

# Numbat: an Entry to CiberMouse@RTSS2006

Leonid Ryzhyk      David Snowdon      Stefan M. Petters

National ICT Australia \*  
Sydney NSW 2052  
Australia

`{firstname.lastname}@nicta.com.au`

## Abstract

CiberMouse is a robotics contest held at the Real-Time Systems Symposium in December, 2006. The contestant's aim is to write a program which can direct a simulated robot within a maze. Within this paper we describe our approach to the problem posed. Our solution involves localisation via dead-reckoning, mapping via a probability field, and a layered path planning approach. These algorithms combine to allow the robot to navigate to its target, and return to its starting point, in a small amount of time.

## 1 Introduction

The CiberMouse contest [1, 2] poses a problem for its contestants: control a robot, such that it can discover and reach a target and return to its start position. Three robots compete in any given round. The robots are navigating a maze using various sensors. The challenge is complicated, since all sensors and actuators are subject to Gaussian measurement and control errors. Each robot contains four infrared sensors to detect obstacles in the maze, a beacon sensor, which is used to track the direction of the target area, a compass, and collision detectors.

There are very few real-time performance concerns, since the processing power available is significantly more than that required, and there is a well-defined time quantum in which a decision about the subsequent actions must be made. The only other issue is the obstacle

avoidance requirement, which necessitates a maximum speed, which may not be exceeded.

The subsequent sections give a brief overview of our approach and then go into detail regarding the individual components of our solution. The conclusion summarises our current state and suggests further optimisations which we intend to implement prior to the competition.

## 2 Approach

The problem can be effectively decomposed. Localisation is the process of estimating where the robot and beacon are. This is essential information, since we must be able to return to the robot's starting position. Mapping is the process of generating a map of the robot's world, given whatever information is available. Developing a model of the world allows for more efficient navigation. When planned navigation is not possible, because, for example, the target beacon has not been observed, a heuristic navigation is necessary.

### 2.1 Localisation

There are two problems to be solved with regard to localisation: the robot's position in the virtual world must be estimated, and robot's target destination (the beacon) must be determined.

The problem of localisation and mapping, which is often required in tandem in an unknown world, is well known. Its difficulty varies depending on the type and accuracy of the sensor and actuator data available. One particularly successful approach, which relies on good odometry and a laser rangefinder, is DP-SLAM [3]. This particle-based approach is inappropriate for the

---

\*National ICT Australia is funded by the Australian Government's Department of Communications, Information Technology, and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Research Centre of Excellence programs.

CiberMouse competition because of the inaccuracy and wide beam of our IR rangefinders. We have also considered the use of Kalman filters [4] for localisation. Kalman filters are useful when measurements for the position and orientation estimates can be taken. The errors in x and y estimation in our approach are more systematic as the subsequent errors in the estimates are highly dependent on the previous estimate. Hence we expect little benefit from the use of Kalman filtering.

We considered two approaches to the robot localisation problem: dead reckoning, and map observation. The latter would involve having an accurate map, and inferring from the observations of the surrounding obstacles exactly where the robot is. The former involves estimating the (linear and rotational) speed of the robot at each time-step given the power provided to the motors (which is set via our motion control algorithms). We can then integrate the speed estimates in order to calculate our position and rotation, relative to the initial position/rotation.

The situation is complicated by errors in the motor power: the motors will not execute exactly what was instructed. This noise is Gaussian according to the documentation. When integrated over the period of the robot's run, this noise can result in a significant error in the position estimation.

In order to estimate the robot's location it is necessary to use the robot's estimated orientation to calculate x and y deltas based on the robot's linear speed. This means that an error in the robot's estimated orientation can have a large effect on the accuracy of the location. In order to reduce this, the robot's compass sensor is used to periodically update the estimate of the robot's orientation. While the compass sensor can have a significant error, the error is Gaussian, and non-cumulative. Therefore the errors in the robot location estimate are also Gaussian.

One issue with using the compass sensor in this fashion is the sensor's latency of nine time-steps. In order to counter for this, the last nine estimates are saved. Each time a compass reading comes in, the estimate from nine time-steps ago is updated to reflect the compass reading, and the subsequent estimates are updated to reflect the newly arrived information.

The second localisation problem, beacon localisation, is based on triangulation. Each observation of the beacon (via the beacon sensor) is recorded, saving the estimated robot position and estimated absolute angle to the beacon (based on the robot orientation estimate).

