Developing an Integrated Software Environment for Mobile Robot Navigation and Control

Zoltán Tuza[1], János Rudan[1] and Gábor Szederkényi[1,2]
[1]Faculty of Information Technology, Pázmány Péter Catholic University
H-1083 Budapest, Práter u. 50/A, Hungary Tel:+36 18864771
[2]Computer and Automation Research Institute, Hungarian Academy of Sciences
P.O. Box 63, H-1518 Budapest, Hungary Tel: +36 12796000; Fax : +36 14667503
E-mail: {tuza.zoltan, rudan.janos}@itk.ppke.hu, szeder@sztaki.hu

Abstract—**A flexible modular robotic software environment based on the popular MRPT toolkit is reported in this paper that is able to integrate path planning, navigation and control algorithms easily from several sources. The different modules (which are responsible for SLAM, trajectory tracking, sensor and actuator handling, visualization etc.) communicate with each other via a carefully developed network based protocol set that ensures transparency and robust operation. The system can also be used as a simulation environment and it is capable of comparative benchmarking of different navigation algorithms. Laser scanner based map building and navigation of an autonomous wheelchair is shown as application examples to illustrate the features of the developed software environment.**

I. Introduction

Currently there are numerous navigation and control algorithms in the field of indoor mobile robotics. The available robotic software toolkits present several implemented algorithms, but system-level integration still remains a challenging task. This inspired us to create a high-level, modular robotic software environment based on a selected toolkit to handle our special application requirements and significantly extend previous functionalities. The presented software system is based on the Mobile Robotic Programming Toolkit (MRPT) [1] package which is developed by the MAPIR Lab at the University of Malaga. MRPT is a software framework containing several modern tools and algorithms, written in C++. Based on this framework a high-level module set can be created to operate on a mobile robot. The software platform composed from these modules can serve as the basis for testing, benchmarking and using several algorithms and robotic tools. The main purpose of the development is to create a flexible and modular software platform which can be used efficiently in a distributed computational environment that are typical in many robotic applications. Our software is a high-level, application-oriented system using algorithms and tools from several sources to control a vehicle in indoor environment. Beyond standard requirements for a robotic software environment such as robustness and portability, our high-level, integrated system was designed to meet the following main requirements:

- strongly modular construction
- multi-host, distributed architecture
- probabilistic computational framework
- an environment where the incorporation of new algorithms and features is easy.

In the following, we will present how the developed software meets these requirements. In Section II some projects with similar goals are introduced. In Section III, we will consider the main properties and foundations of the proposed system. In Section IV, we will enumerate the main design principles. Section V describes some implemented modules in detail and finally in Section VI, we will present applications with the proposed system.

II. Related work

Since a truly autonomous robot is a fairly complex system, a well designed software solution is an essential part of it. Modularization is a common engineering approach to handle large and complex systems. Besides reducing problem complexity, the modularization also gives the opportunity to separate responsibility and the implementation methods of the tasks.

In R. Brooks' seminal paper [2] introduces a layered system architecture. These layers are organized in horizontal hierarchy where the top layer reflects the most desired behaviour. This design principle strongly affects the architecture of robotic software systems since then.

Numerous software libraries and off-the-shelf robotic software products are available currently, we have selected three of them for discussion. These open source programs are widely used by the research community and they also have several similarities to our system.

The Player project [3] focuses on the mobile robot hardware abstraction. It contains several drivers for commercial mobile robots and sensors and it has a framework for developing Player compatible drive for any kind of devices. Similarly to our system, the different modules are communicating with each other via TCP/IP protocol. On the other hand the data distribution is different in Player than in our case. In Player each module can operate in PUSH or PULL data sending mode whereas in our system each module sends the messages to a central module for further distribution. Unfortunately Player does not have a coherent probabilistic framework like the one that MRPT has.

The Carnegie Mellon Navigation (CARMEN) Toolkit [4] has a layered architecture as well, where the base layer is responsible for hardware abstraction and located at the bottom of the hierarchy along with collision avoidance. The middle layer is responsible for localization and navigation, and finally the high level tasks are solved by the top layer. For communication between the modules it uses Inter Process Communication (IPC) system. This type of message distribution is similar to our system. The main problem with CARMEN is the lack of public release since 2008.

The MARIE middle-ware framework [5] focuses on the reusability and integration with new and existing software components. For example, MARIE can use several of CARMEN's applications like path planner or localizer. It supports a wide range of communication protocols, communication mechanisms and robotics standards.

