Simulating Autonomous Robot Teams With Microsoft Robotics Studio

Mr Ross De Rango; Professor Saeid Nahavandi
Intelligent Systems Lab, Deakin University
rdd@deakin.edu.au, nahavand@deakin.edu.au

Abstract. This paper presents an application of Microsoft Robotics Studio (MSRS) in which a team of six four wheel drive, ground based robots explore and map simulated terrain. The user has the ability to modify the terrain and assign destination objectives to the team while the simulation is running. The terrain is initially generated using a gray scale image, in which the intensity of each pixel in the image gives an altitude datum. The robots start with no knowledge of their surroundings, and map the terrain as they attempt to reach user-defined target objectives. The mapping process simulates the use of common sensory hardware to determine datum points, including provision for field of view, detection range, and measurement accuracy. If traversal of a mapped area is indicated by the users' targeting commands, path planning heuristics developed for MSRS by the author in earlier work are used to determine an efficient series of waypoints to reach the objective. Mutability of terrain is also explored- the user is able to modify the terrain without stopping the simulation. This forces the robots to adapt to changing environmental conditions, and permits analysis of the robustness of mapping algorithms used when faced with a changing world.

1. INTRODUCTION

Computer based simulation environments supporting multi-robot simulations have been around for some time [1, 2]. One of the more recent entrants into this field is Microsoft's Robotics Studio [3] package. Each new software package on the market brings scope for increased realism in simulation- providing for the physics of the real world to be more accurately represented, at higher update rates, for more objects, and in more effective dimensions. This paper presents an application in Robotics Studio in which a team of simulated four-wheel drive robots conduct mapping operations on uneven terrain.

Effective operation of a robot team relies on data sharing [4]. The data shared typically includes beliefs about position, orientation, velocity, and intended destination, as well as beliefs about the world in which the robots are operating. In the real world, this generally implies a wireless network of some sort. In this simulated environment, arrays of data are used which are made accessible to all robot team members. The issue of data availability is not considered in this paper; while it is acknowledged that in real-world situations information availability can be critical, this will be addressed in future work.

The chief advantages of simulation over real world experimentation are time and cost- real world testing is expensive and time consuming. Simulations, by contrast, can be adjusted with a few keystrokes if the code associated with them has been written with extensibility in mind. Of course, the simulation is not the real- but simulation work prior to physical work, especially in the software control aspects of the task, saves a lot of effort.

In addition to time and cost advantages, simulations also make it possible to make exact comparisons between robot belief states and the simulated reality over time. If no noise is introduced into the system, then measurements will be accurate to the limit of the floating point numbers used to model the environment. When simulating real systems, noise can be introduced to measurements in the form of Gaussian distributions. This causes discrepancies between the simulated reality and the robot's perceptions of that reality in much the same way that real world sensors are limited in terms of their accuracy. Algorithms written to take noisy signals into account can be rigorously tested in simulation by analyzing the discrepancies

2. SIMULATION ENVIRONMENTS

There are many software platforms available on which work of this nature can be conducted. For non-windows machines, Player/Stage/Gazebo, known as the 'Player Project' [1], is an attractive alternative. These three components comprise a software solution for simulating robot teams. Player is a network server for robot control, providing an interface to the robot's sensors and actuators over an IP network. Stage is a simulation environment, designed for large numbers of robots to be simulated in two dimensions at low computational cost. Gazebo is another simulation environment, designed to allow smaller robot populations to be simulated in three dimensions- unlike Stage, Gazebo supports rigid-body physics interactions. Player can be combined with either Stage or Gazebo, depending on the specific requirements of the user.

Webots is another robotics development and simulation environment, commercially available for PC, Mac, and Red Hat Linux Platforms [5], and used extensively in robotics research. It allows simulation in two or three dimensions of single or multiple robots, custom robot design, custom physics effects, and so on.

