Proceedings of the 2000 IEEE
International Conference on Robotics & Automation
San Francisco, CA • April 2000

It's Time to Make Mobile Robots Programmable

Yutaka J. Kanayama

MotionLab, Inc.
Monterey, CA 93940
kanayama@motionlab.com
www.motionlab.com

C. Thomas Wu

Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943
ctwu@cs.nps.navy.mil

Abstract

This paper discusses the importance and realizability of the "high-level programmability" concept for wheeled mobile robots. A new high-level mobile robot language called Motion Description Language is proposed. This language has a strong and stable theoretical foundation, "robot geometry," which is briefly described. The Java-based object-oriented MDL implements the structured robot geometry into its class hierarchy perfectly. The language MDL is not only "a" mobile robot language, but is a strong candidate for "the standard" mobile robot language. This innovation will thrust the mobile robotics field forward to meet the challenges of the new century head on and make significant contributions to the society.

1 Introduction

Electronic digital computers have been programmable from the time of ENIAC. All the powers of digital computers come from the fact that they are programmable. When we use the term "programmable," we are referring to the programmability above the hardware level, that is, hardware-independent programmability. This concept leads to the development of standard computer languages, such as Fortran, Lisp, Pascal, Ada, C, C++, and Java. The far reaching impact of computers, including the systems embedded in various devices, to the society is undeniable. On the hardware side, it is due mainly to the advances in VLSI, and on the software side, it is due mainly to the existence of the standardized high-level programmability.

It is natural for us to apply the concept of high-level programmability in robot control. In the robot manipulator field, we see a high-level programmability with languages designed and maintained by individual manufacturers. It is realized by many that the existence of a standard is preferable, and there have been several attempts to develop a standard language for the robot manipulators [1,2]. In Japan a standard robot language SLIM was approved as JIS B 8439-1992 "Industrial robots - Programming Language" [3], although a similar attempt at the Organization for International Standards (ISO) failed to reach any agreement. Despite these efforts, there is no significant successful accomplishment in this language-standardization missions.

Unfortunately, the wheeled mobile robotics field does not even support a high-level programmability, except for a simple language with limited capabilities [4]. This lack of high-level programmability is the major reason for the field not being able to make significant contributions to the society. We will propose in this paper a high-level mobile robot language called the *Motion Description Language* (MDL). We will discuss why MDL is suitable as the standard high-level language for mobile robots in Section 4.

2 Proposed Mathematical Model—Robot Geometry

Traditionally, the mobile robotics field has attracted researchers to investigate their own motion-planning algorithms and control architectures [5, 6]. Because of the diversified algorithms and architectures, it is difficult to consider a unified mathematical model in mobile robotics. That is one reason why a high-level robot language was non existent.

Mobile robots are living in the two-dimensional plane in a stream of time. Therefore, geometry is a basic tool for mobile robotics. However, when we spell out mathematical models needed, we find some critical differences between the conventional geometry and the geometry for mobile robotics. We call this specialized geometry the *robot geometry*. Only after we establish such a firm basis, we will be able to achieve high-level programmability.

The role of geometry is more important in robotics than the one of the control theory. Most robotics problems are related to motions whose speeds are relatively low or even zero. The control theory is not much effective if the time-derivatives are small or if the world is static. Another reason is that the control theory is not invented to solve problems related to a two-dimensional plane which is a spatial entity in nature. We need new mathematics to deal with robotic problems. An example is the generalized-voronoi-diagram concept, which belongs neither to the conventional geometry nor to the control theory [7]. We therefore propose the robot geometry to deal with the problems in mobile robotics, replacing conventional disciplines. "Only new wineskins are used to store new wine." (Matthew 9:17)

2.1 Frame and Transformation

First we define a *global* Cartesian two-dimensional frame in the world. A mobile robot is a rigid body, and hence its placement is determined by its position $p = (x, y)$ and orientation θ both in the global frame. Although the positional

information (x, y, θ) is historically called a *configuration* [8], we interpret this information as a *frame* instead.

In the robot geometry, this is considered as a pair of a point and an orientation rather than a triple of the coordinates and an orientation. We assume a robot frame pasted on the robot (Fig. 1). Namely, $R = (p, \theta)$ is a robot frame with respect to the global frame. Specifically, $((0, 0), 0)$ is the *reference frame*.

