. (2004). FastSLAM: An efficient solution to the simultaneous localization and mapping problem with unknown data association, *Journal of Machine Learning Research*

Williams, S.B.; Durrant-Whyte, H.F. & Dissanayake, G. (2003). Constrained initialization of the simultaneous localization and mapping algorithm, *International Journal of Robotic Research*, Vol. 22, No. 7-8, pp. 541-564