Creation and Analysis of a Scenario Based Universal Sensory Driver Layer with Real-time Fault Tolerant Properties

TaeHoon A. Choi, Michael C. Nechyba, Eric M. Schwartz, A. Antonio Arroyo
tae@mil.ufl.edu, nechyba@mil.ufl.edu, ems@mil.ufl.edu, arroyo@mil.ufl.edu

Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611

Abstract

Sensor fusion and sensor integration is becoming an increasingly popular approach in dealing with complex sensor systems in autonomous mobile robots (AMR). However, the procedure for the sensor integration and sensor fusion is a non-trivial process. This paper presents a scenario based approach to sensor fusion based on the Autonomous Evolution of sensory and actuator Driver layers through Environmental Constraints (AEDEC) [1]. Using the scenario based approach, the programmer's work of creating a sensory driver will be eliminated by having the AMR learn the driver on is own. In the process of creating each scenario, sensor fusion is automatically implemented. If sensors change or even if the sensor configuration changes, the driver can be updated by having the AMR relearn the driver over again. Due to the tabular structure of the scenario based sensory drivers, malfunctioning sensors can not only be detected, but the driver can automatically adapt to the malfunctioning sensor in real-time. Furthermore, different AMRs trained using AEDEC architecture will have similar interpretations of its environment. This is guaranteed by having the AMR learn the driver in the same highly structured training environment. The behavioral coding is simplified by eliminating any reference to hardware dependent parameters. Finally, the level of abstraction and the consistency of the highly structured environment will allow for code portability.

1 Introduction

Sensors are a critical component of an AMR. Sensors' performance and the way sensors are utilized can greatly affect the performance of an AMR. Many of the current research has been centered on different methodologies concerning analysis, optimization, and procedure concerning sensor fusion. "Logical sensors" presented by Henderson and Shilcrat [4, 5], COM-based software architecture presented by Wang et al. [6], Hierarchical Phase-Template Paradigm presented by Lou and Lin, Object Oriented Programming [7, 8] are just some of the research which have shown very promising results. For more detailed background refer to Luo and Kay [9], who presents an excellent survey of multisensor integration and fusion. Each of the methodology listed above require non-trivial human planning for implementation.

Traditionally, sensors are painstakingly characterized and studied to find the proper thresholds and use of the sensors. Although it is time consuming, this method has worked well in many applications. The problem arises when there are many sensors. In order to guarantee the same performance for the new sensor, every new sensor had to be characterized, since different sensors can have different characteristics. The problems are further compounded in the cases where the sensors are being added to not just one AMR but multiple AMRs and, even worse, multiple AMRs of different types. When high precision sensors are used, the need for constant characterization of sensors can be reduced. The problem with this option is that the cost of the AMR increases.

Even if high precision sensors are used, sensor's performance and characteristics are greatly effected by how the sensor is physically implemented on an AMR. For example, an infra-red proximity sensor's performance is greatly effected by the orientation of the sensor on the AMR's platform. The variations are caused by different angles with respect to the environment and interference from other physical parts of the AMR causing partial reflections and partial blockage.

Now imagine trying to create a sensory driver layer for a complex sensor suite of an AMR. AMR programmers have spent many tedious hours "tweaking" the sensor parameters of a given AMR to get the AMR working properly. Although tedious and time consuming, it has worked well for many programmers. The problem with this method is that the programmer was constantly "tweaking" sensor parameters. If one of the sensors malfunctioned and needed to be replaced, a programmer had to recalibrate, or if one of the sensor's orientation changed, again recalibration would be required.