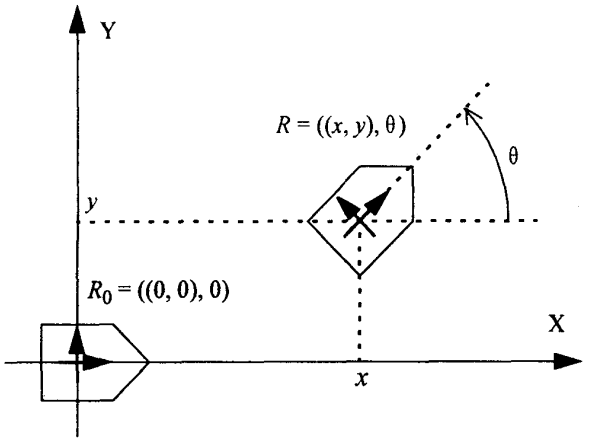


Figure 1. Frame and Transformation

A *transformation* transforms a frame into another frame [9]. A frame $F = (p, \theta)$ can be considered as a transformation T from the reference frame into the frame F. A transformation (p, θ) is the composition of a translation $(p, 0)$ and a rotation $((0, 0), \theta)$. The two basic functions, *compose* and *invert*, are only needed in any complex computation among transformations. Actually the set of all transformation forms an algebraic group with the operation "compose." It turns out this formulation is extremely useful in computing most of the spatial relations in robotics. For instance, the odometry function for a mobile robot is embodied by successive transformations on the robot frame R [8].

An orientation is a real number. For instance, the heading θ of a robot is an orientation. Suppose its initial value is 0 and the robot runs along a circular track counterclockwise. In this situation, it is more natural for θ to increase 2π for every turn than resetting θ to 0 whenever θ comes to 2π. This is the reason why the range of an orientation is the set of real number rather than the range of $[0, 2\pi]$ or $[-\pi, \pi]$. (This is one of the differences between the robot geometry and the conventional geometry.) An *angle* is defined as the difference between two orientations. Since an orientation is any real number, so is an angle. Sometimes we need to normalize an angle into the range of $[-\pi, \pi]$ in the robot geometry.

2.2 Robot's Motion: (v, ω) or (v, κ)

We have seen the static placement of a robot has three degrees-of-freedom (DOF). Likewise a two-dimensional rigid body's motion has basically three DOF. An omni-directional vehicle is a vehicle that has the complete three DOF. On the other hand, a "normal" vehicle such as a car-like vehicle or a differential-drive vehicle has one fixed wheel-axle. Because of the fixed axle, there is a constraint in its motion such that the motion direction at O_R must be equal to the robot X-axis, and its motion has only two DOF [10].

The most general selection of two variables in the "normal" robot's motion is its linear speed v and rotational speed ω. This formulation (v, ω) represents the *spin* motion type with $v = 0$ and $\omega \neq 0$. On the other hand, if we need to deal with only the non-zero linear-speed motions, another pair (v, κ) of the linear speed v and trajectory curvature κ works as well. The relation between the three variables is

$$\omega = \frac{d\theta}{dt} = \frac{d\theta}{ds} \cdot \frac{ds}{dt} = \kappa v \quad , \qquad (1)$$

where t, s, and θ are time, trajectory length, and vehicle heading respectively. Therefore, if $v \neq 0$, the motion (v, ω) is transformed into the second form, $(v, \kappa) = (v, \omega / v)$. Using this mathematical model, it is needed to prepare the linear speed (v) control and curvature (κ) control algorithms. We can prepare the rotational speed (ω) control algorithm if needed for spin motions. The motion (v, ω) or (v, κ) computed is called *commanded motion*, because this is the motion the robot is supposed to perform.

2.3 Continuous Curvature

Since the derivative of the rotational speed ω is proportional to the torque applied to the robot and the torque is a finite value, ω must be continuous. Therefore, by Equation (1), the curvature κ also must be continuous [10]. This property makes the mobile robot control difficult. Tracking a path consisting of only circles and lines cannot be executed precisely by any mobile robot, because such a trajectory has curvature discontinuity. We must come up with an algorithm that produces continuous curvature (See Subsection 2.6).

