Adam Srebro¹

Java simulator for an autonomous mobile robot operating in the presence of sensor faults

Abstract

This paper presents a two-dimensional mobile robot simulator written in Java. This robot performs autonomous navigation based on infrared sensors. Sensing range and angle can be adjusted in real-time. Obstacle detection and avoidance algorithm have been implemented. In this paper, the wall-following problem for mobile robot in the presence of sensors faults is considered in detail. Time related method of robot orientation relative to the wall is shown. A few experimental tests are reported to discuss the robustness of control algorithm and verify simulations results.

Keywords and phrases: autonomous, mobile robots, simulator, wall-following

Introduction

Nowadays mobile robots can perform desired tasks without continuous human guidance. We call this ability “autonomy”. The growing number of autonomous mobile robots applications require dedicated simulators. The computer simulators are used as a tool for testing control algorithms. In addition, simulators also allow researcher to experiment with various sensors type and configurations without the need to do any changes in hardware. The key advantages are reduction of the time of the experiments and reducing the number of issues to be solved in the early stages of the project. There are a few well-known software tools designed for mobile robot simulation. Webots [1] has been developed in collaboration with

¹ Warsaw University of Technology, srebroa@ee.pw.edu.pl
Swiss Federal Institute of Technology in Lausanne. It provides robot libraries for several commercially available robots allowing transfer controllers to real mobile robots. Microsoft Robotic Developer Studio [2] include Concurrency and Coordination Runtime toolkit for applications that require a high level of concurrency. The Player/Stage/Gazebo Project [3] is open source project designed for Multi-Robot Systems. Stage component is designed to simulation multi-agent autonomous systems in a two-dimensional environment. Moreover there is Gazebo component allowing the 3D simulation for outdoor environment. Apart from the mentioned mobile robot simulators there are numerous other commercial and non-commercial tools supporting robot simulations in the 2D and 3D space [4][5][6].

Designing mobile robot simulators is difficult and time consuming task due to variety of factors influencing on robot behaviour. Furthermore if we consider autonomous robot operating in the presence of faults the simulations and control demands increase rapidly. In this paper, the mobile robot simulator designed by the author is presented. In addition the wall-following problem in the presence of sensors faults is studied to show simulator advantages.

1. Simulator for an autonomous mobile robot

Presented 2D autonomous mobile robot simulator was written in Java. This simulator requires only a standard JDK (Java Development Kit 1.6.0 or higher) and can be run on any platform providing this component (e.g. Windows, Linux, Mac OS X). The main requirement that simulator has to satisfy is ability to do more than one thing at a time. It was fulfilled using java.util.concurrent packages which support concurrent programming. Concurrent programming in Java is based on threads sometimes called “lightweight processes” due to fewer resources requirement than creating a new process. To reuse code, the concept of class inheritance was applied.

1.1. Simulator features

The simulator provides following functionalities:
– single robot simulation
– multi-sensors simulation (by default infrared range sensors)
– static and moving obstacles
– robot path drawing
– collision and obstacle detection
– easily extensible graphical user interface

As assumed this simulator is able to simulate single differential drive mobile robot. It provides flexible graphic user interface allowing edit robot parameters such as linear velocity and platform dimensions. Obstacle detection sensors (typically infrared sensors) are adjustable in wide range. Sensor range and radiation cone can be set for each sensor independently. The control panel shown in Fig.1 gives possibility to switch on and switch off every sensor during the simulation. Simulator was design with the idea of indoor application of mobile robots. However, the user has many possibilities of environment model building. There are included several ready to use static obstacle shapes and possibility to add limited number of moving obstacles (by default 3).
Obstacle position and orientation can be changed during the simulation. User interface allows to set trajectory tracking of the robot center point.

Fig. 1. Control panel.

1.2. Collision detection

The program recognizes two types of interaction between a robot and an obstacle. The first one is obstacle detection by an infrared sensor. Every sensor has visible radiation cone. If the radiation cone intersects with obstacle surface, the intersection points are circled. The second interaction is robot body collision with obstacle surface. Similarly to the first case intersection points are circled. However, in this case robot is stooped and first remembered collision point coordinates are displayed on the screen.

Collision detection in graphics applications is very important factor [7]. Almost every 2D or 3D video game include collision detection algorithm. However, because of limited computing time those algorithms are usually relatively primitive and inaccurate. For example objects are closed in virtual spheres or cylinders and program checks if this solids intersects each other. For a large number of moving object in real time the Gilbert-Johnson-Keerthi algorithm is typically used [8][9].
1.3. Position and orientation in 2D space

We can consider robot motion as motion of rigid body (set of points) in global reference frame [9]. In two-dimensional space all movements of rigid body are translations, rotation, or combination of the two. To rotate rigid body about an arbitrary point \( P_1(x_{b1}, y_{b1}) \) by angle \( \theta \) three steps are needed:

1. Translation the robot so that point P will coincide with origin of coordinates \( P_2(0,0) \)
2. Rotation the robot about the origin by angle \( \theta \)
3. Translation the robot back so that point P will coincide with initial point

The result of these transformations is given below.

\[
T(x_{b1}, y_{b1})R(\theta)T(-x_{b1}, -y_{b1}) = \begin{bmatrix}
1 & 0 & x_{b1} \\
0 & 1 & y_{b1} \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
1 & 0 & -x_{b1} \\
0 & 1 & -y_{b1} \\
0 & 0 & 1
\end{bmatrix} =
\begin{bmatrix}
\cos \theta & -\sin \theta & y_{b1} \sin \theta + x_{b1}(1 - \cos \theta) \\
\sin \theta & \cos \theta & -x_{b1} \sin \theta + y_{b1}(1 - \cos \theta) \\
0 & 0 & 1
\end{bmatrix}
\]

(1)

All transformations such as translation, rotation, scaling or shearing are affine transformation. Affine transform is supported in Java via java.awt.geom.AffineTransform class.

