Design and Implementation of a Path Finding Robot Using Flood Fill Algorithm

Semuil Tjiharjadi and Erwin Setiawan
Computer Engineering Dept., Maranatha Christian University, Bandung, Indonesia
Email: semuiltj@gmail.com

***Abstract*—Autonomous robot is a robot that can perform certain work independently without the human help. Autonomous of navigation is one of the capabilities of autonomous robot to move from one point to another. Implementation of Autonomous robot navigation to explore an unknown environment, requires the robot to explore and map the environment and seek the path to reach a certain point. Path Finding Robot is a mobile robot which moves using wheels with differential steering type. This path finding robot is designed to solve a maze environment that has a size of 5 x 5 cells and it is based on the flood-fill algorithm. Detection of walls and opening in the maze were done using ultrasonic range-finders. The robot was able to learn the maze, find all possible routes and solve it using the shortest one. This robot also use wall follower algorithms to correct the position of the robot against the side wall maze, so the robot can move straight. After several experiments, the robot can explore and map of maze and find the shortest path to destination point with a success rate of 70%.**

***Index Terms*—flood fill algorithm, path finding, maze, wall folower algorithm**

I. Introduction

Autonomous navigation is an important feature of mobile robotics. It allows the robot to independently move from a place to target location without a tele-operator. There are several techniques and algorithms have been developed for this purpose, each of them having their own merits and shortcomings [1]-[5].

Path Finding robot is using a structured technique and controlled implementation of autonomous navigation which is sometimes preferable in studying specific aspect of the problem [2]. This paper discusses an implementation of a small size mobile robot designed to solve a maze based on the flood-fill algorithm.

The path finding task is where robots try to solve a maze in the least time possible and using the most efficient way. A robot must navigate from a corner of a maze to the center as quickly as possible. It knows where the starting location is and where the target location is, but it does not have any information about the obstacles between the two. The maze is normally composed of 256 square cells, where the size each cell is about 18 cm × 18cm. The cells are arranged to form a 16 row × 16 column maze. The starting location of the maze is on one of the cells at its corners, and the target location is formed by four cells at the center of the maze. Only one cell is opened for entrance. The requirements of maze walls and support platform are provided in the IEEE standard.

Manuscript received July 1, 2015; revised March 14, 2016.

II. Literature Review

A. Breadth First Search

Breadth First Search uses First In First Out queue. It is used when space is not a problem and few solutions may exist and at least one has shortest path. It works poorly when all solutions have long path length or there is some heuristic function exists. It has large space complexity [6].

B. Depth First Search

Depth First Search uses Last In First out queue and are recursive in algorithm. It is simple to implement. But major problem with Depth First Search is it requires large computing power, for small increase in map size, runtime increases exponentially [6].

C. Heuristic Function

Heuristic function maps problem state descriptor to a number which represents degree of desirability. Heuristic function has different errors in different states. It plays vital role in optimization problem [6].

D. Genetic Algorithm

Genetic algorithm is used to find approximate optimal solution. It is inspired by evolutionary biology such as inheritance, mutation, crossover and selection [7]. Advantages of this algorithm are it solves problem with multiple solutions, it is very useful when input is very large. Disadvantages of Genetic algorithm are certain optimization problems cannot be solved due to poorly known fitness function, it cannot assure constant optimization response times, in Genetic algorithm the entire population is improving, but this could not be true for an individual within this population [6].

E. A* Algorithm

A*combines feature of uniform-cost search and heuristic search. It is BFS in which cost associated with each node is calculated using admissible heuristic [1]. For graph traversal, it follows path with lowest known heuristic cost. The time complexity of this algorithm depends on heuristic used. Since it is Breadth First Search drawback of A* is large memory requirement because entire open-list is to be saved [6].

doi: 10.18178/ijmerr.5.3.180-185

F. Flood Fill Algorithm

Robot maze problems are an important field of robotics and it is based on decision making algorithm [8]. It requires complete analysis of workspace or maze and proper planning [9]. Flood fill algorithm and modified flood fill are used widely for robot maze problem [10]. Flood fill algorithm assigns the value to each node which is represents the distance of that node from centre [6]. The flood fill algorithm floods the maze when mouse reaches new cell or node. Thus it requires high cost updates [3]. These flooding are avoided in modified flood fill [1].

