Autonomic Sonar Sensor Fault Manager for Mobile Robots

Autonomic Sonar Sensor Fault Manager for Mobile Robots

Martin Doran, Roy Sterritt, George Wilkie
Faculty of Mathematics and Computing
University of Ulster
Jordanstown, N.Ireland
doran-M18@email.ulster.ac.uk, r.sterritt@ulster.ac.uk, fg.wilkie@ulster.ac.uk

***Abstract*—NASA, ESA and NSSC space agencies have plans to put planetary rovers on Mars in 2020. For these future planetary rovers to succeed, they will heavily depend on sensors to detect obstacles. This will also become of vital importance in the future, if rovers become less dependent on commands received from earth-based control and more dependent on self-configuration and self-decision making. These planetary rovers will face harsh environments and the possibility of hardware failure is high, as seen in missions from the past. In this paper, we focus on using Autonomic principles, where self-healing, self-optimization and self-adaption are explored using the MAPE-K model and expanding this model to encapsulate the attributes such as Awareness, Analysis and Adjustment (AAA-3). In the experimentation, a Pioneer P3-DX research robot is used to simulate a planetary rover. The sonar sensors on the P3-DX robot are used to simulate the sensors on a planetary rover (even though in reality, sonar sensors cannot operate in a vacuum). Experiments using the P3-DX robot focus on how our software system can be adapted with the loss of sonar sensor functionality. The autonomic manager system is responsible for the decision making on how to make use of remaining 'enabled' sonars sensors to compensate for those sonar sensors that are 'disabled'. The key to this research is that the robot can still detect objects even with reduced sonar sensor capability.**

Keywords—autonomic, self-adaption, self-healing, self-optimization

I. INTRODUCTION

For a mobile robot to be able to navigate within its surrounding terrain, its sensors have to be at optimal performance. Sensors send information, in the form of electronic signals back to the robot controller. This information is then processed and therefore allows the robot to make a decision on its next command action. Sonar sensors can be used to allow a mobile robot to detect objects within its path. Sonar sensors usually are situated on the front and rear of the robot. The sonar sensors are typically arranged in an 'array' configuration, were each sensor is angled separately so that the sensors can cover a 180° range in front of the robot. While the mobile robot is moving, one or more of the sonar sensors can locate an object that may be on the robots path. However, what if there was a hardware issue with some of the sonar sensors? The ability to detect objects would be greatly reduced.

Planetary rovers use both cameras and sensors to navigate the terrain of a moon or a planet. Sensor failure would mean a severe impact on mission objectives. Experimental planetary rovers such as the SR2 [1], use range finders to help them detect objects. Even before the autonomic concept [2], researchers have been looking at *fault tolerance* as a biological unit, were fault detection was handled with *adaptive sensor analysis* [3]. Further has also shown that by comparing the *known* state and actual sensor feedback of a collection of sensor nodes, could lead to the detection of single sensor drop-outs. If sensor failure is identified, then compensation could be possible by using *known* values instead of the measured ones [4].

The autonomic 'self-adaptive' approach implies that even with reduced sensor functionality, it is possible to carry on with mission objectives, by making use of what sensor functionality is still available. Autonomic 'self-awareness can also be employed to detect early signs of degradation in sensors. Using knowledge gathered from previous missions, regular health checks can detect if a particular sensor module is not performing at an expected level. Autonomic *self-adaptive* principle reacts to an unforeseen situation, like damaged caused to sensors. The autonomic *self-awareness* can initiate a change in the mission strategy, if the predicted failure could jeopardize mission objectives.

II. PREVIOUS WORK

Previous work [5] detailed how a damaged wheel on a mobile robot caused the robot to veer off to the left (or right), depending on what wheel was had faulted. Policies were initiated within the autonomic management system to compensate for the *wheel alignment* problem. The mobile robot was able to self-adapt even with the wheel fault and therefore continuing to function. For this paper, we want to explore how a mobile robot can detect objects after it has suffered failure to some of its sonar sensors. Using autonomic principles, we want to investigate how self-analyzing can detect faults within the sonar sensor array and then employ self-configuration and self-adjustment protocols to compensate for the limitation in sensor detection.

III. RELATED WORK

In our research, the detection of sonar sensor faults forms part of our experimentation into *self-diagnosis*. Sensors faults do not always show themselves as simply being *non-functioning* or *disabled*. The *intermittent* fault or *under-performance* fault are the most difficult to detect [13]. In this research, the authors use *Evidence*, *Fault* and *Value* nodes to recognize hardware faults by observing the change of the sensor data over time. They described *small deviation* and *big deviation* to evaluate the extent of the sensor error. In our

approach we use *tolerance ranges* to decide if a sensor is performing correctly.

Detection of abnormal behavior in sensors can also be achieved by comparing sensor data with neighboring sensor data [14]. In this research the authors take input readings of the sensors and subject them to a correlation test that determines which sensors are correlated to each other. In our research, the data from *suspected* sonar sensors is checked by using adjacent sonar sensors; if the results between the sonar sensors do no match, then we can declare the *suspected* sensor as being faulty.

There have been several activities in the US regarding the research in loss of sensor function. For example, in [12], the authors use Organizational-based Multi-agent System model (OMAS), to describe sonar sensor capability loss in robots. If a sensor losses functionality, then another sensor is substituted, that can carry out at least some or all of its predecessor capability. *Organizational rules* are applied to decide what agent roles can one sensor be applied to another sensor.

