International Journal of Engineering & Technology, 7 (2.33) (2018) 1132-1137

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Performance Evaluation of Sensor Programming Patterns in SPL of MSRDS

Jong in Chung*

Faculty of Department of Computer Education, Kongju National University, Kongju, Korea
**Corresponding author E-mail: jichung@kongju.ac.kr*

Abstract

Background/Objectives: SPL of MSRDS provides many functions for sensor programming. The sensor programming can be implemented in two types of patterns: procedure and while–loop patterns. It is known that the while-loop pattern has better performance than the procedure pattern.

Methods/Statistical analysis: To determine if the while-loop pattern has better performance than that of the procedure pattern, we made a simulation environment to evaluate the performance. The simulation environment consisted of a maze and a robot with one of three potential sensors. 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 for the two sensor programming patterns.

Findings: We were able to obtain results that showed that the while-loop pattern does not have better performance than the procedure pattern. However, the while-loop pattern is good for making a program that can sense into user's intent. It is known that programmers prefer the while-loop pattern to the procedure pattern.

Improvements/Applications:It is true that for a long time we have adapted the while-loop pattern to simulate sensor programming, without consideration of other methods. Therefore, from now on, it will be convenient to select an easy pattern to use according to the simulation situation because it is now known there is no difference of performance between the two types of sensor programming patterns.

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

1. Introduction

MSRDS(Microsoft Robotics Developer Studio) enables hobbyists and professional or non-professional developers to create robotics applications that target a wide range of scenarios. MSRDS can support a broad set of robotics platforms by either running directly on the platform if that platform has an embedded PC running Windows or by controlling a robot from a Windows PC through a communication channel such as Wi-Fi or Bluetooth. MSRDS provides simulation robots and environments that enable the basic robot programming without hardware robots. MSRDS provides a Visual Programming Language (VPL) that allows developers to create applications simply by dragging and dropping components onto a canvas and wiring them together. Powered by the NVIDIA PhysX engine, the powerful Visual Simulation Environment (VSE) provides a high-fidelity simulation environment for running game-quality 3D simulations with real-world physical interactions [1,2].

MSRDS includes the following components:

• CCR - Concurrency and Coordination Runtime

• DSS - Decentralized Software Services

• VPL - Visual Programming Language

• VSE - Visual Simulation Environment

• Samples, Tutorials and Documentation

It is possible to use SPL (Simple Script Language) for easy, fun and simplified programming for creative IT, robotics, and embedded and mobile systems. SPL helps users to start programming in an easy and fun way by simplifying complicated coding pattern into a simple script. A novice user with little to no coding experience can start creative IT, robotics, and embedded and mobile system programming right away without any preliminary preparation [3-6].

2. Sensor Programming Patterns

2.1 Sensors in MSRDS

MSRDS provides several sensors for robotics simulation programming; it is possible to use the sensors attached to many robot platforms. Programmers can use Kinect, Bumper (Touch), LRF, IR, Sonar, bright, color, compass, GPS, and RFID sensors on VPL and SPL to control robots. Each sensor is used in different areas: the LRF sensor has the highest performance; the bumper sensor has the lowest performance on the maze explorer [7-9]. SPL, especially, provides several functionalities for easy sensor programming of robots. SPL script editor for MDRDS provides many simulated sensor entities as shown in Figure 1.

Figure 1:. Simulated Sensor Entities in SPL Editor.

It is possible to add a Differential Drive entity in order to use a robot with a motor in SPL. Figure 2 shows the AddDifferentialDriveEntity in SPL script editor.