III. Foundations of the software system

MRPT is an open-source robotic toolkit programmed in C++. This toolkit is under continuous development with strong support of the community. MRPT was selected as basis for our work because this framework uses a coherent probabilistic approach which is a powerful computational paradigm to solve mobile robot navigation tasks. Furthermore, MRPT contains a large amount of implemented algorithms and software tools like Simultaneous Localization and Map building (SLAM) techniques [6], Kalman Filter, Particle Filter, hardware drivers, data structures for several kinds of maps, 3D visualization and many auxiliary utilities. It incorporates several third-party software modules: wxWidgets and OpenGL engine, AriaLab hardware abstraction layer [7] to handle a large variety of commercial mobile robots, drivers for several types of sensors and other tools. Furthermore, it contains a lot of basic level tools: TCP/IP socket handling, operating system independent timing, math libraries, computational and algorithmic packages.

MRPT is built up from a well designed class hierarchy presenting high-level programming methods, e.g. serializable classes, smart pointer implementations, skeletons and other useful features. Several complex data structures like occupancy grid map [8] are incorporated into the class hierarchy, based on the above described programming methods.

MRPT is designed for probabilistic robotics that means probabilistic approach for perception, system modeling and state estimation is fully supported. All the information which is gathered by the sensors can be incorporated into probabilistic density functions. The proper function can represent the certainty/uncertainty over the system states. The Bayesian framework is available [9] to describe all elements of the simultaneous localization and map building problem. Sensor models, motion models for mobile robots and map realizations for feature and grid mapping are also available in this toolkit. For probabilistic density representation, two solutions are available, namely the Gaussian for unimodal densities and the Particle Filter for multimodal densities.

Due to the high activity in the field of mobile robotics, many new algorithms, solutions are published. For that reason we designed an interface, which has the ability to integrate these new algorithms into our system. This setup enables us to investigate their properties and compare them to other existing algorithms.

IV. Design principles and properties of the modular system

In this section we summarize the most important design principles and properties of our software system.

A. Architectural design

Using the algorithms implemented in MRPT, also from third party sources or from our own implementations, a set of separate modules along with software components were created. Each module is responsible for a specific task like SLAM, path planning, trajectory following control, handling of sensors and actuators, visualization, etc. Apart from the above tasks, additional modules were implemented that allow us to use the system as a simulation environment, too. Such modules are responsible for recording the measured data and playing it back, and for monitoring the inner state variables of the software system. The structure of the software system can be seen in Figure 1.

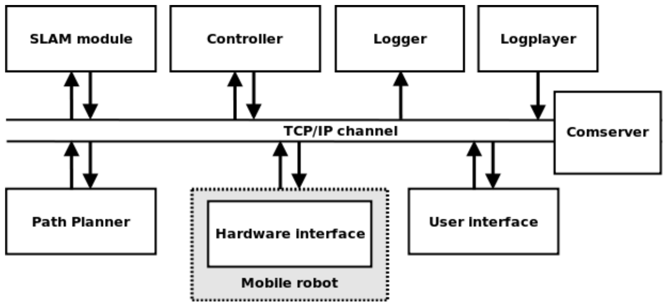

Fig. 1. The block scheme of the proposed modular system. Each module has a specific task. They are connected each other via a TCP/IP channel. Inter-module communication is managed by the Comserver module.

The architecture provides transparency between the real and simulated experiments. Simultaneously running different algorithms on the same data can also be done thus comparative benchmarking is possible. If one wants to integrate a new algorithm into the proposed system, only the used data structures of the new algorithm should be adjusted to an acceptable network message, like robot position, grid map, etc.

Separating functionalities of the system into modules ensures robustness and safety: failure of a single module will usually not cause the failure of the whole system since other independent modules can work properly or they can perform the appropriate shutdown sequence. In addition, modular system offers the opportunity to separately develop, test and tune modules before integrating into the system. The unified form of data representation used at the communication allows us to change specific modules without modifying any settings or parameters in other modules. This form of communication is achieved by using the MRPT's built-in object serialization.

During the design of the modules, we had to consider the desired speed of the operation. The typical sensors in robotics like laser scanners, cameras, etc. operate around 50-100Hz. If on-line data processing is needed, than the modules that are time-critical has to be fast enough to complete the required computations between sensor readings. There can be other modules which can tolerate network delay and message buffering.

The modules communicate with each other via a well-defined interface and protocol set while performing their tasks. The communication protocol defines message types containing specific data (e.g. robot position, planned path, motion command, etc.). Each module's network interface inherits from a common root class. This architecture can be seen in Figure 2.