2. Wheeled mobile robot

The autonomous mobile robot used in experiments is four-wheel differential drive robot. Differential system relies on the velocity difference between the right and left pairs of wheels. If the right and left DC motors turn in the same direction at the same speed, the robot moves forward or backward. As shown in Fig.2 in case of speed difference between them, the robot is steering in the direction of the slower driving wheels. This system allows a robot to pivot in its place when right and left wheels turn with the same speed, but in the opposite direction.

Fig. 2. The moving possibilities for differential steering system.
2.1. Robot kinematics

Let’s assume that robot is moving on a horizontal plane at the low speed and has non-
deformable wheels as shown in Fig.3. The robot position in the global reference frame is
determined by a vector with three elements [10].

\[ q = [x, y, \theta]^T \]  

(2)

The kinematic model of differential-drive mobile robot in the global frame is given by the
following equation.

\[
\begin{bmatrix}
\dot{x}(t) \\
\dot{y}(t) \\
\dot{\theta}(t)
\end{bmatrix} =
\begin{bmatrix}
\cos \theta(t) & 0 \\
\sin \theta(t) & 0 \\
0 & 1
\end{bmatrix}
\begin{bmatrix}
v(t) \\
0 \\
\alpha(t)
\end{bmatrix} \Rightarrow \dot{q} = J(q) \dot{\theta}
\]

(3)

where:
- \( x(t) \) – linear velocity in the direction of the \( X_b \) of the global reference frame,
- \( y(t) \) – linear velocity in the direction of the \( Y_b \) of the global reference frame,
- \( \theta(t) \) – rate of change in angular position in the global reference frame.

![Fig. 3. The mobile robot position in global reference frame.](image)

3. Simulations

The key task in all autonomous mobile robot applications is robot position

GPS is commonly used for outdoor navigation. However, for indoor mobile robot
navigation single, suitable solution doesn’t exist [12]. In the simulation environment the
absolute position is usually computed. However, in practical experiments relative position is
measured more often.
3.1. Wall-following problem

The wall-following problem is connected with building a map of an unknown environment. The goal of a wall-following robot is to find a wall in the open or close space and then navigate along the wall at some fixed safe distance. Most commonly used algorithms are based on multi-sensor navigation [13][14][15]. The General idea is to position vector of movement parallel to the wall. Let’s consider positioning robot with symmetrically arranged sensors as shown in Fig.4. If the robot is turning around its center with the constant angular velocity, then the rotation angles between following sensor measurements are constant. According to this rule robot turning time to the parallel orientation can be defined as

\[ t_p = \frac{t_o}{2} = \frac{t_{F4} - t_{F3}}{2} \]  

where:  
\( t_o \) – time measured between frame F3 and F4

![Fig. 4. Robot positioning parallel to the wall.](image)

Equation(4) is true for whole sensor detection range. Time \( t_o \) is always measured for the left sensor and it doesn’t depend on other sensors disturbances and failures. The Fig.5 shows the results of indoor and outdoor environments mapping.

![Fig. 5. Path of the robot in indoor and outdoor environments.](image)
Typical cases have been simulated for a proposed positioning method. Fig. 6 shows the results for three different angles between the walls. In most simulations scenarios only two sensors are required to carry out a wall-following task. However, in the case of acute angle the system need additional one sensor to keep a fixed distance from the wall.

Fig. 6. Robot positioning parallel to the wall for different angles between the walls.

4. Experiments

The laboratory experiments have been developed using differential drive mobile robot shown in Fig. 7. This vehicle was designed by the author for Sumo Robots Contest and is equipped with eight infrared sensors. For the wall-following task only three of them were used to simulate partially damaged sensor system. The control strategy was the same as the one used in computer simulations.

Fig. 7. Autonomous mobile robot equipped with infrared sensors.
A six state controller has been implemented in order to control the mobile robot for the
wall following task as shown in Fig. 8. The state diagram provides representation of the robot
states and the transition between them in the presence of various events. Each state performs
the defined functions to operate the robot in the specific situation.

Distance control was based on PD controller. The input of the PD controller was the
difference (error) between desired and measured distance from the robot to the wall. The
output of the controller was the positioning action (the proper motors supply voltages).

The experiments showed that proposed orientation method of the robot parallel to the
wall is effective.

![State diagram for the wall-following task.](image)

5. Conclusion

The research carried out using mobile robot simulator shows that it’s an effective tool
for fast control algorithm prototyping, whereas the final experiments can’t be replaced by
simulations.

The presented results from simulation and preliminary experiments on a real robot show
that autonomous mobile robot equipped with a simple sensor system and proposed control
algorithm is able to perform a wall-following task. Furthermore, the positioning method
described in this paper can be used in the multi-sensor applications as a part of hybrid system.
In the case of partial failure of the sensor system the control strategy can be changed from the
multi-sensor to the simpler one. Theoretically, the autonomous mobile robot is able to avoid
obstacles using only one sensor. However, this case has not been investigated.

At the same time practical experiments showed unforeseen problems. Specular
reflection from some polyhedrons, noise, or sensor disturbances initialized by thin objects
such as table base, or cable need to be taken into consideration in a further research.

Fault diagnosis and control systems are still under development [16]. Some of the
difficulties include fault estimation and faulty behaviors prediction in a real world
environment.
References