2.4 Path and Object Boundary

There are two classes of curves in the robot geometry: (a) A *path* is a curve for the robot to track. By nature, a path has a direction. We consider an east bound and west bound paths on the same line distinct. (b) The *boundary* of an object in the world is a curve. Although an object boundary itself does not have any direction, notice that the only one side of an object boundary can be accessed. Therefore, the following convention is extremely useful to be agreed: An object boundary has a direction so that only its right side can be detected by sensors.

In our experience, several elementary geometric entities in robotics has a natural direction (plus/minus, east-bound/west-bound, counterclockwise/clockwise, and so forth). This is another striking feature of the robot geometry. Lines, circles, and polygons are at least needed to describe paths and object boundaries.

2.4.1 Lines

A directed line is defined by a point p on it and its orientation θ (Fig. 2). Namely, specifying a directed line is equivalent to specifying a frame $F = (p, \theta)$. In the ordinary geometry, the distance from a point to a line is always non-negative. However, we consider the distance from a point to a directed line being positive, negative or 0. If the point is on the left (right resp.) side of the directed line, the distance is positive (negative resp.) . This computation is done by the transformation group algebra [9].

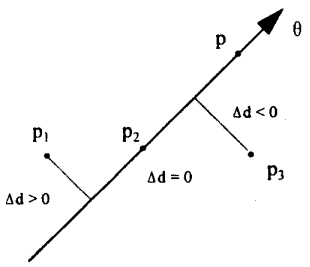


Figure 2. Directed Line (p, θ) and Signed Distance

If a line is the boundary of an indefinitely long wall, this line's right side is open and the left side is filled. A directed line $(p, \theta + \pi)$ is said to be a *complement* of a directed line (p, θ), and vice versa.

2.4.2 Circles

A directed circle can be specified in two ways: (1) A directed circle is defined as a pair (c, r) of its center point c and a non-zero signed radius r. If $r > 0$, the circle is counterclockwise. Otherwise it is clockwise. A circle (c, $-r$) is said to be a *complement* of a circle (c, r), and vice versa. (2) A directed circle is defined as a triple (p, θ, κ), where p is an arbitrary point on the circle, θ the orientation of the tangent to the circle at p, and κ its curvature. If $\kappa > 0$, the circle is counterclockwise; otherwise clockwise. A circle $(p, \theta+\pi, -\kappa)$ is said to be a *complement* of a circle (p, θ, κ), and vice versa. (Fig.3)

The boundary of a circular object (obstacle) is counterclockwise and the boundary of a circular world is clockwise by definition. (Fig. 3)

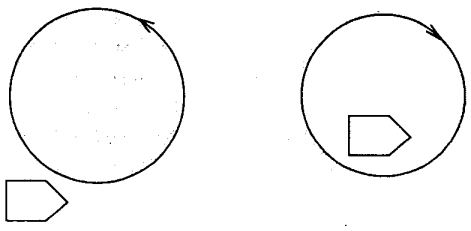


Figure 3. Circular Path and Circular Boundary

2.4.3 Polygons

A polygon is a list $(p_1, p_2, ..., p_n)$ of n points ($n > 2$), where (a) no three consecutive points are colinear, and (b) no two edges share common points, except in the case where the two edges are consecutive and they share a point p_i with $i \in [1, n]$. A line segment (p_i, p_{i+1}) with $i \in [2, n-1]$ or a line segment (p_n, p_1) of a polygon is called an *edge*. A polygon $(p_1, p_2, ..., p_n)$ and a polygon $(p_n, p_{n-1}, ..., p_1)$ are said to be a *complement* of each other.

A polygon can be considered as the boundary of an object. If the order of the points $(p_1, p_2, ..., p_n)$ is counterclockwise, it is the boundary of an object (Fig. 4). On the other hand, if the order of the points $(p_1, p_2, ..., p_n)$ is clockwise, it is the boundary of the world.