The Related Work contributions all involve detection of sensor failure from a *fault tolerance* and *diagnosis* approach. Our research centers on detection, analysis and adjustment of sensor faults using autonomic principles; by employing specialized algorithms, we can adapt the affected hardware systems to continue to function even when functionality has been greatly reduced.

IV. AUTONOMIC MODEL

In 2001, IBM made a commitment to the conceptual ideas of autonomic computing. The main goal was to create systems that could self-manage and take appropriate action when facing system failure [19].

Self-diagnosis is not only concerned with the discovery of potential fault but also the severity and consequences relating to the fault [15]. *Self-configuring* has the ability for a system to automatically adjust itself when faced with changing conditions. *Self-healing* is concerned with recovery and repairing itself when dealing with unexpected faults. *Self-optimizing* has the knowledge of tolerance and performance values. It can then use known policies to maintain optimal performance and employ new policies to improve performance. *Self-protecting* is part of the autonomic system that can detect and mitigate possible threats. It can also establish what could potentially be a threat and use known policies to handle this threat [5].

The autonomic computing system requires sensor channels to sense possible changes within the internal and external environment; it will also require motor channels to react to those changes [16]. Including autonomic principles in the software design of a robotic architecture, could extent the robot operating time in the field. The main challenge in designing an autonomic system is that all possible fault scenarios cannot be anticipated; rather design a system that can detect and resolve problems at run-time [17].

The MAPE-K (Monitor-Analyse-Plan-Execute over a knowledge base), feedback loop model is the standard model to describe autonomic and self-adaptive systems [2]. The Knowledge component collects and maintains data from managed system and retains policies which can be shared with the MAPE components. See Figure 1. The Monitor (M) takes in data in from sensors and stores the data in (K) Knowledge. The Analyse (A) performs analysis to establish if adaptive measures are required. If adaption is required, then the information is passed onto Plan (P) to trigger a policy *algorithm* that will compensate for the fault condition. The Execute (E) will deliver the *policy* commands via the effectors [18].

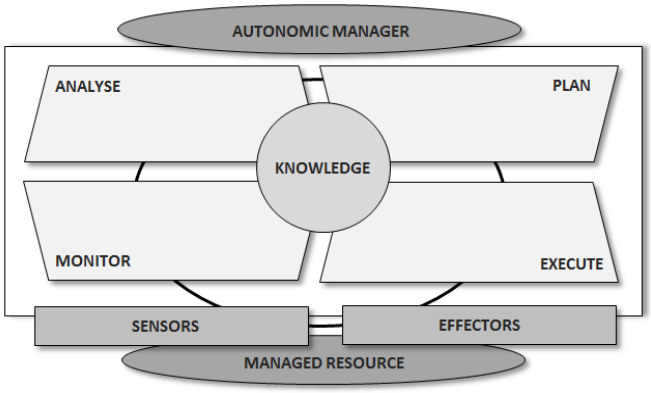

Figure 1. MAPE-K feedback loop.

The components found in the MAPE-K feedback loop can be adapted to form the architecture of the Autonomic Sonar Manager (see Figure 2).

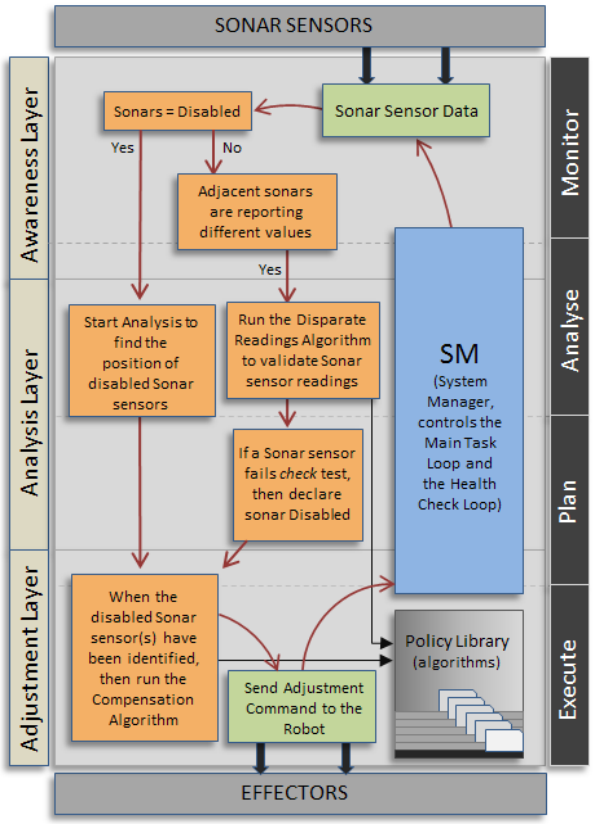

Figure 2. The Sonar Manager Architecture (AAA-3).

The Sonar Manager Architecture comprises of three layers; the Awareness Layer, Analysis Layer and Adjustment Layer (AAA-3). The AAA-3 layers are based on the MAPE-K components from Figure 1. The AAA-3 Layer configuration is used to map our sonar sensor manager architecture, rather than using the traditional MAPE configuration. However, there is over-lapping between the MAPE and AAA-3, regarding components such as Analyse and Execute (see Figure 2). The Main Task loop controls the normal robot operations. The 'health check' loop runs at periodic intervals, to determine the health of the sonar sensors.