While Robotics Studio is a very new entrant into the field of available packages that can perform 3D multi-robot simulation, amongst other features it:

1) Is programmable in the C# language.
2) Runs on a windows XP platform.
3) Is freely available to educational institutions.
4) Permits custom robot design
5) Permits complex terrain in simulation.
6) Utilizes effective simulation and physics engines.
7) Is likely to be well supported in future.

These advantages taken together made it suitable for this work, which has been conducted with no expenditure on software products aside from the operating system and analysis tools already installed on the computer used.

3. TERRAIN

3.1 Generation

The terrain is generated in the simulation engine by providing a large number of points in three-dimensional space, and then interpolating between these points to create a multi-faceted surface. The points are arranged in evenly spaced rows and columns on a theoretical flat ground plane, with variable altitude. The data for the points is drawn from a gray-scale bitmap file, where the rows and columns in the image correspond to the rows and columns of the simulated ground plane, and the intensity of the pixel (0-255) gives the altitude offset from the origin. Once the initial gray scale bitmap file has been read into an array, the file is closed, and the terrain is generated from the data array. The software package supports the creation of a height field entity, which is essentially a structure defining a complex surface as described.

In this work, an image of 260x260 pixels was used, with the data points scaled such that the rows and columns were spaced one metre apart. The pixel intensity was interpreted such that each step in intensity corresponded to a change in altitude of 0.1 metres. The scaling is entirely variable, and can be changed to generate smoother terrain if required. Prior to real world implementation, the map would be scaled to a resolution of less than 50mm, so that objects likely to impede progress would be detected and mapped.

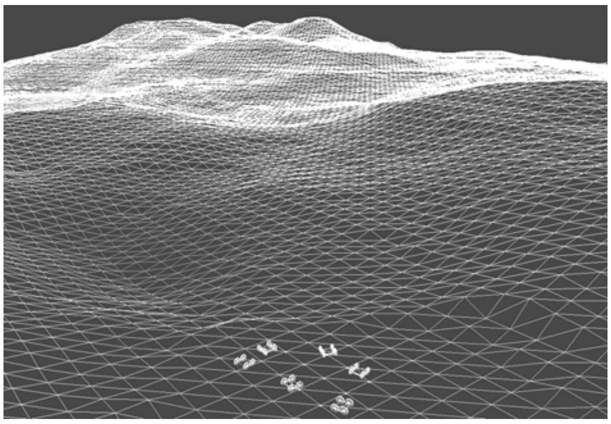


Figure 1: Generated terrain

Figure 1 shows the effect of this process. The vertices in the wire-frame depiction of the terrain are the data points drawn from the analysis of the gray scale image, and the lines between them show the interpolation which generates the faceted surface. It can be seen that the terrain is not made up of smooth curves, but instead of sharp gradient changes along the interpolated lines; this means that some portions of the map may cause robots to become stuck. This is acceptable for this work, as the robots need to be able to evaluate what constitutes drivable terrain, and sensory input will generally be of a step rather than continuous nature anyway [6].

3.2 Manipulation

The altitudes of individual points in the terrain can be modified by changing the members of the height field entity already mentioned. This does not have an impact on the original image file, just on the terrain in the simulation.

When the simulation commences, a graphic user interface (GUI) starts up. With the current GUI, the user is able to get and change the altitude of any given data point used in the terrain. This can be done absolutely, defining a new required altitude, or relatively, by specifying a required offset from the current altitude at the point. These methods permit the user to easily place new obstacles in the environment, such as walls and trenches, which the robots can then detect and navigate around.

In addition to changing specific points, a method has been developed to change a group of points at once, in order to create craters of variable size. The user defines the centre point, the depth of the impact, and the radius of the impact. The simulation code then decreases the altitude of the centre point and surrounding points to make a bowl-shaped depression in the terrain. At this stage, the ‘removed’ ground doesn’t go anywhere, it simply ceases to exist.