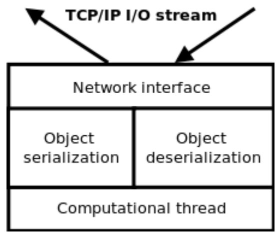

Fig. 2. General structure of a module consists a network interface, and a working thread. Object serialization/deserialization parts are resposible for converting data structure.

The standard TCP/IP protocol is used for message transportation while the distribution of messages is handled by a central module using a publish-subscribe based architecture as described in Section V-A. The network handling layer is completely transparent to each module which means that the modules have (and need) no knowledge of the source of data.

The modular system design and using an IP based protocol to connect each module have the advantage that the modules can be distributed to several computers in order to achieve higher computation power.

B. Analysis of the architecture

The usage of TCP/IP protocol is a solid foundation for a stable inter-modular communication. It is capable of handling packet loss, and checks data corruption. However, it is important that the network protocol itself introduces data flow speed variations which causes network delay and jitter. These have crucial importance in time-dependent robotic systems. Due to the differences in time complexities of the tasks, each module provides messages in a different pace. For that reason the network interface has a message queuing system. With the help of the message queue we prevent TCP/IP buffer overflow on the client side. Depending on the purpose of the module we can process all of the messages in the queue or we can select the last received one for processing. The first scenario is useful during data recording. However, the second is applicable for delay sensitive cases, for instance generating the control signal in the control module.

During a specific system's design it is important to identify the modules which are sensitive to data delay and jitter. After that, there are several ways to handle the problem: in the case of the presence of a slowly processing module, buffering data is possible to ensure the possibility of off-line processing. Another solution can be the sampling of incoming data on a lower frequency. Delays introduced by the network can be handled effectively by running the critical modules on the same computer, or on computers which are connected via a wired network instead of a wireless connection which is more sensitive to this issue.

The used network message distribution technique enables us to run the same modules with different parameter settings on the same data set parallelly. This ensures a good configuration for parallel benchmarking and testing.

The object serialization technique applied is the same for all type of data sent through the network meaning that all data is encapsulated into the same wrapper class. Consequently, none of the packages have priority before the others, and the handling procedure of the packages are the same for each data type. The wrapper class has two attributes: a message type number which describes the type of the data encapsulated into the message, and a data field to store the actual data.

V. Review of important modules

In this section, we will introduce the main properties of some selected modules. These elements provide the necessary functionality for an autonomously navigating robot. As it has been pointed out, further modules can be easily added to the system, by connecting them to the communication channel via the previously introduced interface. Data flow between the selected modules and the content of the network messages can be seen in Figure 3.

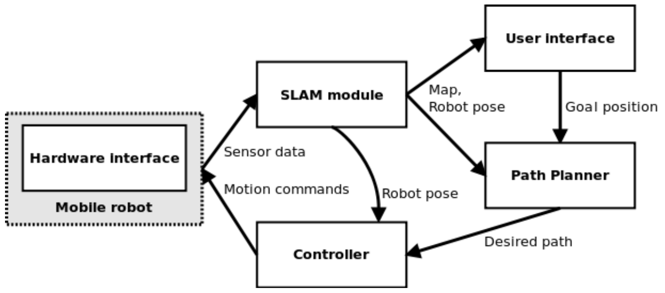

Fig. 3. Data flow in the system. The arrows show the flow of the messages through the system, distributed by the Comserver.

A. Communication server

In the center of the architecture there is a communication server. It is responsible for distributing the network messages based on a publish-subscribe model. This kind of message distribution model works as the following. The modules send a subscribe message to the communication server containing the type number of the message which they want to receive. The communication server registers them as subscribers, and distributes the network messages containing data to the modules subscribed for. If a module has completed its task, then it unsubscribes from the network messages with a specific message sent to the server right before exiting. This means that after the unsubscribing from a message type, the server will not transmit the corresponding packages to the network interface of module.

As it can be seen the communication server handles dynamically the subscribe-unsubscribe requests from the modules.

This scenario ensures the minimal load on the communication channel.

B. The robot interface

The robot interface module is responsible for all hardware-dependent tasks. This can be considered as a hardware abstraction layer, because no other modules have access to the hardware. This module has three main tasks to perform:

- handling the robot's built-in encoders to read odometry information,
- handling the external sensors, specifically in our case, a laser scanner to obtain the distance measurements
- accepting the control commands from the controller module and executing them on the robot.