Problems do not end with tedious characterizations. When a programmer creates a driver for an AMR, the driver is usually robust and reliable. However, it has certain peculiarities and the bias of the original programmer built into the drivers [2]. For example, programmer A's definition of close might mean four inches; while, programmer B's might be 12 inches. Both versions will work as long as the programmer understands these assumptions taken by the original

programmer writing the drivers. The biggest problem arises when the sensors are changed, damaged, incorrectly wired, placed in different locations, etc. Take the instance where the sensors are rearranged to meet a new requirement. This might result in sensor 1, which initially pointed forward, pointing right. With other sensors also changing in a similar fashion, it would require total rewriting of the drivers for the robot, since sensor 1, previously used to detect obstacles in the front, can no longer be used for that purpose. Most programmers of AMRs have experienced some of these problems in one form or another.

2 Innate Knowledge in Sensory Driver Layer

Using Innate Learning [3], certain information and capabilities are hard coded onto the AMR. The AMR has prior knowledge of how many sensor ports it has and how it can get raw sensor data from those ports. However, this does not mean that the AMR is preprogrammed with what type of sensors, how many ports actually have sensor connected, or any information about thresholds or reading ranges. The AMR also has prior information about the complete list of the sensor scenarios. For example, "object to the front," "object to the far front," "object to the right," etc. The AMR only know that this list exist, but it does not have any information about the properties of the scenarios in the list or what they represent. The AMR will learn the meaning of these scenarios through the AEDEC architecture.

Table 1: The partial list of the scenarios and descriptions from [1]

Scenario Name	Description	Scenario Name	Description
NullTemp	Empty environment	WBR30	Wall Back Right 30 degrees
FC	object Front Close	WBR30F	Wall Back Right 30 degrees Far
F	object Front	WLC	Wall Left Close
FRC	object Front Right Close	WL	Wall Left
FR	object Front Right	WLF	Wall Left Far
FLC	object Front Left Close	WFL15C	Wall Front Left 15 degrees Close
FL	object Front Left	WFL15	Wall Front Left 15 degrees
LC	object Left Close	WFL15F	Wall Front Left 15 degrees Far

3 Scenarios

The sensory driver layer is made up of 60 unique scenarios. Scenarios are highly structured environmental situations presented to the AMR during the sensory driver layer learning process. Each scenario was chosen for its uniqueness and for the likelihood of an AMR encountering the scenario during operation. This paper does not make any assertions about the completeness or usability of the given set of scenarios. The set of scenarios was created from the past AMR programming experiences.

The complete list of the scenarios can be found in Table 1. NullTemp was created by having an empty environment and is useful for minimizing detection of "phantom" objects at the extreme edges of the detection range of the sensors. The next 16 scenarios, FC to BL, were created by placing a 5 ¼ X 5 ¼ object in various locations around the AMR. The directions represented by the 16 scenarios are self explanatory. For example, F represent front of the AMR; while L represents left of the AMR. The angle between F and L would be 90 degrees. As for FL, it is in between F and L, 45 degrees from each. As for distance, "far" is placed 18 inches from the center of the AMR; while, normal is place 11 inches from the center of the AMR.

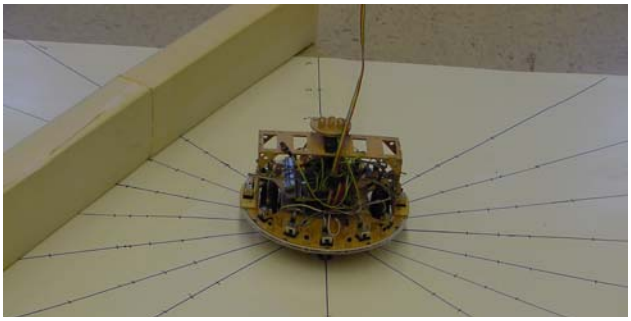

Figure 1: Picture of WFL30F scenario. In the picture Talrik II™ is facing the top of the picture.

