International Journal of Internet, Broadcasting and Communication Vol.13 No.3 56-62 (2021)
http://dx.doi.org/10.7236/IJIBC.2021.13.3.56

IJIBC 21-3-8

Performance Comparison of Sensor-Programming Schemes According to the Shapes of Obstacles

Jong-In Chung[1], Yi-Geun Chae[2*]

[1]*Professor, Department of Computer Education, Kongju National University, Korea*
[2]*Associate Professor, Department of Computer Engineering, Kongju National University, Korea*
Email: {jichung, ygchae }@kongju.ac.kr*

Abstract

MSRDS(Microsoft Robotics Developer Studio) provides the ability to simulate these technologies. SPL(Simple Programming Language) of MSRDS provides many functions for sensor programming to control autonomous robots. Sensor programming in SPL can be implemented in two types of schemes: procedure sensor notification and while–loop schemes. We considered the three programming schemes to control the robot movement after studying the advantages and disadvantages of the sensor notification procedure and the while-loop scheme. We also created simulation environments to evaluate the performance of the three considered schemes when applied to four different mazes. The simulation environment consisted of a maze and a robot with the most powerful sensor, i.e., the LRF(Laser Range Finder) sensor. We measured the required travel time and robot actions (number of turns and number of collisions) needed to escape the maze and compared the performance outcomes of the three considered schemes in the four different mazes.

Keywords: *MSRDS, Sensor Programming Scheme, Maze, Simulation, SPL*

1. Introduction

Intelligent robots are smartly providing innovations to our lives as we evolve into a society where humans and robots coexist through the convergence of advanced technologies such as artificial intelligence (AI). Intelligent service robots refer to robots capable of recognizing external environments, judging situations without assistance, and reacting autonomously to movement. They are used in various fields, such as disaster relief, healthcare, the medical field, silver mining, education, defense, construction, marine industries, and agriculture. MarketsandMarkets forecasts that by 2022, the total service robot market, including autonomous driving, will reach $ 239 billion [1].

As such, service robots are facing a major turning point in the craze of the fourth Industrial Revolution, while robot companies in advanced countries such as Japan and Germany are leading the market, and barriers to entry are increasing. For this purpose, the Korean government has decided to develop and commercialize service robots actively in five promising areas, smart homes, medical and rehabilitation services, disaster relief and safety, unmanned transportation, and agriculture, all of which have great growth potential.

Manuscript Received: June. 7, 2021 / Revised: June. 12, 2021 / Accepted: June. 15, 2021
Corresponding Author: ygchae@kongju.ac.kr
Tel: +82-41-521-9233
Associate Professor, Department of Computer Engineering, Kongju National University, Korea

Given that autonomous robots recognize roads and obstacles through radar, cameras, and sensors, many areas of technology are involved in the development of robots [2,3]. If the robot is developed after its hardware is manufactured, the development process will be high due to the necessary trial-and-error process. These robot development costs can be reduced because the development results can be predicted through the concurrent development and testing of the hardware and software via robot simulations that match the conditions of the actual field [4,5].

Microsoft focused on robotic application development tools to resolve these issues, resulting in the creation of MSRDS [6-8]. SPL of MSRDS provides an environment for simulating autonomous robots. A robot with a sensor must recognize an event issued from the sensor continuously when the robot is driving. An event generated from a sensor refers to a situation in which the changing status of the robot is traced periodically. Therefore, a sensor must generate events to sense the status of the robot when the robot is driving.

SPL uses two programming schemes: the procedure sensor notification scheme and the while-loop scheme to detect the values of sensors mounted on the robot [9,10].

The procedure sensor notification scheme is the easiest way to use the "/Procedure_SensorNotify" attribute of the sensor entity, allowing the sensor to acquire sensing data and execute driving procedures at the same time. The "/Procedure_SensorNotify" attribute refers to the ability to activate the procedure specified when an event occurs according to the sensor.

In the procedure sensor notification scheme, the system cannot execute the specified procedure again when the sensor takes measurements very frequently and notifies the system of new sensing data while the specified procedure is being executed. This scheme can incur a synchronization problem between the procedure execution and sensor notification steps. Therefore, the procedure sensor notification scheme can lead to abnormalities when attempting to control the robot.

