Autonomic Wheel Alignment 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*—In pursuit of future space exploration, NASA has described the concept of Multiple Rovers; this concept was based on the fact that multiple rovers are capable of completing more tasks and covering a larger area than a single rover. However, the amount of expenditure put into each rover will be greatly reduced compared to that of a single dedicated rover. These lower spec. rovers would be more vulnerable to hardware faults. However, if the software system built into each rover is based on autonomic principles, then the ability of the rover to continue to operate would be greatly increased. When studying the features of mobile robots using the X80-h from Dr. Robot, it was found that this type of Robot can suffer from a wheel alignment issue – were, when given a command to move forward by a given distance, it would either veer off to the left or to the right. This resulted in the Robot not arriving at the expected destination. The alignment 'error' was not always consistent for each Robot; it was found that after each attempt the Robot was at various distances from the expected destination point. To plot the path of the Robot at each attempt, Indoor GPS was used. The Indoor GPS provides x, y and degree of angle data which can then be recorded into a database. The goal is to use this data to create an algorithm to compensate for the alignment error; this will therefore improve the Robots reliability to reach its expected destination.**

Autonomic, Alignment, GPS, Dead-reckoning

I. INTRODUCTION

In 2003, NASA put forward the concept of Autonomous Nano-technology Swarm (ANTS). This involved using multiple spacecraft to complete space missions by a means of collaboration and redundancy [1]. This concept was taken further by including multiple robotic rovers for use in planet and moon exploration [2]. However, using multiple rovers would mean less expenditure on each individual rover; therefore, it would be assumed that the reliability and robustness of the rover would be greatly reduced [3]. If the rover develops a fault during the task, then depending on the nature of that fault, it could either delay the mission or perhaps damage the Rover permanently. However, if the rover is equipped with a *reflective* and *self-aware* software system [4], it may be able to limit the damage or provide an alternate strategy for completing the mission.

In [10], a navigation architecture for an Autonomous Exploration Rover (AERO) is described. This robot participated in the NASA Robot Centennial Challenge in 2013. The authors describe how AERO needed to use dead-reckoning for extended periods and how a fibre optic ring gyro was employed to assist with this task by providing acceleration and angular velocity information. Such gyroscopic equipment is an added expense, which could become considerable in the situation where a swarm of robots is to be engaged. Ideally, it would be good to achieve a more basic form of dead-reckoning - relying on calculations from direct angular and movement instructions communicated to the robot.

Robots that employ differential drive wheels such as the X80-h [11], are susceptible to hardware issues such as a *wheel alignment error*. This can be the result of many hours of use or perhaps a component fault or mal-adjustment. The problem causes the robot to slew to the right or left when it is commanded to move in a straight line. This problem clearly needs to be addressed if the basic form of dead reckoning is to have any chance of success.

Under laboratory conditions, it is easy to rectify the wheel alignment error by adjusting a Motor Calibration Board. However, if the Robot is inaccessible, then the onboard software system would need to check for the fault and provide a suitable work-around strategy where necessary. Detection of this problem would be provided by the Robot's self-monitoring system. While the Robot is in operation, it will be periodically checking each system for an 'I AM OK' response from its various sensors and visuals. Self-optimization [5] allows the Robot to analyze the data therefore checking that all systems are performing as expected. In the case of a *wheel alignment error*, the Robot would have to decide if it is safe to continue to operate; does the terrain allow for alignment compensation adjustments to be made, which would depend upon existing path obstacles, walls, cliff edges and slopes. The Robot would implement a basic strategy of self-preservation but at the same time evaluate the possibility of completing the mission using an alternative, compensated, path.

II. ARCHITECTURE OF SYSTEM

A. Autonomic Architecture

One of the fundamental aspects of autonomic computing technology is known as CHOP – configure, heal, optimize and protect [6]. Self-configuring: can dynamically adapt to changing environments. In the case of a Robotic device, this could be loading in a new component to cope with surface changes i.e. sandy surface to a rocky surface. Self-healing: can discover, diagnose and react to disruptions. If a component is not responding then performing a *restart* of that particular system may alleviate the problem. Self-

Optimizing: can monitor and allocate resources automatically. If one of the sensors fails on a Robot, the remaining sensors are re-calibrated to compensate for the loss. Self-protecting: can anticipate, detect and identify possible threats to the software or hardware systems; for example, if a component is over-heating, then the system containing that particular component can be shutdown thus preserving the remaining components.