AMRs tend to encounter "wall" type scenarios more frequently and it is more important to be able to recognize different orientations of the "wall" than a small obstacle. Hence, over half, 33 to be exact, are "wall" scenarios. In fact the remaining scenarios (corners and dead ends) can also be classified as "walls," putting majority of the scenarios in the "wall" category. All the "wall" scenario names start with W. 30 out of 33 "wall" scenarios are to the left and right of the AMR with the other three in the front. All "wall" scenarios were created using a "wall" 24 inches in length. WR represents a wall that is parallel to the axis line running from the front to the back of the AMR. 15 and 30 degrees represent the angle of the "wall" with respect to WR and WL. The distance represented by "close," "normal," and "far" for the "walls" are nine, 11, and 13 inches, respectively, from the center of the AMR. An example of a wall scenario (Wall Front Left 30 degrees Far) is shown in Figure 1.

"Corners" are implemented using two 12 inch walls. They are placed at 90 degrees respect to each other

to create a corner, as shown in Figure 2. Each of the walls making the corner is placed 11 inches away from the center of the AMR. The direction represents the position of the corner of the walls respect to the AMR, as shown in Figure 2 (Corner Back Left). The final category of scenarios is DF and DB. In the case of DF, three 12 inch walls are used to surround the front of the AMR to create a "dead end."

There are enough variations in the given set of scenarios to represent most of the real world environment that an AMR might face. Furthermore, from this basic set of 60 scenarios, more complex scenarios can be created by "merging" two or more scenarios together.

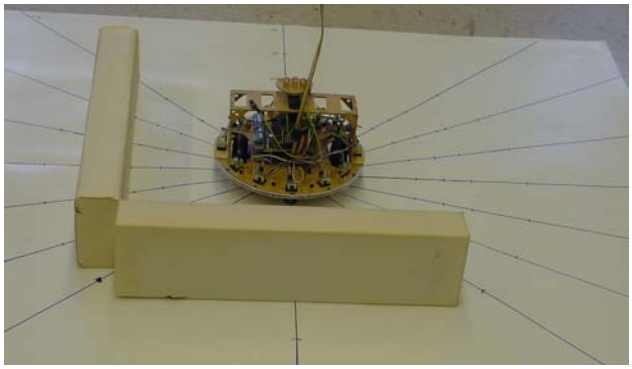

Figure 2: A picture of CBL scenario. In the picture Talrik II™ is facing the top of the picture.

4 Learning Algorithm for the Sensory Driver Layer

For each scenario, the appropriate environment is created according to the specifications of the scenarios as described in the previous section. The AMR is programmed to ask for a scenario and then the scenario is presented to the AMR. The AMR takes a complete reading of all the sensor ports and creates a template for the given scenario. This process is repeated for every scenario in the list. For this experiment, a human helper laid out the scenarios for the AMR. Future implementations will incorporate an automatic environment controlled by the AMR to lay out the environment for each scenario.

In order to create the templates, six different readings are made for the same scenario. Then for each sensor port, the highest and the lowest values are dropped and the four remaining readings are averaged to create each scenario template. This filtering process was implemented to minimize the environmental noise error, sensor fluctuations, etc.

Once the scenario templates are complete, templates could be used to find the closest match for any new scenario that the AMR might encounter. The best match was found using a modified Euclidean distance. For example, AMR would take a sensor sweep of all it sensors and then find the modified Euclidean distance of the new scenario with respect to all of the scenarios in the sensory driver layer. The scenario with the smallest distance is then selected as the match. The modified Euclidean distance is given in Equation 1, where N = number of sensors; n = sensor number; X = database of template values; Y = new sensor readings to be matched.

$$MED = \sum_{n=0}^{N-1} (X_n - Y_n)^2 \qquad \text{Equation 1}$$

5 Analysis of the Sensory Driver Layer

Once the scenario templates were created, series of experiments were performed to test the performance and usability of the sensory driver layer.

5.1 Recognition of the Original Scenarios

The AMR was given the same set of scenarios to see if it could properly identify the scenarios that it learned. In order to analyze the data, a table of modified Euclidean distance (MED) was made. A complete table is 60 by 60, where horizontal headings represent the scenarios in the database and the vertical headings represent scenarios presented to the AMR for identification. Each cell represents the MED of the corresponding vertical and horizontal headings. For readability, the cell containing the smallest MED is highlighted with thicker borders. For 60 scenarios tested, the AMR correctly identified all 60 scenarios, represented by the smallest MED occurring on the diagonal of the table.