1. *Awareness Layer*

The Awareness Layer can only perform a limited amount of processing. The main function of the Awareness Layer is to decide if there is a failure within the sonar sensors. If failure is detected, then the information is passed to the Analysis Layer for processing. The Awareness Layer can detect if there are unusual readings between adjacent sonar sensors; the Awareness Layer will then record those sonar sensors under suspicion and pass this information to the Analysis Layer.

2. *Analysis Layer*

The Analysis Layer uses data received from the Awareness Layer to establish the extent of the sonar sensor failure. This Layer will map out what sonar sensors have been disabled and pass this information to the Adjustment Layer. The Analysis Layer will also check those sonar sensors that have been marked as *suspicious*; it will use a *checking* algorithm to verify if a sonar sensor is performing within expected parameters. If a sonar sensor if reporting invalid data, then that sonar sensor is marked as being disabled.

3. *Adjustment Layer*

The Adjustment Layer receives data from the Analysis Layer showing what sonar sensors are currently disabled. The Adjustment Layer will then decide what algorithm (from the policy library) is appropriate to handle the fault condition. When the algorithm has been executed, the instructions are passed from the Adjustment Layer to the Effectors.

V. PIONEER P3DX ROBOT AND SONAR SENSORS

The Pioneer P3DX is a research laboratory robot that has two independent drive systems for each wheel. The robot contains an on board computer that can be uploaded with user defined programs such as a Microsoft Windows operating systems and .Net application. The P3DX is equipped with two sets of Polaroid sonar sensors arrays. The Polaroid sonar sensor array comprises of 8 electrostatic transducers and a sonar ranging module, see Figure 3 and Figure 4. The individual transducers are controlled by the ranging module. The 'echo' signals captured by the transducers, allows the ranging module to calculate ranges from 6" to 35ft [6].

Figure 3. Pioneer P3DX fitted with the Polaroid Sonar Array (front and rear). There is also a Bumper Array fitted.

Figure 4. Polarid Sonar Sensor Transducer is used for operations in air at ultrasonic frequencies.

The Sonar sensor array is manufactured in such a way that each of the sonar transducer are set at different angles from the center of the robot, see Figure 5. This allows for maximum detection of the surrounding terrain and obstacles within the robots path. Each sonar sensor is allocated a number; 0-7 for the front array and 8-15 for the rear array.

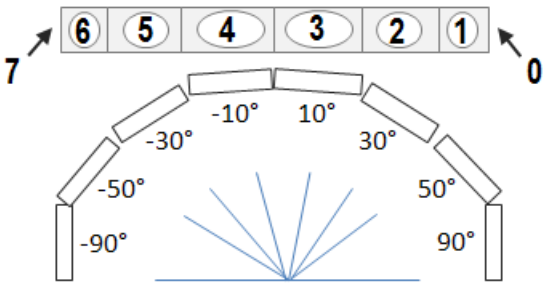

Figure 5. The sonar sensors are arranged at different angles from the center of the P3DX robot, so that maimum dectection coverage is achieved.

VI. SONAR SENSOR PROBLEMS

Ranging sensors like sonar are widely used in research and industrial robotics. They allow a robot to see an object without actually coming into contact with it. However, sonar sensors are limited in range and can also suffer from 'Ghost' echoes, where there is dense obstacle distribution and complex surfaces on objects [10]. In this Paper, we are concerned only with sonar sensors detecting objects; rather than the performance level of sonar sensors detecting obstacles of different shape and texture, located in varying environments.

If sensor hardware fails (or loses its calibration), then there is no choice but to abandon the sensor [8]. When a sonar sensor(s) becomes faulty, it can impact the robots ability to navigate in various ways. A single sonar sensor fault would only result in a minor reduction in the robots object detecting ability. The faulty sensor can then be compensated for, if necessary, by a neighbouring 'working' sensor. Detection of a faulty sonar sensor on the P3-DX is typically discovered by reading a value of '5000', from the sonar array readings. This can be a result in the failure of the sonar micro-controller or where the physical connection to the sonar has been severed [11]. However, a more difficult sonar fault to detect is where the sonar is reporting some data but this data is inaccurate, due to an impact (from the surrounding terrain) on the sensor itself, which has distorted the readings. Figure 6 shows how we *classed* various sonar sensors failures (states), for the P3-DX robot.

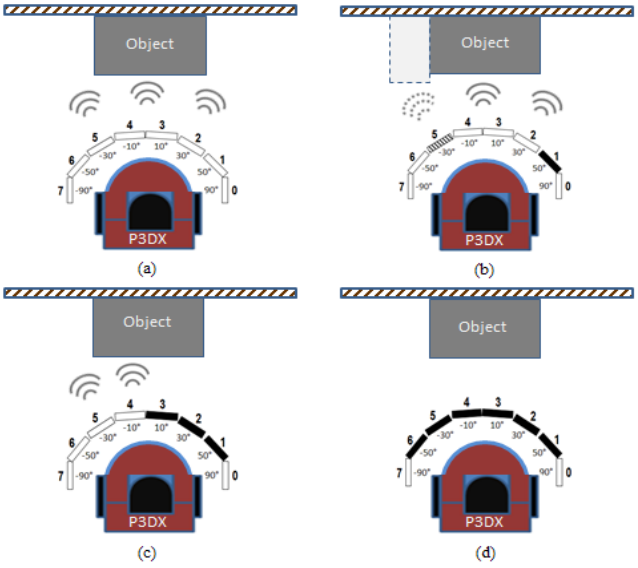

Figure 6. Failure states for the Sonar Sensors on the P3-DX Robot. (a) IsNormal – all sonar sensors are working as expected. (b) IsMinor – one or two sonars are either disabled or reporting errornous data. (c) IsMajor – a loss of 3 or more (but not all) sonars, providing only limited sensing ability. (d) IsCatatrosphic – all forward facing sonars are disabled. No ability to detect objects on immediate path.