The CHOP attributes can then be combined with an Autonomic Management system. This is known as MAPE – monitoring, analysing, planning and executing [6]. For example, in the situation were a Robotic device has a system virus, the autonomic manager would perform a monitor and analyse function and determine that there is a risk to the software system. Then, the autonomic manger uses the self-protecting mechanism to inform the self-configuration mechanism to perform an installation patch to rectify the issue [8].

The Intelligent Machine Design (IMD) architecture from an autonomic computing system is closely related to how biological systems work [7]. The architecture proposes three distinct layers; The Reaction layer, the Routine Layer and the Reflection Layer. The lower layer, the Reaction Layer, is connected to the sensors and effectors. In relation to the *Wheel Alignment* experiment, Fig.1., this would relate to the sensors on the X80 Robot such as the infrared, ultrasonic and camera, which would be used to spot path obstacles/hazards in real-time, as the robot progresses along a prescribed path.

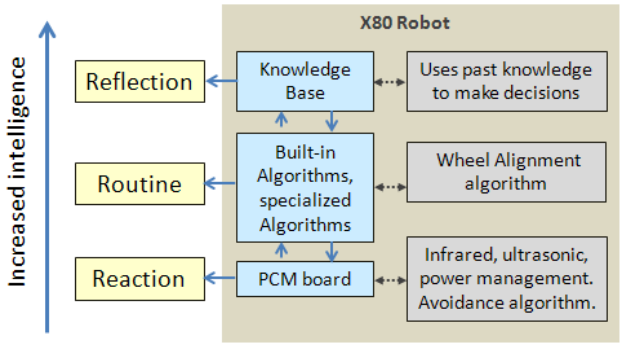

Figure 1. Autonomic Intelligent Machine Design for X80 Robot.

The Routine Layer will handle know situations either learned or hard-wired, such as following a compensated path calculated from known wheel alignment issues. The Reflection Layer makes decisions based on a knowledge base collected over a period of time. The Reflection Layer uses data to decide the best course of action. In the case of the *Wheel Alignment* error, the Reflection Layer can call on past experience (database store), to inform the Routine Layer what process to engage – in this case what *Wheel Alignment* algorithm to use, which might depend upon the nature of the terrain such as solid or loose or whether there is a incline or decline, all of which may require differing degrees of wheel alignment compensation when plotting a compensated path. The reflection layer would also be concerned with post-evaluating the performance on the finished task in order to decide if further refinement should be made to the known wheel alignment algorithms.

The learning process involved in developing or refining a *Wheel Alignment* algorithm, is described in Fig.2. This is part of the Reflection layer. The process is currently achieved indoors. It is currently partly manually achieved, but the intention is that this would be automated for eventual field use. The process consists of a GUI, which allows the User to plot a Robot journey and to display the Robot system data; this includes a set of coded routines, which send the specific commands to the Robot. The GUI interface is coded in .net C#. Communication to the Robot is carried out via a wireless router, which passes commands that directly control all the hardware modules inside the Robot, such as motion, power and cameras. Indoor GPS, is supplied by a module on the Robot, which uses passive Landmarks attached to the ceiling.

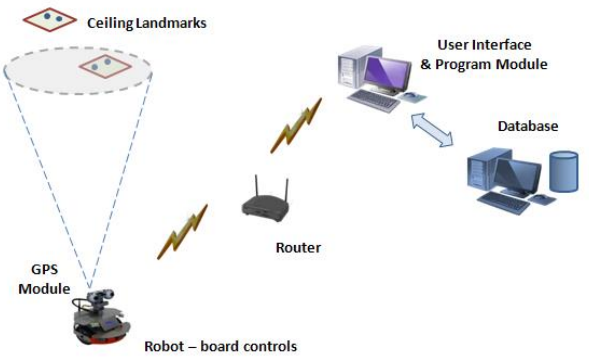

Figure 2. Architecture used in the Autonomic Wheel Alignment .