Table 2 Partial table of modified Euclidean distance of each scenario respect to the database of scenario templates from [1]

	NullTem	FC	F	FRC	FR	FLC	FL	LC
NullTem	3	1183	206	1010	254	1442	224	742
FC	1133	9	328	1164	962	1206	834	1842
F	290	372	3	871	365	1075	269	999
FRC	999	1261	816	4	252	2348	1200	1738
FR	286	1056	325	253	3	1707	503	1025
FLC	1331	1031	942	2218	1562	18	452	1230
FL	307	799	246	1278	552	476	10	716
LC	713	1859	916	1720	964	1058	532	12

For easy reference, a small part of the table is shown in Table 2. In this table the highlighted cells representing matches mostly have values in the single digits, while other cells average in the hundreds and sometimes in the thousands. Larger differences represent less likelihood for misidentification. For example, reading across the row labeled NullTemp in Table 2, the

shaded first cell represents the match between NullTemp and NullTemp with a value of three. The next smallest value in the row is 39, representing the MED between NullTemp and B (object to the Back). Although 39 is an order of magnitude larger than the matching value of three, 39 is relatively smaller because B is a small object relatively far away from the AMR. Consequently, the AMR makes a smaller distinction between NullTemp and B than between NullTemp and BC (object to the Back Close).

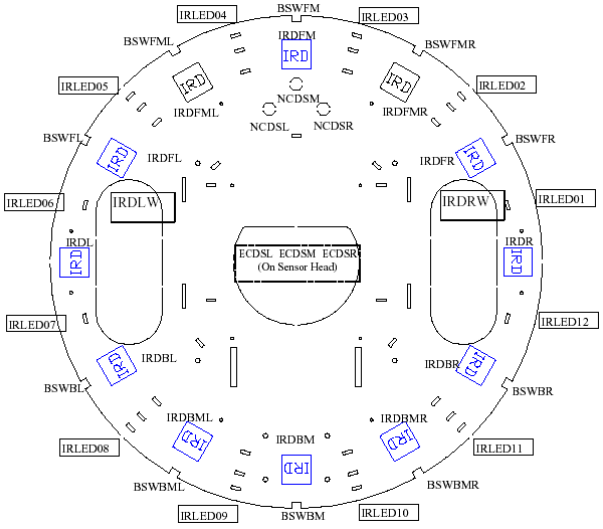

Figure 3 Talrik II™ Sensor Layout Diagram

Table 3 Partial table of modified Euclidean distance of each scenario respect to the database of scenario templates with one sensor malfunctioning from [1]

	NullTen	FC	F	FRC	FR	FLC	FL	LC
NullTen	2	1270	245	1021	255	1453	227	711
FC	735	489	250	786	570	1752	854	1414
F	209	687	66	790	284	1444	386	918
FRC	991	1113	728	28	260	2318	1198	1730
FR	306	1040	323	239	5	1759	541	1045
FLC	782	1540	785	1661	1009	637	413	681
FL	130	1274	315	1155	399	967	123	539
LC	678	1916	921	1697	931	1147	545	7

5.2 Recognition of the Original Scenarios with One Malfunctioning Sensor

To observe how a malfunctioning sensor would affect the performance of the sensory driver layer, one of the sensors was disabled from getting any readings. IRDFML, a "front left" sensor shown in Figure 3, was disabled and the previous experiment described in section 5.1 was repeated. The partial results are found in Table 3. The experiment resulted in 15 misidentifications out of 60.

