Autonomic Self-Adaptive Robot Wheel Alignment

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*—The Autonomic Computing paradigm was first presented almost 15 years ago as a 20-30 year long research agenda. Organizations like NASA have explored the possibilities of an autonomic system along with the biologically inspired Swarm and Agents approaches. These Systems are unfolding circumstances rather than preconceived scenarios. This paper focuses on the aspect of the Autonomic System, were, adaptive self-optimization are explored using an intelligent machine model layers such as Reaction, Routine and Reflection. In the experimentation, a Pioneer P3-DX Robot is used to simulate a Planetary Rover. The Pioneer Robot has been the subject of hardware faulting – in the case Wheel Integrity. Using the Microsoft Robotics Developer Studio (MRDS) and Microsoft SQL Server, a framework has been developed, that records the Pioneer Robot's sensor data before and after hardware faulting. The Autonomic system elements, such as self-adaptive and self-optimizing are used to process the data. The Reaction, Routine and Reflection responses are determined at the appropriate response rates from the evaluation of these and form part of the self-healing, self-configuring, self-optimizing and self-protecting strategy to enable the Robot to continue to function, even with hardware issues such as a faulty wheel.**

Keywords—autonomic, self-adaptive, self-optimizing

I. INTRODUCTION

The future of space exploration will most certainly involve the deployment of vehicles such as planetary Rovers. Although Rovers like Curiosity and Opportunity have been successful, they have a limit in mobility and surface coverage. NASA in the past had researched the idea of Swarm Agents, such as multiple Rover deployments [1]. Recently, NASA have begun testing with "Swarmies" [2], were multiple Rovers can be deployed to gather resources on Moons or on Planets. Other scientific research has put forward the idea of "honeybee search strategy" [3], were large quantities of robots can been deployed to gather important data in special areas of interest. Fundamentally, the design of each individual Rover would be relatively basic to keep costs at a minimum. If each Rover is functioning under an Autonomic Management System, then the faults that occurred would be managed to the extent that the Rover could still function in its mission goals. Current NASA missions have reported hardware faults. The Curiosity Mission reported faults on all six wheels on the Rover; were the rubber casing on each wheel had been punctured through by sharp rock material [4]. Consequently, Mission Control was forced to plan alternate routes to avoid certain types of rocky outcrops. This ultimately has an impact on mission time and mission objectives. Could the wheel issue with Curiosity been identified earlier if the Rover had an Autonomic Self-protecting System [5] onboard? This paper investigates the implementation an Autonomic System and how it can deal with hardware failures.

Using a mobile robot to simulate planetary Rovers, we exposed the robot to hardware failure such as a wheel fault. At first, the robot is tested using two wheels in perfect order. The robot is given a task to travel a given distance, from a start point and then onto an end point. Each journey is recorded in a SQL database, using the robot sensor data. The robot is then fitted with one slightly damaged wheel. The robot is then given the same task as previous explained, with the results recorded in a SQL database. This paper focuses on implementing an Autonomic System to monitor and analyze the data, plan any necessary changes and execute those changes [6].

The structure of this paper is as follows. Section II documents related work in autonomic detection of sensor faults. Section III documents the autonomic system architecture. Section IV documents the autonomic management system and framework architecture. Section V documents the robotic hardware used in the research. Section VI documents wheel damage scenarios. Section VII documents autonomic failure detection in robotic mobile hardware. Section VIII documents the software framework and state machine used to create tasks including robot motion, laser readings and database recording. Section IX documents processing of the data collected from the tasks performed by the robot. Section X documents the equation used to compensate for the robot wheel alignment error. Section XI concludes the paper and outlines future work.

II. RELATED WORK

Since the introduction of autonomic computing [6], there have been a number of approaches on the subject of hardware fault-detection in mobile robots. Loss of sensor data is a typical example. The software framework in a mobile robot can be setup to ask for sensor information regardless of the sensor device that supplies it. If an *interface* program requires an object-detection *service*, then the "distance" value can be acquired from say, a laser range-finding sensor; however, if the laser sensor would fail, then the *service* can switch to an ultra-sonar sensor to obtain the same "distance value" [20].

The study of the types of failure that occur in mobile robots, libraries or classes of autonomic properties could be

 ISBN: 978-1-61208-463-3

developed that address each type of failure. Services can be provided that correspond to the type of failure found, such as service oriented architecture or a multi-agent system [21].