For this project, the *Wheel Alignment* algorithm is held on a PC and all recorded data is stored on a SQL Server database. The Robot commands are sent via the application software to the Robot using a WIFI network. Sensor data from the Robot is sent via the WIFI network back to the PC and processed by the *Wheel Alignment* algorithm. Fig.3. shows the workflows involved in processing the *Wheel Alignment*. Their interaction is as follows;

During the learning process, when the wheel alignment algorithm is being compiled, the robot is instructed to move in a straight line to a target position, a known distance from the starting point. The actual path the robot takes is recorded and the degree to which the robot deviates from the direct path is determined. This information is used in compiling the wheel alignment algorithm. It is envisaged that different algorithms will be required to handle surfaces with different frictional characteristics and also for surfaces presenting either an incline or a decline. Further research is required to determine how these parameters affect the wheel alignment algorithm.

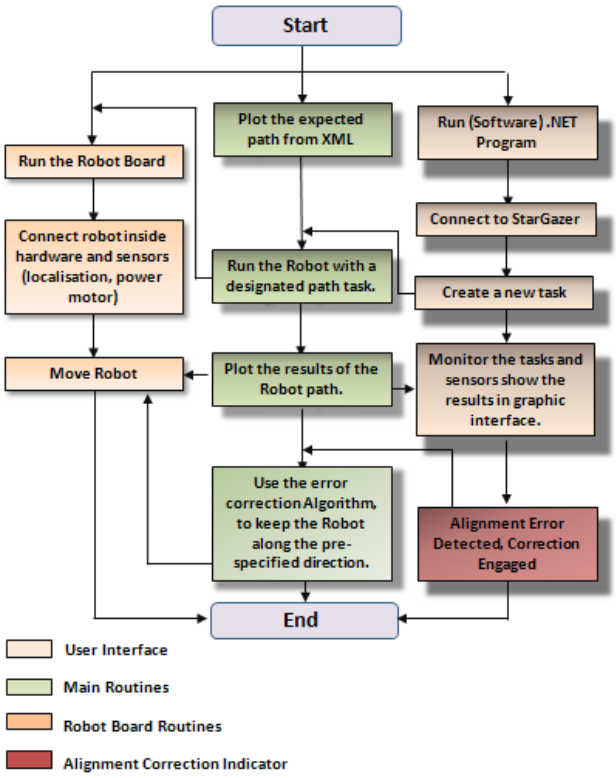

Figure 3. Process flow of the proposed wheel alignment program .

The User Interface provides a means of specifying direction and distance parameters for the Robot. A Wi-Fi connection is made to the Robot and GPS via the router. Once a new task is initiated, the main routines in the program will pass commands to the Robot to move forward using the supplied parameters. Once a task is completed, the Robot can analyze the data and check that all systems are performing correctly. If an alignment error is detected, then a programmed routine (the wheel alignment algorithm) is employed to re-align the Robot therefore increasing the destination accuracy. However, the result can be impacted by the physical environment such as obstacles, walls and cliff edges.

III. INDOOR LOCALIZATION

The indoor GPS used in this project, is the StarGazer, supplied by the Hagisonic Company, Korea. It comprises of an infrared camera (installed on each Robot) and a number of ceiling (passive) Landmarks. It can measure a series of images, which are reflected from the ceiling Landmarks unique reference IDs (see Fig.4.) The Landmarks are placed on the ceiling at 2 meters apart; close enough to avoid 'dead zone' – i.e. a space where the StarGazer cannot read any of the ceiling Landmarks. In-order for the StarGazer to establish a local GPS reading, the room must be mapped using the ceiling Landmarks. One of the Landmarks is used as a reference marker; then, using commands supplied by the Hagisonic Company, the Robot is moved from one Landmark to the next until all the ceiling Landmarks are encoded into the StarGazer memory. The Landmark information remains in the device memory until another room mapping is performed. In [9] an experiment showed that the maximum position error and maximum direction error of the StarGazer localization were 2cm and 5 degree, respectively. This accuracy is sufficient for most localization cases.

