Robot Localization and Map Construction Using Sonar Data

Vassilis Varveropoulos
vassilis@users.sourceforge.net
The Rossum Project
http://rossum.sourceforge.net

Introduction

An important challenge in small-scale robotics is finding a robot's position when only limited sensor information is available. There are many technologies available for robot localization, including GPS, active/passive beacons, odometry (dead reckoning), sonar, etc. In each approach, however, improvements in accuracy come at the cost of expensive hardware and additional processing power. For the robotics enthusiast, the key to successful localization is getting the best results out of cheap and widely available sensors.

This paper presents a method for localization and map construction of a mobile robot using data from a sonar-based range sensor. No prior knowledge of the environment is assumed. The map is constructed autonomously by the robot.

This method has been implemented and tested using both the Rossum Playhouse simulator (see reference [3]) and an enhanced Rug Warrior robot. Experimental results from both the simulator and the robot are presented at the end of this paper.

This paper and the positioning application bundled with the simulator, is publicly available from [1]. The source code will also be released.

The Positioning Algorithm

In its localization phase, the algorithm determines the robot's position by correlating a local map (generated by a range sensor sweep), with a global map. While the global map can be supplied in advance, this algorithm does not require prior knowledge of the robot's environment. Instead, it uses sensor data to construct the global map dynamically.

The algorithm estimates the robot's location by comparing the global and local maps. To do so, it computes positions called *feasible poses*, where the expected view of the robot approximately matches the observed range sensor data. It then selects a best fit from the feasible poses.

To evaluate feasible poses efficiently, the algorithm represents the global map as an occupancy grid (a matrix of cells, each having a value that indicates whether that cell is empty or occupied). Using its sensors, the robot determines range vectors (distance and bearing from detected objects or features) which are then compared against all occupied cells in the grid. If a range vector can be overlaid on the grid without interference by other occupied cells, it indicates a feasible pose. In addition to its occupancy value, each cell in the grid is also assigned a *certainty value* indicating the likelihood that the robot is located at that position. Each time a feasible pose is identified, the certainty value of the corresponding cell is incremented. After all feasible poses are considered, the grid cell with the highest certainty value is selected as the robot's present position.

Ensuring that the algorithm identifies feasible poses requires information about the robot's orientation. Orientation can be measured from a digital compass, a gyro or even calculated from odometry data. While systems for measuring orientation are often prone to error, the algorithm is quite robust and can tolerate considerable inaccuracy. The algorithm can also be extended to provide corrections for measured orientation.

To create a useful local map, the algorithm requires range measurements in a number of different directions. Such measurements are readily obtained by sweeping the sensor. The robot used in this project features a sonar sensor that is mounted on a servo, so a 180-degree sweep is possible.

More information about this localization algorithm can be found in reference [4].

The Software

A complete description of the positioning and robot control software is out of the scope of this paper. Detailed information can be found from link [1]. This paper provides an overview of the software components relevant to the localization and map construction process including:

- Grid Map representation and manipulation
- Occupancy algorithm
- Dead Reckoning algorithm
- Localization algorithm

Grid Maps

As mentioned above, a 2-dimensional grid is used to provide a map of the robot's environment. The grid map consists of a matrix of cells, each containing an occupancy value and a certainty value. These values are used by the occupancy and localization algorithms respectively.

Occupancy Algorithm

The occupancy algorithm creates a map of the environment by integrating data collected over time. As the robot explores its environment, information from range sensor sweeps is combined with information about the robot's location to update the occupancy values for the global grid map. Thus the occupancy value for each cell in the grid indicates whether the cell is occupied, empty or unexplored.

Due to the inaccuracy of sonar range sensors it is not sufficient to simply mark the cell at the end of the range vector as occupied. Instead we need to be able to specify an occupancy probability and integrate range readings over time. Bayes' rule is used to estimate this probability. The occupancy value ranges from 0 to 1. An initial value of 0.5 is used to mark this cell as undecided (unexplored). Then thresholds can be used to determine if the cell is occupied, empty or unexplored.

Another function of the occupancy algorithm is to appropriately update the occupancy values from a range sensor sweep (array of range sensor vectors). Very few range sensors can localize a detection to a single point. The inaccuracies inherent in a sonar measurement require that the algorithm do more than simply mark the occupancy value of the cell at the end of a detection vector. The energy radiated by a sonar transducer is not focused in a straight line, instead it is distributed in a cone. The size of the cone is measured by the beam-width angle of the sonar transducer; for example the Polaroid sonar used in this project has a half beam-width angle of 12.5 degrees. The result is a large angular uncertainty of the range reading (i.e. even though the sonar is pointing at 90 degrees the actual reflection might be coming from an object somewhere between 77.5 degrees and 102.5 degrees). In a two-dimensional space the object that caused the reflection will be laying somewhere on an arc (with radius equal to the distance measured and angle equal to ±12.5 degrees). We must now consider the probability that each point in the arc will contribute to the occupancy value. This probability varies along the arc as illustrated in Figure 1.

