November-December 2019
ISSN: 0193-4120 Page No. 2226 - 2233

Efficient Sensor Programming Patterns in SPL of MSRDS

Jong-In Chung
Department of Computer Education,
Kongju National University, Korea,
jichung@kongju.ac.kr

Article Info
Volume 81
Page Number: 2226 - 2233
Publication Issue:
November-December 2019

Abstract
SPL of MSRDS provides many functions for sensor programming. The sensor programming can be implemented into two types of patterns: procedure and while–loop. The easiest way to check the sensor's measured value is to use the procedure sensor notify pattern in SPL. But the pattern can have a synchronization problem between procedure execution and sensor notification. Therefore, the procedure sensor notify pattern can lead to an abnormality of robot control. However, the while-loop pattern is known to cause system overload because the robot control routine is executed whenever the while-loop block executes. This study suggests the efficient programming scheme to control the robot movement. This study consider three efficient programming schemes to control the robot movement by studying the advantages and disadvantages of the procedure and while loop scheme and also makes a simulation environment to evaluate the performance for the three considered schemes. The simulation environment consists of a maze and a robot with one of three potential sensors. This study measures the required travel time and robot actions (number of turns and number of collisions) needed to escape the maze and compares the performance for the three considered schemes

Keywords*: MSRDS, Programming Pattern, Sensor Programming, Simulation,SPL*

Article History
Article Received: *5 March 2019*
Revised: *18 May 2019*
Accepted: *24 September 2019*
Publication: *12 December 2019*

1. INTRODUCTION

Recently, primary schools, secondary schools, and universities are attempting to create interest in students and raise class participation by using robots in their existing classes [1-3]. There are especially many cases for robot simulations as alternatives to the constraints of purchasing physical robots that will be tested and developed in working fields. The simulation robot evokes a student's curiosity for new fields and provides a new environment that they have never experienced. It helps students understand their expectations and achievements and provides a positive leaning effect [4, 5].

If the robot is developed after its hardware manufacture, the robot development will have a high cost because of the trial and error. With real-world robot simulations, robot development cost can be saved because you can predict the development result through the concurrent development and test of the hardware and software [6, 7].

MSRDS(Microsoft Robotics Developer Studio) of Microsoft[8], ERSP of Evolution Robotics, ROS of Willow Garage, OROCOS of Europe, and OpenRTM-aist of AIST in Japan are typical global robotics platforms. MSRDS provides a development and simulation environment that

can predict the hardware factors such as the manipulator and the software factors such as kinematics. It also provides the development framework that can easily combine the diverse factors for developing different intelligent services and human-friendly technologies because it provides common message protocols and interworking technologies based on the modularized services. Therefore, the MSRDS platform provides a simulation robot environment that can simulate robot programming without a hardware robot. A special simulation robot can be made, such as space a shuttle and submarine.

2. SPL

MSRDS provides the VPL(Visual Programming Language) which allows developers to create applications simply by dragging and dropping components onto a canvas and wiring them together. You can also use the SPL (Simple Programming Language) for easy, fun and simplified programming for creative IT, robotics, embedded and mobile programming. SPL helps people begin programming in an easy and fun environment by simplifying complicated programming patterns into simple scripts. A novice user with little or no programming experience can start creative IT, embedded, mobile and robotics programming right away without any preliminary preparation.

Sensors are attached on the robot to effectively control it in MSRDS. MSRDS provides several sensors for robotics programming and you can use the attached sensors on many robot platforms. Programmers can use the Kinect, Bumper(Touch), LRF, IR, Sonar, bright, color, compass, GPS and RFID sensor on VPL and SPL to control the robot. Each sensor is used for its different areas but the LRF sensor has the highest performance and the bumper sensor has the lowest performance on the maze explorer[9].

SPL has better procedure statements than any other computer programming language. A procedure can be enabled to make several small program units. Thus, it is easy to control and manage a large program if you use the procedure in the computer program.

You can add the differential drive entity to use the robot with motors in SPL. The differential drive robot is controlled by the power of the left and right motor.

To make the sensor program on a robot with two wheels in SPL, Differential Drive entity and sensor entities are needed. For example, the robot with an LRF sensor can be written as follows:

```
AddDifferentialDriveEntity robot1
  /Position:0 0 4
  AddLaserRangeFinderEntity lrf1
  /Position:0 0.4 0
  /ParentEntity: robot1
  /Procedure_SensorNotify:proc1
robot1. GoTo(5, 0.5)
robot1.Turn(30, 0.2)
```

The example given in the script above shows that a robot,“robot1”, with an LRF sensor and goes 5m distance with the power of 0.5 and turns 30 degrees with the power of 0.2.