VII. SOFTWARE FRAMEWORK

Software Development for this paper is carried using the MRDS (Microsoft Robotics Developer Studio). MRDS is a .Net based programming environment for building robotic applications. Code development was implemented using C# in Microsoft's Visual Studio environment [7].

To create robot movement commands and sonar sensor readings, it required implementation of event driven commands that are integrated into state-based processing behaviors using a standard state machine concept [7]. The user Interface provides a means of controlling the movement of the robot and also monitoring the sonar values from the front and back sonar arrays. The user interface also provides error reporting for any faulty sonar sensors; this includes sonar sensors that are not showing any readings – (total failure) or sonar sensors that are not reporting data as expected.

VIII. SONAR SENSOR FAULT EVALUATION

From the *failure states* shown in Section VII, we will evaluate states *IsNormal*, *IsMinor* and *IsMajor*. The *states IsMinor* and *IsMajor* both involve sonar sensor faults but these *states* are recoverable, in that, it is still possible to detect objects even if some of the sonar sensors are disabled.

1. Frontal sonar sensor test (IsNormal)

In the first experiment, we tested the effectiveness of the sonar sensor array to detect objects. A regular object was placed at different angles from the robot; these angles were calculated using the fixed position of each of the sonars on the robot (see Figure 5.). Sonars 1-6 on the sonar array are only used, as they are the *forward* facing sensors. As the robot moves, sonar data is analyzed; when the data reported back from the sonar sensors reaches a *object range* value, then the robot is issued a STOP command. In this experiment, the *object range* value was set at 250mm. Even with a STOP command, there is some additional forward moment due to the robots momentum before it comes to a complete stop. Table I shows the values for each of the sonar sensors as they detect the *object* in their path.

TABLE I. FRONTAL SONAR SENSOR TEST (NO SONAR FAULTS)

Angle of the Object to the Robot	Sonar Position on the array					
	1	2	3	4	5	6
50°	983	982	982	983	983	232
30°	817	818	817	817	237	818
10°	711	712	712	238	714	713
-10°	703	704	242	704	704	704
-30°	783	240	784	784	784	784
-50°	241	989	987	988	987	987
	The grey cells denote the value of the sonar readings in mm, when an object has been detected.					

2. Single sonar sensor evaluation (IsMinor)

If a sonar sensor has become disabled, due to an internal electrical fault or an impact from an object in the surrounding environment, this is reported to the system 'manager' as a reading of '5000'. However, in some cases, the sonar sensor is reporting what looks like a valid reading (not 5000) but in fact this reading could be false. The sonar may have received some slight damage or there could be a possibility of electrical data transmission becoming unstable.

The Awareness Layer, discussed in Section IV, is responsible for investigating *suspicious* readings reported

from the sonar array. While the robot is performing its allotted tasks, a 'health check' loop is performed to assess the data reported by the sonar sensors; for example, Sonar 4 in the array is reporting value of 415, Sonar 5 a value of 245 and Sonar 6 a value of 417; then Sonar 5 may need checking, as its value is considerably lower than Sonar 4 and Sonar 6. However, Sonar 5 could be detecting an object and reporting a correct reading; this can be verified by using the adjacent sensors to check the reading is valid.

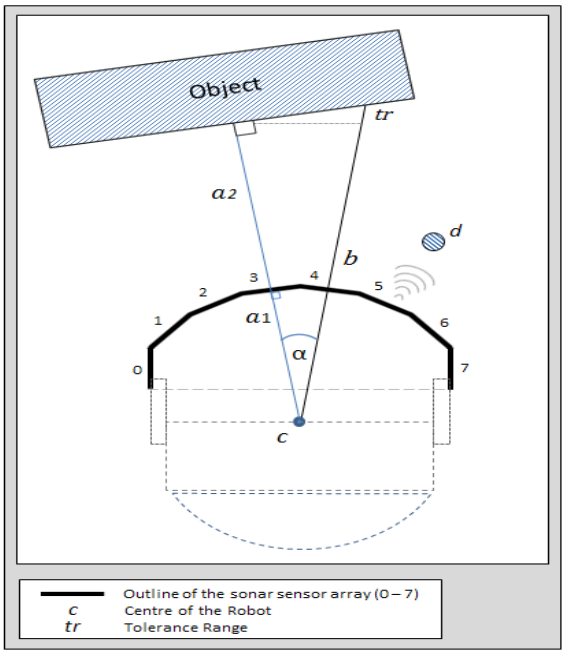

Figure 7. Shows how the Sonar sensor *tolerance range* is calculated. This calculation is used when comparing the distances between neighboring sonar sensors and an object – show as (***d***).

