
Rat's Life Competition Entry

Thavidu Ranatunga
ANU-UCSD EAP Exchange Student
Department of Computer Science
University of California, San Diego
tranatun@ucsd.edu

Abstract

This report presents an entry into the Rat's Life robotics competition. The competition pits pairs of robots in a maze with the objective of out lasting the other robot. The task is measured by an energy level which may be recharged from sources within in and unknown maze configuration. The robots are equipped with limited vision and distance sensors. We discuss the methods and strategies involved in an entry; covering the obstacles and various aspects that are required including motor control, mapping, and vision and distance-based sensing.

1 Introduction

1.1 Competition

The objective of the project is to create a competitive entry into the Rats Life competition running at

<http://www.ratslife.org>

The competition is defined in both simulated and real platforms however we will be referring to the simulated platform hence forth.

The competition runs matches of two robots in a random maze configuration containing power sources ('Feeders'), walls, landmarks and open spaces. The robots have a simulated 'energy level' which depletes over time and the robots must recharge at power sources in order to stay active. 'Feeders' require time to recharge once they have distributed their allocated power. The objective of the competition is to outlast the opposing robot.

In the current state of the competition, there are only 6 maze configurations being selected from, however there are intended to be more.

1.2 Robot Description

The robot is designed to reflect the e-puck educational robot (www.e-puck.org) and as such has 3 types of sensors:

- 1 Camera
- 8 Distance Sensors
- 1 Accelerometer

- 2 Wheel encoders

The camera is mounted on the front of the robot, centered near its top. It can only output a resolution of 52x39 and returns an RGB colour format.

The distance sensors are IR-based and only have a specified range of 4cm. This represents roughly one tile on a standard map. They are positioned around the circumference of the robot.

The wheel encoders measure the turn of the wheels not necessarily the actual movement of the robot.

The accelerometer returns the acceleration in the x, y and z axis directions in units of m/s^2 .

The robot also has 2 outputs: 10 LED lights positioned around the circumference of the robot and 2 wheels. The 2 wheels are positioned at the base of the robot such that turning on the spot is possible.

1.3 System Setup

We used Webots version 6.0.1 for the simulating the robot and its environment. The original competition ran in Webots 5.8.0. The code for the robot controllers are written in Java 1.6 update 11.

2 Methods & Strategies

2.1 Movement

The robot moves based on how much power is given to the wheels. Basic motor control is required to turn and move the robot into various positions. Turning can be achieved with applying an opposite polarity to the power of each wheel. Movement back and forth is done by having the same polarity on the power of each wheel. This is illustrated in *Figure 1*. The movements of the wheels are measured by the wheel encoders; however it only measures the turn of the wheels as opposed to actual movement of the robot. As such this leads to inconsistencies and is not always able to be a good measure of actual motion. To reduce this in post-movement it is possible to use the accelerometer to slightly enhance this functionality but this returns the acceleration values and may not be fine enough to be very useful. Alternatively it is possible to use the distance sensors pre-movement to test that there aren't any obstacles in the direction of movement such that for example the robot doesn't try to push against a wall (which would increase the wheel encoder amount but not actually move the robot).

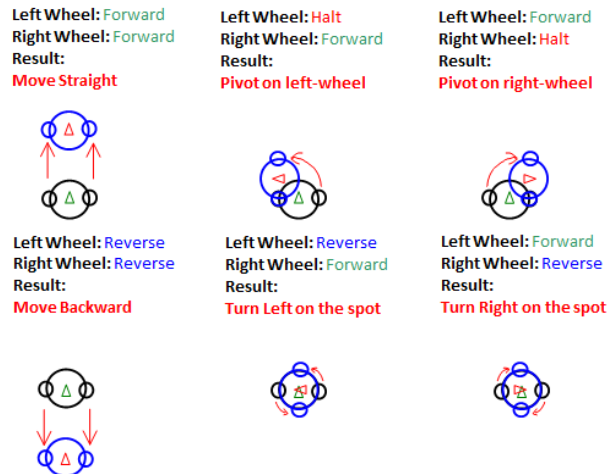


Figure 1. Basic Motor Control

All sensors are only updated every cycle of the controller's time step and thus it is not possible to stop moving at an exact measurement of the wheel encoders. However this can be compensated for and the robot can be re-aligned by other techniques such as using the camera for a vision-based measurement. There are black painted crosses at constant intervals on the ground to form tiles. These can be used to aid in measuring actual motion of the robot.