3. COMPARISON OF SENSOR PROGRAMMING PATTERNS

The robot must recognize the event issued from the sensor continuously whilethe robot is driving. The event generated from a sensor is the situation that traces the changing status of a robot periodically. Therefore, a sensor must generate events to sense the status of a robot while the robot is driving.

To get the sensing values generated from the sensors, SPL uses two kinds of the sensor programming patterns: the procedure sensor notify pattern and the while loop pattern. The

easiest way to check the sensor's measured value is to use the procedure sensor notify pattern in SPL. The procedure sensor notify pattern can implement the robot sensing routine using the sensor's measured values. To implement the robot sensing routine in the procedure sensor notify pattern, it is necessary to use the "/Procedure_SensorNotify" option in the sensor entities of the SPL editor. The "/Procedure_SensorNotify" option means that the system jumps to and executes the specified procedure whenever the sensor measures any value. Figure 1 shows the process of procedure sensor notify pattern.

In the procedure sensor notify pattern, the system cannot execute the specified procedure again when the sensor makes very frequent measurements and notifies the system of new sensing data while the specified procedure is being executed. This pattern can cause a synchronization problem between procedure execution and sensor notification. Therefore, the procedure sensor notify pattern can lead to an abnormality of robot control.

The while-loop pattern is good for making a program that can sense the user's intent. The system can obtain measured data and execute the robot control routine in a while-loop block. There is no synchronization problem in the procedure sensor notify pattern. However, the while-loop pattern causes system overload because the robot control routine is executed whenever the while-loop block executes.

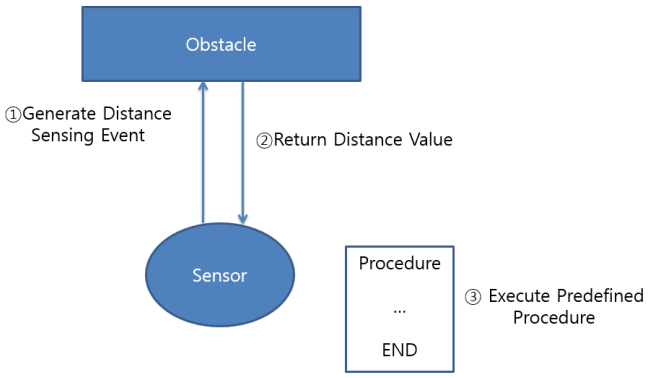

Figure 1: Process of procedure sensor notify pattern

To see if there is any performance difference between the two sensor programming patterns of MSRDS, Chung [10,11] made a simulation environment to evaluate performance. The simulation environment consisted of the maze and the robot with any sensor. The robot with any sensor travels the maze to escape from the start point to the end point. He measured the required traveling time and the robot's actions(turning number and collision number) for escaping the maze and compared the performance of the two sensor programming patterns for three different sensors(LRF, IR, bumper). He concluded that there is no performance difference between the while-loop pattern and the procedure sensor notify pattern for each sensor.

4. SIMULATION

This study suggests efficient sensor programming schemes to control the robot movement and also compares and analyzes the following three possible programming cases:

(Case 1) Scheme using the flag in Procedure_SensorNotify

(Case 2)Scheme using the non-flag in Procedure_SensorNotify

(Case 3) Scheme using the non-flag in NonProcedure_SensorNotify

This study makes the simulation environment as follows:

1. make a maze for a robot to trace.
2. put the robot with an LRF sensor at the starting point.
3. The robot escapes the maze by moving along the maze wall from the starting point to the exit.

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

The procedure sensor notify pattern needs an added sensor entity and the predefined procedure being executed when an event is generated. The sensor_entity named,"sensor-name", must be defined with the "/Procedure_SensorNotify" attribute previously.