In order to estimate the beacon location, the estimates

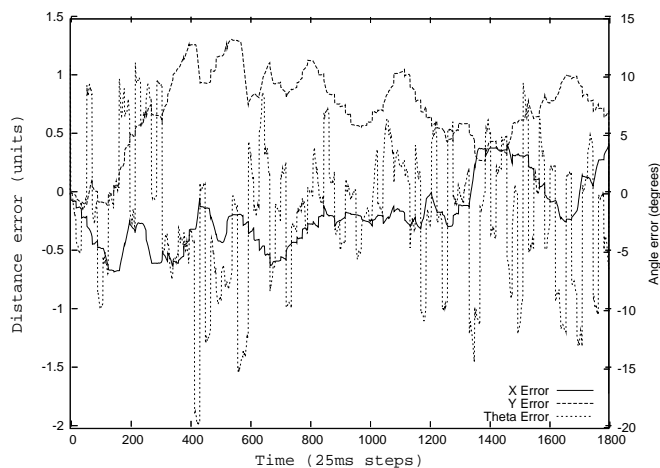


Figure 1: X, Y, Theta errors in localisation for a typical run

are taken in pairs. Linear equations are derived from the position and angle of each observation, and the point at which the lines intersect is calculated. This is performed for each pair of observations with a difference in angle of greater than 30 degrees (since for small differences in angle, a small error in the angle gives a large error in the position estimation). This problem can be solved more generally via the use of a number of criteria (e.g. distance-to-beacon) which can be used to calculate the potential error, allowing for a weighted average based on their potential error.

## 2.2 Mapping

The map is the virtual representation of the environment the robot moves in. As such certain abstractions can be made to minimise the memory consumption and run time of the mapping algorithm itself as well as the run-time of the planning algorithm on top. As with almost all robotics mapping problems a discrete representation was chosen. The mapping problem can be separated into several subproblems which need to be resolved.

- map granularity
- map size
- adding obstacles

At the time of writing, we were experimenting with the granularity of the map. The current implementation is a *0.1 ul* granularity, i.e. the minimum distance between two points in the map is one tenth of the robot diameter. Due to ease of internal representation and

complexity of operations performed the map has been chosen to be Cartesian.

The grid in the simulator may be configured at race start time, but the size of the playing field in the simulator is provided as  $28\text{ ul}$  in the x dimension and  $14\text{ ul}$  in the y dimension in the race rules [1]. As a result we have chosen the map to be  $56\text{ ul}$  by  $28\text{ ul}$ , placing our robot grid position in the centre of the map as depicted in Figure 2. This allows the simulator to have arbitrary grid positions, while ensuring that any playing field which is within the restrictions of the race rules, may be represented in our map. Additionally it allows for a position error of at least than 7 % as a result of the way we deal with obstacles described below. Since the grid is unlikely to be positioned on the very edge of the map, the real position error to be tolerated is even larger, but dependent on the grid layout.

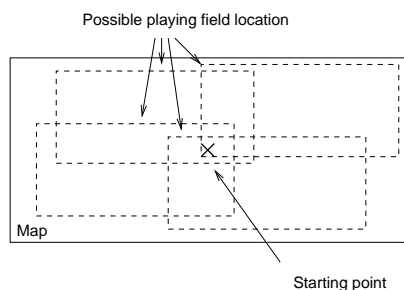


Figure 2: Location of Real Playing Field in Virtual Map

The process of adding obstacles to the map has to deal with four fundamental problems:

1. the robot localisation has errors;
2. the IR sensors have errors;
3. the IR sensors provide a single obstacle distance value for the the circle segment they are covering (e.g. they have a wide beam width);
4. obstacles may be moving (e.g. other robots).

The input data used for localisation is subject to Gaussian noise. During our experiments we have perceived a number of cases where the localisation error averages out. However, obstacles in the robot's map may be mapped in a different location to the simulator's representation because of the instantaneous localisation error. A similar problem occurs because of the IR sensors, which are also subject to Gaussian noise. Furthermore, the sensors measure the closest object within the circle segment defined by the IR beam width. Therefore, for

any given reading of the IR sensor, an obstacle can only be placed on a semi-circle, and nothing can be inferred about the obstacles occluded by that semi-circle. This results in obstacles being mapped in locations where there are no obstacles and vice versa. Finally the other robots form moving obstacles as there is no reliable way of distinguishing reliably between obstacles and opponents.

To circumvent the above problem we choose a probabilistic map. Therefore each piece of discretised space is assigned the probability of containing an object instead of a simple binary map containing either an obstacle or free space. In simple terms: when an obstacle is detected by the sensor, the map increases the probability of an obstacle in the corresponding location in the map and vice versa the probability is decreased when no obstacle is observed. The increment used for obstacles is weighted inversely to the distance between robot and obstacle to ensure we're not steering the robot into an obstacle. To reduce false positives the robot only adds obstacles to the map which are closer than  $1\text{ ul}$ . The probability of an obstacle is set to 0 in the position of the robot.