Further work can be seen in the use of Distributed Integrated Affect Reflection Cognition Architecture (DIARC). DIARC is knowledge-based architecture that can employ human-robot interactions without any structural modifications. DIARC employs MAS (multiple agent systems), which allows the distribution of components over multiple hosts and thus providing support for the autonomic detection of component faults and for subsequent error recovery [22].

Other fields of work such as those found in Organic Computing (OC), have experimented with similar attributes found in autonomic computing such as *self-adapting* and *self-healing*. Experiments using *hexapod* robots show that even with the amputation of one of the legs, the robot can still function by re-configuring the neighboring legs to compensate for the missing leg. [24].

The Related Work contributions all involve the detection of hardware failure using autonomic principles. However, they make no argument for trying to compensate for the error detected by employing specialized algorithms that will make use of the affected hardware, even if the *sensor* or *effector* operational ability is greatly reduced.

III. AUTONOMIC SYSTEM

The autonomic system can be summarised by four objectives: self-configuring, self-healing, self-optimizing and self-protecting; additionally, with four attributes: self-awareness, self-situated, self-monitoring and self-adjusting [7]. Self-configuring has the ability to automatically make adjustments when faced with changing circumstances. Self-healing is concerned with dealing with unexpected faults. It can recover from these faults and where possible, repair the faults. Self-optimizing has the knowledge of expected performance values. It can use policies to maintain optimum performance but also flexible to employ new policies to enhance performance. Self-protecting can deal with external attacks. It can establish what could potentially be a threat and how to deal with those threats. The autonomic manager describes the Monitor Analyse Plan Execute (MAPE) loop. This 'loop' is connected to a *knowledge* block [6]. This connection indicates that knowledge is used throughout the autonomic manager.

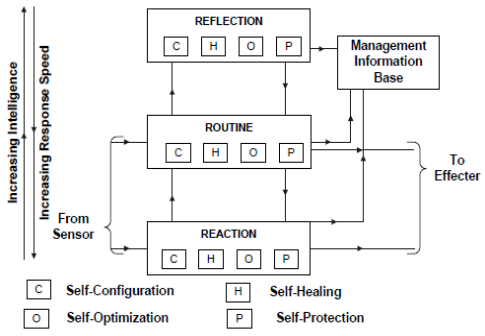


Figure 1. IMD Architecture.

The Intelligent Machine Design (IMD) architecture provides the distinct layers (see Figure 1): Reaction, Routine and Reflection [8].

The Reaction Layer is connected to effectors and sensors. This is the lowest level were no learning occurs. The Routine Layer will handle know situations. Input is received from both the Reaction and Reflection layers. The Reflection Layer makes decisions based on knowledge collected over a period of time. Operations at this level are executed based on experiences, current behavior and current environment. The lowest intelligence is found in the Reaction Layer, whereas important decision making (higher intelligence), is found in the Reflection Layer.

IV. AUTONOMIC ANALYSIS

In this section, we look at how the combination of the MAPE-K Loop [6] and the IMD Reflection [8], can be used to design an Autonomic Management System (see Figure 2).

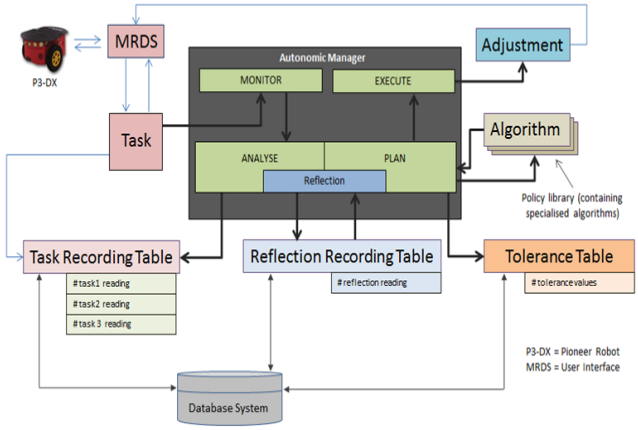


Figure 2. Autonomic Management System.

Figure 2 shows how the Pioneer P3-DX robot is controlled using the MRDS interface [15] (detailed in Section VIII). To emulate a robot mission, the P3-DX robot is set a number of tasks. The data relating to these tasks is recorded into a Database Management System. The tasks are closely monitored by the *MAPE-k* Monitor component; monitoring information can then be passed onto the Analyze function. The *MAPE-k* Analyze component can then use the data supplied by the *Reflection Layer* to evaluate if there is any anomalous behavior in the P3-DX *task* data. The *MAPE-k* Plan component will then use the data supplied by the *Reflection Layer*, to decide what *algorithmic policy* is required to compensate for the anomaly discovered. The *MAPE-k* Execute component will initiate the *policy* into the program *loop* - these *adjustment* values are then passed onto the MRDS to instruct the P3-DX robot command functions.