On a larger perspective of movement, we consider two ways of moving through the maze: fluid motion and block motion. The difference is illustrated in *Figure 2*.

In block motion the robot only moves in square tiles or blocks, as designated by the black painted crosses on the maze floor. All robots start in the middle of one such square in all maze configurations. This method is considerably easier to create and maintain a map with as it is possible to just store and reference it as an array or table. The drawback of the method is due to the inaccuracies of movement described above. It is difficult to get the robot to move in discrete squares and as such it needs to be compensated for and realigned over time.

In fluid motion the robot moves fluidly without any restrictions. This is faster as there is little to no stopping whilst travelling, and movement isn't restricted to moving horizontally and vertically. However this inherently makes it much harder for mapping and localization, and techniques such as dead reckoning are required to be used. The problems regarding alignment within the tiles no longer exists, however the inaccuracies with the wheel encoders still exist. This can have less impact here than in block motion, as it is more easier compensated for, by having an acceptable region of error.

Both methods of motion are viable with different strategies, but we decided to use block motion as it we estimated it would have a better return on interest for the amount of time available for this project.

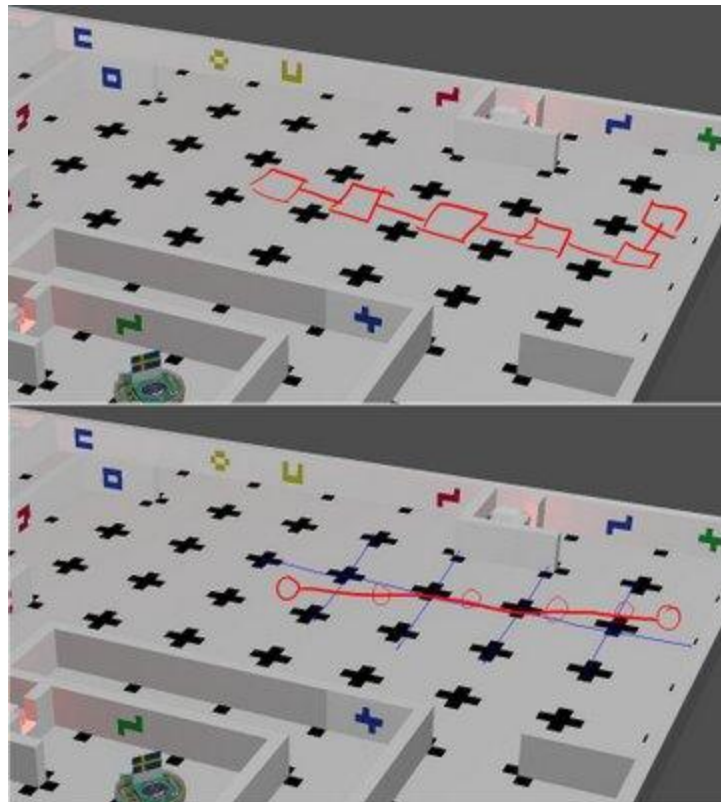


Figure 2. Top: Block Motion, Bottom: Fluid Motion

2.2 Distance Sensors

The distance sensors on the robot are IR-based and short range, with the specification listing it as being about 4 cm. In terms of actual maze tiles however, the robot is at best able to only detect the 4 walls on sides of the square. The e-puck robot is circular and the sensors are positioned along its circumference. During normal operation without obstacles nearby the sensors output a value somewhere between 0 and 80, this value changes constantly. At closest the value ranges into the 3000s and at worst the lowest value you can hope to get with a wall nearby is around 100. **Figure 3** shows an example reading of sensors from the e-puck; the distance sensors are in blue.