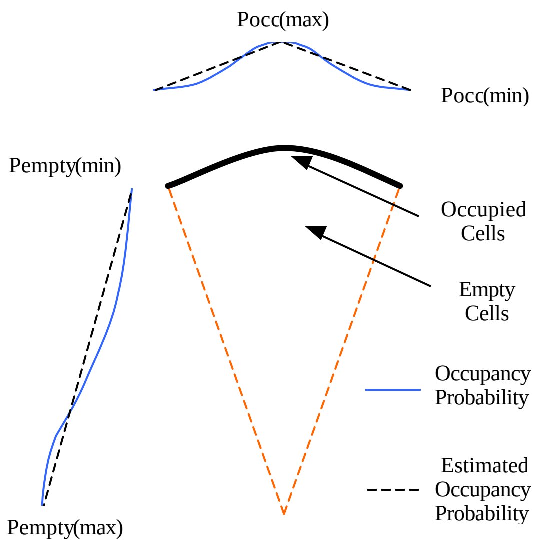


Figure 1 – Sonar modeling

The probability that a cell is occupied along the arc drops exponentially to Pocc(min). The maximum probability is in the middle of the arc and it is Pocc(max). Ideally the probability distribution should be modeled as a Gaussian distribution. For efficiency, a linear distribution was chosen instead (since in order to determine the probability of a cell a simple multiplication is required).

For the probability of the empty cells, a similar approach is used. Close to the arc the sensor's range error must be taken into account, thus the probability that a cell is empty rises when the distance from the arc decreases. The probability range starts from Pempty(min) close to the arc, increasing up to Pempty(max) as the distance from the arc decreases.

The occupancy of each cell in the pie is finally updated based on the previous value and the one calculated from the range reading.

Figures 2 and 3 illustrate how the occupancy of each cell is updated from a single range sensor vector. The color of each cell reflects its occupancy value, starting from white for empty cells (occupancy close to 0) to black for occupied cells (occupancy close to 1). Unexplored cells are displayed as gray (occupancy value close to 0.5).

In order to make the map responsive to changes in the environment but at the same time reduce the noise of false range sensor readings (due to multiple reflections and transducer accuracy), a weight is given to each new occupancy update. We distinguish between two modes: exploration and localization. During the exploration mode any changes in the occupancy values will have the maximum effect, but during the localization mode each update will only have a reduced effect in the global map. This way the map constructed during exploration is preserved and at the same time updated for changes made in the environment.

Information from sources other than sonar could also be used to update the global map. For example many robots are equipped with infrared proximity sensors and bumper switches. These sensors can be used to update the occupancy value of the cells adjacent to the robot's present position.

Dead Reckoning

Location estimation using odometry information is a very common approach. Although fast and relatively easy to implement dead reckoning suffers from accumulation of errors. Dead reckoning is only useful if a position correction can somehow be given, so that any errors accumulated since the last correction are zeroed out.

In this application dead reckoning is used as a secondary source of location information. Combined with the range sensor based localization algorithm, dead reckoning can provide a close to real-time location estimate. At regular intervals a range sensor localization is performed that yields a more accurate position. The dead reckoning algorithm then uses this position for future updates, effectively eliminating any accumulated errors.