The *knowledge* contained into the database system, forms an integral part of the autonomic management system. The *Reflection System* maintains a model of self-representation. Reflection enables inspection and adjustment of a system at

run-time [19]. The P3-DX robot tasks history is recorded into database *tables*; analyzing this history, allows the autonomic management system to make *self-adjustments* based on hardware sensor performance of the robot.

Solutions for traditional fault tolerance systems are usually designed and configured at design time. The Developer is tasked with identifying in advance the most critical components and then, decides what strategies to use to overcome possible faults [18]. Autonomic Systems are designed to look for subtle changes in behavior or inconsistent performance data. The autonomic element has its own manager system. The managed system looks after the controller. The controller consists of two loops – the local loop and the global loop [17]. The global loop will run constantly, gathering data from sensors and storing this data in the database system. The global loop manages the behavior of the whole system. The local loop is *blind* to the overall system loop. The local loop will focus on analyzing data in the database and look for discrepancies. Discrepancies can be identified by comparing the data with known tolerance values. If the tolerances are above the limits that the local loop can maintain, then this can affect the performance of the overall system. This change in performance will trigger the global loop. The global loop will then implement a policy to deal, in this case, with the wheel alignment issue.

V. PIONEER P3-DX MOBILE ROBOT

The Pioneer P3-DX is a mobile robot with two independent drive wheels, plus an additional caster for stability (see Figure 3). The internal drive uses Proportional-Integrated Derivative (PID) system with a wheel encoder feedback to adjust a pulse-width-modulation (PWD) at the motor *drivers* to control the power of the motors [9]. The P3-DX is also fitted with a LMS 200 Laser. The LMS 200 is capable of measuring out to 80m over 180° arc. The sensor operates by shinning a laser of a rotating mirror. As the mirror spins, the laser scans 180°, effectively creating a fan of laser light (see Figure 4). Any object that breaks this fan reflects laser light back to the sensor. The distance is calculated based on how long the laser takes to bounce back to the sensor [10].

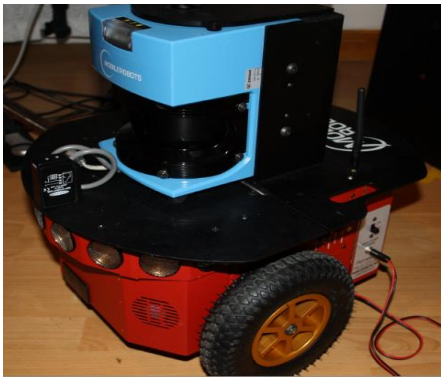


Figure 3. Pioneer P3DX mobile robot fitted with LMS 200 laser.

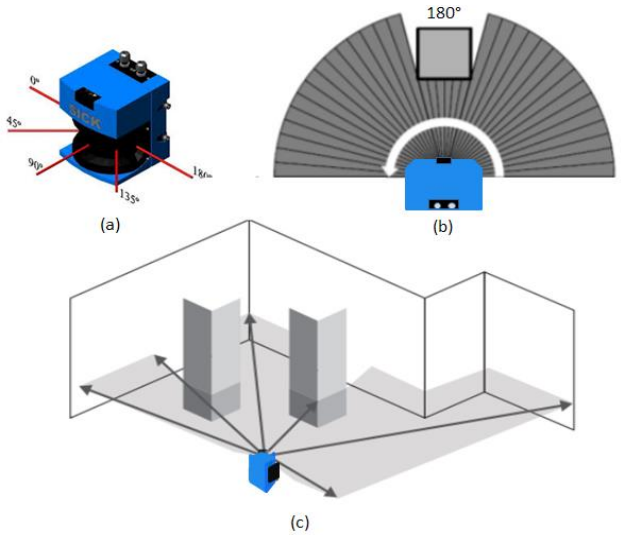


Figure 4. (a) The LMS 200 laser has a 0° to 180° field of view. (b) The laser creates a fan of laser light that scans from right to left. (c) Objects are detected by breaking the laser fan projection. The distance is detected by the time it takes for the laser to bounce back to the sensor.