After reading out the encoder data and other sensory information, these are serialized into the proper wrapper class, and sent through the network to the SLAM module.

The robot interface module obtains the motion commands from the network originating from the controller module and executes them. The control command network message incorporates linear and an angular speed values. The robot interface transforms the values according to the moving robot's physical model, to use them as a motion command.

Raw sensor data is used for collision detection. If there is a laser distance measurement which has a lower value than a given threshold, the module immediately stops the robot by setting the wheels' speed to zero. The operator can enable the command execution again, but the robot will move only if there is no distance measurement under the distance threshold. The threshold for collision detection can set according to the given experimental situation and the physical dimensions of the moving robot. This type of collision detection is practical in very dense indoor environments, where there is no possibility for replanning due to time restriction or lack of free route.

C. Map building and localization

This module has a central role in the whole system's functionality. After receiving the odometry and laser data it performs the simultaneous localization and map building task which is fundamental for the navigation of the mobile robot. We selected a scan-matching based approach for the SLAM problem [10]. The output of this module is an occupancy grid map and a probabilistic density function describing the estimated robot pose.

The localization part is solved by the Iterative Closest Point algorithm [11]. ICP itself is incapable to track multi-hypothesis of the robot pose. For that reason, we used a particle filter together with ICP to have multi-hypothesis of the estimated pose. The ICP algorithm provides a measure of the quality of the registration process, a "goodness" value to show the fitting of the last measurement set to the previously built map. During the tests, we compared the properties of the classic ICP and the Levenberg-Marquardt optimization [12]. Based on these results we selected the second one for further usage.

The map building part differentiates free space from the obstacles by integrating the sensor measurements into an occupancy grid map using the estimated robot pose. This map serves as a basis of path planning. The result of the map building is an updated metric map that integrates the new sensor readings.

The algorithm can be parametrized via an initialization file where several options can be changed, namely the properties of the generated map, the cell update properties of the map, the type and parameters of the ICP algorithm, etc.

The SLAM module generates precise log files during its work that helps to analyze the performance and quality parameters of the algorithm with the applied parameter set on the given data. With the help of this, some useful information were obtained about the properties of the algorithms, for example memory usage, execution time.

The MRPT contains the implementation of the two traditional SLAM approach, namely the feature-based SLAM [6] and a scan-matching based SLAM . The measurement result of a laser rangefinder can be easily processed by a scan-matching algorithm. The particle filter based ICP is implemented in the MRPT. The design of the software system enables to test more map building techniques, and use the one which seems to be optimal considering the given project's specialties.

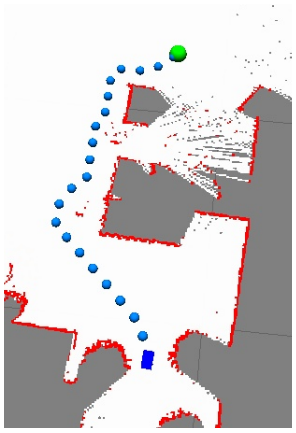

Fig. 4. Scan-matching based SLAM built metric map. In the bottom region the robot, in the top region the goal point is located, between them, a planned path can be seen.

D. Path planning

Path planning is a task where an autonomous mobile robot is calculating a collision-free trajectory between two points in its environment. To solve the trajectory planning problem in a dynamic environment, one needs to know the properties of the autonomous system (e.g. the motion model of the robot), and the robot has to be able to map its operation environment. Then the task is to plan an appropriate motion between two given configurations such that some additional constraints (e.g. collision avoidance, maximal allowable control signals) are satisfied as well. A planned path on a map can be seen in Figure 4.

The path planner module subscribes for the messages originating from the SLAM module: the current map and the estimated position of the robot on it, and from the user interface: the desired goal position. The main function of the path planner is the low-frequency path planning, which means finding a free route from the current position to the desired target. The output of this module is the planned path, which will be used by the trajectory following controller to drive the robot along it. The planned path is a set of free points in the map, in an equal distance from each other. This discrete set of points has to be transformed into a continuous and smooth path to drive on; this task is completed by the controller module.

In the case of mobile robots in dynamic environment one of the most important task of the path planner is replanning. The change in the map based on sensory data, or the change of the goal position by a user command results that the current path is not free or valid any more. In this case the replanning of the path is needed. Some algorithm used in this field is developed for fast replanning, by using the previous planning step's results to speed up the procedure.

The path planning task can be considered as a searching problem in the configuration space determined by the current map. Two main methods are used in our system: the A* [13] and the D* Lite algorithm [14]. D* Lite is a state-of-art search algorithm, capable for fast replanning.