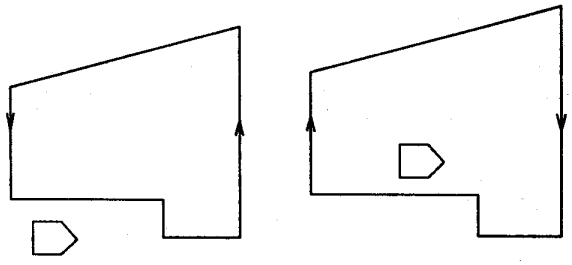


Figure 4. Polygon Boundaries

2.5 Geometrical Model of World

A world model for a robot is described by points, circles, polygons, other geometric entities, voronoi graphs, and Homotopy classes [11]. A robot can perform useful tasks if it is supplied with an appropriate world model.

2.6 Motion Control Models

The heart of the mobile robot functions is the set of motion control capability. As discussed in Section 2.2, a normal mobile robot has two DOF. What control rules for these two variables are needed to perform useful, safe, or interesting? Typical motions include: (a) A geometrically defined motion, (b) Spinning, (c) A person-following motion using sonars, (d) A wall-hugging motion, (e) A planned motion with a world model, (f) A random motion, (g) A remote controlled motion, (h) Exploring in an unknown world, (i) A cooperated task execution with other robots, (j) A hostile behavior against humans or other robots, and (k) Dancing.

A motion control model is any algorithm that may use the sonar information and/or the world model. Therefore, the set of mathematical models must allow a wide variety of motion selections for any application.

The most fundamental motion control model is the path-tracking model. A path is a sequence of directed lines and circles and a robot is supposed to track it. The more detailed

description of the algorithm is given in [12, 13]. (See the program examples in Section 3.2)

2.7 Sonar Model

An autonomous mobile robot with no external sensors can perform only a very limited set of motions. We propose the inclusion of ultrasonic range finders (sonars) as the standard components of a mobile robot. Advantages of sonars are that they are inexpensive and that they do not require significant CPU processing time. When a sonar obtains a return echo, the estimated position of the object that produced the echo is computed by

$$F = R \cdot S \cdot ((d,0),0),$$

where F is the frame of the object, R the robot frame, S the sonar frame local to the robot frame, and d the range detected by the sonar. Each of the two dot-operators in the right hand side is the compose operator in the transformation group [9]. A sequence of these points will give a least-squares fitted line-segment result, [14] which eventually contributes to real-time localization.

3 Motion Description Language

3.1 Overview

We must design a concrete mobile robot language based on the robot geometry for the theory to be useful in practice. To build a mobile robot language, one can either build a completely new language or use an existing programming language as its basis. The first approach is simply too time-consuming and labor-intensive. Moreover, there's no compelling technical advantage to design a special-purpose robot language. We therefore decided on the second approach. In selecting the existing programming language, we established the following criteria:

1. The language is powerful enough to implement effectively all the mathematical models embodied in the robot geometry.
2. Language is easy to understand and use so the user can concentrate on robot control, not on the language syntax and semantics.
3. Compiler and other development tools for the language are widely available.

To satisfy Criterion 1, the language must be object-oriented. Traditionally, the C programming language was used most often for programming robots because it is effective in writing low-level routines. However, the C language does not provide a mechanism powerful enough to model high-level concepts such as transformation, path, segment, polygon, and others. The object-oriented class library is the most natural and effective way to model our proposed mathematical models. Object-orientation also allows the user to extend the predefined capabilities using the inheritance and other OOP mechanisms.

To satisfy Criterion 2, the language must not be too complex or esoteric to a majority of programmers and engineers. The language must not possess too many "tricky" features that can interfere with the user's development. The language should not require a steep learning curve before the user can become proficient in the language. We want the user to program a mobile robot without being a programming language expert. For example, C++ has many difficult-to-understand features which the user must grasp correctly before being able write programs.

To satisfy Criterion 3, the development tools must be available from many different vendors. We do not want, for example, a language sold only by a single company no matter how good is the language. We intend our selected language to be used worldwide, so the availability of software development tools for the language is very important

After consideration, we chose Java as our implementation language, C++ being our second choice. Java met all three criteria far better than any other languages. Moreover, in addition to satisfying the above criteria, Java gives us the following advantages: (1) Java is executable on all types of platforms, from the embedded real-time OS to the desktop system such as Windows. This feature will allow the user to keep open a wide range of options in choosing the hardware in creating a mobile robot. (2) Java allows the user to write a program with its modern, feature-rich capabilities, for example, writing a program to control the robot via the Internet.