The Pioneer P3-DX robot is used to simulate a planetary rover. The robot has an on-board PC and a WIFI connection. The software framework used to control the robot is MRDS.

VI. WHEEL DAMAGE

One of the most important aspects of a planetary rover mission is its ability of movement on the surface. The rover is reliant on optimal wheel performance, in-order to reach locations according to the mission goals. Possible wheel damage to a rover on a mission is very difficult to anticipate. In our own everyday lives, wheel damage can occur in with our cars, motorbikes and pedal bikes – hitting objects on the road or potholes, causes the most damage. The possibility of damage to a wheel on a rover mission is very high, as we have witness with the Curiosity Mission (see Figure 5) [13].

Figure 5. Curiosity wheel damage – many cracks like this have been found on all six wheels of the rover [13].

NASA engineers are looking to provide a software update to Curiosity rover; the software could provide the ability to

match electrical current with wheel drive [14]. This type of damage can lead to issues such as wheel alignment. If the wheel can't drive in a straight line as expected, then there would need to be a *self-adjusting* process employed to allow the mission to continue.

VII. AUTONOMIC FAILURE DETECTION

Physical failures in a planetary rover can affect systems, such as effectors, sensors, power and communication [11]. In this paper, we investigated the failure in the effector systems centering on wheel degradation. The investigation was begun by fitting two wheels in perfect working order, onto the Pioneer P3-DX Robot. The robot is then given a series of tasks where it is required to travel from known start-point and then move to a known end-point. For tests purposes, the route that the robot is traveling along is parallel to the laboratory wall. The test consisted of the robot being positioned at the start-point exactly parallel to the wall at a given distance. The onboard LMS 200 laser is used to accurately record the distance from the robot to the wall. The robot is then given a command to move a given distance. When the robot arrived at the destination, the LMS 200 laser is used to record the distance from the robot to the wall. The results of the laser data were recorded into an SQL Server database. These tests were repeated multiple times to give us an accurate evaluation of the robot performance with two perfectly operational wheels. Standard Deviation equation is applied to the test results to give an indication of the accuracy of the robot's movement from a start-point to an end-point. The Standard Deviation results were then evaluated using an algorithm to represent the Autonomic Intelligent Machine model Reflection layer. This *reflection algorithm* processed the data to establish if the robot wheel alignment was accurate against expected tolerance values. This self-monitoring of the robot over a given period of time, confirms that the robot is operating as expected. The Pioneer Robot is then fitted with a slightly damaged wheel. The robot is then evaluated using the same test process. The purpose of this action was to process the data within the *reflection algorithm*, and initiate the Autonomic *self-monitoring* to cause the system to identify there was a problem and consequently put in motion, policies that could repair the issue or at least compensate for the error. The evaluation done in the Reflection layer would subsequently initiate a policy within the Routine layer, which in turn would cause a physical implementation in the Reaction layer.

VIII. SOFTWARE FRAMEWORK

Software Development for this paper is carried using the MRDS framework. MRDS is a service-oriented programming model that allows the creation of asynchronous and state-driven applications [15] and [16]. Code development is carried out using C# language. Database work was completed using Microsoft SQL Server and User defined stored procedures.

To create robot tasks (explained in Section VII), it required implementation of an event driven and state-based behavior processing, using a state machine as shown in Figure 6. Between each state, transitions are added. These transitions are triggered by notifications received from partner services [12].

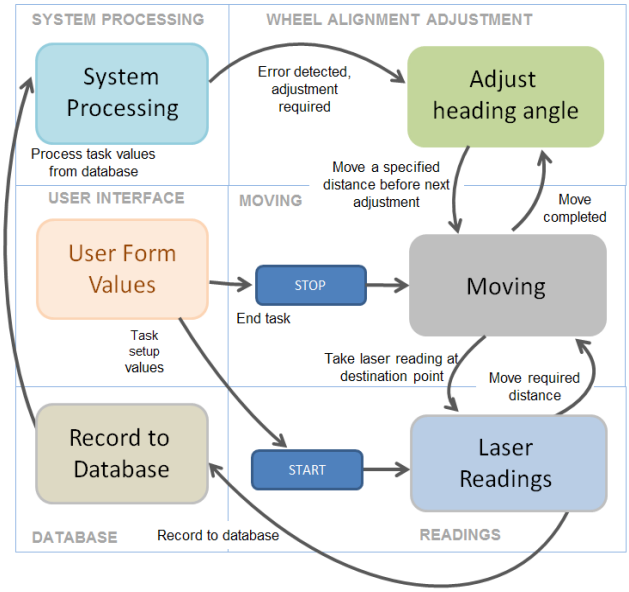


Figure 6. Shows the state machine used to create the robot tasks, process database information and make necessary adjustments if an error is detected.