Changes to the terrain can be recorded, as well. The GUI has an option to save the terrain, which generates and saves a new bitmap file based on the current height field entity.

4. ROBOT DESIGN

4.1 Four Wheel Drive

The main robot design used in the simulation work presented here is comprised of a rectilinear box representing the chassis, 300mm long, 200mm wide, 50mm high; and four cylinders representing the wheels, 200mm diameter, 70mm width, aligned on two parallel theoretical axles spaced 240mm apart. This configuration is designed for 'skid-steering'; changes in orientation are achieved by applying the same torque to both wheels on one side, and a different torque to both wheels on the other side [7]. The design is depicted as a wire-frame image in figure 2.

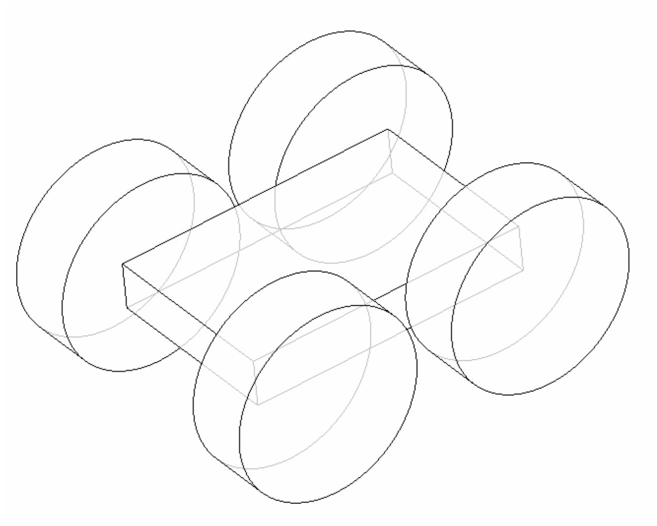


Figure 2: Simulated four wheel drive robot design

Note that the simulated robots do not utilize a tread pattern, suspension, or a pneumatic tire model. This has reduced the computational complexity of the model, in order to achieve a higher simulation frame rate. The simplicity of the simulated robot design means that there will inevitably be some variance in performance of the simulated robot compared to real ones, but the basic control structure will still be transferable.

For the purposes of this exercise, the robots are assumed to be equipped with sensory hardware which scans the environment in a wedge to the front of the robot. The size of this wedge is user definable, defaulting to 120 degrees from left to right, 30 degrees declination, and 30 degrees elevation, with the assumed centre point above the middle of the top front edge of the chassis depicted in figure 2, 200mm above ground level. This implies that on level ground the sensor will be able to 'see' the ground in an arc commencing approximately 350mm in front of the sensor, and that in instances where the ground is dropping away in front of a robot on flat ground, a point 1 metre in front of the robot will be within the detection area provided that it is no more than 377mm *below* the present ground level. Default sensor maximum range is 5 metres, and again this is user adjustable.

4.2 Other robot designs

The software package lends itself to the designs of many robot types. Included in the samples provided with the software are programmatically built models for the Pioneer 3DX, the Lego NXT, and a Kuka robot arm [3]. MSRS allows the user to programmatically design a physical representation of robots of almost any configuration, including wheeled and tracked as shown here, through to multiple jointed legged and humanoid designs [8]. Other work by the author has involved teams of simulated tracked robots, generated by simulating ten overlapping wheels on each side of the robot's chassis, as shown below in figure 3.