When comparing the values of neighboring sonar sensors, we need to take into account the location of the sonar sensors on the P3-DX robot. The forward facing sensors on the sonar array (1-6), are arranged as part of an octadecagon design. Therefore, if a particular sonar sensor has detected an object *square-on*, then the neighboring sensor can also detect this object but at *extended* distance value. Figure 7 shows how the *difference* value between two sonar sensors looking at the same object, is calculated. The value is described as the *tolerance range*.

$$b = \left(\frac{a1+a2}{Cos(\alpha)}\right) \quad (1)$$

$$tr = (b - a1) - (a2) \quad (2)$$

The *tolerance range* value can now be applied to Algorithm 1 (Table II), were all the sonar sensors are checked for any unusual values. In Figure 7, Sonar *five* has detected object (***d***), we therefore can use Sonar *four* and Sonar *six* to check the *distance* value reported by Sonar *five* is indeed correct. The *Highlight Disparate Readings* algorithm can identify what readings are significantly different from their immediate neighbors – see Table II (Algorithm 1). The *Highlight Disparate Readings* algorithm is contained in the Awareness Layer. If a sonar sensor requires checking, then this information is passed to the Analysis Layer for processing (see Table III).

TABLE II. (ALGORITHM 1) - HIGHLIGHT DISPARATE READINGS

```
1:   sr[] = sonarReadings[ ]  (readings from  P3-DX  sonars 1-6)
2:   tr = tolerance range
3:   sonarCheck[ ][ ]  (write sonar position and distance reading)
4:   x = 0
5:   sn = 1
6:   for (sn < number of  sonars) do
7:    if ( sn is == 1) then
8:     differenceValue = (sr[sn+1] – (sr[sn ]
9:    else if (sn is == 6)
10:    differenceValue = (sr[sn-1] – (sr[sn ]
11:   else (sn is > 1 &&  sn is < 6)
12:    adjacentDiffValue = (sr[sn-1] – (sr[sn + 1])
13:     if (adjacentDiffValue is < tr) then
14:      differenceValue = (sr[sn+1] – sr[sn]
15:     end if
16:   end if
17:   Check the differenceValue  a greater than  the tolerance  range
18:   If it is greater, then  that sonar will need checking
19:   if (differenceValue  >  tr)
20:     sonarCheck[x][0]= sn
21:    sonarCheck[x][1]= sr[sn]
22:   end if
23:   x = x + 1
24:  end for
25:  return (sonarCheck)
```

If during the *Highlight Disparate Readings* process, a sonar sensor is identified for checking i.e. *sonarCheck[x]*, the *Checking Sonar Readings* algorithm is then deployed. The *Checking Sonar Readings* is contained in the Analysis Layer and will issue *commands* to the robot to use the neighboring sonar sensors adjacent to '*sonarCheck[x]*' to check if the reading reported by *sonarCheck[x]* was indeed correct.

TABLE III. (ALGORITHM 2) – CHECK SONAR READINGS

```
1:   sonarCheck[ ][ ] contains the sonar position and readings
2:   sr[] = sonarReadings[ ]  (readings from  P3-DX  sonars 1-6)
3:   sc = 0
4:   col = 0
5:   ra = 20°
6:   for (sc < number of  sonarCheck rows) do
7:      if ( sonarCheck[sc][col] == 1) then
8:        RotateRobot(-ra)
9:        checkReading = sr[sc+1]
10:     end if
11:     if ( sonarCheck[sc][col]== 6) then
12:       RotateRobot(ra)
13:       checkReading = sr[sc-1]
14:     end if
15:     if ( sonarCheck[sc][col] > 1 && sonarCheck[sc][col] < 6 ) then
16:       RotateRobot(-ra)
17:       SonarReadingA = sr[sc-1]
18:       RotateRobot(ra)
19:       SonarReadingB = sr[sc-1]
20:       checkReading = SonarReadingA + SonarReadingB / 2
21:     end if
22:     diffValue = (checkingReading – sonarCheck[sc][col +1]
23:     if (diffValue  >  tr) then
```

```
24:         sonarCheck[sc][col] = disabled
24:       end if
26:     sc = sc + 1
27      end for
28:     return (disabled Sonars)
```

The process performed by Algorithm 2 (Table III), involves using two neighboring sonar sensors to be rotated to the original position of '*sonarCheck[x]'*. If the readings reported by both sonar sensors are different from '*sonarCheck[x]'*, then '*sonarCheck[x]'* will be tagged as being *disabled*. If '*sonarCheck[x]'* is at position *one* or *six*, then it will have only one neighboring sonar sensor available for checking. If the reading reported by this *one* sonar is different from '*sonarCheck[x]'*, then '*sonarCheck[x]'* will be tagged as being *disabled*. All sonar sensors tagged as being *disabled* will be handled in section IX.

3. *Frontal sonar sensor evaluation (IsMajor)*

If a sonar sensor becomes disabled, it returns a value of '5000' as a default to the sonar reader. Sonars sensors can also become *disabled* if, when processed through Algorithm 2, their *distance* readings proved to be unreliable. To emulate sonar sensor failure, rubber caps can be placed over the transducer to disable it. If the robot loses 50 percent of the sonar sensors, it can be completely blind on one side. For the robot to detect an object on its now 'blind' side, it will have to rotate on its center and use those remaining sonar sensors to locate the object. The amount of rotation required depends on the number of sonar sensors that have become disabled and the location of each sensor on the sonar array (see Figure 5). We use the Awareness Layer to establish if there is a problem with one or more of the sonar sensors. If a reading of '5000' is reported by one or more of the sonar sensors, the P3-DX is issued a STOP command; likewise, if the *bumper* sensor on the P3-DX robot is also triggered, this action will also issue a STOP command. The Analysis Layer is then notified that there is a fault with one or more of the sonar sensors. Analysis is then performed to establish what sonars are disabled and their position on the sonar array. Each sonar position on the sonar array also carries an *angle* value relative to the center of the P3-DX (See Figure 5). This information is then passed to the Adjustment Layer, so that calculations can be performed to establish how the remaining *enabled* sonars sensors can be used to compensate for the *disabled* sonar sensors.