The System Processing state is executed after the robot has completed a task; this *self-monitoring* attribute can then initiate a *self-adjusting* process if an error is detected.

IX. DATA EVALUATION

The graphs in Figure 7 shows the readings taken from the robot tasks (see Section VII). Graph (a) shows how the robot performs with both wheels fully functional. The robot stays within the limits of the *expected* path. Graph (b) shows how a damaged wheel causes the robot to veer off to the right.

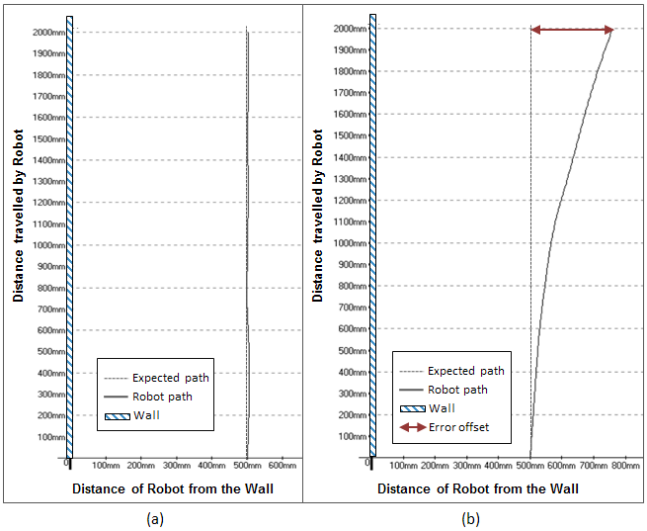


Figure 7. Graph (a) shows the path of the robot with both wheels at optimal performance. Graph (b) shows the path of the robot with one wheel in a damaged state.

Using Standard Deviation equation (non-grouped data), the results from two types of tests (explained in Section VII) were calculated.

$$\sigma = \sqrt{\frac{\sum(x-\bar{x})^2}{n}} \quad (1)$$

The first sets of tests are conducted using the Pioneer P3DX robot with two wheels in perfect condition – Test Scenario A. The second sets of tests are conducted using one wheel with slight damage – Test Scenario B. Table I shows the standard deviation value for Test Scenario A and how adding the results of Test Scenario B affect the standard deviation value.

TABLE I. TEST VALUES FOR WHEEL ALIGNMENT CALCULATIONS

Pioneer P3DX wheel alignment testing - the numbers represent the amount in millimeters (mm) that the robot was from its required destination point, after each task.										
Test Scenario A: for a Robot with two wheels in optimal condition										***SD***
2	9	-9	4	-5	-7	-22	-7	-6	-5	8.63
-8	-13	-13	11	-14	-4	-13	-2	-6	-6	
3	-21	-12	-10	4	-10	9	-11	-4	-21	
-5	10	-8	2	-7	-12	3	-5	-14	10	
6	-2	-7	4	-13	5	8	3	-4	7	
Adding another one test result to Scenario A, does not affect the SD value significantly										***SD***
									-5	8.54
Test Scenario B: for a Robot with one slightly damaged wheel, the overall SD changes significantly and thus would be flagged as a fault										***SD***
								35	49	12.02

Significant changes to the standard deviation (SD) value shown in Table I indicated that there was a problem with wheel alignment in the Pioneer P3-DX robot. The state machine System Processing discussed in section VIII was implemented to identify changes in SD values. Identifying the error is achieved by comparing the SD value to the set *tolerance* value. If the SD is within the tolerance value range, then no action is needed; if the SD is above the tolerance value, then an *error* task was executed. This then resulted in the robot initiating a *self-adjustment*, were an algorithm is used to determine the *degree* of the error and calculate the values needed to compensate for that error.

X. WHEEL ALIGNMENT ERROR EVALUATION

To compensate for the wheel alignment fault, an algorithm is required to work out the compensation value needed to correct the robot trajectory. The aim of this process is to keep the robot functioning even with a damaged wheel. As the robot moves, the damaged wheel is constantly pulling it away from its expected path. To correct the wheel alignment error, the robot needs to adjust its heading at calculated intervals during its journey. To achieve this, the robot would travel a certain distance, stop, then, turn itself back toward its expected path, then, move again. The downside of this particular strategy is that it will add significant distance and time to the robot mission. However, this is justified, in that the robot will arrive at its expected destination point rather than being significantly off-course.