III. Hardware Design

Mobile robot base construction was made using miniQ 2WD robot chassis. It was a product from DFRobot as shown in Fig. 1. In the product consists of 1 robot chassis with a diameter of 122mm. 2 wheels with a diameter of 42mm, 1 piece ball caster and 2 DC motors which have been furnished by the gearbox as well as two pieces of the DC motor bracket to pair on the chassis.

Figure 1. 12WD miniQ robot chassis.

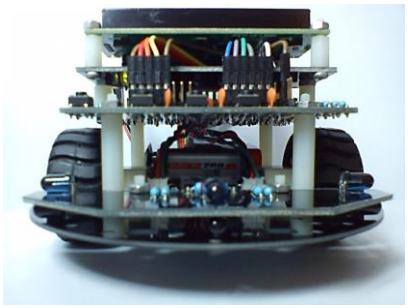

Figure 2. Mobile robot from side view.

In this maze solving robot had 2 pieces rotary encoder. Rotary encoder used is miniQ robot chassis encoder which is also a product from DFRobot. Rotary encoder is compatible with 2WD products miniQ robot chassis. Rotary encoder attached to the DC motor to calculate the rotation of the wheel as shown in Fig. 2. [11]

The whole hardware system of this mobile robot can be seen in the block diagram at Fig. 3 and Fig. 4 shows the main program. Mobile robot used three infrared sensors to detect maze wall at right, left and front position. Driver L293D controled the direction of rotation and speed of a DC motor [12]. Rotary encoder is used to calculate the rotation of the right and left wheels. Push button was used to instruct the robot to start. The system output would drive two DC motors that served as actuators to move the right and left wheels, so that the robot can move forward, spun to the right, turned to the left, and rotates reverse [13]. ATmega324 microcontroller serves to process the signal-sinyalinput, perform processing algorithms, and generates output signals to control a robot [9]. Information about all actions that had been taken by the robot, would be displayed on the LCD 16 x 2 at Fig. 5.

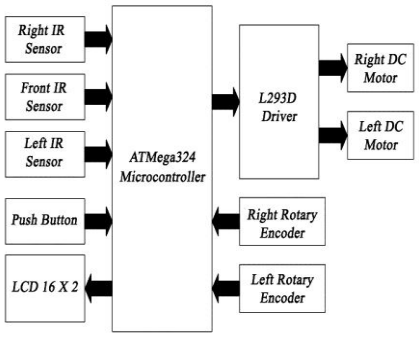

Figure 3. Block diagram of mobile robot.

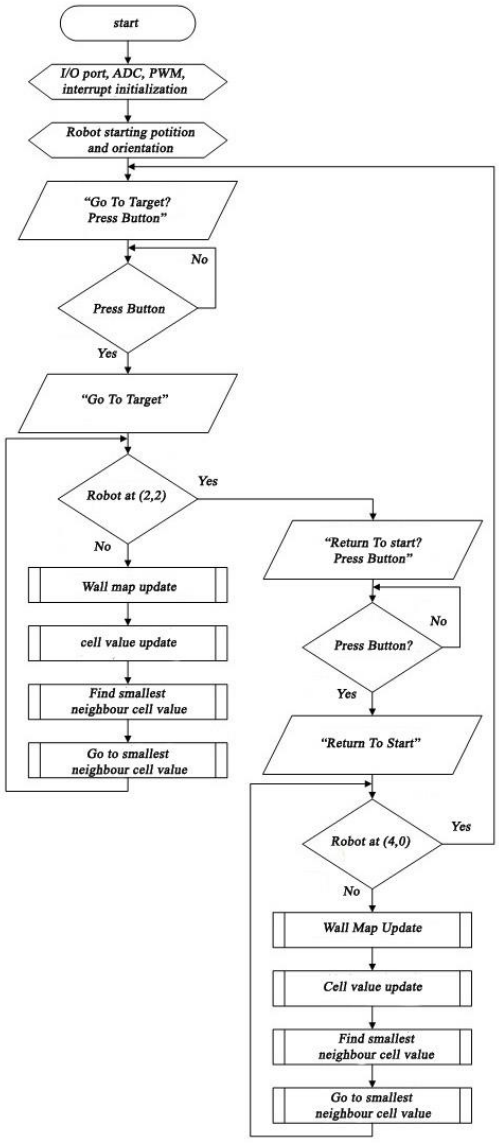

Figure 4. Flowchart of the main program

The maze designed for the robot to solve is of the size of 5×5 cells as shown in Fig. 6. The actual maze constructed, as shown in Fig. 7, has a physical size of about 1.32 m2. The maze was designed so that it will have two paths in order for it to be solved. One of the paths is longer than the other. The robot (Fig. 2) must decide which one of the paths is shorter and solve the maze through that path.