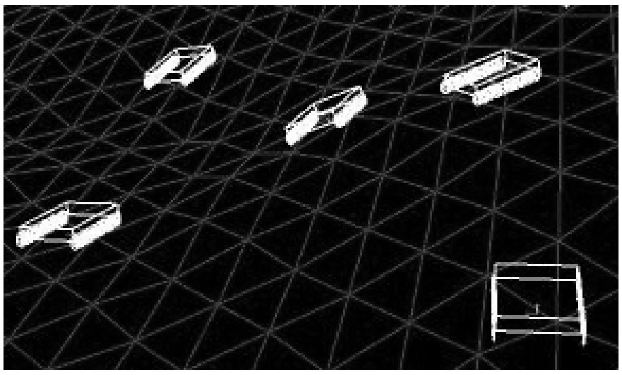


Figure 3: Simulated track-drive robot design

5. ROBOT OPERATIONS

In the present simulation environment, six four wheel drive robots are instantiated two metres apart around the origin point in the simulation world co-ordinate system. The simulation engine keeps track of their position and orientation in three axes, and their linear and angular velocities.

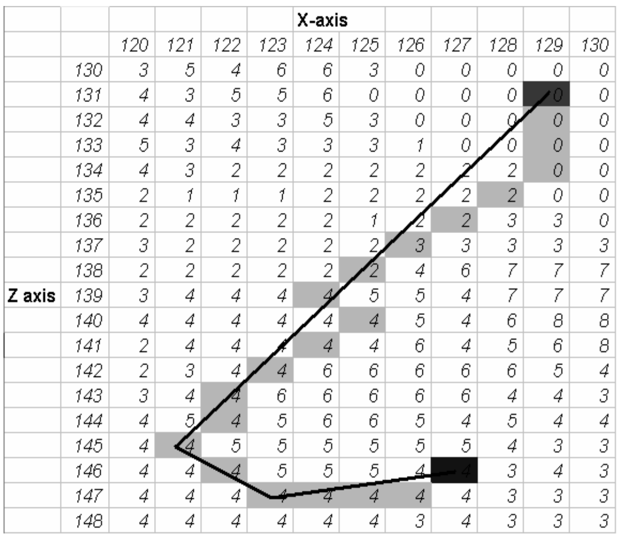
The robots communicate via a notice board arrangement, to which they all have read/write access. It is assumed that the robots have known locations, which they can communicate to each other via this notice board. This paper does not address the SLAM problem [10]; it is assumed that the robots derive their location data from another source, such as differential GPS, and that errors in this location data are negligible.

Instructions to move are given to the robot team in terms of Cartesian co-ordinates of the team objective, a specific formation to take up on arrival at the destination, and specific spacing between robots that is expected at that objective. Currently supported formations are a hexagon surrounding the objective point, a pentagon with one robot on the objective point and the remainder surrounding it, and a single rank with the centre of the line at the objective point. Orientation is also controllable, so the single rank can be instructed to arrive and form up facing in a given direction.

In addition to taking terrain features into account, the robots must avoid crashing into each other. This is achieved by disallowing movement along bearings where team mates are close by. To clarify: assume robot A is attempting to drive forward, and robot B is in the way. If robot B is within 2 meters of the robot making a bearing decision (robot A), the bearings in the direction of robot B from robot A will be disallowed in an arc. The angular width of this arc is inversely proportional to the separation distance, starting at 0 degrees at two meters and increasing to 180 degrees (effectively a wall) at one meter. Changing the parameters by which bearings are disallowed lets the user trade off between faster, more efficient achievement of target formations and increased risks of collisions.

When instructed to move to a location as a team, the robots analyze the map information currently available to check the drivability of the terrain between their current location and the objective. The drivability is chiefly governed by the variation in altitude from one vertex to the next. Each vertex has eight immediate neighbours. If any of those neighbours has an altitude differing by more than 400mm from its own (i.e. a maximum local gradient of more than 4 in 10), then that vertex is considered to be non-navigable.

If a fully mapped, navigable route exists, the robots will find it during the path planning process. They will then generate waypoints along that route and proceed to the objective. The precise methodology behind the path planning process is the subject of another paper, but in brief, the path planning method is a best-first, heuristically based process, utilizing forward and reverse passes [11, 12]. Figure 4 shows an example of a planned path in an array of navigability (maximum local gradient) data. In this diagram, the Cartesian axes, x and z, are consistent with the rows and columns in the terrain. The numeric data is the maximum local gradient, with any value greater than four considered non navigable. The path planning process starts at the vertex (129,131), and reaches (127,146) via a series of steps, shown shaded in the diagram. Vertices from the path are chosen as waypoints, such that a linear interpolation between these waypoints does not come within 0.5 metres of any vertex with a maximum local gradient greater than four. The path taken by the robot in this simulation was very close to the path indicated by the black line in figure 4.