Using the values from Test Scenario B, an *average* distance from which the robot was from its expected destination is calculated. Using Right-Angled Triangle equation, the angle between the *Hypotenuse* side and the *Opposite* is calculated, see Figure 8.

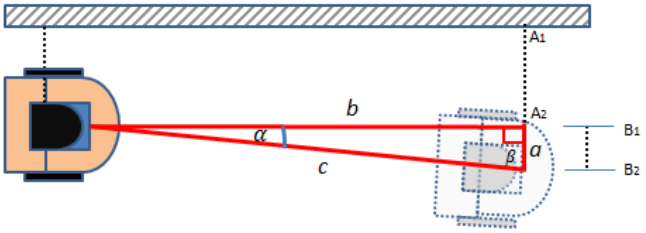
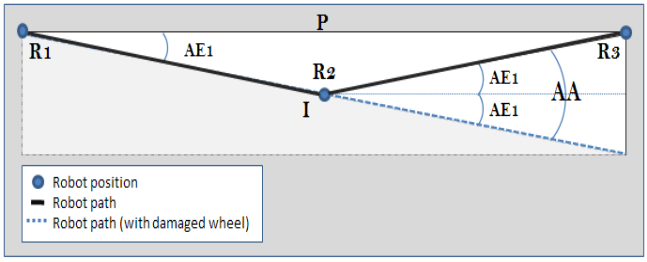


Figure 8. The Pioneer P3DX robot with a damaged wheel: this caused to the robot to slew to the right. A1 to A2 represents the expected distance the robot should be from the wall. B1 to B2 represents the average distance the robot was offset from the expected destination point.

$$\alpha = sin^{-1}\left(\frac{a}{c}\right) \quad (2)$$

The equation (2) is used to calculate the *angle* of the initial error-offset discovered from results in Figure 7 (b). This *angle error* value is then used to establish the *angle of turn* needed for the robot to make its heading adjustment; see Figure 9.



(a) Robot error correction using 1 interval

Figure 9. Represents how the *angle of turn* is calculated.

The *angle of turn* calculation is formulated using the values represented in Figure 9. The 'R', represents the robot position. When the robot reaches the 'I' position (*interval*), the robot is stopped. The robot *heading angle* is then adjusted and the robot continues its journey. The 'AE' represents the *angle* of the wheel alignment error calculated using equation (2). The 'AE' angle value is then *doubled*. The reasoning behind this is that two 'AE' values are required to bring the robot back to the expected path. The two 'AE' values are then divided by the number of *intervals* the robot is required to stop. The 'AA' represents the *angle*

of turn needed to allow the robot to re-establish the expected journey path marked as 'P'.

$$AA = \frac{2AE}{I} \quad (3)$$

From the values represented in Figure 9, we can derive the equation (3). The *interval* 'I' represents the number of times the robot with stop and adjust its heading angle. The more *intervals* the robot uses, the more accurate the robot will be in terms of keeping to the original journey path. In equation (4), *interval distance* is represented by 'ID' and *total distance* is represented by 'TD'. The *interval distance* is calculated as follows:

$$ID = \frac{TD}{I} \quad (4)$$

Figure 10 shows how the number of intervals used decreases the *error-offset* value. Table II shows a robot with wheel damage, driven over a fixed distance. The robot is stopped and adjusted according to the number of intervals applied. The offset value is the maximum distance the robot is from the expected path.

TABLE II. COMPARE OFFSET VALUES USING A GIVEN INTERVAL #

Pioneer P3-DX wheel alignment testing. Error offset decreases as the number of intervals increases. The maximum offset is measured from the expected path value.			
Distance of Journey	Number of Intervals	Angle of adjustment	Maximum offset error value
2000 mm	1	12°	44 mm
2000 mm	2	6°	26 mm

The graphs in Figure 10 show the comparison of the robot path when used with different *interval* values.