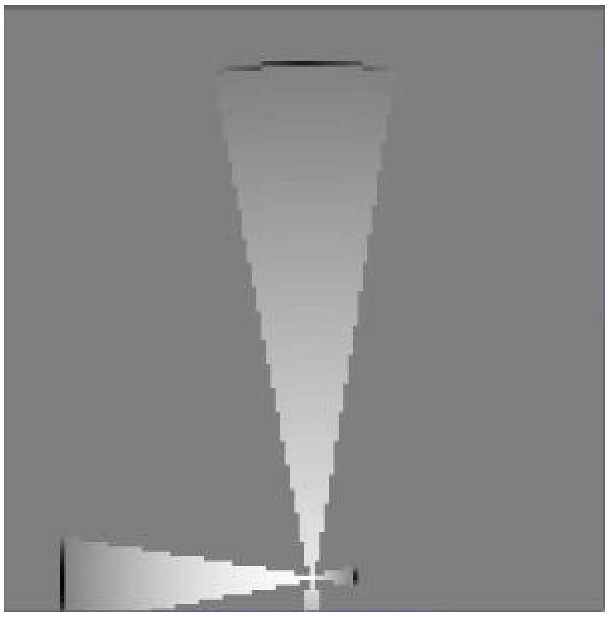


Figure 2 – Occupancy update of an unexplored area

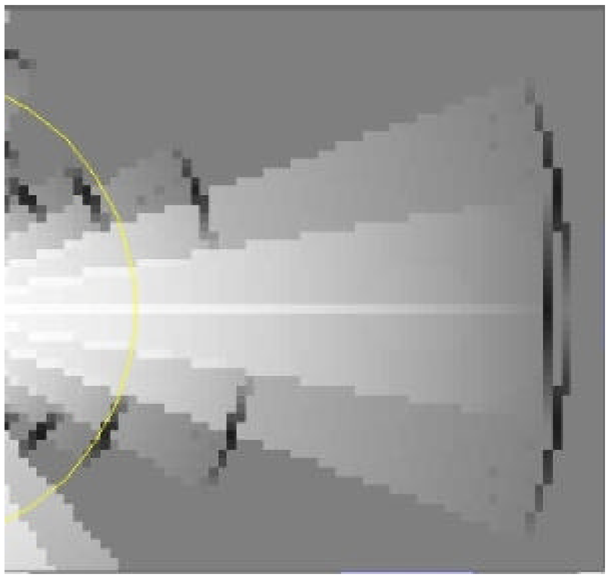


Figure 3 – Overlapping occupancy updates.

In order to know the approximate error of the location calculated from the dead reckoning algorithm, an error radius measurement is kept. In this application it has a fixed start-up value that is incremented every time a new location is calculated. The range sensor based localization algorithm can then use this information to search for a location within this circle.

The dead reckoning algorithm used in this application is based on the one described in paper [5]. It is very easy to implement and it doesn't require much processing power for each update. The accuracy of the location that is calculated from the dead reckoning algorithm is not very important in this application.

Localization

The localization algorithm is the heart of the positioning application. It requires as input: the global map with the occupied cells, a range sensor sweep (consisting of an array of range vectors) and optionally an estimated location (from the dead reckoning algorithm). If the estimated location is not available localization will be performed in the complete map; otherwise the localization will be restricted around the estimated location (within a circle with the estimated location at its center).

Figure 4, shows how localization is performed for a simple map with only three occupied cells and two range vectors.

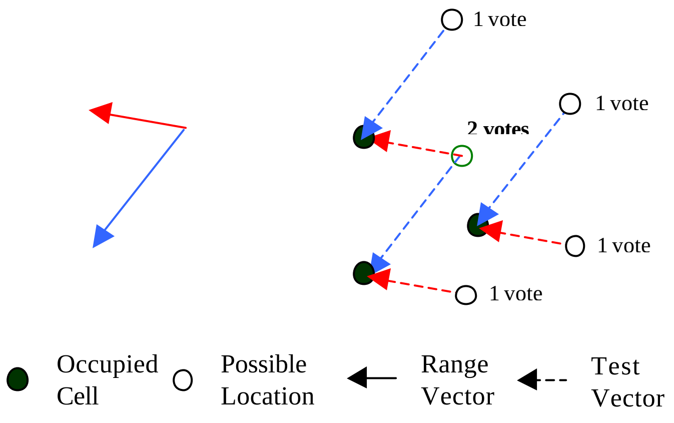


Figure 4 – Localization in a map with three occupied cells (right) and two range vectors (left). The location with the most votes is selected as the present location.

The algorithm relies on the assumption that each range vector ends at an occupied cell and that the path between the origin of the vector and the occupied cell is unobstructed. The procedure is quite simple, apply the end of each vector to an occupied cell, if the vector is unobstructed (i.e. does not cross another occupied cell) then increase the certainty value (or vote) of the cell at the start of the vector. This procedure is repeated for all occupied cells and range vectors. The end result is a map as shown in Figure 5.