IV. Algorithm

Choosing an algorithm for the maze robot is critical in solving the maze. In this exercise, flood-fill algorithm was chosen to solve the maze due to its balance in efficiency and complexity.

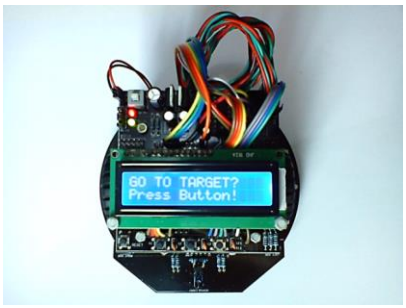

Figure 5. Mobile robot from above view.

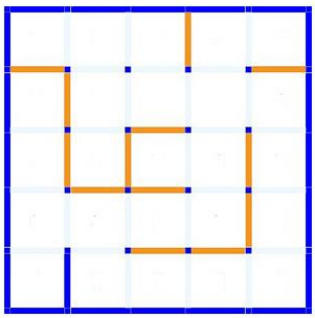

Figure 6. Design of the maze.

Figure 7. The maze.

Mapping the maze which has size of 5x5 cells is accomplished by using two-dimensional memory array with a size of 5x5. Artificial intelligence program requires two memory arrays 5x5. The first memory array is used to store information in each cell walls of the maze. The second array of memory function is used to store the cell value information in each cell. The position of the robot in the program expressed by the coordinates (row, column). The movement of the robot in the array is done to position the robot as in Fig. 8.

If the robot moves one cell to the south, then the coordinates of the line increases 1. If the robot moves one cell to the West, then the coordinates of the column will be reduced by 1. If the robot moves one cell to the North, then the coordinates of the line will be reduced by 1. If the robot move one cell to the East, the coordinates of the column will increase 1. The initial conditions of the robot, already has information about the initial position, the initial orientation, the size of the maze, and the existence of the outer walls of the maze.

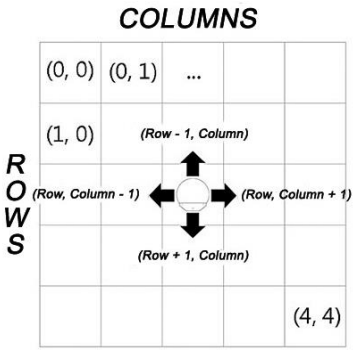

Figure 8. Array of robot movement

There are four main steps in the algorithm; wall data updates, cell value updates, the smallest neigbour cell calculation, and moving to the smallest neighbour cell.