The while-loop scheme is a structure that continuously reads sensor values and executes driving procedures using while loop control statements. The get() method of the sensor entity is used in the while-loop block to read the sensor value and control the operation according to this value. The while-loop scheme is suitable for making a program that can sense the user's intent. The system can obtain the measured data and execute the robot control routine in a while-loop block. There is no synchronization problem as in the procedure sensor notify scheme. However, the while-loop scheme causes a system overload because the robot control routine is executed the whenever the while-loop block executes.

To determine if there were any performance differences between the two sensor-programming schemes of MSRDS, Chung [11-15] created a simulation environment to evaluate the performance capabilities of both schemes. The simulation environment consisted of a maze and a robot with a sensor. The robot with a sensor travels through the maze to traverse from the start point to the end point. The required traveling time and robot actions (numbers of turns and collisions) were measured as the robot escaped the maze and the performance outcomes of the two sensor-programming schemes were compared for each of three different sensors (LRF, IR, and bumper types). He concluded that there is no performance difference between the while-loop scheme and the procedure sensor notification scheme for the three different sensors.

This study presents three programming schemes for controlling the robot's movement and compares while also analyzing codes written in three programming languages for various mazes. The performance of the suggested programming schemes was compared in terms of the time taken to escape the maze, the number of turns of the robot, and any abnormal behavior that occurred.

2. Maze Generation

SPL's Maze Builder service was used to create the mazes to be used for the simulation in this study. Maze

Explorer is created when the Maze Builder service executes in the SPL editor. Mazes in Maze Explorer can be both drawn and erased. They can be saved as XML files and used via the /FileName attribute in StartSimulationEngine whenever necessary. Here, we generated four mazes, each with their own characteristics, and three programming schemes were applied to each maze to compare the time required to escape the maze and the number of turns and abnormal behaviors by the robots.

Case 1 was designed to create a maze with straight line obstacles to ensure that the robot avoids them. Case 2 was designed a maze with curved obstacles. Maze Builder's graphics consist of thick points; accordingly, when drawing a curve, the curve is created with stairs. Case 3 involved a maze with zigzag obstacles, and Case 4 was designed as a maze with rectangular obstacles with different passage sizes. Figure 1 shows the mazes designed for the simulation in this study.

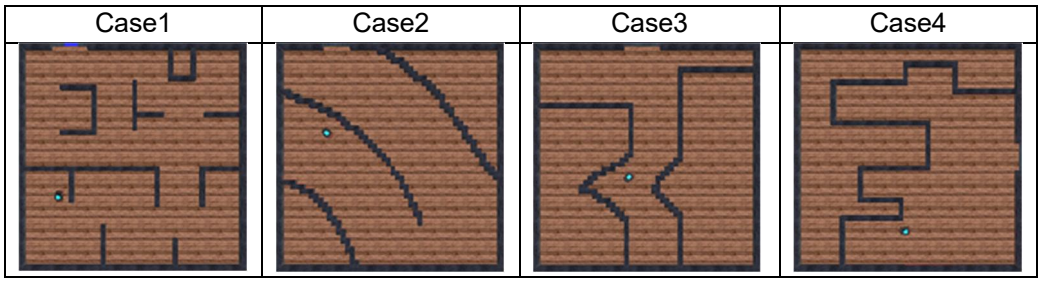

Figure 1. Various mazes designed for simulation

3. Simulation

We suggest efficient sensor-programming schemes that can be used to control the movement of a robot. Four mazes, each with their own characteristics, were generated and three programming schemes were applied to each maze to compare the time required to escape the maze and the number of turns and abnormal behaviors of the robot.

In this study, three programming schemes were used.

(Type 1) Scheme without a flag in Procedure_SensorNotify

(Type 2) Scheme with a flag in Procedure_SensorNotify

(Type 3) Scheme with the while-loop logic

We created the simulation environment as follows.

1. Created four mazes for the robot to trace.
2. Put the robot with an LRF sensor at the starting point.
3. The robot escapes the maze while moving along the maze wall from the starting point to the exit.

The initial conditions of the simulation are for the robot to have power of 0.2 and 45 degrees for each rotation in the left or right direction respectively.

Type 1 executes the predefined procedure specified in the Procedure_SensorNotify attribute when the sensor recognizes the distance. However, unlike Type 2, if a new event of the sensor occurs while a procedure is being executed, the currently executing procedure is aborted and the new procedure specified is executed. The following is the pseudo code for Type 1.