Finally the issue of effective path planning has to be addressed. The robot has a physical dimension which is much larger than one location in our map. This can be addressed in three ways:

1. take the physical dimensions of the robot into account when planning;
2. assume a robot which has a physical size of one field, moving; the sensors into the centre of the robot
3. *grow* the obstacles after detection by the radius of the robot consider the robot to be single field in size for the planning.

The first option has the drawback of being very *expensive* in the planning stage, while the second option tends to distort the map. the third option has the problem of a continuity gap between free area detected in front of the robot and the location of the robot marked as free. We have chosen the third option as depicted in Figure 3 as we consider this to be the most effective solution. The growing is achieved marking a robot sized area in the map as obstacle for each obstacle identified. This adds an additional buffer to compensate for estimation error, as the obstacle may now be outside the playing field spreading into the playing field as it us *grown*.

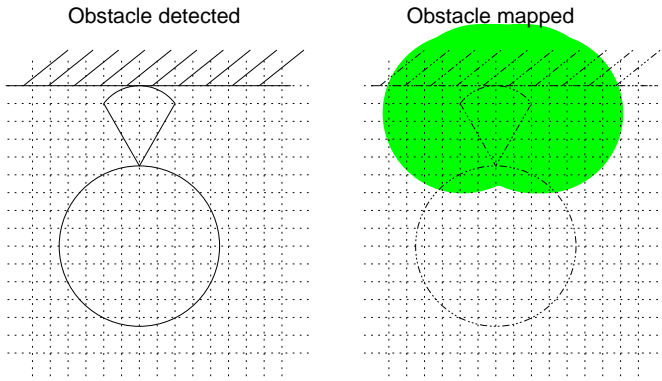


Figure 3: *Growing Obstacles in the Map*

## 2.3 Navigation

The planning component is responsible for controlling the robot based on information provided by localisation and mapping components. It implements a three-level control strategy as follows:

**Level 3: (Top level)** This level is responsible for updating the current *global goal* — the location on the map which the robot is planning to visit next, e.g., the beacon or the initial position. It also requests and reacts to readings from the ground sensor and controls the LEDs. It provides the second-level control algorithm with coordinates of the current global goal.

The global goal is computed as follows: when searching for the beacon, and if an estimate of the beacon location is available, it is used as the global goal. If only the direction to the beacon is known, the global goal is set to the current robot location plus  $1.0\text{ ul}$  in the direction of the beacon. If no information about the beacon location is available, the algorithm just continues exploring the maze. To this end, the maze is covered by an exploration grid with  $1\text{ ul}$  cells and the first unexplored node of the grid is used as an intermediate global goal. When returning to the initial position, coordinates of the initial position are used as the global goal. In case the robot is surrounded by obstacles, the map is cleared and mapping of obstacles starts again.

**Level 2:** This level determines the immediate direction in which the robot should move. It computes the shortest path to the global based on the map on each time step. The resulting path may consist of many small steps. Such a trajectory cannot be efficiently navigated, therefore the control algorithm computes the longest

straight segment of the path, originating in the current robot location, that does not intersect with any obstacles. This straight segment, called the *local goal*, is used as the input for the first-level controller.

**Level 1:** This level computes the power that needs to be applied to the wheels on the next step in order to move towards the local goal, based on the current robot orientation, output power generated on the previous step and the location of the local goal.

## 3 Conclusion

We have presented the state of our approach to controlling the robot during the CyberMouse contest. We have split the problem into three parts: localisation, mapping, and navigation. Currently our robot navigates the maze and returns to the starting point and is thus fulfilling the basic requirement of the contest. However, we think several optimisations are possible and we aim to implement those. Currently, planning is done using many small steps when following a wall. One optimisation is to step more boldly when faced with unknown territory and thus avoid approaching the wall every two or three fields in the map. Other particular areas of improvement include the heuristic used in the absence of a beacon signal.

## References

- [1] *Ciber-Mouse: Rules and Technical Specifications*, 2006.
- [2] L. Almeida, J. Azevedo, B. Cunha, P. Fionseca, N. Lau, and A. Pereira, *Micro-Rato Robotics Contest: Technical Problems and Solutions*, 2006.
- [3] A. Eliazar and R. Parr, “DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks,” in *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pp. 1135–1142, Morgan Kaufman, 2003.
- [4] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *Transactions of the ASME - Journal of Basic Engineering*, vol. 82, pp. 35–45, 1960.