3.2 MDL Classes

In this subsection, we will describe briefly some of the key MDL classes that implement the mathematical models described in Section 2. The user writes his *user program*, or task program in ISO standard [15], in Java using the classes from the MDL class library (*package* in the Java terminology). The high-level programmability provided by MDL separates the system level and application level programming. A user can program the robot for an individual application quickly because he does not have to deal with the system level details. All the system level details are hidden inside the MDL classes. This separation makes possible to use the same application level program on mobile robots with different hardware architecture.

To present the classes in an organized manner, we will explain the classes in logical groups:

1. Transformation
2. Segment
3. Robot
4. Motion
5. Sonar
6. World
7. Hardware Dependent Vehicle

The *Transformation* group includes classes for handling the positions, frames, and transformations of robots.

The *Segment* group includes classes for representing the segments that the robot will track. Four classes in this group are Segment, Line, Circle, and Path. The Segment class will derive no instances, and as such, the class is declared as abstract. The Segment class includes code common to its

two subclasses Line and Circle. A Path object represents a sequence of segments which the robot will track.

The *Robot* group includes classes for individual robots. At present, we have three classes: Robot, Swan, and SimulatorRobot. The Swan class represents the prototype robot named *Swan* that MotionLab has built for experimentation and demonstration. The superclass Robot is an abstract class that includes code common to its subclasses Swan and SimulatorRobot.

The *Motion* group includes the classes for carrying out various motion functions. The Motion classes include information related to motion modes, path, teaching data, odometry, speed/acceleration, and so forth.

The *Sonar* group includes the hardware-independent classes for carrying out sonar-related functions. The Sonar classes encapsulate sonar specific details, such as its local (robot) frame, object range and position, linear-fitting, and localization. These values are accessible from the user program in real-time.

The *World* group includes the classes for defining the geometrical relations of entities in the world which the robot lives in, and their generalized voronoi graph.

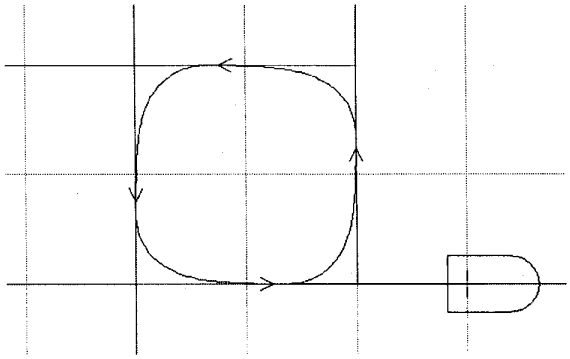
The *Hardware Dependent Vehicle* group includes the classes that encapsulate hardware specific details. At present we have three classes Vehicle, SwanVehicle and SimulatorVehicle that capture the hardware information. The actual hardware information of our prototype robot *Swan* are coded in the SwanVehicle class The SimulatorVehicle is an idealized vehicle used by the SimulatorRobot. The Vehicle class is an abstract superclass of the other two classes. Since all other classes are common to any mobile robot, the user has to define only a very small number of classes in the hardware independent vehicle group in order to port his program to a new mobile robot with a different hardware. This software reusability will greatly enhance the end-user productivity.

3.3 Sample User Programs

To illustrate the use of the MDL class library, we present two sample user programs. The first sample code makes the robot tracks a sequence of lines that form a square. Here is the program:

```
 1: double size             = 100.0;
 2: double smoothnessFactor = 0.08;
 3:
 4: robot.setSmoothness( smoothnessFactor * size );
 5: robot.setFrame( 0.5* size, 0.0 , 0.0 );
 6:
 7: Line line;
 8: line    = new Line( 0.0, 0.0, 0.0 );
 9: robot.track(line);
10:
11: line = new Line( size, 0.0, Math.PI/2.0);
12: robot.track(line);
13:
14: line = new Line( size, size, Math.PI);
15: robot.track(line);
16:
17: line = new Line( 0.0, size, 3.0*Math.PI/2.0);
18: robot.track(line);
19:
20: line = new Line(0.0, 0.0, 2.0*Math.PI);
21: line.setStopPoint( new Position( 0.5*size, 0.0) );
22: robot.track(line);
```