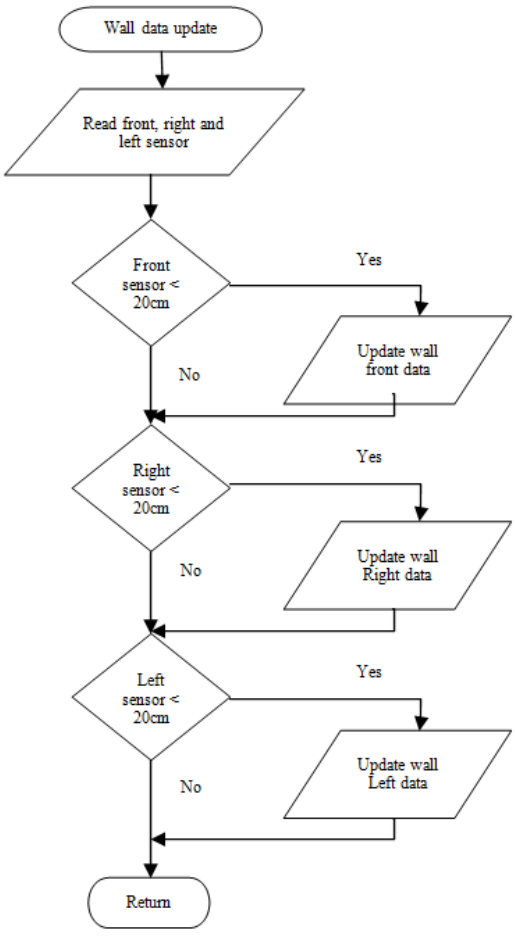

Figure 9. Flowchart for updating wall location at each cell

A. Wall Data Update

If robot decides where it wants to move to, it will check if it is surrounded by any walls in any of the three directions: front, right nad left. The robot will read the distance of any obstacle at each direction and check if the distance in each is more than 20 cm. The ones that exceed 20 cm are updated as "wall" on their respective side. It shows by the flowchart in Fig. 9. Robot also needs to know which direction it is facing. There are four orientations for the robot: north, south, east or west, as shown in Table I. Initial orientation was set at start and the robot keeps tracking of any changes.

TABLE I. ROBOT DETECTION WHEN IT DETECT WALL.

Robot Orientation	Detection Sensor		
	Right	Front	Left
South	West wall	South wall	East wall
West	North wall	West wall	South wall
North	East wall	North wall	West wall
East	South wall	East wall	North wall

B. Cell Value Update

Update value of the cell (restocked every cell with the new value) serves to adjust the value in each cell of the position of the wall that has been updated by the robot. The value stored in the array 2 dimensions of memory cell with size 5x5. Update the value of the cell is done by using the flood fill algorithm.

Update cell values subroutine works by resetting the values of the previous cell, then it will give a value of 255 in each cell, then fill in the values of these cells gradually, start value (level) 0 to all the cells filled grades. The cells that will be updated is the current_level array while neighboring cells will be inserted into the next_level array. After value fill in process is completed, then the cells are in next_level array will be moved to an array of fill in current_level to do next value. The update process will be complete if the value of the cell array next_level empty.

C. The Smallest Neigbour Cell Calculation

Subroutine specify the smallest neighboring cells function to search for a neighboring cell which has the smallest value. The smallest neighboring cell search is done on a priority basis, so that if there is more than one neighboring cell that has the smallest value, then the selected cells are cells that have a higher priority.

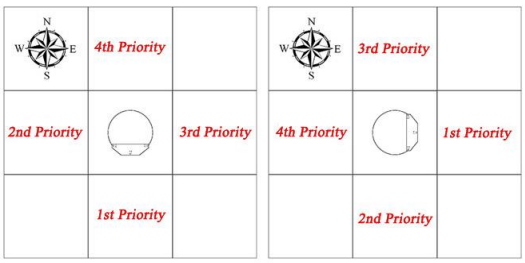

Figure 10. Priority of Neighbour cell

Prioritization is based on the movement of the robot is moving forward one cell has the first priority, move one cell to the right has a second priority, move one cell to the left has a third priority, and moving backward one cell has the fourth priority. For example, if the robot were facing the South, the South cells have a first priority, the second priority of the West has a cell, the cell has a third priority East and North cells have fourth priority as in Fig. 10. If the robot was facing the East, the East cells have a first priority, South cells have second priority, the North has a third priority cells, and cells West has fourth priority.

D. Moving to the Smallest Neighbour Cell

Subroutine moves to the smallest neighboring cells function to move the robot towards neighboring cells which have the smallest value, after the robot finds the neighboring cells. To perform movement to the cell, the robot should know the location of the cell. Furthermore, the robot will move to the cells by observing the orientation. For example, if the South cell is the smallest cell and orientation of the robot was facing west, then to move to the position of the cell, the robot must be turning left, then move forward as in Fig. 11. If the South cell is the smallest cell and robot orientation was facing East, then to move to the position of the cell, the robot must be spinning right, then move forward.

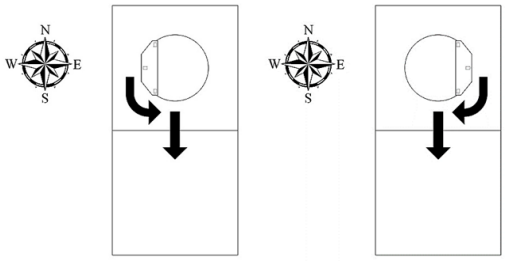

Figure 11. Moving to smallest neighbour cell.