Pseudo code for Type 1
move generate a sensor event periodically call the predefined procedure

```
Procedure
{
   measure distance to 180°, 135°, 90° direction
   if(distance to 180° direction< 0.4m or distance to 135° direction< 0.6m
        or distance to 90° direction< 1m)
       {
        right turn 45° then go straight
       }
   if(distance to 90° direction> 1m)
     {
        left turn 45° then go straight
     }
}
```

Type 2 executes the predefined procedure specified in the Procedure_SensorNotify attribute when an event occurs in which the sensor recognizes the distance. It uses a flag to prevent any application to new events of the sensor while the procedure is running. The pseudo code for Type 2 is shown below.

Pseudo code for Type 2

```
move
generate a sensor event periodically
call the predefined procedure
flag=0

Procedure
{
   measure distance to 180°, 135°, 90° direction
   if (flag==0) {
      flag=1
      if(distance to 180° direction< 0.4m or
         distance to 135° direction< 0.6m or distance to 90° direction< 1m)
         {
          right turn 45° then go straight
         }
      if(distance to 90° direction> 1m)
         {
          left turn 45° then go straight
         }
      flag=0
   }
}
```

Type 3 does not use the Procedure_SensorNotify attribute, and it executes the while-loop block. Therefore, a new while-loop block can be executed upon the completion of the execution of the previous while-loop block. The following is the pseudo code for Type 3.

Pseudo code for Type3

```
move
while(true)
{
   generate a sensing event
   measure distance to 180°, 135°, 90° direction
   if(distance to 180° direction< 0.4m or distance to 135° direction< 0.6m
      or distance to 90° direction< 1m)
      {
       right turn 45° then go straight
      }
   if(distance to 90° direction> 1m)
     {
       left turn 45° then go straight
     }
}
```

4. Performance Comparison

In order to evaluate the performance of Types 1, 2 and 3 for four different mazes, this study measured the total travel times, the number of turns of the robot, the number of abnormalities such as a collision with the wall, and the number of times the robot failed to escape the maze. Simulation data was recorded five times for the performance evaluation. Tables 1, 2, 3 and 4 show the performance comparison results for Types 1, 2, 3 for the four different mazes.

Table 1. Comparison of performance of Type 1, 2, 3 for Case 1

Type \ Factor	Total Travel Time[sec]	# of Turns	# of Abnormality
Type 1	79.60	38.80	2 collisions
Type 2	87.80	32.40	0
Type 3	85.20	32.40	0

Table 2. Comparison of performance of Type 1, 2, 3 for Case 2

Type \ Factor	Total Travel Time[sec]	# of Turns	# of Abnormality
Type 1	∞	∞	2 collisions, 1 stop
Type 2	∞	∞	5 collisions, 1stop
Type 3	121.80	37.00	1 collision

Table 3. Comparison of performance of Type 1, 2, 3 for Case 3

Type \ Factor	Total Travel Time[sec]	# of Turns	# of Abnormality
Type 1	80.20	44.40	1 collision
Type 2	∞	∞	3 collisions, 2 stops
Type 3	67.80	16.00	0

Table 4. Comparison of performance of Type 1, 2, 3 for Case 4

Type \ Factor	Total Travel Time[sec]	# of Turns	# of Abnormality
Type 1	115.40	63.40	1 collision
Type 2	∞	∞	5 stops
Type 3	114.40	40.20	0

In Type 1, the robot stopped only in Case 2, but for the remaining cases, there were no severe abnormalities except for a few collisions. Type 2 crashed and stopped in all cases except Case 1. Type 3 had only one collision for Case 2 but no collisions for all other cases, and it started stably from the starting point to the destination without stopping.

In Type 1, while the procedure is being executed by the previous event, a new event is detected, the procedure currently executing stops, and a new procedure is then performed, causing the number of turns of the robot to increase. Moreover, because the previous procedure was halted and the new procedure was subsequently executed, consistent operation logic for escaping the maze cannot be executed. Therefore, many collisions with the wall occurred.

Type 2 uses a flag to make up for the shortcomings of Type 1, allowing the procedure to complete even if a new event comes in while the procedure is executing. Therefore, we thought that Type 2 would outperform Type 1 such that no collisions or anomalies would occur. However, unexpectedly, Type 1 outperformed Type 2.

Type 3 reads the sensor value in the loop and executes the while-loop block that controls the robot's movement based on this value. Therefore, there are no abnormalities such as wall collisions or unintended stoppages.

As noted above, Type 3 is most effective for robot sensor programming in the SPL of MSRDS. That is, the while-loop scheme is more efficient than the procedure sensor notification scheme. This study concluded that the while-loop scheme has better performance than that of the procedure sensor notification scheme and that the while-loop scheme is suitable for making a program capable of sensing the user’s intent.