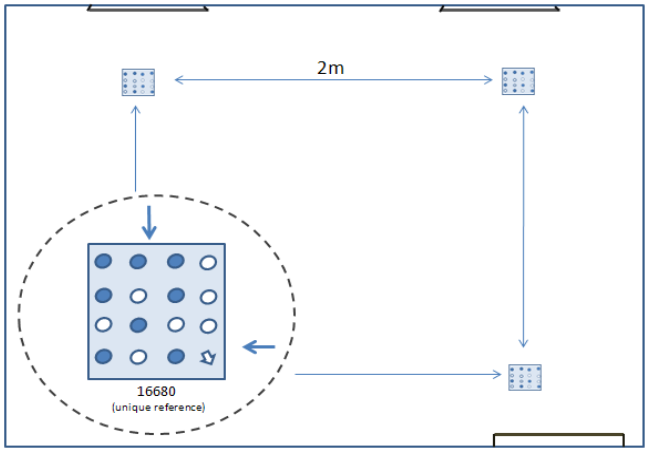

Figure 4. Overhead room view showing the GPS StarGazer ceiling Landmarks .

To retrieve the GPS data from the StarGazer, a .Net C# Windows Form interface is currently used. StarGazer functions can be accessed via the StarGazerGPS.dll. Connection is made via a Wi-Fi router using a dedicated port number on the Robot. When the StarGazer device detects a ceiling Landmark, it calculates the X and Y co-ordinates. The StarGazer device is located on a North–South axis on the Robot; the direction *angle* is then calculated between the StarGazer device and the current Landmark detected on the ceiling above.

IV. ALIGNMENT EVALUATION

Detailed analysis of the alignment error is required for each Robot used. To find the *mean* alignment error value, each Robot was tested 12 times. This gave a *mean* set of values that were used to generate a wheel alignment algorithm. The tester entered a desired distance and direction they wished the Robot to follow; this path was plotted on the User Interface. These values were then used as a marker against the actual path values returned from the Robot's indoor GPS. After the Robot had completed each test run, the path taken by the Robot was plotted on the User Interface. Each test was recorded in a SQL database. Fig.5. shows the results from each test together will the expected path values, for one robot. The averaged values from the SQL database, shown in Table 1 were then used to derive the wheel alignment algorithm, making note of the frictional

characteristics and angular properties of the terrain over which the measurements were made. Eventually it is our intention that this process will be automated so that the reflection layer of the autonomic software will be able to automatically refine the wheel alignment algorithms over time.

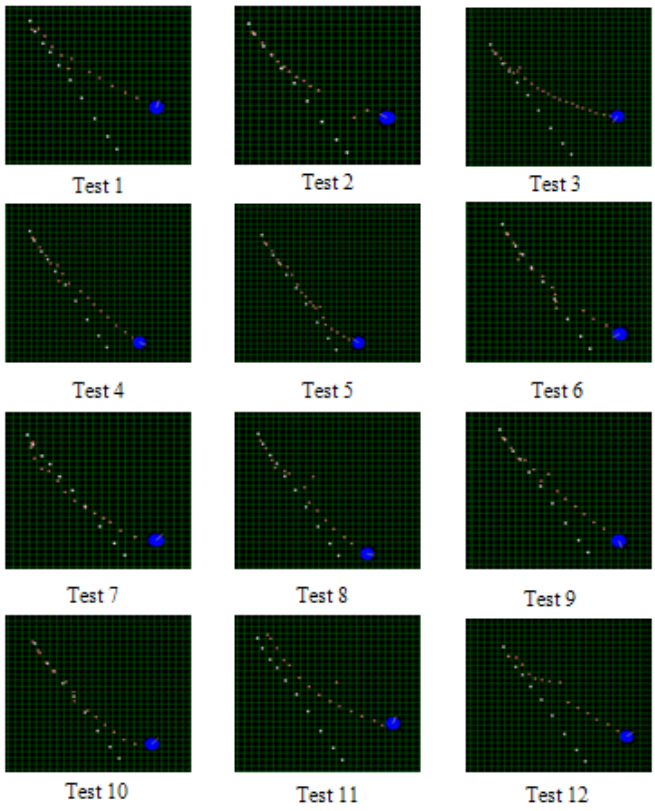

Figure 5. Test data alignment error plotting: at -30° direction for a distance of 2 meters.