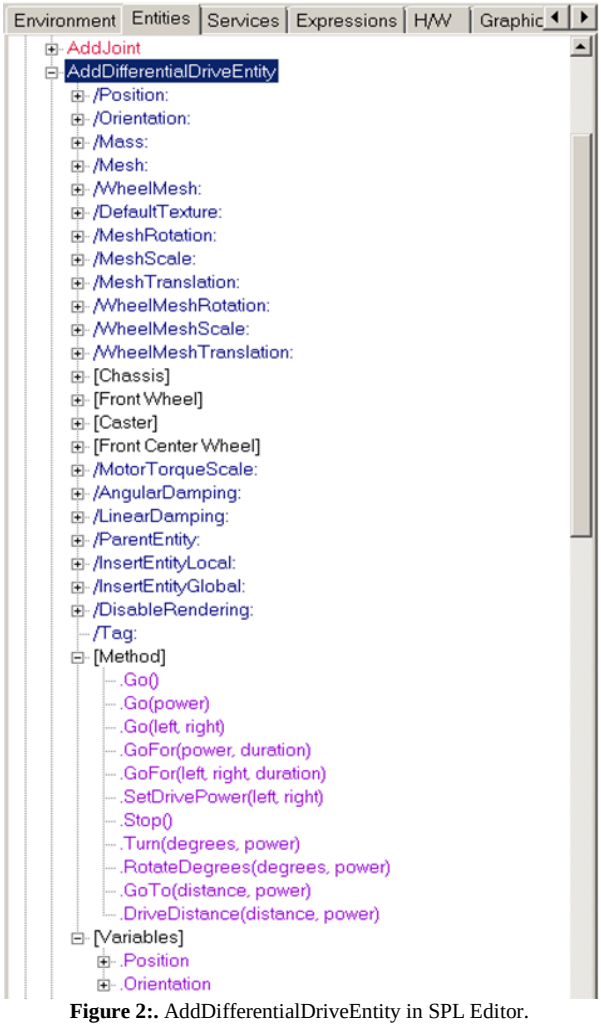

Figure 2:. AddDifferentialDriveEntity in SPL Editor.

To make a sensor program work on a robot with a sensor and motor in SPL, a Differential Drive entity and a sensor entity are needed. For example, the program for a robot with a bumper sensor can be written as follows.

AddDifferentialDriveEntity robot1
/Position: 0 0 4
AddBumperEntity bumper1
/ParentEntity: robot1
/FrontPosition: 0 0.05 -0.3
/RearPosition: 0 0.05 0.3

2.2 Operation of Robot

To control the action (go, power setup, rotation, driving distance setup) of a robot in SPL, the methods and attributes of AddDifferentialDriveEntity are used. As can be seen in Figure 2, it is possible to use the methods of AddDifferentialDriveEntity to control the operation of the robot body.

For example, if a robot named “robot1” is turned 30 degrees after setting its power to 0.5 and moving 5m, it is possible to make scripts as follows.

AddDifferentialDriveEntity robot1
/Position: 0 0 4
robot1. GoTo(5, 0.5)
robot1.Turn(30, 0.5)

By obtaining information from the sensors, the robot can run autonomously ~~automatically~~. Robot can be controlled by information obtained from the methods and values in Procedure_SensorNotify of the simulated sensor entity. For example, if robot “robot1” with bumper sensor “bumper1” is in conflict with an obstacle it must go back 0.5m and turn 180 degrees, and then go 3m and stop. It is possible to implement this situation using the following script.

If (bumper1.Pressed)

robot1.DriveDistance(0.5, -0.3)
robot1.RotateDegrees(180, 0.3)
robot1.DriveDistance(3.0, 0.3)
robot1.SetDrivePower(0.0, 0.0)

2.3 Procedure Pattern

The robot has to trace a particular situation continuously using the sensor’s measured values. The easiest way to check the sensor’s measured value is to use the procedure pattern in SPL. The procedure pattern can implement the robot sensing program using the sensor’s measured values. To implement the robot sensing program in the procedure 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.
The following is the structure of the procedure pattern.

sensor_entitysensor_name
/Procedure_SensorNotify: procedure_name
Procedure procedure_name
End

Let us consider a situation in which a robot has to avoid an obstacle. Figure3 shows the situation in which the robot goes back, turns 90 degrees to the left, goes straight, and turns 90 degrees to the right when it finds itself in conflict with an obstacle.