IX. SONAR SENSOR FAULT COMPENSATION

1. *Frontal sonar sensor compensation (handling disabled sonar sensors)*

The *compensation policy* deals with faults for the six forward facing sonar sensors. The two *side* sonar sensors zero and seven (see Figure 5), are not required for this demonstration.

Compensation for the faulty sonar sensor will require a deliberate 'stop' and 'rotate' strategy. The fully working sonar sensors will need to be rotated to a position where they can replace the faulty sonar sensor(s). The more sonar sensors that are lost, then the more rotation commands by the robot are required to locate an object. Using the six sonar sensors in an array, there can be sixty-four possible combinations using binary notation. Combination '1' = 000000, all sonars are working correctly (no action required) and combination '64' = 111111, all sonars sensors are disabled (the robot has no ability to detect an object); this leaves sixty-two possible fault combinations. Table IV shows an example of how much the robot needs to rotate (clockwise or anti-clockwise) in-order to compensate for the loss of some sonar sensors. A single sonar fault will only require *one* rotation of the robot whereas a loss of *three* or more sonar sensors could require the robot to rotate at three different stages. It must be noted, that if for example, the robot is required to rotate +20 degrees to compensate for a disabled sonar sensor: after the compensation *reading* has been checked, the robot will be rotated back to its original position. This guarantees that the robot is always pointing to its original heading angle.

TABLE IV. FRONTAL SONAR SENSOR FAULT

Table IV shows the amount or rotation (degrees) required by the P3-DX robot, in-order to locate an object using the available sonar sensors.			
Enabled sonar sensor position used to compensate	Angle of enabled sensor on the sonar array	Disabled Sonar Sensor position (and angle).	Rotation(s) required
Scenario 1 – the sonar sensor at position 3 has become disabled			
2	30°	3 (10°)	-20°
Scenario 2 – the sonar sensor at position 3 and 2 have become disabled			
4	-10°	3 (10°)	+20°
1	50°	2 (30°)	-20°
Scenario 3 – the sonar sensor at position 2, 4, 5 and 6 have become disabled			
1, 3	10°, 50°	2(30°), 4 (-10°)	-20°
3	10°	5 (-30°)	-40°
3	10°	6 (-50°)	-60°
Scenario 4 – the sonar sensor at position 1, 2, 3 have become disabled			
4, 5, 6	-10°, -30°, -50°	3 (10°), 2(30°), 1 (50°)	+60°

Figure 8 shows Scenario 4 (from Table IV), how the robot is rotated to compensate for the disabled sonar sensors.

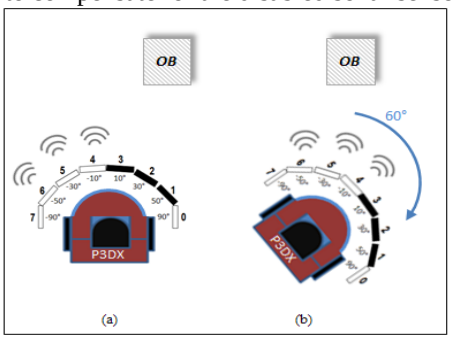

Figure 8. Shows sonar sensors (1-3) as disabled; they are 'blind' to object *OB*. (a). The *Compensation Policy* is used to establish that a 60° clock-wise rotation, can allow the P3-DX robot to detect object *OB*.

2. *Frontal sonar sensor compensation (algorithm)*

When *disabled* sonar sensors are first discovered, the P3-DX robot is stopped and analysis takes places to evaluate the extent of the fault. Table IV showed examples of what

rotation commands are required for various sonar fault scenarios. The sixty-two possible sonar sensor fault combinations will require different robot *rotation* calculations, so that the P3-DX robot can utilize the remaining enabled sonar sensors to compensate for the disabled sonar sensors. Algorithm 3 (Table V), shows the rotation *angles* are calculated for any of the sixty-two possible sonar sensors fault scenarios.