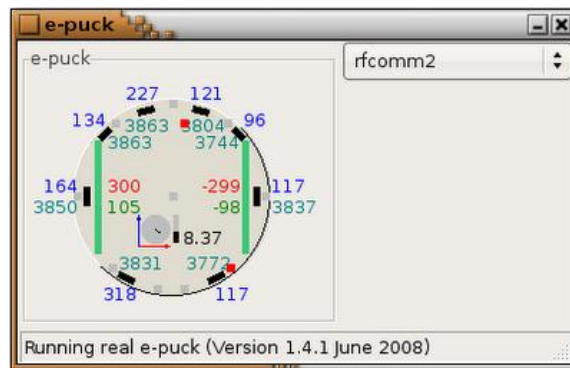


Figure 3. Example sensor reading from e-puck

The sensors can be used to detect obstacles, and fluid motion is especially dependant on the distance sensors for traversing the maze without getting stuck. For both fluid and block motion, the sensors are essential in the case that the opposing e-puck robot is in your robots path. If the robot doesn't stop when an obstacle is detected then the wheel encoders will continue to increase but the robot would not move.

When travelling with block motion the distance sensors can be very useful for mapping, however if the robot varies out of alignment too much, they are become horribly inaccurate. In order to compensate for this, a combination with vision based techniques using the on-board camera is effective.

2.3 Vision

The e-puck robot contains a small camera capable of outputting at 52x39. The resolution is rather small so it is difficult to do complex computer vision techniques with it. However it still provides some very useful information and the small size actually allows for really fast calculations.

The top line of the walls can be used in contrast with the background of the simulation to obtain an estimate of how far away a wall is from the robot. Firstly the vertical line directly in the center of the image can be used to detect how far away the closest wall directly in front of the robot is. Using a single vertical scanline and detecting at how many pixels from the top the grey level changes from below 100 to above 150 will tell you the distance. It is noted however that the e-puck robot seems to bob up and down whilst hovering on one spot, and as such there is sometimes a frame which has a half grey pixel between 100 and 150; this offsets the recorded distance values by 1. After this first wall it can then be determined how many walls to the left and right of the center wall we can obtain information about. We use more vertical scanlines starting from the outside and working towards the center to determine the distance of any other discernable nearby walls. The **Figure 4** below illustrates this flow of detection.






	2 Square gap between robot and wall 16 pixels from top 3 walls visible (either front 3, or straight + 2 sides) +4 unclear walls (due to resolution)
	2 Square gap between robot and wall 16 pixels from top 3 walls visible (either front 3, or straight + 2 sides) +2 unclear walls (due to resolution)
	2 Square gap between robot and wall 16 pixels from top 3 walls visible (either front 3, or straight + 2 sides)
	1 Square gap between robot and wall 8 pixels from top 1 wall visible (one in front)
	0 Square gap between robot and wall 0 pixels from top 1 wall visible (immediate only)

Figure 4. Vision based wall distance detection

Another use of the camera is to re-align the robot after traveling. A frame can be examined and split into portions as shown below in *Figure 5*. The amount of dark (non-grey) pixels that are visible from the painted crosses in the image can be counted and then be used to determine if there is imbalance to one side or the other; ie. whether the robot is sitting too far to the left or the right in its square). The left and right speeds for the wheels can be adjusted during the next movement to compensate for this.



Figure 5. Divisions for re-alignment

Finally, probably the most important use of the camera is to search for an active feeder. Feeders emit a bright red light when active (Shown in *Figure 6*), and turn it off returning to its dull red paint when it is recharging. The simple sample robot that comes with the Webots package uses a simple but efficient pixel scan of the vertically-middle 3rd of the image to search for an arbitrary number of bright pixels.