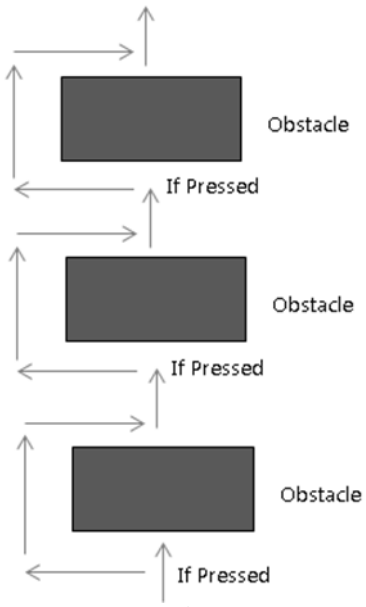

Figure 3:. Traveling path of a robot with bumper sensor.

The following is the program written for the procedure pattern shown in Figure3.

```
AddDifferentialDriveEntity base1
/Position: 0 0 0
AddBumperEntity bumper1
/Procedure_SensorNotify:BumperEvent
FlushScript
WaitForServiceCeation bumper1
base1.SetDrivePower(0.2, 0.2)
//procedure for bumper1 notifying
Procedure BumperEvent
If (value.Pressed)
{
base1.DriveDistance(0.5, -0.2)
base1.RotateDegrees(90, 0.2)
base1.DriveDistance(1.0, 0.2)
base1.RotateDegrees(-90, 0.2)
base1.DriveDistance(2.0, 0.2)
base1.RotateDegrees(-90, 0.2)
base1.DriveDistance(1.0, 0.2)
base1.RotateDegrees(90, 0.2)
base1.SetDrivePower(0.2, 0.2)
}
End
```

The procedure BumperEvent is executed whenever the sensor "bumper1" senses any value. If the value of "value.Pressed" is true then the following block is executed; otherwise, the block is bypassed. The execution frequency of the specified procedure depends on how often the sensor makes its measurements. The procedure pattern can have some trouble in case in which there is a complicated routine for the robot control. The system cannot execute the next specified procedure again when the sensor makes its measurements very frequently and notifies the system of new sensing data while the specified procedure is being executed. Therefore, this pattern can lead to a synchronization problem between the procedure execution and the sensor notification.

2.4 While-loop Pattern

The while-loop pattern is another pattern for robot control. The pattern, which has a while-loop block, obtains sensing data from the sensor and executes the routine for robot control in the while-loop block in SPL. This pattern controls the robot according to the sensed data using the Get() attribute function of the sensor entity in the while-loop block.

The following is the structure of the while-loop pattern.

```
sensor_entitysensor_name
…
while (true)
{
sensor_name.Get()
if (…)
{
}
```

The following is the program written in the while-loop pattern for the situation shown in Figure3.

```
AddDifferentialDriveEntity base1
/Position: 0 0 0
AddBumperEntity bumper1
FlushScript
WaitForServiceCeation bumper1
base1.Go(0.2, 0.2)
while (true)
{
bumper1_pressed=bumper1.Get()
if(bumper1_pressed==true)
{base1.DriveDistance(0.5, -0.2)
base1.RotateDegrees(90, 0.2)
base1.DriveDistance(1.0, 0.2)
base1.RotateDegrees(-90, 0.2)
base1.DriveDistance(2.0, 0.2)
base1.RotateDegrees(-90, 0.2)
base1.DriveDistance(1.0, 0.2)
base1.RotateDegrees(90, 0.2)
base1.SetDrivePower(0.2, 0.2)
}
}
```

The drawback of this pattern is that it overloads the system because the specified procedure is executed whenever the while-loop block executes.

3. Comparison of Two Types of Patterns

There are two types of patterns for robot sensing programming: procedure and while-loop patterns. It is known that the while-loop pattern has better performance than the procedure pattern. The while-loop pattern is good 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 there is when using the procedure pattern. However, the while-loop pattern is known to cause system overload because the robot control routine is executed the whenever the while-loop block executes[10].

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

It is known that programmers prefer the while-loop pattern to the procedure pattern. We can enumerate the following characteristics for the two types of patterns.