TABLE V. ALGORITHM 3 - COMPENSATION FOR DISABLED SONAR SENSORS

```
1:   sonarArray[6]  enabled/disabled sonar sensor positions
2:   disabledArray[ ]  disabled sonar angle position values
3:   enabledArray[ ]  enabled sonar angle position values
4:    lsa  = -50°  lowest sonar sensor angle
5:    hsa  = 50°  highest sonar sensor angle
6:    ia  = 20°   incremental angle
7:    av = 0      angle value for each  sonar sensor
8:    Calculate the array angle position for enabled/disabled sensors
9:    i  =  0
10:  for (av = lsa ; av < hsa + 1;  av = av + ia) do
11:     if ( sonarArray[i] == disabled) then
12:        disabledArray[i]= av
13:     end if
14:     if ( sonarArray[i] == enabled) then
15:        enabledArray[i]= av
16:     end if
17:     i = i + 1
18:  end for
19:  Combine disabledArray[] and  enabledArray[] values to establish
20:  the difference value required for an enabled sonar array to take
21:  the place of a disabled sonar array
22:  combinationArray[ ]  combined disabled/enabled  array values
23:  ii  =  0    inner index
24:  oi = 0     outer index
25:  av = 0     reset angle value
26:  for (da < number in disabledArray) do
27:    for (av = ia ; av < hsa + 1;  av = av + ia) do
28:      if ( enabledArray[ii] == (disabledArray[oi] + (-av))) then
29:         combinationArray[ii] = av
30:      end if
31:      if ( enabledArray[ii] == (disabledArray[oi] +  (av))) then
32:         combinationArray[ii] =- av
33:      end if
34:      ii = ii + 1
35:     end for
36     oi = oi + 1
37:  end for
38:  Sort the CombinationArray[] according to the values closest to
39:  Zero (The Robot centre line 0°). This ensures the robot will rotate
40:  the minimum of times in-order to compensate for the disabled
41:  sonar sensors. Store the results in the calcArray[]
42:  calcArray[ ]  sorted angle values needed for compensation
43:  for (ca < number in combinationArray) do
44:    var nearest = ca.OrderBy (x => math.abs(long) x-0)).First()
45:    Remove the nearest value found from the combinationArray[]
46:    combinationArray.RemoveAll(item => item == nearest
47:    calcArray.Add(nearest)
48:  end for
49:  Use the calcArray[] to work out  the rotationCommand values
50:  foreach(int calc in calcArray) do
51:    ii = 0 inner index
52:    for(ea < number in enabledArray) do
53:      if(disabledArray.Contains (ea[ii] + calc) then
54:        disabledArray.Remove(ea[ii] + calc)
55:        rotationCommand.Add = calc
56:      end if
57:      i = i + 1
58:  end for
```

When Algorithm 3 has been executed, it will return the *rotation* values required to compensate for the fault (depending on the number of sonar sensors that are disabled).

3. *Frontal sonar sensor compensation (rotation patterns)*

Figure 9 shows a chart plotting the number of robot rotations required for a particular sonar sensors fault scenarios. Figure 9 shows thirty-one sonar sensor *fault* combinations (alternate combinations from the sixty-two possible sonar fault combinations on the P3-DX robot sonar array). The position of the disabled sonars sensors on the robots sonar array can result in different *rotation* requirements. For example, in Figure 9, scenario 11 has three disable sonar sensors and scenario 13 has also three disabled sonars; however, it only requires one robot rotation to compensate for scenario 11, whereas it takes two rotations to compensate for scenario 13.

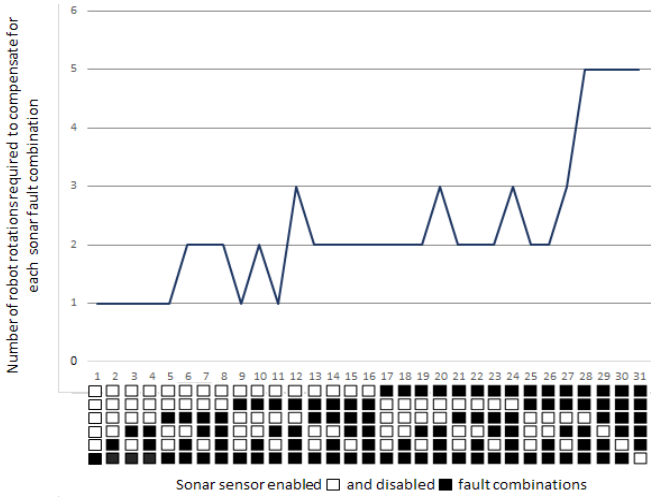

Figure 9. Shows how the increased number of disabled sonars sensors will also result in a increased in robot rotations to compensate for the fault.

Discovery of a sonar sensor fault causes the robot to STOP and triggers an evaluation process to establish the extent of the fault. When the fault has been analyzed and the *compensation* policy has calculated the robot *rotation(s)* (see Algorithm 3) required, the robot can continue its allocated task. However, because the P3-DX robot is in a *failure* mode, the robot is stopped at pre-defined intervals to check if there any obstacles in its path. Figure 10 shows that the P3-DX robot is stopped every 200mm intervals; this is to ensure the robot does not strike an object while in *sonar failure* mode. When the robot is stopped, it then rotates on its axis according to the *rotation* instructions established from using the *compensation* algorithm (see Algorithm 3, Table V). The robot will only declare an object has been detected, if that object is within a certain *threshold distance*. If, after a robot *rotation* has been executed and an object detected, then the robot can apply its Obstacle Avoidance *policy*. The robot will have to maintain the sonar sensor fault *compensation policy* for the remainder

of its task while the sonar sensors are reporting a fault condition. Extensive sonar sensors faults will result in multiple rotations by the robot at each STOP *interval* and consequently result in the task taking a longer time to complete. On a shorter journey this may not present any issues but if the robot is executing a task involving a long distance, then this could have an impact on resources like power consumption.

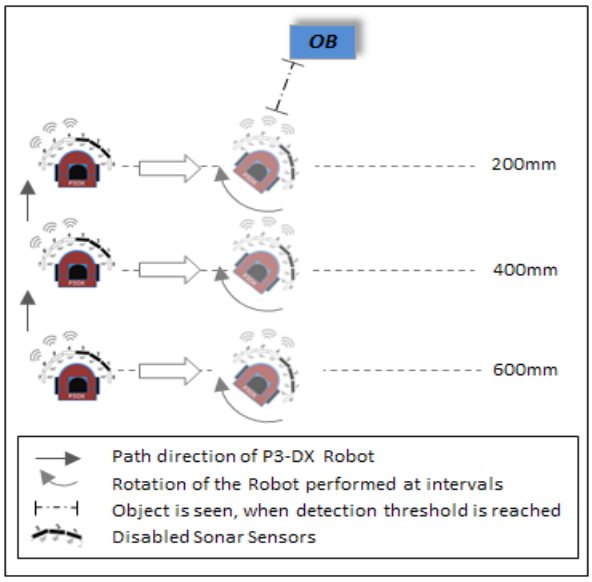

Figure 10. When the robot has entered *sonar failure* mode; the robot is stopped and rotated at specific intervals during its task; an object **OB** can be discovered during a *rotation* event.