Figure 6. An Active Feeder

2.4 Mapping

Maintaining and using a dynamic map of the maze is essential for the any robot to be successful. Since we chose to use block motion for the robot, it is relatively easy to map using a multi-dimensional array. The existing maze configurations all have a size of 10x10 tiles total, with walls being in-between tiles instead of occupying one square. An example maze is shown in **Figure 7**. The starting location is chosen at random from a given set of coordinates for a maze and this set can contain coordinates from anywhere within the maze. Since the initial environment is unknown to the robot, it must allocate an area double the size of mazes (10x10 currently) to be able to start at any tile in the maze. It can then go to the edge of the maze in any direction from any start tile and not be out of bounds.

The map is built from information gathered from the vision and distance sensors at each step taken through the maze. As described in the vision section the robot can estimate the distance of certain walls in certain conditions and be able to contribute to the map. The distance sensor can provide information regarding the immediate surroundings of the current square. Combining this information it is possible to build a map to work with.

Currently the contest only has 6 maze configurations and as such it is possible to integrate the known maze configurations as background information to the robot prior to competing. This is better than a blank map, and once identified as the correct configuration, is a huge source of information that greatly aids the strategy of the robot. The solution however is not dependant on this information being available, and is designed to work in a blank map situation too. In that case a standard graph search algorithm such as a modified breadth first search can explore the maze by moving to the closest unexplored square. Once more and more information is known a standard graph search algorithm, A*, is used to travel to the nearest feeder, recharge, and continue exploring again. An alternative to the closest unexplored square is to use A* to path to the center of the most unexplored area of the maze. The robot can keep track of the time since the feeder was last deactivated and travel in between to compensate. This is discussed more in the strategy section.

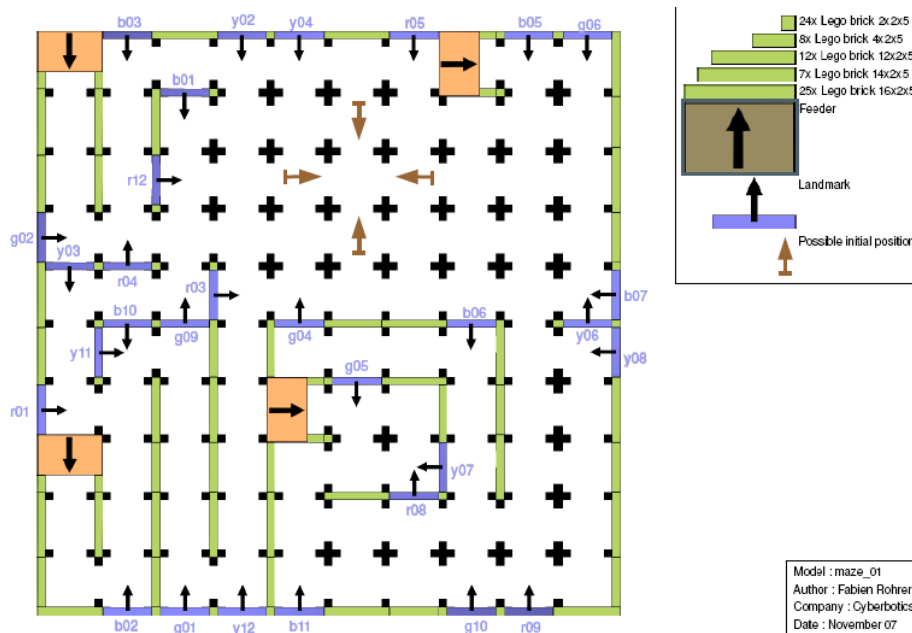


Figure 7. Example maze configuration. Source: www.ratslife.org

2.5 Strategy

In order to be able to be competitive, the robot needs to be able to

- Create and maintain a dynamic map of its surrounding environment

- Identify ‘feeder’ power sources and recognize whether they are an active or inactive state
- Use the gathered information to traverse the maze efficiently and without crashing into walls
- Manage its power supply so as to stay active as long as possible

Possible strategies for entries to Rat’s Life vary from random/luck-based to more systematic approaches, and both have their benefits depending on which maze configuration arises. Thus a balance needs to be sought between the two, and a good strategy is possibly one that can switch or adapt based on the observed state of the maze as it happens. Several of the strategies that appeared in the original Rat’s Life competition as it evolved are described in [1].