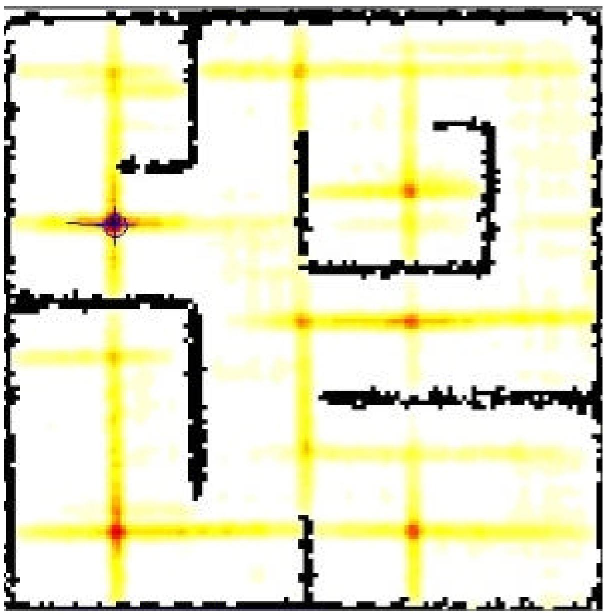


Figure 5 – Overlaid occupancy and certainty maps.

The certainty values (or votes) are represented in colors. The higher the certainty value is, the darker the color will be. In this map, hotspot-like features are clearly visible. These hot spots are feasible poses for the set of range vectors used to perform the localization. But only one has the highest certainty value and this is chosen as the present location (upper left corner).

Selecting the best location is not always as straight forward as simply choosing the cell with the highest certainty value. In many cases more than one cell will have the maximum certainty value. In real world environments the votes will also be distributed around the present location due to the noise in the sensor readings. The localization algorithm must take these in to account before choosing the best location. The procedure followed in this application is the following:

- Average occupancy map. This is done in order to strengthen spots were the votes are not concentrated in a single cell but are distributed around a hot spot.
- From the filtered map find the cells with the highest certainty value (will usually be more than one).
- For each of these cells, increase the certainty value by the sum of the neighbor cells. Select as present location the one with the highest certainty value. If more than one then find the average location.

In order to speed-up the localization process and to increase the accuracy of the calculated position, the location estimated by the dead reckoning algorithm can be used. Since an estimate is available, localization will only take place within the boundaries of a circle centered at the estimated location with a radius of the location error calculated by the dead reckoning algorithm (Figure 6).

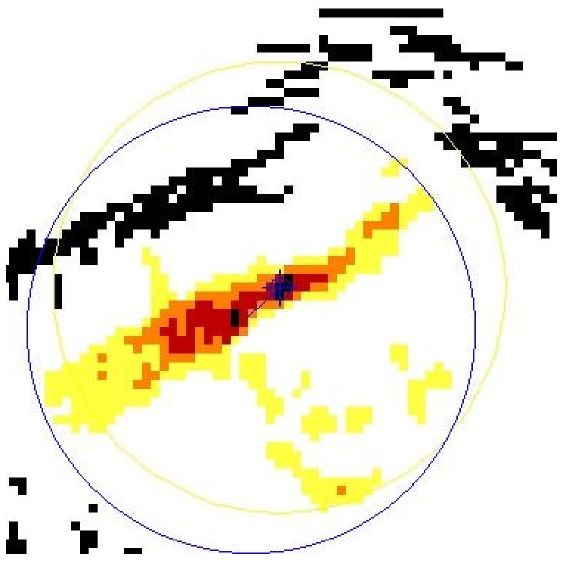


Figure 6 – Certainty map when an estimated location is available. Localization is only performed within the blue circle.

Another way to increase the accuracy of the localization algorithm is to localize incrementally. To do this more than one range sensor sweep is used to perform localization. Each sweep is done from a separate location so that a more complete view of the environment can be seen. This process can be thought as building a local map and then comparing this map with the global map. Normally a single sweep will be used to build the local map; but if more sweeps are added, the localization process will be more accurate. In order to build the local map, the estimated location from the dead reckoning algorithm is required (since localization can not be performed in the local map). It was found that best results are observed when between two and four sweeps are used to build the local map. If more are used, the map will be distorted due to the positioning errors of the dead reckoning algorithm. This will in turn lead to errors in the localization process.

Figure 7 illustrates how the incremental localization can improve the overall certainty. If this map is compared with the one in Figure 5, it can be clearly seen that the strength and number of the false hot spots is greatly reduced.