Upon closer analysis, one can observe that the 15 misidentifications are not random; instead, they are related closely to the correct scenario. For example, two of the misidentifications are shown in Table 3. First, FC (object to the close front) is incorrectly identified as F (object to the front). Although the distance of the object was wrong ("normal" distance rather than "close" distance), the sensory driver layer correctly identified the direction of the object ("front"). Complete analysis shows that most of the errors occur while trying to identify different types of "walls." This is a consequence of the fact that as discussed in section 3, "wall" scenarios consist of "walls" shifted two inches from each other and rotated 15 degrees from each other. These "wall" scenarios that physically differ from each other in small variations cause the sensory driver layer to make mistakes. However, these mistakes are expected due to a malfunctioning sensor. It should be noted that the mistakes are closely related to the correct scenarios, demonstrating the flexibility of the sensory driver layer. Other errors include wall being mistaken for an object, a wall being shifted at an angle, etc. But in all cases, relative direction of the obstacle is consistent.

Table 4 Partial table of modified Euclidean distance of each scenario respect to the database of scenario templates with two sensors malfunctioning from [1]

	NullTen	FC	F	FRC	FR	FLC	FL	LC
NullTen	15	1153	196	860	188	1430	232	724
FC	744	494	259	687	525	1761	863	1423
F	242	668	75	703	261	1403	393	921
FRC	927	1175	726	14	228	2234	1120	1636
FR	314	1000	307	235	11	1693	523	1023
FLC	806	1480	771	1543	961	659	437	765
FL	169	1193	302	990	338	932	136	548
LC	522	1676	727	1379	697	1083	413	237

5.3 Recognition of the Original Scenarios with Two Malfunctioning Sensors

To take the analysis further, a second sensor was disabled in addition to the sensor disabled in section 5.2. The second sensor was chosen form the same quadrant of the first sensor and from one of the primary axis sensors (sensor located in the front, left, right, and back). The second sensor chosen was the left sensor, IRDL shown in Figure 3. Again the experiment performed in sections 5.1 and 5.2 was repeated. The partial results are given in Table 4. With two sensors disabled, this experiment resulted in 20 errors, five more than with just one sensor disabled. Analysis of the data showed similar results with the experiment with one sensor disabled, discussed in section 5.2. Majority of the misidentifications made errors in distance and the "wall" scenarios. In comparing

Table 4 (MED data of two sensors disabled) with Table 3 (MED data of one sensor disabled), mistakes were made in direction of the object as well. This is different from the one sensor disabled case, in which direction was pretty consistent. Results in the complete table show further degradation in the performance of the sensory driver layer, due to two sensor malfunction.

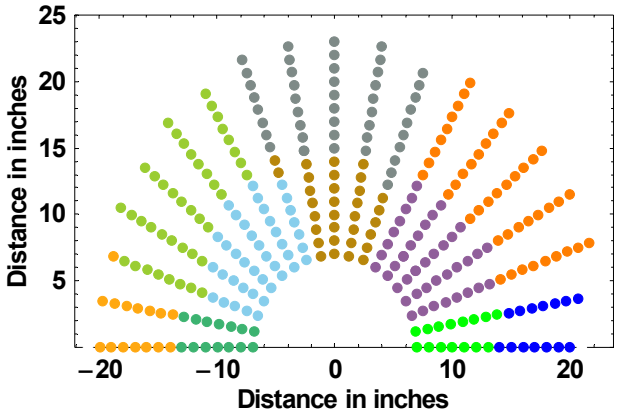

Figure 4 Scenario mapping of a 5 ¼ x 5 ¼ inch block in the 180 degree region in front of the AMR. Each radial line represents 10 degree offsets and each point represents one inch offsets

5.4 Sensor Template Mapping for Objects and Walls

Thus far in the proceeding sections, namely5.1, 5.2, and 5.3, sensory driver layer was tested with scenarios identical to the scenarios used during the learning process. The next few experiments test the sensory driver layer with scenarios different from the scenarios used during the learning process. The first of these experiments creates a sensor template map covering front half of the AMR. The sensor template map was created by placing a 5 ¼ x 5 ¼ inch block in different locations within a 180 degree region about the AMR. The block was placed every inch starting from seven inches to 24 inches in a line pointing radially out from the center of the AMR. This was done for every 10 degrees for the 180 degree region around the front of the AMR.