Navigation:

Even though mapping wherever you move is essential, choosing where to move isn’t so clear. If the map is preloaded into the robot as described in the previous section, then movement is simply just navigating to the nearest feeder. However when the environment is being discovered as the robot moves, it has to decide on a method to explore. One such method is the simple but very efficient right-hand method, where you simply follow the first wall on your right hand side (or left) and thus map out a lot of the walls in the maze (the exception being islands of walls separate from the rest). You could also go to the nearest unknown square on the map you are generating or navigate to the area of most unexplored squares. You can always just travel randomly too.

Feeders:

Feeders, being the most important part of the maze towards our objective, form an important point in strategy. Feeders deactivate once used and take some time to recharge. Each rat has 2 batteries (2 lives?) which drain sequentially, each initially containing an arbitrary a level of ‘100’. The feeder recharges the active battery to full but if a battery has been drained already then it is unaffected. One popular strategy noticed in other competitors (see results) was to sit in front of a feeder, just out of its charging range but blocking all other rats from reaching it. Then when the current battery is about to run out, it will move forward slightly and charge it, before moving onto another feeder. This makes the most efficient use of the feeder since your battery will be full while you travel to find the next feeder, and keeps the rat alive for longer (taking the most charge out of the feeder). However the time for a feeder to recharge is faster than the time it takes for you to lose one battery, so at most you can get 2 charges from sitting at one feeder the entire match.

3 Results

3.1 Sample Robot

The sample robot controller/rat that comes with the Webots package is a simple robot that consists of a 2 part strategy:

- Use distance sensors to avoid obstacles; it multiplies the distance sensor readings with a set of weights so as to slow down wheel power respectively and avoid. If an object is directly ahead, it will randomly choose to turn left or right.
- Identifying feeders; Scan for a red light in the middle section of the screen, if found it adjusts wheel power to try and make the average amount of red pixels to be in the center of the screen.

Other than the above, its movement is fairly random. Hence in this section we are using the remaining battery life (of the winner) when we win as a metric for the effectiveness of the bot.

The robots that I used were the following:

- tMaze - A robot with the 6 mazes preloaded into memory, it spins around identifying which maze it is in at the start, then uses a search algorithm to navigate to the nearest feeder. After deactivating, it finds a new feeder.
- tMaze, Sit - Same as tMaze, however once it reaches a feeder it sits there and waits for it to recharge.
- tNoMaze, Random - A robot without the mazes preloaded, it uses its camera and distance sensors to detect walls, map and navigate its environment. Its movement is random until it finds a feeder, at which point it will charge from it and record its location.
- tNoMaze, Nearest - Same as tNoMaze,Random, except that it travels to the nearest unexplored square each time.

Some test matches were run against the sample bot on Mazes 1 & 2 which have a balanced amount of open space and tight corridors. 5 examples matches per pair were done and the results were averaged, as shown in *Table 1*.

Table 1. Results against the sample bot

Maze	Bot0	Bot1	Winner	Avg Remaining Battery
1	tMaze	Sample	tMaze	50
1	tMaze,Sit	Sample	tMaze,Sit	40
1	tNoMaze,Random	Sample	Draw	-
1	tNoMaze,Nearest	Sample	Draw	-
2	tMaze	Sample	tMaze	60
2	tMaze,Sit	Sample	tMaze,Sit	50
2	tNoMaze,Random	Sample	Draw	-
2	tNoMaze,Nearest	Sample	Draw	-

Unfortunately even with all the attempts to rectify going out of alignment, the robot bugs and gets stuck incredibly often. At first it is okay but usually about halfway between finding the first feeder and travelling to the second one, it has gotten too out of alignment and gets stuck on a wall, or maps itself incorrectly.

Because of this failure, sitting at the first feeder guaranteed it lived for 2 lives, but no more than that (as the feeders recharge slower than your battery drains). However constantly moving to a new feeder runs the risk of it getting stuck but with the prospect of being able to live indefinitely.