Case 1 executes the predefined procedure specified in the Procedure_SensorNotify attribute when an event occurs in which the sensor recognizes the distance. It uses the flag to prevent it from being applied to new events of the sensor while the procedure is running. The Following is the code for case 1:

```
## Code for case1
AddDifferentialDriveEntity base1
   /Position:-1.3 0.2 3.7
   /Orientation: 0 0 0

AddLaserRangeFinderEntity lrf1
   /Position:0 0.2 0
   /ParentEntity:base1
   /Procedure_SensorNotify:LRFEvent

FlushScript
WaitForServiceCreation lrf1
Wait 1000
base1.SetDrivePower(0.2 ,0.2)
```

```
int busy=0
Procedure LRFEvent
   d180 = value.DistanceMeasurements[355]
   d135 = value.DistanceMeasurements[270]
   d90 = value.DistanceMeasurements[180]

   if(busy==0) {
    busy=1
    if(d180 < 400 || d135 < 600 || d90 < 1000 ) {
     base1.Turn(-45, 0.2)
     base1.Go()
     }

    if( d180 > 1000) {
     base1.Turn(45, 0.2)
     base1.DriveDistance(0.2, 0.2)
     base1.Go()
     }
    busy=0
}
End
```

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

```
## Code for case2
AddDifferentialDriveEntity base1
   /Position:-1.3 0.2 3.7
   /Orientation: 0 0 0

AddLaserRangeFinderEntity lrf1
   /Position:0 0.2 0
   /ParentEntity:base1
   /Procedure_SensorNotify:LRFEvent

FlushScript
WaitForServiceCreation lrf1
Wait 1000
base1.SetDrivePower(0.2 ,0.2)

Procedure LRFEvent
   d180 = value.DistanceMeasurements[355]
   d135 = value.DistanceMeasurements[270]
   d90 = value.DistanceMeasurements[180]

   if ( d180 < 400 || d135 < 600 ||d90 < 1000 )
   {
      base1.Turn(-45, 0.2)
      base1.Go()
   }
   else
   {
      if (d180 > 1000)
      {
         base1.Turn(45, 0.2)
         base1.GoTo(0.2, 0.2)
         base1.Go()
      }
   }
End
```

Case 3 does not use the Procedure_SensorNotify attribute and calls the procedure in the loop. Therefore, the next procedure can be called after the execution of the procedure is completed. The following is the code for case 3:

```
## Code for case3
AddDifferentialDriveEntity base1
   /Position:-1.3 0.2 3.7
   /Orientation: 0 0 0

AddLaserRangeFinderEntity lrf1
   /Position:0 0.2 0
   /ParentEntity:base1

FlushScript
WaitForServiceCreation lrf1
Wait 1000
base1.SetDrivePower(0.2 ,0.2)

for (i = 0; i < 100000000; i++)
{
Call proc1
}

Procedure proc1
{ distance=lrf1.Get()
```

```
  d90 = distance[180]
  d135 = distance[270]
  d180 = distance[355]
    if (d180 < 400 || d135 < 600 || d90 < 1000)
    {
      base1.Turn(-45, 0.2)
      base1.Go()
    }
    else
    {
      if (d180 > 1000)
      {
        base1.Turn(45, 0.2)
        base1.DriveDistance(0.2, 0.2)
        base1.Go()
      }
    }
  }
End
```

5. COMPARISON OF PERFORMANCE

In order to evaluate the performance of cases 1, 2 and 3, we measured the total travel time, number of turns of the robot, number of abnormalities such as collisions with the wall, and failure of the robot to escape the maze. Simulation data was measured five times for the objectivity of the performance evaluation. Table 1, 2, and 3 show the simulation data for cases 1, 2 and 3, respectively. Table 4 shows the performance comparison of cases 1, 2, and 3 based on Table 1, 2, and 3.

Table 1: Simulation data for case 1

	1st	2nd	3rd	4th	5th	Average
Total Travel Time	74	74	74	79	74	75
# of Turns	24	23	24	28	24	24.6
# of Abnormalities	0	0	0	0	0	0

Table 2:Simulation data for case 2

	1st	2nd	3rd	4th	5th	Average
Total Travel Time	73	76	∞	86	75	∞
# of Turns	39	39	∞	64	43	∞
# of Abnormalities	0	1	stop	3	1	1 stop, 5 collisions

Table 3:Simulation data for case 3

	1st	2nd	3rd	4th	5th	Average
Total Travel Time	72	76	76	74	75	74.6
# of Turns	22	27	26	26	26	25.4
# of Abnormalities	0	0	0	0	0	0

Table 4: Comparison of performance of Case 1, 2, 3

	Total Travel Time[sec]	# of Turns	# of Abnormalities
Case 1	75	24.6	0
Case 2	∞	∞	1 stop, 5 collisions
Case 3	74.6	25.4	0

The travel time of case 1 and 3 are almost similar, and the number of turns of case 1 and 3 are fewer than case 2. It also do not have any collision with the wall while moving in maze. Case 2 had 5 collisions with the wall, and in some cases it is stopped.

