

Robert or Pitty-Mouse? Depends on Win or Lose!

Nevine AbouGhazaleh, Ali Alanjawi, Peter Djalaliev, Frank Liberato, Thomas D. Torsney-Weir, and Daniel Mosse

Department of Computer Science
University of Pittsburgh, Pittsburgh, PA 15260

1 Introduction and General Strategy

This cyber mouse following the beacon (or is it bacon? should it not be cheese?) is an extremely interesting exercise. It has taken many interesting nights to figure out what are seemingly simple points. However, they are more complex than we initially thought, due to several issues. Here are some of them:

- sensors are very noisy: this turned out to be bad for programming, but very good for thinking.
- We have not yet considered where on the robot we will place the sensors. There are several approaches for this particular problem, but we have not yet tackled this problem.
- discover beacon to start with: Section 2
- threaded software architecture: high, medium, and low priority threads. High-priority thread is the receiving of information and feeding this information to other threads. The medium priority is the sending of commands to the simulator (see Section 4), every 23 ms. Clearly, the medium priority is the highest priority in the system, since it strictly follows the high-priority thread. Low-priority threads are those that update the map (see Section 3), discovery the best path (see Section 3).

2 Beacon Discovery

We need to locate the bacon to direct the cyber mouse towards it. Our strategy for beacon discovery is based on two facts: (a) the sensors, including the beacon sensor, can be very noisy, and (b) the robot cannot go towards the beacon unless the robot knows at least approximately where the beacon is (duh! descubriu a pólvora!).

The main idea behind beacon discovering is through locating the beacon direction from two robot positions, called reference points. Reference points are any points on the robot path where beacon is visible. Beacon discovery undergoes two main phases: detecting a beacon signal, and locating the beacon using reference points.

Detecting beacon signals is possible if beacon is within the angular sensing range of the mouse and there are no high obstacles between mouse location and the beacon. Prior to locating the beacon, multiple beacon

readings are required. In case the robot is located where it can not sense the beacon, we start by rotating the robot 360 degrees, and sense whether the robot can see the beacon. If it detects a beacon, the robot stops turning and starts sensing the direction for 9-13 cycles, to get rid of the Gaussian noise in the sensors. We reduce the reading error by average multiple readings. The longer the sensing time, the higher the accuracy of determining the beacon direction. However, the sensing time create a trade-off between the accuracy of locating the beacon and the robot progress in reaching the beacon. Computing the confidence of the results, we have noticed that we can ensure that, after 13 readings, the beacon is within 3 degrees of the average of the readings, with (approximately) 80% confidence. We repeat this process for the two reference points.

Given the co-ordinates and the beacon direction at the two selected reference points, we can compute the location of the beacon using triangulation as shown in Figure 2. We construct two straight lines. Each line connects one of the reference points to the beacon. The beacon location is the intersection of these two lines. We use the obtained beacon directions (relative to the +ve x-axis), α_1 and α_2 , at the reference points to determine the slope of each of the two lines.

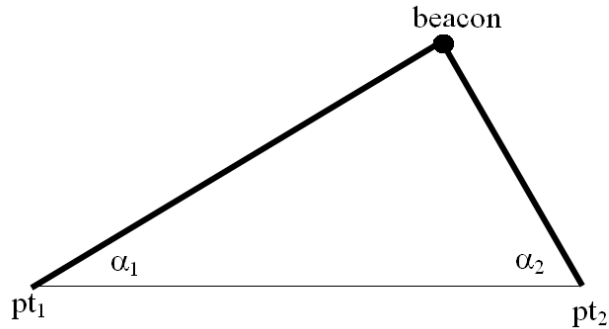


Figure 1: Locating the beacon position using two reference points (pt_1 and pt_2).

We compute the beacon coordinate (x, y) using the equation below. Once the beacon position is identified, the navigation of the cybermouse is steered towards this location.

$$x = \frac{c_2 - c_1}{\tan(\alpha_1) - \tan(\alpha_2)} \text{ and } y = \tan(\alpha_1) * x + c_1 \quad (1)$$

where c_1 and c_2 are the line constants.

3 Map and Path Discovery

We decided to follow the Pitt strategy in 2005's competition, and subdivide the area into squares (cells). Each cell has a .25u side.

Our map discovery, much like the beacon discovery, is based on how bad sensors are (and boy, are they unreliable!). Because of the wide angle for the sensors, and the noise added to the measurements, our wall detection algorithm only trusts measurements that are returned within 1u of the wall (after conversion). This allows us to create a fairly good map, but has the disadvantage of taking a long time to populate the map (cells).