Table 1. Comparison of Sensor Programming Patterns

Type	Sensing method	Abnormality	Detailed control	Preference	Overload
Procedure	/Procedure_SensorNotify	much	No	less	less
While-loop	Get()	less	Yes	much	much

4. Performance Evaluation

To determine if there is any difference of performance between the two sensor programming patterns of SPL, we implemented a simulation environment to evaluate the performance. The simulation environment consisted of a maze and arobot with one of three potential sensors. The robot with this sensor travels in the maze, attempting to escape the maze by traveling from the start point to the end point. We implemented six simulation environments so as to consider a total of three sensors (LRF, Bumper, IR sensor) and two sensor programming patterns.

We measured the travel time required to move from the starting point to the end point and robot actions (number of turns and number of collisions) needed to escape from the maze and compared the performances of the two sensor programming patterns.

4.1 Simulation Design

We created the maze using Maze Explorer and Maze Builder services in SPL of MSRDS, as shown in Figure 4. A robot with one of the sensors is put at the starting point of the maze.

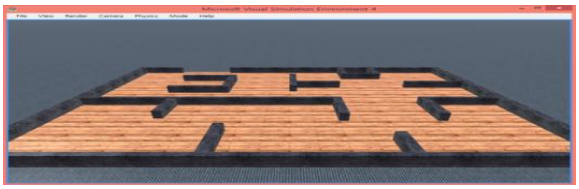

Figure 4.: Maze generated by Maze Builder.

4.2 Simulation Conditions

We performed this simulation on a personal computer with 2GBof main memory. The initial simulation values are as follows.

(1) Robot forward speed: 0.3

Robot has to remain at only an adequate speed because a higher speed could cause a conflict with a wall of the maze. Robot speed can have values in the range of -1 ~ +1. When a robot with an LRF sensor comes into conflicts with a wall, at that point it does not recognize any other obstacles. Therefore, we set an adequate forward speed.

(2) Robot backward speed: -0.2

The backward speed of the robot has to be set lower than the forward speed because it is easier for the robot to come into conflict with obstacles during backward movement than it is during forward movement.

(3) Robot turning speed: 0.2

If the robot turning speed is high, then the robot can go in wrong directions little by little whenever it turns. Therefore, we set the turning speed so as to reduce the direction error and to limit the travel time as much as possible.

When the robot with a bumper sensor is simulated on the procedure pattern, the robot can come into conflict with obstacles while the robot is turning. As the result, the robot can acquire a new, wrong direction and then move straight. Therefore, we added a robot control routine such that the sensor does not operate while the robot is turning.

4.3 Robot Traveling Algorithm

4.3.1 Bumper Sensor

The robot with the bumper sensor travels according to the following algorithm.

```
//principle: robot goes along the left wall of maze.

move forward 0.3m
if(robot conflicts with wall)
{
move backward 0.3m
turn 90° to right
move forward 0.3m
if (robot does not conflict with wall)
{
turn 90° to left
move forward 0.3m
}
}
```

4.3.2 LRF Sensor

The robot with the LRF sensor travels according to the following algorithm.

```
//principle: robot goes along the left wall of maze.

if(distance between wall and robot is less than
0.15m or distance to 135° wall is less than 0.3m,or
forward direction distance to wall is less than 1m)
{
turn 30°to right and go straight
}
else
if(distance between robot and left-side wall is
greater than 1.5m)
{
turn 30°to left and go straight.
}
```

4.3.3 IR Sensor

The traveling algorithm of the robot with the IR sensor is the same as the previous bumper sensor algorithm, except that the IR sensor measures the straight distance. The IR sensor measures the distance only in the straight forward direction.

```
//principle: robot goes along the left wall of maze.

move forward 0.3m
if( distance between robot and wall is less than 0.5m)
{
move backward 0.3m
turn 90° to right
move forward 0.3m

if(distance between robot and wall is greater than
0.5m)
{
turn 90° to left
move forward 0.3m
}
}
```

5. Simulation Results and Analysis

5.1 Simulation Results

We performed the same simulation five times to ensure the objectivity of this simulation. We obtained simulation results as shown in Table 2-7.