V. RESULTS AND DISCUSSION

In this experiments, Robot will learn to find the shortest path from the starting cell (line 4, column 0) to the destination cell (row 2, column 2) and then back again to the initial cell. The initial orientation of the robot is facing the North.

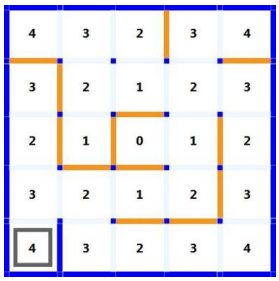

Figure 12. Simulation search path to cell (2,2)

The maze simulator program aims to facilitate the observation on how the flood fill algorithm. Fig. 12 is a view maze simulator program. Maze blue wall is a wall that position known to the robot. While the maze walls are colored orange wall position is not known by the robot.

Robot will perform a search of the initial cell lines (4.0) to the destination cell (2, 2). Flood fill algorithm

simulation results when a search of the cell lines (4, 0) to the cell (2, 2) are shown in Fig. 12 to 22.

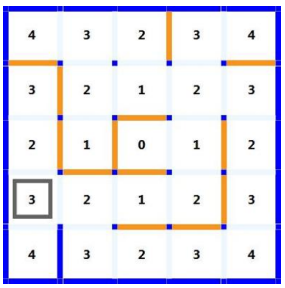

Figure 13. Simulation search path to cell (2,2)

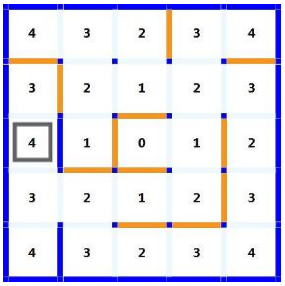

Figure 14. Simulation search path to cell (2,2)

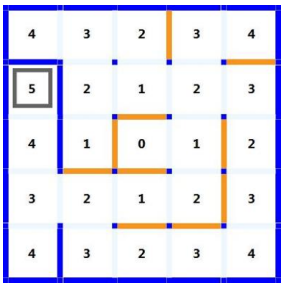

Figure 15. Simulation search path to cell (2,2)

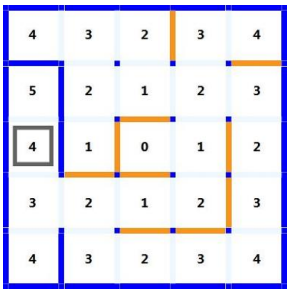

Figure 16. Simulation search path to cell (2,2)

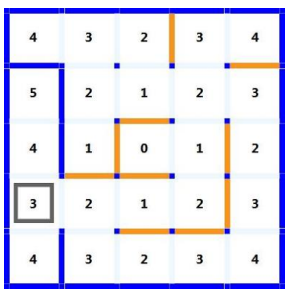

Figure 17. Simulation search path to cell (2,2)

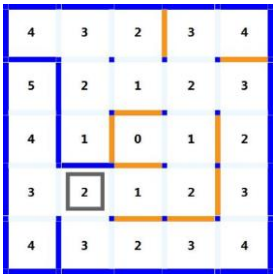

Figure 18. Simulation search path to cell (2,2)

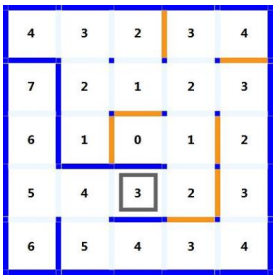

Figure 19. Simulation search path to cell (2,2)

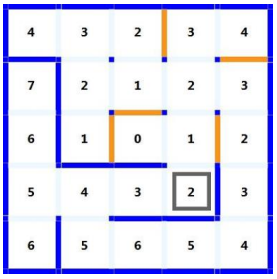

Figure 20. Simulation search path to cell (2,2)

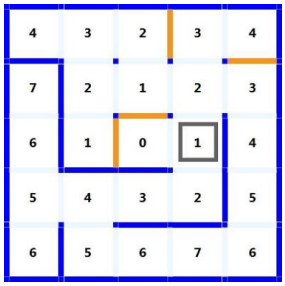

Figure 21. Simulation search path to cell (2,2)

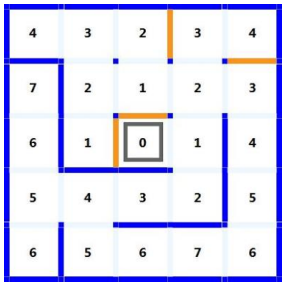

Figure 22. Simulation search path to cell (2,2)

After robot run the search and update his wall data, then it knows the shortest path to go to cell (2,2). It is shown in Table II.