						X-axis						
		120	121	122	123	124	125	126	127	128	129	130
	130	3	5	4	6	6	3	0	0	0	0	0
	131	4	3	5	5	6	0	0	0	0	0	0
	132	4	4	3	3	5	3	0	0	0	0	0
	133	5	3	4	3	3	3	1	0	0	0	0
	134	4	3	2	2	2	2	2	2	2	0	0
	135	2	1	1	1	2	2	2	2	2	0	0
	136	2	2	2	2	2	1	2	2	3	3	0
	137	3	2	2	2	2	2	3	3	3	3	3
	138	2	2	2	2	2	2	4	6	7	7	7
Z axis	139	3	4	4	4	4	5	5	4	7	7	7
	140	4	4	4	4	4	4	5	4	6	8	8
	141	2	4	4	4	4	4	6	4	5	6	8
	142	2	3	4	4	6	6	6	6	6	5	4
	143	3	4	4	6	6	6	6	6	4	4	3
	144	4	5	4	5	6	6	5	4	5	4	4
	145	4	4	5	5	5	5	5	5	4	3	3
	146	4	4	4	5	5	5	4	4	3	4	3
	147	4	4	4	4	4	4	4	4	3	3	3
	148	4	4	4	4	4	4	3	4	3	3	3

Figure 4: Path planning example

If a fully mapped route does not exist, the robots work as a team to map their environment until a route does exist, as described in section 6. This means that each robot acts to increase the total map knowledge, based on what is currently known, and the most efficient way to expand the map.

6. MAPPING OPERATIONS

At the commencement of the simulation, the robots do not have knowledge of their surroundings, so they cannot plan a path to anywhere. Therefore, on receiving an instruction to move, they begin mapping.

6.1 Visible vertices

The first time any vertex falls within the detection area of any robot, its actual altitude (drawn from the terrain data) modified by noise is stored as the mapped altitude at that point, along with the associated 95% confidence intervals. The degree to which Gaussian noise impacts the measurement is determined by the programmer; at this stage a standard deviation of 0.02 metres in measured height is used. The Gaussian noise is approximated using the Box Muller algorithm, where U_1 and U_2 are random numbers.

$$Z_0 = R\cos(\Theta) = \sqrt{-2\ln U_1}\cos(2\pi U_2) \quad [13]$$

On subsequent occasions, the newly measured data is compared to the currently held belief about the altitude of the point. If the new data is within the 95% confidence interval of the current belief, then the new data is incorporated into the existing belief statistically, forming an updated belief. A Kalman filter approach is used here, since the assumption is that the object being measured has a steady-state location, and noise in the samples is Gaussian.

If the new data falls outside the 95% interval, it is necessary to determine if the new data is incorrect, or if the terrain has actually changed, rendering the existing belief obsolete. If the terrain has actually moved, it will be counter-productive to utilize Kalman methods, which assume that previous states have some bearing on the current state [9]. Four additional readings with associated noise are taken of the point concerned, and considered with the outlier recently recorded. If the average of these five readings falls outside the 95% interval, then the altitude belief regarding that vertex (that is, it's mapped altitude) is changed to the average just computed, and the previous belief is discarded. Otherwise, the five data points just collected are added into the existing belief state using the Kalman technique discussed in the previous paragraph.

6.2 Invisible vertices

In some instances, vertices that would be detected on flat ground may be above or below the wedge shaped detection area. In instances where the vertices are above the detection area, the robot can be permitted to recognize that a vertically protruding obstacle has been found. This is akin to a scanning laser range finder detecting a wall- the fact the vertex (the top of the wall) is not in the detector range is not particularly important, as the terrain leading up to the vertex *is* in the detector range.