	Readings are taken from a -30° start direction, for a distance of 2 meters									
Test #	.2m	.4m	.6m	.8m	1m	1.2m	1.4m	1.6m	1.8m	2m
Test 1	-30.1°	-30.5°	-32.6°	-35.7°	-44.3°	-47.9°	-51.5°	-54.5°	-58.6°	-60.8°
Test 2	-30.0°	-30.1°	-30.2°	-32.5°	-38.4°	-42.7°	-47.4°	-52.9°	-55.1°	-58.5°
Test 3	-30.1°	-30.3°	-31.9°	-34.8°	-42.5°	-45.7°	-49.8°	-53.1°	-56.7°	-61.2°
Test 4	-30.0°	-30.0°	-31.1°	-33.5°	-35.3°	-37.1°	-39.9°	-42.2°	-44.3°	-47.7°
Test 5	-30.0°	-30.0°	-31.2°	-31.5°	-31.8°	-32.6°	-34.5°	-37.7°	-39.0°	-42.6°
Test 6	-30.0°	-30.1°	-32.2°	-34.1°	-37.8°	-39.9°	-41.7°	-43.4°	-45.9°	-48.8°
Test 7	-30.0°	-30.7°	-31.8°	-32.5°	-33.7°	-35.6°	-38.1°	-39.0°	-41.5°	-43.1°
Test 8	-30.0°	-30.1°	-31.7°	-32.9°	-34.7°	-37.4°	-38.9°	-41.3°	-43.8°	-45.9°
Test 9	-30.0°	-30.3°	-32.7°	-34.8°	-38.2°	-40.0°	-42.3°	-44.7°	-46.1°	-49.2°
Test 10	-30.0°	-30.1°	-31.4°	-31.9°	-32.3°	-34.1°	-36.9°	-39.9°	-41.2°	-44.3°
Test 11	-31.2°	-32.3°	-33.7°	-36.4°	-42.7°	-48.2°	-52.5°	-57.6°	-60.1°	-64.8°
Test 12	-30.1°	-30.6°	-31.8°	-34.1°	-38.3°	-35.4°	-42.2°	-45.5°	-49.1°	-56.9°

TABLE I. TEST VALUES FOR WHEEL ALIGNMENT ERROR PATH

V. WHEEL ALIGNMENT ERROR DETECTION

A feature of autonomic behavior is for the Robot to establish that there is an error in the first place. The self-evaluation process, employed in the Robot's function, will expect certain parameters to be fulfilled with an expected tolerance value. In the case of the *alignment error*, the Robot needs to establish that after it has executed its journey or a check-point within that journey, that its actual position is not the same as its expected position. In a laboratory situation, this could easily be established using indoor GPS to work out the *difference* value between actual position and expected position; however, in the field, such as a space mission, GPS may not be available. In this situation the Robot may have to rely on dead reckoning and hence the importance's of the Robot being aware of any wheel alignment errors and the changes in this characteristic over time and surface characteristics.

VI. WHEEL ALIGNMENT ALGORITHM

The StarGazer GPS module can be used to supply x, y and direction data of the Robot; this data can then be collected and stored in a database. The data can also be represented graphically (see Fig.5.). Algorithm 1 shows how these methods can be achieved.