The scenario map from the experiment is presented in Figure 4. From this plot, one can observe that the region surrounding the original location of a scenario is also identified as the same scenario. For example, gray dots in Figure 4 represents "object to the front" (F). Original location of the F scenario is directly in front of the AMR 18 inches away from the center of the AMR. Notice that many other dots in the region also share the same classification of F. These groupings of regions around the original location of the scenarios indicate that similar scenarios will be properly identified as intended.

One will notice that not all measurements extend out the same distance. The lack of a plot represents the distance at which the sensory driver layer started identifying a NullTemp, an empty environment. The differences in this and in other characteristics are due to inherent differences in the sensor and/or their alignment.

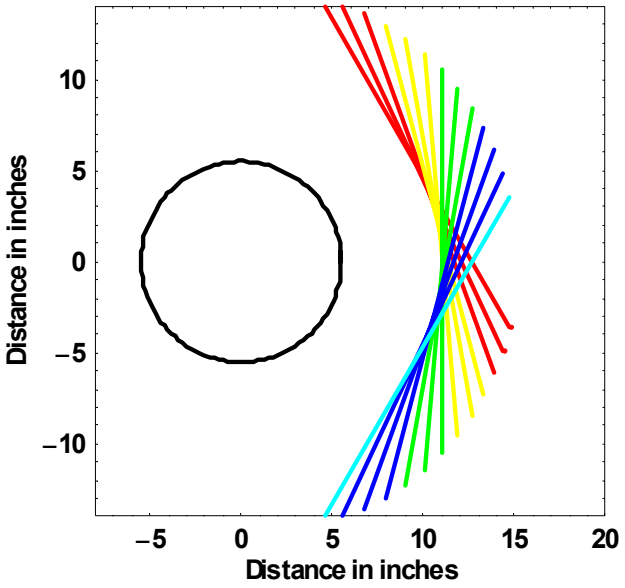

Figure 5 Scenario mapping of a wall being rotated around an AMR every five degrees

Next, the usability of the sensory driver layer was tested by rotating a wall around the AMR to verify that different wall angles could be identified. A wall was rotated every five degrees starting from 30 degrees to -30 degrees, where the vertical line represents 0 degrees. One can observe from Figure 5, the consistent transition from one scenario to the next. Figure 5 shows that the wall was classified as "wall front right 30 degrees," "wall front right 15 degrees," "wall right," "wall back right 15 degrees," and "wall back right 30 degrees" as the wall is rotated around the AMR.

The results presented in Figure 4 and Figure 5 show the adaptability of the sensory driver layer to properly identify untrained scenarios by finding the nearest trained scenario.

5.5 Object Morphing

The scenario mapping experiment showed the effects of varying the location of objects throughout the region. This section presents experiments that vary the shapes and sizes of the objects. In the first experiment a block (5 ¼ x 5 ¼ inch), used as an obstacle during the learning procedure, was lengthened by two inch increments to observe at what length the object would start to be recognized as a "wall." Figure 6 presents a graphical result of the experiment. The objects were placed 11 inches from the center of the AMR. The object was identified as an obstacle from 5 ¼ inches in length

to11 ¼ inches, as a "wall far away" from 13 ¼ inches to 19 ¼ inches, and as a "wall" from 21 ¼ inches to 27 ¼ inches. The dimensions of the objects used during the learning process for an obstacle was 5 ¼ inches, while a wall was 28 inches in length. The sensory driver layer identified the object as a wall starting at 13 ¼ inches.