1) Bumper Sensor

(1) while-loop

Table 2:. Simulation data for robot with bumper sensor using while-loop pattern

	1st	2nd	3rd	4th	5th	Average
Travel time(s)	289.1	298.9	292.4	254.0	267.0	280.28
Number of Turns	133	137	135	118	125	129.60
Number of Collisions	67	69	68	59	63	65.20

(2) procedure

Table 3:. Simulation data for robot with bumper sensor using procedure pattern

	1st	2nd	3rd	4th	5th	Average
Travel time(s)	290.0	287.4	285.2	289.7	286.4	287.74
Number of Turns	137	135	133	137	133	135.00
Number of Collisions	70	68	67	70	67	68.40

2. LRF sensor

(1) while-loop

Table 4.: Simulation data for robot with LRF sensor using while-loop pattern

	1st	2nd	3rd	4th	5th	Average
Travel time(s)	83.2	85.3	84.8	84.7	84.6	84.52
Number of Turns	9	9	9	9	9	9
Number of Collisions	0	0	0	0	0	0

(2) procedure

Table 5:. Simulation data for robot with LRF sensor using procedure pattern

	1st	2nd	3rd	4th	5th	Average
Travel time(s)	82.1	82.1	82.0	82.7	81.2	82.02
Number of Turns	9	9	9	9	9	9
Number of Collisions	0	0	0	0	0	0

3. IR sensor

(1) while-loop

Table 6:. Simulation data for robot with IR sensor using while-loop pattern

	1st	2nd	3rd	4th	5th	Average
Travel time(s)	293.4	289.9	286.5	279.5	283.8	286.62
Number of Turns	117	113	115	108	109	112.40
Number of Collisions	1	3	0	2	1	1.40

(2) procedure

Table 7.: Simulation data for robot with IR sensor using procedure pattern

	1st	2nd	3rd	4th	5th	Average
Travel time(s)	290.4	284.9	286.1	277.8	285.2	284.88
Number of Turns	115	113	112	110	113	112.60
Number of Collisions	2	1	0	1	2	1.20

5.2 Comparison and Analysis

LRF sensor is best one, requiring the least travel time and the least number of turns; it also has a conflict number of zero, as shown in Table 2-7.The IR sensor has a slightly less impressive performance than that of the LRF sensor; the bumper sensor has the worst performance.

Each robot, equipped with three different sensors and simulated on the while-loop pattern traveled for nearly the same time as it did on the procedure pattern, as shown in Figure 5-7. Also, the numbers of collisions and turns using the while-loop pattern were the same as using the procedure pattern. We were able to conclude that there is no performance difference between the while-loop and the programming patterns for three different sensors (LRF, IR, bumper).

It is known that the while-loop pattern has better performance than that of the procedure pattern and that the while-loop pattern is good for making a program that can sense the user's intent. However, we were able to obtain results in which the while-loop pattern does not have better performance than the procedure pattern.

It is true that for a long time we have adapted the while-loop pattern to simulate sensor programming without consideration of other methods. Therefore, in the future, it will be convenient to be able to select an easy pattern to use according to one's own simulation situation.

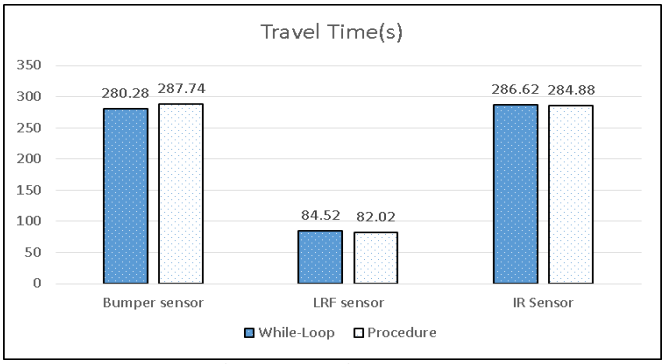

Figure 5.: Comparison of Travel time for the two sensor programming patterns.