To execute this code we must place it in the right Java program structure (e.g., defining a class with the main method). Instead of showing a complete program, we will only list and discuss the core statements that are relevent to the MDL concepts. In the program the variable robot refers to a robot, either the simulator robot or the real *Swan* robot. In Lines 4 and 5, we set the robot's smoothness and frame. From Lines 8 to 21, we create a sequence of lines that form a square. After each line is created, we let the robot track it (Lines 9, 12, 15, and 21). For the very last line, we designate a stop point (Line 22), so the robot will stop tracking when it reaches the designated stop point. Figure 5 shows the simulation result.



Each grid is 50 cm by 50 cm. The arrowheads indicate the point the robot switches from the current segment (line) to the next segment (line).

Figure 5. The simulation result of tracking the square.

The second sample code is slightly more involved. This program makes the robot exhibit a motion that simulates the vacuum cleaning of a room. Here is the program:

```
 1: double size             = 80.0;
 2: double verticalIncrement = 20.0;
 3: double radius           = 0.8 * verticalIncrement;
 4: double smoothnessFactor  = 0.5;
 5:
 6: int    numberOfTurns     = 4;
 7:
 8: robot.setSmoothness( smoothnessFactor * radius );
 9: robot.setFrame( -size, 0.0 , 0.0 );
10:
11: Line   line;
12: Circle circle;
13: int    sign = +1;
14:
15: for (int i = 0; i < numberOfTurns; i++ ) {
16:   line = new Line( 0.0,
17:                    2*i*verticalIncrement,
18:                    (i%2)*Math.PI );
19:
20:   circle = new Circle( new Position( sign*size,
21:                        (2*i+1)*verticalIncrement),
22:                       sign*radius );
23:
24:   robot.track(line);
25:   robot.track(circle);
26:
27:   sign = -sign;
28: }
29:
30: line = new Line( 0.0,
31:                 2*numberOfTurns*verticalIncrement,
32:                 (numberOfTurns%2)*Math.PI );
33: line.setStopPoint( new Position( sign*size,
34:                 2*numberOfTurns*verticalIncrement) );
35: robot.track(line);
```

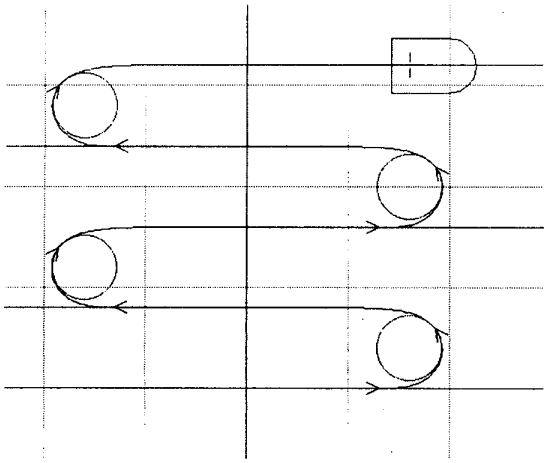


Figure 6. The simulation result of vacuum cleaning the room.

Figure 6 shows the simulation result. This rather complex motion is expressed by a straightforward MDL program. The for loop (Lines 15 through 28) is repeated four times, with each iteration creating a pair of a line and a circle. The mathematical expressions used for the parameters in creating lines and circles compute the positions so the pairs are stacked vertically. The last line is created outside of the loop (Lines 30 - 32), and the stop point is attached to it (Lines 33 - 34).