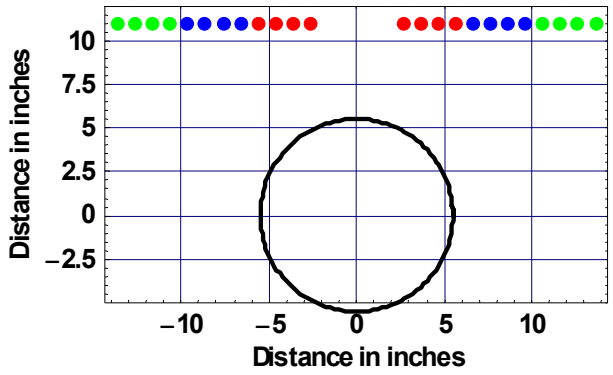

Figure 6 The plot of lengthening a 5 ¼ inch object to a 27 ¼ inch object in 2 inch increments

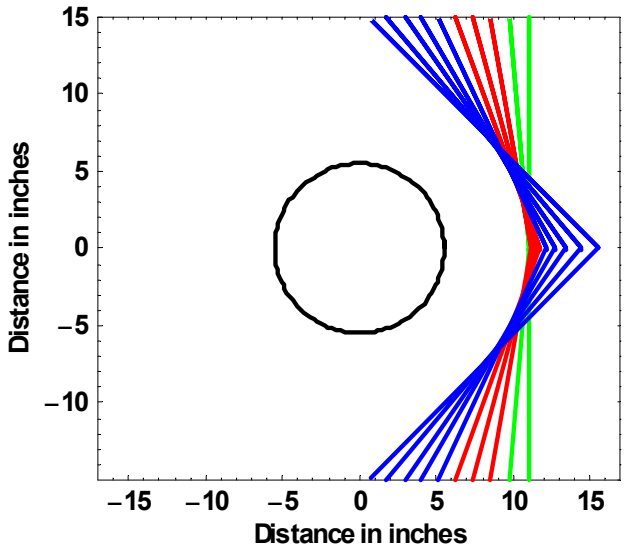

Figure 7 The plot of bending a wall into a corner by 10 degree increments.

In the second experiment, a wall was bent ten degrees at a time into a corner. The results shown in Figure 7 shows that the transition from "wall right," "wall right close," and "corner right" is consistent and smooth.

These object morphing experiments show that the sensory driver layer can deal with objects of different shapes and sizes. The object mapping and object morphing experiments demonstrate that the sensory driver layer based on the AEDEC architecture is highly robust and highly adaptable to different environmental scenarios, even for the untrained scenarios.

Table 5 Partial table of modified Euclidean distance of each scenario respect to the new database of scenario templates with two sensors malfunctioning from [1]

	NullTem	FC	F	FRC	FR	FLC	FL	LC
NullTem	3	1183	206	1010	254	1442	224	742
FC	1133	9	328	1164	962	1206	834	1842
F	290	372	3	871	365	1075	269	999
FRC	999	1261	816	4	252	2348	1200	1738
FR	286	1056	325	253	3	1707	503	1025
FLC	1331	1031	942	2218	1562	18	452	1230
FL	307	799	246	1278	552	476	10	716
LC	713	1859	916	1720	964	1058	532	12

6 Sensory Driver Layer's Ability to Compensate for Sensor Malfunctions

Recall the experiments performed in sections 5.2 and 5.3, where one and two sensors were disabled, respectively. With one sensor disabled, it resulted in 15 errors out of 60; while, with two sensors disabled, it resulted in 20 errors out of the same 60. Normally, the problem would be resolved by replacing the malfunctioning sensor or sensors and recalibrating the parameters. But what if for some reason the sensors could not be replaced. Then it would require reprogramming of the AMR. The programmer has to manually compensate for the malfunctioning sensors by recreating the drivers. This would be a time consuming and complex process. Even worse, if the code depends on direct sensor values, much of the code probably has to be rewritten to compensate for the bad sensors. These options would not be acceptable in most situations.

The sensory driver layer, presented by AEDEC architecture, allows for quick and easy method for compensating for sensor malfunctions and/or changes by simply relearning the scenarios. In the first experiment, AMR with one bad sensor relearned the 60 scenarios. Then it was tested, as described in section 5.1, to see if any improvements could be made on the 15 errors it had before the retraining. It passed with flying colors, improving from 15 errors to none. It properly identified all 60. This process was extended to the case where two sensors were bad, which previously resulted in 20 out of 60 errors. Even with two bad sensors, it properly identified all 60 scenarios after repeating the learning process. Partial results are given in Table 5.