If the vertex is *below* the detection area, we cannot make any immediate conclusions. It is possible that a sheer cliff or trench exists in front of the robot, but it is also possible that the vertex is drivable, but not visible because of the nature of the terrain. For example, consider a robot at the lip of a crater facing toward the centre, where the initial downward angle approaches 30 degrees. The robot will not be able to perceive vertices immediately to its front, but as the gradient of the crater becomes shallower towards the centre, other vertices will become visible. The non-detectable vertices to the front are allocated an altitude of five metres below their nearest mapped neighbour, with a 95% confidence interval of one metre each way. This ensures that they will be considered non-navigable unless and until they are viewed from a new angle, and re-defined. To determine the altitude of the points in front of it, the robot would need to move to a location that would provide an increased vertical detection range at the vertex concerned. In this example, the robot could move around the rim, then turn to face the area that it could not previously 'see'. The increased range may now mean that the vertex is within the detection area, and could be accurately mapped.

If at a later time a robot is moving through the same area, and the shortest route involves driving over the previously mapped lip of the crater, the classical technique would be for it to depend on previously mapped information, and not be concerned with the fact that certain areas are non-visible. This can be considered 'leap of faith' mode, and is one behaviour option available to the user.

It this simulation, one of the concepts is that the terrain is not fixed- in the time between when it was last mapped and the present time, it may have changed. 'Look before leap' mode takes this into account- regardless of previously mapping, the robots will not drive over vertices that they cannot 'see'. In this mode path planning includes the imperative that each vertex along the route be visible to the robot as it proceeds along the planned path. Switching between these modes is manually controlled by the user.

6.3 Destination selection

In instances where the robot cannot plan a route to it's objective due to an incomplete map, it must select a point to which it can go in order to make the map it is using more complete. Map completion is an iterative process, and is treated here as two functions:

1) If turning on the spot will permit previously unseen terrain to be mapped, do so.
2) Push back map edges

Function one essentially means that each robot should move the minimum possible amount to map the maximum amount of new terrain. At the commencement of the simulation, the robots spin on the spot to map the area immediately around their starting location, as this represents the least effort for the most return.

Function two then plans routes for the robots to any edges of the current map where impassable terrain is *not* indicated (i.e. no consistently steep terrain or sudden drops). The robots then move to those locations and map what is visible from there. Since more than one robot is likely to be engaged in this activity at the same time, some co-ordination is required so that the robots do not end up converging on the same location. This co-ordination prioritizes the point on the edge of the existing map that is closest to the objective and dispatches the nearest robot to that point. The other robots are assigned other points on the perimeter of the mapped area, spaced at a minimum of three metres apart, according to how close they are to the indicated perimeter locations.

This process of mapping what can be seen from the current position, then moving to a map edge and repeating the process, continues iteratively until the objective is reached, or the mapped area is completely surrounded by non-navigable terrain and the objective has *not* been reached.

6.4 Map display

Figure 1 shows generated terrain, prior to map generation. As each vertex in the terrain is mapped, small marker spheres come into existence vertically aligned with the vertices. These markers are used to provide a visual indication to the user of the robot's belief state regarding altitude of the vertex- the marker will appear with it's centre at the altitude the robots believe the ground surface to be at, rather than where the ground actually is. In cases where the marker sphere is above the ground surface, an additional offset of 0.5m is used to prevent the markers obstructing the robot's movements. In subsequent versions, purely visual entities with color animation will make it possible to communicate more information more effectively.

The map, like the terrain, can be saved at any point by a GUI command for later analysis.