X. CONCLUSION AND FUTURE WORK

The purpose of this research paper was to apply autonomic principles to the problem of managing sonar sensor hardware failures. In our approach, we extended the current autonomic MAPE architecture by introducing the AAA-3 layered architecture. This approach gave us the ability to detect sonar sensor faults, process the extent of the fault and finally make the necessary adjustments to allow the P3-DX robot to detect objects, even with reduced functionality. However, our experiments showed that as the number of disabled sonar sensors increased, then the time for the robot to complete its task greatly increased. Recording the *journey time* and *power* usage, was not part of this research paper but they would have to be seriously considered if the experiment was extended for *real-time* tasks.

An important lesson learned during this research is that hardware failure cannot always be observed by the User, especially those in sub-systems [9], as we found in sonar sensors that reported inconsistent data.

In the future, we would like to adapt our Sonar Sensor framework architecture to other mobile robot sensors, including laser and stereo cameras. In the past we have experimented with mobile robot wheel faults [5]. Our main goal is to develop a autonomic generic framework that can handle varying types of sensor and effector faults.

REFERENCES

[1] D. P. Miller, T. Hunt, M. Roman, S. Swindell, L.Tan and A. Winterholler,"Experiments With a Long-Range Planetary Rover," University of Oklahoma Norman, OK, 73019 USA

[2] D. M. Chess, A. Segal, I. Whalley, and S. R. White, "An architectural blueprint for autonomic computing," IBM Corporation, 2004.

[3] T. Huntsberger,"Fault Tolerant Action Selection for Planetary Rover Control," University of South Carolina, Columbia, SC 29208, USA

[4] T. Kohler, E. Berghofer, "Sensor Fault Detection and Compensation in Lunar/Plantary Robot Missions," University of Bremen, 28359, Germany

[5] M. Doran, R. Sterritt, G. Wilkie, "Self-Adaptive Wheel Alignment For Mobile Robots," IARIA Conference, Rome, 2016

[6] Adept Mobile Robots. Pioneer 3 Operations Manual, Version 6, 2010.

[7] Microsoft. Microsoft Robotics Developer Studio. [Online]. Available from: http://www.microsoft.com/robotics/ [Accessed 10 September 2016]

[8] N. K. Melchior and W. D. Smart, "Autonomic Systems for Mobile Robots." Department of Computer Science and Engineering, Washington University, MO, 63130 USA

[9] D. Crestani, K. Godary-Dejean, "Fault Tolerance in Control Architectures for Mobile Robots: Fantasy or Reality?," Laboratoire Informatique Robotique Microélectronique de Montpellier Université Montpellier Sud de France

[10] M. K. Habib, "Real Time Mapping and Dynamic Navigation for Mobile Robots," International Journal of Advanced Robotic Systems, Vol. 4, No. 3 (2007) ISSN 1729-8806, pp. 323-338

[11] Sensor failure detection through introspection. [Online]. Available from: http://hdl.handle.net/10945/3518 [Accessed 3 September 2016]

[12] E. Matson, S DeLoach, "Enabling Intra-Robotic Capabilities Adaptation Using An Organization-Based Multiagent System," IEEE International Conference on Robotics and Automation (IEEE ICRA 04) on, May 2004, pp 2135-2140.

[13] O. Zweigle, B. Keil, M. Wittlinger, K. Haussermann and P. Levi, "Recognizing Hardware Faults on Mobile Robots Using Situation Analysis Techniques," International Conference IAS-12 on, June 2012, pp 397-409

[14] E. Khalastchi, M. Kalech, L. Rokach, Y Shicel and G. Bodek, "Sensor Fault Detection and Diagnosis for Autonomous Systems," 22nd International Workshop on Principles of Diagnosis, October, 2011

[15] Y. Dai, Y. Xiang and G. Zhang, "Self-healing and Hybrid Diagnosis in Cloud Computing," DBLP Conference: Cloud Computing, First International Conference, CloudCom, December, 2009, pp. 45 – 56

[16] M. Parashar and S. Hariri, "Autonomic Computing: an Overview," Proceedings of the 2004 international conference on Unconventional Programming Paradigms, September 2004, pp. 257 – 269

[17] M. Scheutz and J. Kramer, "Reflection and Reasoning Mechanisms for Failure Detection and Recovery in a Distributed Robotic Architecture for Complex Robots," in Robotics and Automation, 2007 IEEE International Conference on, April 2007, pp. 3699-3704.

[18] P. Arcaini, E Riccobene and P Scandurra, "Modeling and Analyzing MAPE-K Feedback Loops for Self-adaptation," Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, June 2015, pp. 13 – 23

[19] Computerworld. IBM Adds Autonomic Tools to Speed Up Error Detection. [Online] Available from: http://www.computerworld.com/article/2557731/networking/ibm-adds-autonomic-tools-to-speed-up-error-detection.html [Accessed 27 September 2016]