Similarly to other modules, changing the core algorithm in the module is quite easy. The parameters of the path planning are described in a separated initialization file.

Trajectory is a path which is an explicit function of time. The motion is defined by a path geometry and by a velocity profile which describes the time distribution along the reference path. Considering this statement, it is essential to add a time parameter to the planned path to ensure that the controller can drive the robot on the path. This time scaling problem is an important part of motion planning because the capabilities of the vehicle, the control algorithm and the requirements of the task have to be considered at the same time.

E. Trajectory following control

Controlling a mobile robot on a reference path requires the usage of a proper control algorithm. The controller generates the motion commands on a way to minimize the difference between the actual position of the robot and the desired path. Feedback control is needed to handle the complex requirements introduced by the disturbances, difficult reference paths and other problems. Because of all these things the designing, tuning and implementing of a well-operating controller is a challenging task.

The incoming data from the path planner is a discrete finite set of free points along the desired path. With the help of curve fitting the set of points is converted into a continuous line. In this system a B-spline interpolation technique is used to get the curve. As a result the paths have a smooth curvature with continuous derivatives. Constraints such as velocity and acceleration limits or maximum curvature could be considered, too [15]. The output of the module are the motion commands, namely the desired angular and longitudinal speed values. These commands are received and executed by the robot interface.

The control algorithm is using a virtual vehicle approach. This means that there is a virtual vehicle moving on the desired trajectory. The control task is to minimize the error between the virtual vehicle's state and the real vehicle's state. In such a problem, the controller has to deal with two problems. In one hand, it is needed to control the virtual vehicle's speed according to the trajectory's properties and the distance between the virtual and the real vehicle. On the other hand, it calculates the real vehicle's motion commands to decrease the error which consists of the position and orientation differences between the real and the virtual vehicle. The control of the virtual vehicle is based on [16].

In our case, the controlled vehicle is a differential driven robot. The controller which is used in our implementation is detailed in [17]. First the position error values are calculated comparing the desired and estimated position. With the properly selected feedback gains the feedback input can be calculated. Finally the motion commands' values are computed.

During the tests the controller worked on 20Hz, and it drove the robot fast and accurately without vibration or other noises and could handle complex motion tasks, e. g. Y-turns.

The modular architecture of the system enables us to change the controller module also. MRPT contains a tool to navigate differential driven robots, called Reactive Navigation package.

F. Further auxiliary modules

In our system, there are several auxiliary modules which help handling the robot and the generated data. Namely, there is a data recorder module (called logger), a data playback module (called logplayer), a visualization module as a user interface, and a module to monitor the state of selected modules on-line.

Logger and logplayer modules are using the MRPT built-in class serialization techniques to write the data into files and read it back.

MRPT has a built-in OpenGL abstraction layer for 3D object visualization, which is used in the user interface module. The 3D objects in this case are the robot, the map itself, the goal position and the elements of the planned path. The user can select a point on a map as a goal position which triggers the re-planning of the path by the path planner module.

Monitoring the output of selected modules is useful when the robot operates in a real-world environment: with the help of the proper module, the operator can observe the trends in different module's latencies, motion command values, properties of the paths and other important values.

Further modules (e.g. parameter server, alternative control interfaces, etc.) can be implemented and integrated into the system, as it is needed by the given project.

VI. Applications

Our long term aim is to develop a robotic wheelchair system. Autonomous robotic wheelchairs are receiving increasing attention all over the world due to the need of easier wheelchair mobility caused by longer life spans in ageing societies. There are many people with partial disabilities who could benefit from semi-autonomous/autonomous wheelchairs. Standard electric wheelchairs have been in use for a long time, but many disabled people cannot control an ordinary electric wheelchair while independent mobility would be a huge advance in their life.

These motivations encouraged us to create the previously introduced mobile robotic software. The highlighted requirements introduced in Section I are crucially important in the case of an autonomous wheelchair.

The developed system was widely tested in several environments. The robustness of the modules were tested with several parameter combinations and also in difference circumstances, e.g. unexpected module shutdown, with various network delay, etc.

A commercially available PowerBot robot was used as a testbed. All the functionalities of the Powerbot are accessible via AriaLab [7]. The hardware abstraction module was located on the on-board computer of the PowerBot. The robot was equipped with Sick LMS-100 laser rangefinder which performed with high accuracy in indoor environment [18].