We also tried different algorithms for the map discovery, including populating the cells with a probability of having a wall, depending on the distance from the reading. For most readings, this is pretty accurate, but it turns out that the outliers create havoc in the conversion of values. It is also hard to reconcile different values read from different positions on the grid/map, and our collective hours dedicated to this problem yielded no great solution.

Every time the map gets updated, we run an exhaustive search algorithm in the approximately 10,000 cells, and determine what the best path to reach the beacon is. Clearly, our path finding is very myopic, given that the map discovery is also myopic due to the noise in the sensors that provide a distance to the next obstacle. Nevertheless, we will use the approach by the RTSS 2005 MaRTian Task winners: first start with the target and expand paths outward; as the robot progresses, check if a shortcut can be applied to the best path so far.

4 Robot Movement

Going and Stopping To start moving and stopping the robot, we implemented certain functions, namely to make the robot go a certain distance, turn (in both rads and degrees!), carry out a spiral (move in a curve, but with an increasing radius), among others. These building blocks help the development of the program.

Following Walls The main goal of following a wall algorithm is to find a passage to go behind it, as soon as possible. Assuming that we know the location of the beacon (beyond the wall), a wall becomes an obstacle that we need to overcome, thus, passing a wall is a goal as if we have to go through it. If the wall is simple with no corners, and the robot location is close to the wall and facing it, the algorithm would start by choose a side to turn to (left or right), and drive in a straight line until it can make two turns on the other direction. That will place the robot in a location behind the wall, and the wall is no more an obstacle in the path of the robot.

We implemented a more complex wall-following algorithm, based on past CiberRato competitions: start a counter at 2 when turning left, and decrease the counter every time the robot turns right. This will allow for the mouse to be placed “on the other side” of the wall. The algorithm will terminate when the counter’s value is one less than its starting value. This simple algorithm will work even in a complex wall setup, like the one in Figure 4, and all it needs is a simple counter that keep track of the turns.

Figure 4 illustrate the expected behavior of the robot implementing this algorithm. The numbers near the corners depict the value of the counter. Assuming the counter starts with a value of one, then the algorithm will terminate if the counter is zero.

A problem with this algorithm arose when we have corners which are not at right angles. This problem can be solved by using information from the map discovery algorithm to figure out the angle of the corners, and hence the degree of the turns. Another problem is when the wall’s interior is not the same as its exterior. In the case the algorithm will terminate in the robot being in a location exterior to the wall, but not at the goal location, which is just behind the wall in a point opposite to the starting location. The use of the path discovery comes to the rescue to control the robot to drive to its target, since we are not trapped inside a wall chamber.

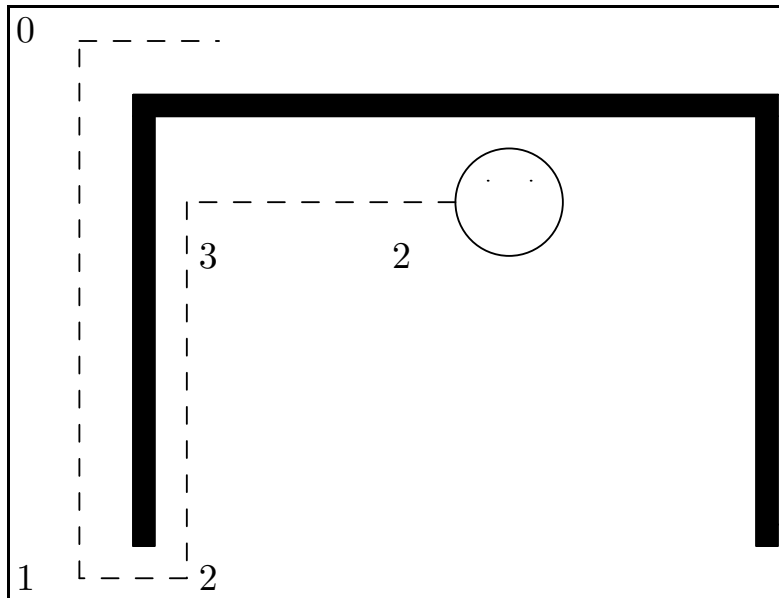


Figure 2: Illustration of the follow-wall algorithm.

5 Conclusions

This is a very interesting exercise and the authors are thankful to the organizers for putting it together. It is a great brain teaser, specially with this much noise in the sensors (the authors are **NOT** thankful to the organizers for putting in this much error!!).