TABLE II. FIRST AND SECOND ROUTES OF ROBOT EXPERIMENT

	Routes	Number of steps
First run	(4,0) →(3,0) → (2,0) → (1,0) → (2,0) → (3,0) → (3,1) → (3,2) → (3,3) → (2,3) → (2,2)	10
Return home	(2,2) → (2,3) → (3,3) → (3,2) → (3,1) → (3,0) → (4,0)	6
Second run	(4,0) →(3,0) → (3,1) → (3,2) → (3,3) → (2,3) → (2,2)	6

Wall map data will be updated when the robot go to cells that have not been visited before. Flood fill algorithm will update the value of the cell based on the position of the wall that has been mapped out by the robot.

Robots always perform movement to neighboring cells which have the smallest value. If there is more than one neighboring cell that has the smallest value, then the cell selection will be done on a priority basis. Go foward has first priority, turn to the right has the second priority, turn to the left has a third priority, and move backwards has a fourth priority.

The value is changed in accordance with the position of the wall that has been mapped out by the robot. Cell values represent the cell distance to the destination cell.

VI. CONCLUSION

This design and implementation of the robot is a study about the ability to equip a small mobile robot with the ability to learn how to navigate in unknown environment based on its own decisions. The flood-fill algorithm was found to be an effective tool for maze-solving of a moderate size. For the robot to make its decisions it relies on inputs from several sensors, namely the ultrasonic range sensors and wheel rotation decoders.

The robot has successfully able to map the maze in the first, return home and second runs. In its second run it reaches its target cell through the shortest route it has mapped in the previous first run and return home.

Future works may include to studying the robot's maze solving capability in a bigger and more complex maze. In order to improve the quality in wall detection, better object sensor, such as a laser range finder, is needed. It is much more costly but it have ability to scan its surrounding at a wirde angle plane, so it will help a lot in search ability at bigger and more complex maze.

REFERENCES

[1] B. S. Harapan, *Pencarian Shortest Path Dinamik dengan Algoritma Bellman Based Flood Fill dan Implementasinya pada Robot Micromouse*, Institut Teknologi Bandung, 2009.

[2] I. Elshamarka and B. S. S. Abu, *Design and Implementation of a Robot for Maze-Solving using Flood-Fill Algorithm*, Universiti Teknologi Petronas. 2012.

[3] I. Elshamarka and A. B. S. Saman, "Design and implementation of a robot for maze-solving using flood-fill algorithm," *International Journal of Computer Applications*, vol. 56, no. 5, pp. 8-13, October 2012.

[4] A. Ansari, M. A. Sayyed, K. Ratlamwala, and P. Shaikh, "An optimized hybrid approach for path finding," *International Journal in Foundations of Computer Science & Technology*, vol. 5 no. 2, pp. 47-58, March 2015.

[5] K. Sharma and C. Munshi, "A comprehensive and comparative study of maze-solving techniques by implementing graph theory," *IOSR Journal of Computer Engineering*, vol. 17, no. 1, pp. 24-29, 2015.

[6] R. K. Sreekanth, "Artificial intelligence algorithms," *IOSR Journal of Computer Engineering*, vol. 6, no. 3 September-October 2012.

[7] C. David, *Intermediate Robot Building*, New York: Apress, 2010.

[8] M. Per. *Design of an H-Bridge*. [Online]. Available: http://axotron.se/index_en.php?page=34

[9] M. A. Mazidi, S. Niami, and Sepehr Niami, *The AVR Microcontroller and Embedded System*, New Jersey: Prentice Hall, 2011.

[10] B. Thomas, *Embedded Robotics*, Berlin: Springer, 2006.

[11] Rizqiawan, Arwindra. *Sekilas Rotary Encoder*. [Online]. Available: http://konversi.wordpress.com/ 2009/06/12/ sekilas-rotary-encoder/

[12] S. Paul, *Practical Electronics for Inventors*, New York: McGraw-Hill, 2000.

[13] G. W. Lucas. (June 2014). A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators. [Online]. Available: http://rossum.sourceforge.net/ papers/DiffSteer/