For a testing environment a small-scale, dense environment and a large scale indoor environment were used. In a small-scale scenario the task of the robot was to navigate in a classroom, where several objects were located. The long term behaviour of the system was investigated in our university main hall and in an exhibition center.

The trajectory following control tests were done in both scale. We manually selected a sequence of goal points on the map. We paid lots of effort to solve the problem of the synchronization of the virtual vehicle and the actual robot, which is essential for a smooth control. Path planning can not produce on-line speed in large environments due the size and density of the map.

In these scenarios the modules performed their tasks on the desired level, the current state of our system enable us to navigate several types of robots with high accuracy.

VII. Conclusion

In this paper, we presented the design and implementation of a high-level, modular mobile robotic framework, based on the MRPT toolkit. The contributions of the work presented in this paper are the following.

The main design principles and architectural requirements were considered. The analysis of the architecture is presented to show the main properties of the designed system.

To show the usability and effectiveness of the proposed system model and the selected toolkit, several modules' implementation and functions are detailed. Each of them plays an important role in a mobile robot navigation task, and together can operate reliably as a coherent software environment.

As a future work, further modules will be created, e. g. dynamic object detection and tracking module with the help of the Joint Probabilistic Data Association Filter [19]. Further developments in the system architecture can improve performance and can facilitate integration new algorithms and tools.

Acknowledgements

This work was partially supported by the Hungarian National Fund (OTKA K-67625) and by the NTP-OKA-I. Gábor Szederkényi is a grantee of the Bolyai János Research Scholarship of the Hungarian Academy of Sciences. Special thanks to György Cserey (Head of PPCU Robotlab) and Sick GmbH. We would like to thank György Lipták's work.

References

[1] *The Mobile Robot Programming Toolkit*. http://www.mrpt.org.

[2] R. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2:14–23, 1986.

[3] Toby H. J. Collett and Bruce A. Macdonald. Player 2.0: Toward a practical robot programming framework. In *in Proc. of the Australasian Conference on Robotics and Automation (ACRA)*, 2005.

[4] Michael Montemerlo, Nicholas Roy, and Sebastian Thrun. Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (CARMEN) toolkit. In *In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS*, pages 2436–2441, 2003.

[5] Carle Côté, Yannick Brosseau, Dominic Létourneau, Clément Raïevsky, and François Michaud. Robotic software integration using marie. *International Journal of Advanced Robotic Systems*, 3(1):055–060, March 2006.

[6] M. W. M. Gamini Dissanayake, Paul Newman, Steven Clark, Hugh F. Durrant-whyte, and M. Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on Robotics and Automation*, 17:229–241, 2001.

[7] *ARIA Developer's Reference Manual*. http://turing.bard.edu/~sven/robotics/Aria/docs/main.html.

[8] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, Jun 1989.

[9] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (slam): part i-ii. *Robotics & Automation Magazine, IEEE*, 13:108–117, 2006.

[10] Jorge L. Martínez et al. Mobile robot motion estimation by 2d scan matching with genetic and iterative closest point algorithms. *Journal of Field Robotics*, 2006.

[11] P.J. Besl and H.D. McKay. A method for registration of 3-d shapes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14(2):239–256, Feb 1992.

[12] Andrew W. Fitzgibbon. Robust registration of 2d and 3d point sets. In *In British Machine Vision Conference*, pages 411–420, 2001.

[13] A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, February 2007.

[14] S. Koenig and M. Likhachev. D* Lite. In *Proceedings of the National Conference on Artificial Intelligence*, pages 476–483. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002.

[15] Szádeczky-Kardoss Emese. *Real-Time Motion Planning for Nonlinear Nonholonomic Mechatronic Systems Using Time-Scaling*. PhD thesis, Budapest University of Technology and Economics Department of Control Engineering and Information Technology, April 2009.

[16] M. Egerstedt, X. Hu, and A. Stotsky. Control of mobile platforms using a virtual vehicle approach. *IEEE Transactions on Automatic Control*, 46(11):1777–1782, 2001.

[17] G. Matko Klancar and S. D. Blazic. Mobile robot control on a reference path. *Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation*, 2005.

[18] J. Rudan and Z. Tuza. Using LMS-100 laser rangefinder for indoor metric map building. In *Proceedings of 2010 IEEE International Symposium on Industrial Electronics (ISIE2010)*. IEEE ISIE, 2010.

[19] Dirk Schulz, Wolfram Burgard, Dieter Fox, and Armin B. Cremers. People tracking with a mobile robot using sample-based joint probabilistic data association filters, 2003.