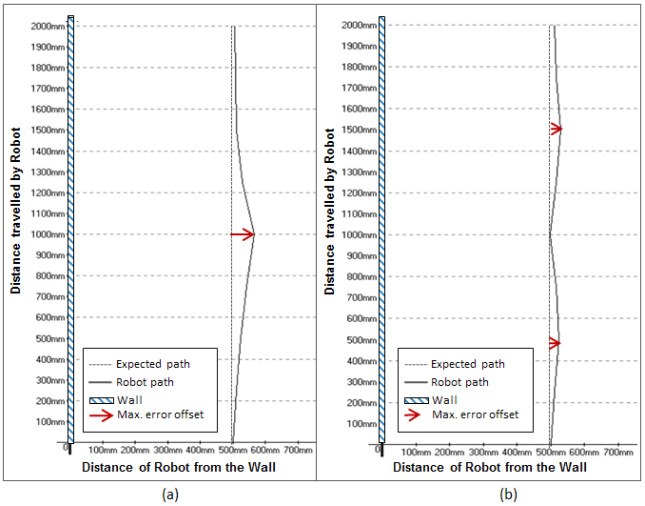


Figure 10. Using the compensation algorithum, the robot journey accuracy is increased when the number of intervals is also increased. (a) Robot journey uses one interval. (b) Robot journey uses two intervals.

This compensation method reflects the ability of the robot to *self-adjust* itself to arrive at the expected destination point even with a damaged wheel. Figure 11 shows how the robots journey is divided into *intervals*. This process is repeated until the robot reaches its destination point.

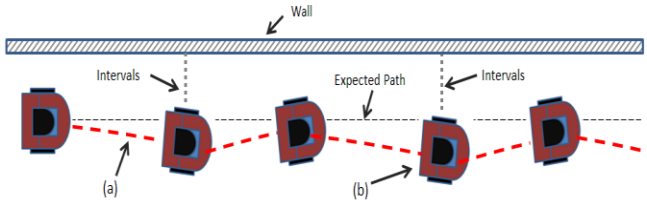


Figure 11. Shows the path of the Pioneer P3-DX robot when it implements the compensation algorithm. (a) Wheel damage causes the robot to 'slew' away from the expected path. (b) When the robot reaches an interval point, the robot is stopped and then turned on its axis at the required angle for adjustment..

The severity of the alignment error will have an effect on the 'actual' distance and journey time of the robot; as the *angle* of compensation increases, the journey time also increases. On a shorter journey, this may not be a factor but if the robot needs to travel for long distances, then could have an impact on resources like power consumption.

Algorithm 1 Wheel Alignment Compensation

```
1:  Read test data from SQL Tables.
2:  for (ts = number of tests) do
3:    for (vs = each value in test[ts]) do
4:      Calculate the average value (av) of each test vs[ts]
5:      Use SD equation to test if the av > tolerance range
6:    if ( tolerance range is exceeded == true)
7:       Alignment Error Angle (ae) = sinθ (Right-Angle) equation
8:       Angle of Adjustment (aa) = 2 * ae/number of intervals
9:    end if
10:   Enter the journey values for the Robot.
11:   dis = distance to travel
12:   ni =number of required intervals
13:   aa= alignment error angle
14:   cd = current Robot distance
15    iv (Interval value) = dis/ni
16:   iv = interval value
17:   while cd< dis do
18:   Adjust the Robot direction at interval setting (iv)
19:   if (dis % iv == 0)
20:      StopRobot()
21:      Rotate robot to new angle direction(aa)
22       RotateRobot(aa)
23:      MoveRobot()
24:   end if
25:   end while
```

Figure 12. Shows pesudo code for the Robot Wheel Alignment compensation.

Figure 12 shows a representation in pseudo code for the Robot wheel alignment data processing and compensation functions. The initial SQL data from the robot tasks is processed and checked against know tolerance values. If the tolerance values are exceeded, then the *compensation algorithm* is employed to re-establish the robot to its expected journey path.

 ISBN: 978-1-61208-463-3

TABLE III. TEST VALUES WITH WHEEL COMENSATION ALGORITHM

Pioneer P3-DX wheel alignment testing - the numbers represent the amount in millimeters (mm) that the robot is from its required destination point, after each task.										
Test Scenario C: for a Robot with damaged wheel and using compensation algorithm										*SD*
2	9	-9	4	-5	-7	-22	-7	-6	-5	8.63

Table III shows how the *compensation algorithm* restored the robots wheel alignment measurement (SD), within the expected tolerance values.

XI. CONCLUSION AND FUTURE WORK

The purpose of this research paper is to identify how the autonomic model can be applied to dealing with robot hardware issues, such as *wheel alignment*. If NASA decides to deploy *swarm* planetary rovers in the future, then autonomic systems will need to be seriously considered, to deal with possible hardware degradation and software system failures.