7 Real-time Self Correcting Sensory Driver Layer

In section 6, sensory driver layer easily adapted to the malfunctioning sensors by relearning the sensory driver layer with the malfunctioning sensors. Although it

is a simple and efficient method, it does require a relearning process. If an AMR can detect and compensate for a malfunctioning sensor in real-time it would be that much more useful. A closer comparison of the new sensory driver layer to the original resulted in a surprising but an obvious result. The new drivers were almost identical to the old, with the exception of the malfunctioning sensors. The net effect of the relearning process was equivalent to just dropping the malfunctioning sensor from the input sensor vector and dropping the malfunctioning sensor's component from each of the previously learned scenarios. Detection of a malfunctioning sensor is surprisingly easy. During the initial boot up procedure, an AMR can make sensor readings and compare against its scenarios and just disable any sensor with improper readings.

As long as an AMR can detect a malfunctioning sensor, it can ignore that sensor reading and also ignore the MED generated by the offending sensor respect to each of the scenarios. In effect, adapting the sensory driver layer in real-time to malfunctioning sensor or sensors.

8 Conclusion

This paper outlines a methodology of autonomously creating a sensory driver layer using a scenario based approach. The supporting analysis demonstrated that the resulting drivers are robust and adaptable. It has been shown that the resulting sensory driver has real-time discrete error detection and correction capabilities. Without any human intervention, a robust sensory driver is created through sensor fusion. It presents an alternative to explicitly creating a sensor fusion architecture.

Reference:

[1] T. A. Choi, "Autonomous Evolution of Sensory and Actuator Driver Layers through Environmental Constraints," *Ph.D. Dissertation* (University of Florida, Gainesville, FL 32611, December 2002).

[2] T. A. Choi, E. A. Yim, and K. L. Doty, "Environmental Reinforcement Learning: A Real-Time Architecture for Primitive Behavior Refinement," *ROBOLEARN 96: An International Workshop on Learning for Autonomous Robots* (Key West, Florida, May 19-20, 1996), pp. 20-27.

[3] T. A. Choi, E. A. Yim, A. A. Arroyo, and K. L. Doty, "Automatic Configuration of Sensors and Actuators Through Innate Learning," *Robotics 98: The 3rd International Conference and Exposition on Robotics for Challenging Environments* (Albuquerque, NM, 26-30 April 1998), pp. 64-70.

[4] T. Henderson and E. Shilcrat, "Logical Sensor System," *J. Robot. Syst.,* vol. 1, no. 2, pp. 169-193.

[5] T. Henderson and E. Shilcrat, "A Fault Toerant Sensor Scheme," *Proc. 7th Int. Conf. Pattern Recognition,* (Montreal, PQ, Canada, July 1984), pp. 663-665.

[6] J. Wang, J. Su, Y. Xi, "COM-based Software Architecture for Multisensor Fusion System," *Information Fusion, vol. 2* (2001), pp. 261-270.

[7] T. Henderson and E. Weitz, C. Hanson, and A. Mitiche, "Multisensor Knowledge Systems: Interpreting 3-D Structure," *Int. J. Robot. Res.,* vol. 7, no. 6, (1988), pp. 114-137.

[8] J. C. Roger and R. A. Browse, "An Object-based Representation for Multisensory Robotic Perception," *Proc. Workshop Spatial Reasoning and Multi-Sensor Fusion,* (St. Charles, IL., Oct. 1987), pp.13-20.

[9] R. C. Luo and M. G. Kay, "Multisensor Integration and Fusion in Intelligent System," *Systems, Man, and Cybernetics,* vol. 19, no. 5, (Sept./Oct. 1989), pp. 901-916.