Algorithm 1 Plotting the Robot Path

Input: *Direction of Robot (°), Distance to travel(meters)*

a = direction of Robot
d = distance to travel
c = current Robot distance
gpsX = x-coordinate from the GPS StarGazer module
gpsY = y-coordinate from the GPS StarGazer module
gpsDir = direction of the Robot from GPS StarGazer module
Take a reading from GPS every 0.2 meters until the distance travelled is equal to the SET distance.
while c < d **do**
  if *timerCounter % 10 = 0* **do**
    Add gpsX reading to xPlot Array
    Add gpsY reading to yPlot Array
    Add gpsDir reading to dirPlot Array
    Write data to SQL database Table
  end if
end while
return *plot result*

The SQL data, stored from testing can be used in the Wheel Alignment (Arc Method) see Fig.6.

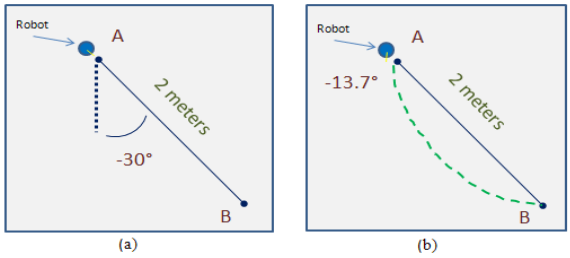

Figure 6. In this example, the direction and destination are given at -30° and 2 meters respectively bewtween A and B (a) - this represents the Robot with perfect alignment; (b) represents the Robot with an alignment error. The Robots' direction is adjusted to – 13.7°, to compensate for the alignment error. This is the Arc Method – see Algorithm 2.

Algorithm 2 shows how the data is retrieved and manipulated to calculate a *new* angle of direction to compensate for the alignment error. For the tests executed in Fig.5 an average value is calculated for each test. Then a further average is calculated over all the tests (12 in this case). The *new* start angle value is calculated by subtracting the *average angle* (from the test results) from the *default start angle*. Therefore, when the robot is given a new destination point, the robot will set its navigation direction to the *new* start angle value (calculated from the tests results), before it begins to move. This will improve the robots chances of arriving close to the designated destination point. This method however, relies on the terrain being clear of obstacles and walls.

Algorithm 2 Wheel Alignment (Arc Method)

Input: *Direction of Robot (°), Distance to travel(meters)*

Read test data from SQL Tables.
for (ts = number of tests) do
  for (vs = each value in test[ts]) do
    Read each test value into an array
    Calculate the average value of each test vs[ts]
    Store the average value in an array
    Calculate the Mean value of all tests
    Alignment Error Angle = Mean value of tests
Enter the journey values for the Robot.
sa = start angle
dir = direction to travel
dis = distance to travel
aa= alignment error angle
cd = current Robot distance
while cd < dis **do**
Set Robot angle
  sa= aa - dir
  Turn robot to new angle direction
  Move robot forward
end while
return *end position*

The main drawback of the Arc Method was that it relied on an area along the 'arc' path that was free of obstacles. If the journey between two points is more restrictive, then the alignment adjustments would need to be made at a regular interval. Fig.7. shows the Wave Method, were the alignment adjustments are made a certain intervals along the journey.

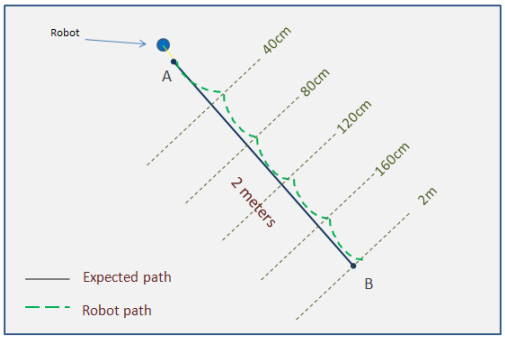

Figure 7. The Alignement Wave Method. Adjustments to the Robot are made a regular intervals, so that the Robot travels as close to the expected path as possible.

At each interval the Robot is turned back towards the original expected A to B path. As the Robot heads back towards the expected path, the alignment error begins to pull it away from the path. At the next interval, the Robot is turned once gain toward the expected path. These adjustments give the Robot path a *wave* appearance.