8 MDL as the Standard Mobile Robot Language

MDL is just "a" mobile robot language, but we believe MDL will eventually become "the" standard mobile robot language. There are several reasons:

1. MDL is based on the highly advanced robot geometry. The robot geometry is built on a sound and complete theory. It can be used to describe any motion-related tasks. Moreover, we are confident that the robot geometry cannot be simplified any further without losing its expressiveness and power. In other words, the robot geometry is minimal, although there may be additions.
2. The robot geometry has been taught to the students with diverse programming skills at the Naval Postgraduate School (NPS) for many years. The students found the robot geometry easy to understand and program in any language including C, C++, Lisp, Prolog, and Java. Less reliance on any particular programming language improves the likelihood of MDL being accepted by many as the standard.
3. A portion of the robot geometry was successfully implemented in the mobile robot language MML constructed on the C language at the University of California and at NPS [7, 8, 9, 10, 12, 14]. This experience demonstrates that the mathematical model is extremely suitable for building the robot software system.
4. Comparing the past experience with C and the current experience with Java gives us an added confidence that the current object-oriented implementation will contribute significantly to the user productivity, making MDL more appealing and more structured to many users. (Our current implementation is done in Java, because it is the best language currently available, but it is not a requirement. MDL can be implemented in other object-oriented programming languages as well.)

5 Conclusion

Despite the many high hopes, the mobile robotics field has yet to devise a technology that contributes significantly to the society. We proposed in this paper a new technology—a high-level programmability for mobile robots—that will be a significant contribution to the society. Our proposed high-level programmability is based on the robot geometry and the corresponding MDL, which is a strong contender to become the standard mobile robot language.

6 References

[1] S. Mujtaba and R. Goldman, "AL Users' Manual," Stanford University, Report No: CS-TR-81-889, December 1981.

[2] R. A. Volz, "Report of the Robot Programming Language Working Group: NATO Workshop on Robot Programming Languages," IEEE Journal of Robotics and Automation, vol. 4, no. 1, pp. 86-90, 1988.

[3] A. Matsumoto and T. Arai, "Japanese Standard Robot Language SLIM and its Educational System," Proc. Symposium on Manufacturing Application Programming Language Environments (MAPLE'93), Ottawa, Canada, October, pp.117-126, 1993.

[4] Y. Kanayama and M. Onishi, "Locomotion Functions in the Mobile Robot Language, MML," Proc. IEEE International Conference on Robotics and Automation, in Sacramento, California, April 9-11, pp. 1110-1115, 1991.

[5] J. Latombe, "Robot Motion Planning," Kluwer Academic Publishers, 1991.

[6] S. Iyengar and A. Elfes, "Autonomous Mobile Robots," vol. 1, Perception, Mapping, and Navigation, and vol.2, Control, Planning, and Architecture, IEEE Computer Society Press, 1991.

[7] F. Preparata and M. Shamos, "Computational Geometry, an Introduction," Springer-Verlag New York Inc., 1985

[8] T. Lozano-Pérez, "Spatial Planning: A Configuration Space Approach," IEEE Transactions on Computers, C-32(2), pp.108-120.

[9] Y. Kanayama and G. Krahn, "Theory of Two-Dimensional Transformations," IEEE Journal of Robotics and Automation, vol.14, no.5, pp. 827-834, October 1998.

[10] Y. Kanayama, "Two Dimensional Wheeled Vehicle Kinematics," Proceedings of International Conference on Robotics and Automation, in San Diego, California, May 8-13, pp. 3079-3084, 1994.

[11] P. Hilton, "Algebraic Topology," Courant Institute of Mathematical Sciences at New York University, 1969.

[12] Y. Kanayama and F. Fahroo, "A New Line Tracking Method for Nonholonomic Vehicles," Proc. the IEEE International Conference on Robotics and Automation, in Albuquerque, New Mexico, April 21-27, pp. 2908-2913, 1997. Also will appear in Advanced Robotics journal.

[13] Y. Kanayama and F. Fahroo, "A Circle Tracking Method for Nonholonomic Vehicles," Proc. the Fifth IFAC Symposium on Robot Control, Nantes, France, September 3-5, pp. 551-558, 1997.

[14] Y. Kanayama and T. Noguchi, "Spatial Learning by an Autonomous Mobile Robot with Ultrasonic Sensors," Technical Report of Computer Science Department at the University of California at Santa Barbara, TRCS89-06, February, 1989

[15] International Standards: ISO 8373:1994, "Manipulating industrial robots -- Vocabulary," Organization for International Standards, 1994.