The Intelligent Machine Design theory proves that responses such as Reflection can over time, analyze data and predict possible issues at an earlier stage. This can be put to the test by carrying out more extensive testing in the future and building up a knowledge database. The *compensation algorithm* we applied in these test cases is only one of many ways which the robot wheel alignment issues could have been solved. In the future, we would like to investigate the use of *wheel velocity variation* for dealing with wheel alignment issues; were small increments of power can be applied to a damaged wheel. Unfortunately at present, such robots like the Pioneer P3-DX do not possess the *fine granularity* in wheel velocity, which is required for such an experiment.

REFERENCES

[1] R. Luna, A. Oyama, and K. Bekris, "Network-Guilded Multi-Robot Path Planning for Resource-Constrained Planetary Rovers," IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'10) on, September 2010, pp. 776-783.

[2] National Aeronautics and Space Administration. NASA: Meet The Swarmies-Robotics Answer To Bugs. [Online]. Available from: https://www.nasa.gov/content/meet-the-swarmies-robotics-answer-to-bugs/2016.02.02

[3] Á. Kisdi and A. R. L. Tatnall., "Future robotic exploration using honeybee search strategy: Example search for caves on Mars," Acta Astronautical , vol 68, issues 11-12, pp. 1790-1799, June-July 2011.

[4] National Aeronautics and Space Administration. NASA: How Wheel Damage Affects Mars Rover Curiosity's Mission. [Online]. Available from: http://www.space.com/26472-mars-rover-curiosity-wheel-damage.html/2016.02.02

[5] R. Sterritt and M. G. Hinchey, "Engineering ultimate self-protection in autonomic agents for space exploration missions, "Engineering of Computer-Based Systems," 2005. ECBS '05. 12th IEEE International Conference and Workshops on, April 2005, pp. 506-511.

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

[7] R. Sterritt and M. G. Hinchey, "Biologically-inspired concepts for self-management of complexity," Engineering of Complex Computer Systems, 2006. ICECCS 2006. 11th IEEE International Conference on , August 2016, pp. 65-70.

[8] H. Shualib, R. J. Anthony, and M. Pelc, "Framework for Certifying Autonomic Computing Systems," The Seventh international Conference on Autonomic and Autonomous Systems on, January 2011, pp. 122-127.

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

[10] J. Bedkowski, M. Kretkiewicz, and P. Maslowski, "3D Laser Range Finder based on rotated LMS SICK 200," unpublished.

[11] E. Matson and S. Deloach, "Integrating Robotic Sensor and Effector Capabilities with Multi-agent Organizations," Proceedings of the 2004 International Conference of Artifical Intelligence (IC-AI-04) on, June 2004, pp. 481-488

[12] J. S. Cepeda, L. Chaimowicz, and R. Soto, "Exploring Microsoft Robotics Studio as a Mechanism for Service-Oriented Robotics," (LARS) on, October 2010, pp. 7-12.

[13] NASA/JPL-Caltech/MSSS; Sol 962, April 21, 2015.

[14] National Aeronautics and Space Administration. NASA: Mars Rover Curiosity Dealing With Damage. [Online]. Available from: http://www.space.com/29844-mars-rover-curiosity-wheel-damage.html/ 2016/02/04

[15] Microsoft. Microsoft Robotics Developer Studio. [Online]. Available from: http://www.microsoft.com/robotics/2016/02/04

[16] K. Johns and T. Taylor, Professional Microsoft Developer Studio, Wiley Publishing Inc., 2008.

[17] M. Parashar and S. Hariri, "Autonomic Computing: An Overview," International Workshop UPP on, September 2004, pp. 257-269.

[18] R. Almeida, J. Briot, S. Aknine, Z. Guessoum, and O. Marin, "Towards Autonomic Fault-Tolerance Multi-Agent Systems," The 2nd Latin American Autonomic Computing Symposium (LAACS'2007), Petropolis, Rio De Jeniro, Brazil, September, 2007.

[19] P. Maes, "Computational Reflection," The Knowledge Engineering Review on, November 1988, pp.1-19.

[20] N. A. Melchior and W. D. Smart, "Autonomic systems for mobile robots," in Autonomic Computing, 2004. Proceedings. International Conference on, May 2004, pp. 280-281.

[21] C. Rouff, J. Rash, and W. Truszkowski, "Overcoming Robotic Failures through Autonomicity," in Engineering of Autonomic and Autonomous Systems, 2007. EASe '07. Fourth IEEE International Workshop on, March 2007, pp. 154-162.

[22] 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.