5. Conclusions

MSRDS is a very suitable tool for creating robot simulations. The user can control the motion of a robot using various sensors in SPL and VPL. In particular, the procedure sensor notification scheme in SPL is a popular programming scheme because it is easy to program and has good readability of programming. To find the most efficient programming scheme, we suggested the three programming schemes for proper control of the movement of a robot. We compared and analyzed the programming types for four different mazes in each case.

Type 2 uses a flag to make up for the shortcomings of Type 1, allowing the procedure to complete even if a new event arrives while the procedure is executing. Therefore, we considered that Type 2 could outperform Type 1 such that no collisions or anomalies would occur. However, unexpectedly, Type 1 outperformed Type 2.

Type 3 had only one collision for Case 2, but no collisions for any other cases, and it started stably from the starting point to the destination without stopping. Among the three types, Type 3 performed very well.

In general, the procedure sensor notification scheme and the while-loop scheme are feasible for robot control. It can be seen from the results here that the while-loop scheme is superior in terms of performance.

In the future, it will be necessary to study how to improve the while-loop scheme for proper synchronization between the predefined procedure execution and the sensor notification. Moreover, in order to improve the performance of Type 3, it will be necessary to optimize the driving speed, direction angle of rotation, and the wait time.

References

[1] *http://www.aitimes.kr.*

[2] J. S. Park, "Discrete-Time Sliding Mode Control for Robot Manipulators," *Journal of the Korea Industrial Information System Society*, Vol. 16, No. 4, pp. 45-52, Dec. 2011.
DOI: https://doi.org/10.9723/jksiis.2011.16.4.045

[3] M. Y. Kim, S. T. Ahn, and H. S. Cho, "Bayesian Sensor Fusion of Monocular Vision and Laser Structured Light Sensor for Robust Localization of a Mobile Robot," *Journal of Institute of Control Robotics and Systems*, Vol. 16, No. 4, pp. 381-390, April 2010.
DOI: https://doi.org/10.5302/J.ICROS.2010.16.4.381

[4] S, H. Cho, "The Effect of Robotics in Education based on STEAM," *Journal of Korea Robotics Society*, Vol. 8, No.1, pp. 58-65, Feb. 2013.
DOI: https://doi.org/10.7746/jkros.2013.8.1.058

[5] D. G. Shin, S. H. Lee, S. Y. Yi, and B. W. Choi, "Network Control for Virtual Robot in MSRS Simulation Environment," *Journal of Korea Robotics Society*, Vol. 2, No. 3, pp. 242-248, Sep. 2007.

[6] *http://www.helloapps.com.*

[7] *http://www.microsoft.com/robotics/.*

[8] Y. J. Kim, "MSRDS Simulation Environment and External Interface," *Robot and Human, Korea Robotics Society*, Vol. 7, No. 2, pp. 16-22, May 2010.

[9] S. Y. Hong, "Learning Method using RDS for Creative Problem Solving," *Journal of KIISE*, Vol. 16, No. 11, pp. 1126-1130, Nov. 2010.

[10] S. Y. Hong, *Intelligent robot programming for SMART creative engineering*, Bookshollic Publishing, 2012.

[11] J. W. Lee and J. I. Chung, "Comparative Analysis of the Performance of Robot Sensors in the MSRDS Platform," *Journal of the Korea Industrial Information Systems Research,* Vol. 19, No.5, pp. 57-67, Oct. 2014.
DOI: https://doi.org/10.9723/jksiis.2014.19.5.057

[12] J. I. Chung, "Comparison of Sensor Programming Schemes in MSRDS," in *Proc. ICIECT* June, 2016.

[13] J. I. Chung, "Performance Evaluation of Sensor Programming Patterns in MSRDS," *International Journal of Engineering and Technology*, Vol. 7, No.2.33, pp. 1132-1137, July 2018.
DOI: http://dx.doi.org/10.14419/ijet.v7i2.33.17925

[14] J. I. Chung, "Comparison of Programming Schemes for Robot Control in SPL of MSRDS," in *Proc. IICCC*, July 2019.

[15] J. I. Chung, "Efficient Sensor Programming Patterns in SPL of MSRDS," *TEST Engineering & Management,* Vol. 81, pp. 2226-2233, Dec. 2019.
DOI: https://doi.org/10.1080/17415993