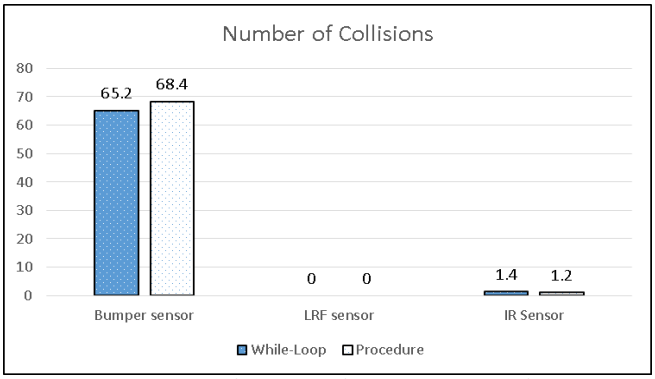

Figure 6:. Comparison of Number of Collision number for the two sensor programming patterns.

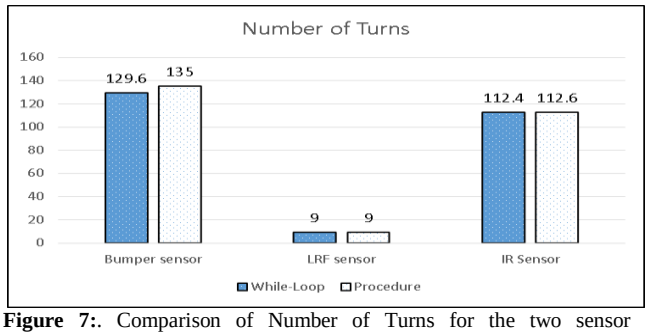

Figure 7:. Comparison of Number of Turns for the two sensor programming patterns.

6. Acknowledgments

This work was supported by the research grant of Kongju National University in 2016.

7. Conclusion

It is known that the while-loop pattern has better performance better than that of the procedure pattern. The while-loop pattern is good for making a program that can sense the user's intent. User can obtain measured data and execute the robot control routine using the while-loop block. There is no synchronization problem with the procedure pattern, but the procedure pattern can have a synchronization problem between the procedure execution and the sensor notification. Therefore, the procedure pattern can lead to abnormality of the robot control. It is known that programmers prefer the while-loop pattern to the procedure pattern.

To determine if there is any difference of performance between the two sensor programming patterns of SPL, we made a simulation environment to evaluate the performance. The simulation environment consisted of a maze and a robot with one of three sensors. A robot with one sensor travels in the maze, attempting to move from the start point to the end point. We measured the required travel time and robot actions (number of turns and number of collisions) necessary to escape from the maze and compared the performance of the two sensor programming patterns.

We were able to conclude that there is no performance difference between the while-loop pattern and the procedure pattern for the three different sensors (LRF, IR, bumper).

It is true that for a long time we have adapted the while-loop pattern to simulate sensor programming without consideration of other methods. Therefore, from now on, it will be convenient to select an easy pattern to use according to the simulation situation because it is now known that there is no difference of performance between the two types of sensor programming patterns.

8. References

[1] Microsoft Robotics Developer Studio 4. http://www.microsoft.com/robotics/. Date accessed: 04/01/2014.
[2] Kim Y, Chung J. Robot Programming for Logical Thinking Improvement: HongRung Pub, 2012.
[3] STEM Education. http://www.helloapps.com. Date accessed: 04/01/2014.
[4] Hong S. Intelligent Robot Programming for SMART Creative Engineering: IT Holic Pub, 2012.
[5] Park I, Kim D, Oh J, Jang Y, Lim K, Learning Effects of Pedagogical Robots with Programming in Elementary School Environments in Korea. *Indian Journal of Science and Technology.2015 Oct*, 8(26).
[6] Jang Y, Lee W, Kim J, The Changes of Middle School Students' Perception and Achievement based on the Teaching Method in Physical Computing Education. *Indian Journal of Science and Technology*.2016 June, 9(24).
[7] Lee J, Chung J, Comparative Analysis of the Performance of Robot Sensors in the MSRDS Platform. *Journal of the Korea Industrial Information Systems Research*.2014 Oct, 19(5), pp. 57-68.
[8] Kim H. Sensor Engineering: HongRung Pub, 2010.
[9] Kim J. Sensor Fundamental and Applications for Worker: Bokdoo Pub, 2012.
[10] Chung J, Comparison of Sensor Programming Schemes in MSRDS. *Proceedings of 2nd ICIECT, Vietnam*, 2016, pp.165-172.