Algorithm 3 Wheel Alignment (Wave Method)

Input: *Direction of Robot (°), Distance to travel(meters)*

Read test data from SQL Tables.
for (ts = number of tests) do
  for (vs = each value in test[ts]) do
    Read each test value into an array
    Calculate the average value of each test vs[ts]
    Store the average value in an array
    Calculate the Mean value of all tests
    Alignment Error Angle = Mean value of tests
Enter the journey values for the Robot.
sa = start angle
dir = direction to travel
dis = distance to travel
aa= alignment error angle
cd = current Robot distance
is = interval setting(cm)
while cd < dis **do**
Adjust the Robot direction at interval setting
if cd % is == 0 **do**
Set Robot angle
  sa= aa - dir
  Turn robot to new angle direction
  Move robot forward
  moveRobotForward()
end if
end while
return *end position*

VII. Experiments

The Arc method (see Algorithm 2), was implemented using .net C# platform. The following set of tests (see Fig.8), show the Arc method applied to an X80 Robot.

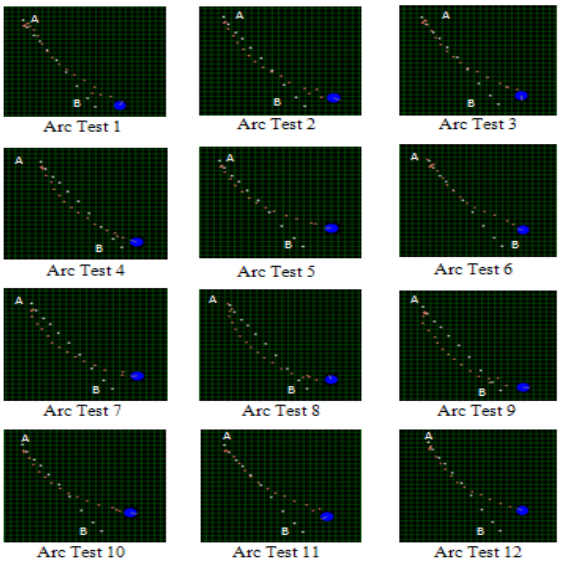

Figure 8. The above tests show the Arc method being applied to the X80 Robot. The white dashed line A – B, shows the expected path. The orange dashed line, shows the actual Robot path.

The Wave Method (see Algorithm 3), was implemented using .net C# platform. The following tests Fig.9 shows the Wave method applied to an X80 Robot.

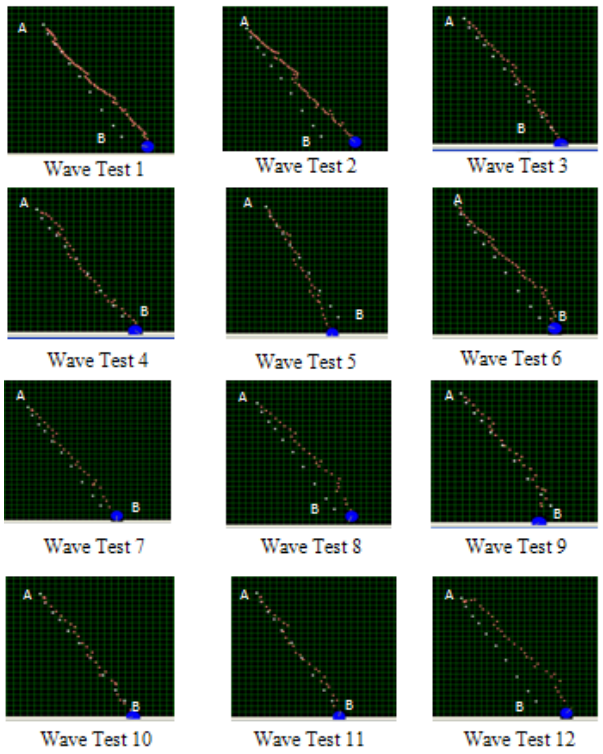

Figure 9. The above tests show the Arc method being applied to the X80 Robot. The white dashed line A – B, shows the expected path. The orange dashed line, shows the actual Robot path.