7. FUTURE WORK

Future steps in this simulation works will include:

1) More highly developed sensor models, coupled with higher resolution terrain, to map the terrain interpolated between the vertices rather than just the vertices themselves.
2) Creating a crater will generate new physics entities with defined masses and velocities, so that simulated rocks and coarse aggregate will fly out of the new depression in parabolic arcs, and then remain on the surface.
3) Increased intelligence programmed into the simulated robots, to allow more complicated problem solving.
4) Integration between simulated airborne and ground based autonomous vehicles, such that ground based vehicles can call on air based vehicles to assist in terrain mapping and path planning
5) Manipulation of terrain by simulated robots, to represent construction of trenches etc.

ACKNOWLEDGEMENTS

This work was conducted under the auspices of the Intelligent Systems Lab. at Deakin University, Australia, led by Saeid Nahavandi. The author wishes to thank the development team at Microsoft Robotics Studio and the community posting to the related forums [14].

REFERENCES

1. Gerkey, B.; Vaughan R.T.; Howard, A.; (2002) “The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems”, IEEE Transactions on Robotics and Autonomous Systems, Vol. 18, no. 5, October, pp 796–812.

2. Billard, A.; Ijspeert, A.J.; Martinoli, A.; (1999) “A Multi-robot System for Adaptive Exploration of a Fast-changing Environment: Probabilistic Modeling and Experimental Study” *Connection Science*, Taylor and Francis, Vol. 11, Nos. 3-4, December, pp. 359-379.

3. Microsoft Corporation, Microsoft Robotics Studio – Website (2007): http://msdn.microsoft.com/robotics/ (accessed 11.04.2007).

4. Fujii, T.; Asama, H.; Fujita, T.; Asakawa, Y.; Kaetsu, H.; Matsumoto, A.; Endo, I.; (1996) “Knowledge sharing among multiple autonomous mobile robots through indirect communication using intelligent data carriers”, Intelligent Robots and Systems '96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on, vol. 3, November, pp 1466-1471

5. Magyar, B.; Forhecz, Z.; Korondi, P.; (2003) “Developing an efficient mobile robot control algorithm in the Webots simulation environment”, Industrial Technology, 2003 IEEE International Conference on, vol. 1, December, pp 179-184.

6. Thrun, S., et al, (2006) “Stanley: The Robot that Won the DARPA Grand Challenge”, Journal of Field Robotics, vol. 23, no. 9, pp 661-692.

7. Shamah, B.; (1999) “Experimental Comparison of Skid Steering Vs. Explicit Steering for a Wheeled Mobile Robot”, master's thesis, tech. report CMU-RI-TR-99-06, Robotics Institute, Carnegie Mellon University.

8. Doh!Bots, Dortmund University, (2007) http://www.dohbots.com/dev_simulator (accessed (11.04.2007)

9. Gutmann, J.S. (2002) "Markov-Kalman Localization for Mobile Robots," 16th International Conference on Pattern Recognition (ICPR'02) - Volume 2. p 20601.

10. Dissanayake, M.W.M.G.; Newman, P.; Clark, S.; Durrant-Whyte, H.F.; Csorba, M.; (2001) “A solution to the Simultaneous Localization and Map Building (SLAM) Problem”, IEEE Transactions on Robotics and Automation, Vol. 17, no. 3, pp 229-241.

11. Korf, R.E.; (1985) “Depth-first iterative-deepening, an optimal admissible tree search”, Artificial Intelligence, vol. 27, no. 1, September, pp 97-109.

12. Kuan., D.; Zamiska. J.; Brooks, R.; (1985) “Natural decomposition of free space for path planning”, 1985 IEEE International Conference on Robotics and Automation, vol. 2, March, pp168-173

13. Box, G.E.P.; Muller, M.E.; (1958), sourced from website http://en.wikipedia.org/wiki/Box_muller (accessed 11.04.2007)

14. Microsoft Corporation, MSDN Website (2007): http://forums.microsoft.com/MSDN/default.aspx?ForumGroupID=383&SiteID=1 (accessed 11.04.2007).