In case 2, while the procedure is being executed by the previous event, a new event is detected, and the executing procedure is stopped. Then a new procedure is performed, so the number of turns of the robot increases. Since the previous procedure is stopped and the new procedure is executed, the consistent operation logic for escaping the maze cannot be executed. Therefore, many collisions with the wall occur.

Case 3 reads the sensor value in the loop, and calls the procedure that controls the robot's movement based on the value. Consequently, there are no abnormalities such as collisions with the wall and unintended stoppage.

Therefore, case 1 and case 3 are more effective for robot sensor programming in SPL of MSRDS. That is, it is efficient to use the flag in the procedure sensor notify pattern or to call the procedure without using the flag in the loop statement.

6. CONCLUSION

MSRDS is a very suitable tool for creating robot simulations. You can control the motion of the robot using various sensors in SPL and VPL. In particular, the procedure sensor notify pattern in SPL is a popular programming pattern because it is easy to program and has good readability of programming. To find the efficient programming scheme, this study comparedthree cases, and suggested the efficient programming schemes to control the robot's movement [13 – 14].

The scheme using the flag in Procedure_SensorNotify and the scheme using the non-flag in NonProcedure_SensorNotify have better performance than the scheme using the non-flag in Procedure_SensorNotify in travel time, number of turns and number of abnormalities. In the scheme using the non-flag in Procedure_SensorNotify, while the procedure is being executed by the previous event, a new event is detected, and the executing procedure is stopped. Then a new procedure is performed, so the number of turns of the robot increases [15].

In the sensor programming pattern for controlling the robot, using the flag in the procedure sensor notify pattern or calling the procedure without using the flag in the loop statement was found to be more efficient.

Future research topics need to be studied to compare the performance evaluation of how the procedural programming method proposed in this study is influenced by various types of mazes.

REFERENCES

1. J. I. Chung and Y. J. Kim.***Robot programming to improve logic***,Korea: Hongreung Science Publishing, 2012.
2. ***http://www.helloapps.com***
3. S. Y. Hong.**Intelligent robot programming for SMART creative engineering**,Korea: Bookshollic Publishing, 2012.
4. Y. J. Kim.**MSRDS Simulation Environment and External Interface**, *Robor and Human, Korea Robotics Society*, vol. 7, no. 2, pp. 16-22, 2010.
5. S, H. Cho.**The Effect of Robotics in Education based on STEAM**, *Journal of Korea Robotics Society*, vol. 8, no.1, pp. 58-65, 2013.
6. J. S. Park.**Discrete-Time Sliding Mode Control for Robot Manipulators**, *Journal of The Korea Industrial Information System Society*, vol 16, no. 4, 2011.
7. S. P. Kim.**Kinematic and dynamic analysis of a spherical three degree of freedom joint rehabilitation exercise equipment**, *Journal of The Korea Industrial Information System Society*, vol. 14, no. 4, pp.16-29, 2009.
8. ***http://www.microsoft.com/robotics/***
9. Hilkevics, S.; Semakina, V. 2019. The classification and comparison of business ratios analysis methods, Insights into Regional Development 1(1): 48-57. https://doi.org/10.9770/ird.2019.1.1(4)

10. Umana, S. I., Akpbio, N. O., & Mbong, S. E. (2018). **Extended Stanford University Interim Model Loss Stanford University Interim Propagation Loss Model for a Gmelina Arborea Tree-Lined Road.** Review of Computer Engineering Research, 5(2), 57-63.
11. Pechancová, V., Hrbáčková, L., Dvorský, J., Chromjaková, F., Stojanovic, A. (2019). Environmental management systems: an effective tool of corporate sustainability. Entrepreneurship and Sustainability Issues, 7(2), 825-841. http://doi.org/10.9770/jesi.2019.7.2(3)
12. 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.
13. Jabarullah, N. H., Jermsittiparsert, K., Melnikov, P. A., Maseleno, A., Hosseinian, A., & Vessally, E. (2019). Methods for the direct synthesis of thioesters from aldehydes: a focus review. Journal of Sulfur Chemistry, https://doi.org/10.1080/17415993.2019.1658764
14. J. I. Chung.**Comparison of Sensor Programming Schemes in MSRDS** in *ICIECT*, June 2016.
15. J. I. Chung.**Performance Evaluation of Sensor Programming Patters in MSRDS**,*International Journal of Engineering and Technology*, vol. 7, no.2.33, pp. 1132-1137, July 2018.