3.2 Proving Grounds Official Ladder Result

There is an unofficial contest for the out-of-season period (of the official contest) at

<http://www.theprovinggrounds.net/ratslife/>

They run one match a day in a ladder system such that the lowest person fights the next person above them and so on. Thus each rat gets one more match for every win, but only one match when you lose. A rat similar to tMaze in the previous section was submitted here for 2 days as of writing, but unfortunately the first day's rounds weren't met properly and the second day was the rat lost terribly and thus only faced one match. The main reason for the loss is that on the 3 later maps (Mazes 3-6), the configuration has many tight corridors and not very much room to move. This also makes identifying the mazes difficult in close quarters and there was a bug unfortunately rendering it unable to do so.

3.3 Proving Grounds Competitors

Thanks to Anthony Morse of the provinggrounds.com we were able to obtain 4 anonymous competitor's entries to test against. The competitors were matched similar to section 3.1 and the results are shown in Table 2. This time average battery remaining wasn't used a metric since all competitors are intelligent enough to find feeders fairly regularly and as such the battery was always being refilled. Each match was run 4 times instead and win rate is listed instead. A description of each bot by observation is as follows:

- Competitor A Uses the left hand wall method, follows each wall very tight and moves very fast. Waits at feeder once found for battery to drain and blocks competitors from reaching it.
- Competitor B Similar to A however its wall following isn't as efficient and it gets confused going in loops occasionally, or gets stuck in wall corners.
- Competitor C Also uses the left hand wall method, but is incredibly fast and due to its speed and the relatively close starting positions- always seems to rush forward and confuse the opposing robot as a wall, running around it in a circle continuously! It then panics when less than 30 battery and darts in a random direction, usually finding a real wall, and then winning.
- Competitor D Similar to A. However not as fast, and doesn't wait quite as long in front of a feeder before using it.

Maze	Bot0	Bot1	Winner	Win Rate
1	A	tMaze	A	100%
1	B	tMaze	tMaze	75%
1	C	tMaze	C	100%
1	D	tMaze	D	100%
3	A	tMaze	A	75%
3	B	tMaze	tMaze	75%
3	C	tMaze	C	100%
3	D	tMaze	D	75%

All four of the competitors were incredibly fast as they were only checking for a feeder and following a wall. Other than that they each gave attention to their battery level and did a random dart when it had gotten low without them spotting a feeder. Their strategy was simple and effective. The block motion I used was terribly ineffective as it often caused the robot to get bugged out over time since it had lost its alignment over time. It was also a bit slower, and the opposing robots sometimes beat it to reaching a feeder when it was almost there. Opposing robots also tended to block the feeders quite often preventing my robot from getting there (as it was slower). It was unclear which of the other robots mapped and which just flowed from wall to wall.

Maze 3 faired better for me as there were wall islands in places and a lot more convenient arrangements of walls in my favour. The competitors would sometimes get stuck in loops.

Finally, since starting positions of the 2 rats are often very close to each other. Competitors A and C ran so fast that they confused my robot as a wall initially, and ran circles around me in a loop until both battery levels were less than 30! This was horrible as (though hilarious to watch) as both robots were stuck. My robot couldn't identify its surrounding maze as whenever it turned it would always see the competitor and take bad readings. The competitor would just run around and around until it reached its panic level, at which point it would leave and win on its second battery, whilst mine was too confused to continue.

4 Future

If there was more time I would like to have attempted to see what may come of using fluid motion in a lot more detail instead of blocks. It would require a whole new approach as the strategies and techniques would be considerably different.

Block motion was a large part of the downfall of my robot as it really is too difficult to keep in alignment. Even if you can fix it without outside interference, another robot can nudge you or block your way and mess with your alignment (even if by accident, in circles...).

In the future of the actual competition there has been announced that more contrast in the colour scheme will be given so as to aid vision based solutions. There is also more planned changes to make the contest more interesting however they have not released details on what those might be. The second edition of the contest is set to happen sometime in the first half of 2009.

References

[1] Michel, O., Rohrer, F. (2008) The Rat's Life Benchmark: Competing Cognitive Robots. *European Robotics Symposium*