The Wave Method test results along with the Arc Method test results are displayed in a chart: see Fig.10. Included in the chart are the original *Alignment Error* test results and also the perfect *Alignment* values.

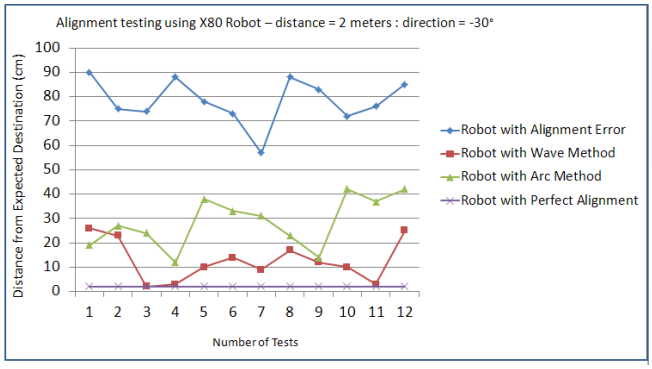

Figure 10. This chart shows the how the Arc Method and the Wave Method, improves the average distance between the expected destination point to the actual Robot destination point.

VIII. Conclusion and Future Work

The purpose of this research is development of algorithms to deal with *Wheel Alignment* issues, found in Mobile robots. As explained previously, low cost *swarm* robots will not have the sophistication as described in [10], and therefore will rely on autonomic solutions to continue to perform in the field, even with defects such as wheel alignment. The experiments described in this paper show how a low cost mobile robot, with a wheel alignment defect, can be instructed to overcome this defect. It was found that the algorithms *Arc* and *Wave*, greatly improved the Robots ability to arrive at or near the expected destination point. However, these tests were conducted on a flat uninterrupted surface; future development will introduce different surfaces, inclines, declines and obstacles. The Autonomic Reflection Layer will be represented in future development, as a Robot that retains its historical data and uses this information to evaluate possible strategies when confronting defects such as wheel alignment errors.

References

[1] D. A. Norman, A. Ortony, and D. M. Russel, "Affect and machine design: Lessons for the development of autonomus machines," IBM Systems Journal, vol. 42, no. 1, pp.38 – 44, 2003.

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

[3] G. Punzo, G. Dobie, D. J. Bennet, J. Jamieson, M. Macdonald, "Low-Cost, Multi-agent systems for planetary surface exploration", 63rd International Congress, 2012.

[4] R. Sterrit, M. Parashar, H. Tianfield, R. Unland, "A concise introduction into autonomic computing", Advanced engineering Informatics, 19, pp.181-187, 2005.

[5] W. F. Truszkowski, M. G. Hinchey, J. L. Rash, C. A. Rouff "Autonomous and Autonomic Systems: A Paradigm for Future Space Exploration Missions", IEEE Transactions on Systems, Man, and Cybernetics - TSMC , vol. 36, no. 3, pp. 279-291, 2006.

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

[7] H. Shualib, R. J. Anthony, M. Pelc, "Framework for Certifying Autonomic Computing Systems", The Seventh international Conference on Autonomic and Autonomous Systems, 2011

[8] C. C. Insaurralde, "Autonomic Management Capabilities for Robotics and Automation", 1st Global Virtual Conference, Electronics, Electrical Systems, Electrical Engineering, 16, pp. 518-523, 2013

[9] I. Ul-Haque and E. Prassler, "Experimental Evaluation of a Low-cost Mobile Robot Localization Technique for Large Indoor Public Environments", in Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK), 2010, pp. 1–7.

[10] V. Dimitrov, M.DeDonato, A.Panzica, S.Zutshi, M.Wills and T.Padir, "Hierarchical Navigation Architecture and Robotic Arm Controller for a Sample Return Rover", IEEE International Conference on Systems, Man, and Cybernetics, IEEE Computer Society, 2013, pp.4476 – 4481, DOI 10.1109/SMC.2013.761.

[11] Dr. Robot Inc. X80 Reference Manual, Version 1.0.3, Canada, 2010.