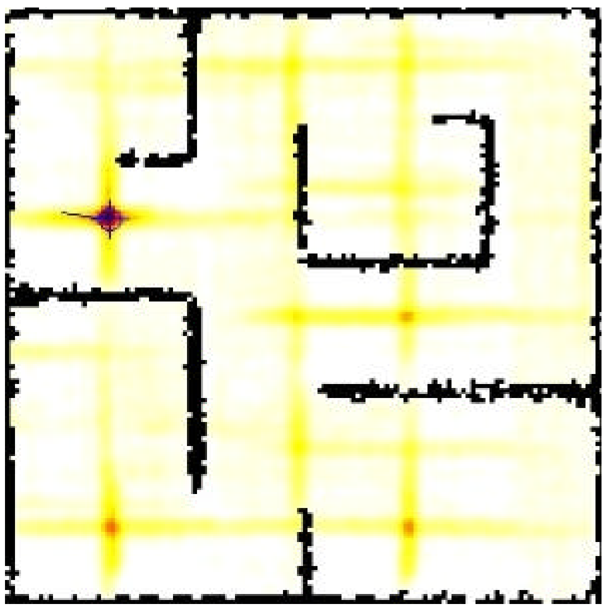


Figure 7 – Certainty map after an incremental localization (3 false hotspots compared with 8 in Figure 5).

Experimental Results

The algorithms described in the previous section have been implemented and tested in an indoor environment using a small mobile robot equipped with an ultrasonic range sensor. In the following sections the robot used in the experiments will be first briefly described, finally the results obtained from a run in the simulator and in a real world environment will be presented.

The test platform can be separated into two parts, the actual robot and a remote PC (PII 350Mhz, Red Hat Linux and Sun JDK 1.3), running the Java based positioning software.

The Robot

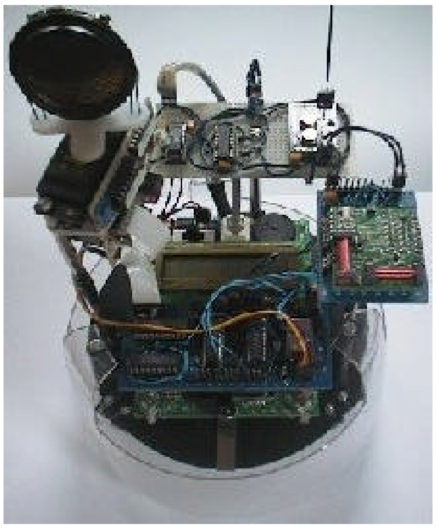
The robot is based on the Rug Warrior Pro platform, but with quite a few modifications and improvements.

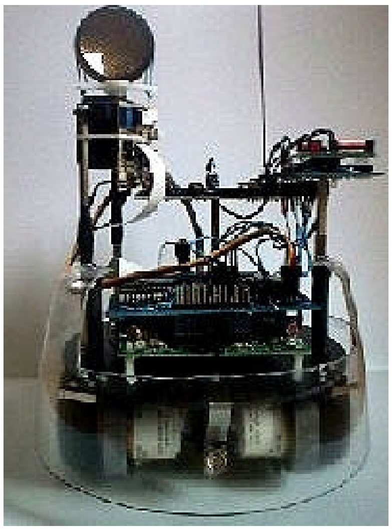
In order to take the processor intensive tasks off the on board microcontroller a RF link is required. The Radiometrix BIM transceivers are used in this robot. One transceiver is connected to the 68hc11 serial port and the other to the Linux box, both running at 9600bps. Since the transceivers can not handle re-transmission and error detection, a small protocol had to be improvised in order to detect corrupted packets and re-transmit them if necessary. The robot is responsible of all low level control tasks, such as motor and sensor control. All high-level control tasks (positioning and map building) are running in a remote PC.

The popular Polaroid ultrasonic range sensor is mounted on a RC servo so that a 180-degree sonar sweep is possible. The Polaroid sensor can measure distance from about 15cm to 10m with an accuracy of about 2cm.

A digital compass is necessary to get an orientation fix since the Rug Warrior's wheel encoders have very limited resolution and suffer from noise when the motors are running. The Rug Warrior's wheel encoders are not reliable for long range dead reckoning (but are good enough when combined with a more accurate positioning source). The Vector 2x digital compass is used. This compass gives up to 2 degrees accuracy and is controlled through an SPI port. Like any compass a magnetic field will influence the reading. It was found that when the motors are running, a 5-10 degree error is produced. Large metallic objects can also influence the reading. The localization algorithm does not require precise orientation information. In a robot with more reliable wheel encoders dead reckoning could be used instead. The localization algorithm could then be used to produce an orientation correction.

Although the wheel encoders are not very reliable in this robot, they are still used for short-range dead reckoning. During the exploration phase (map building), the localization algorithm produces a position correction every 50 to 100cm. This correction effectively erases any errors accumulated by the wheel encoders.


Picture 1 – The Rug Warrior based robot.


Picture 2 – Another view of the robot.

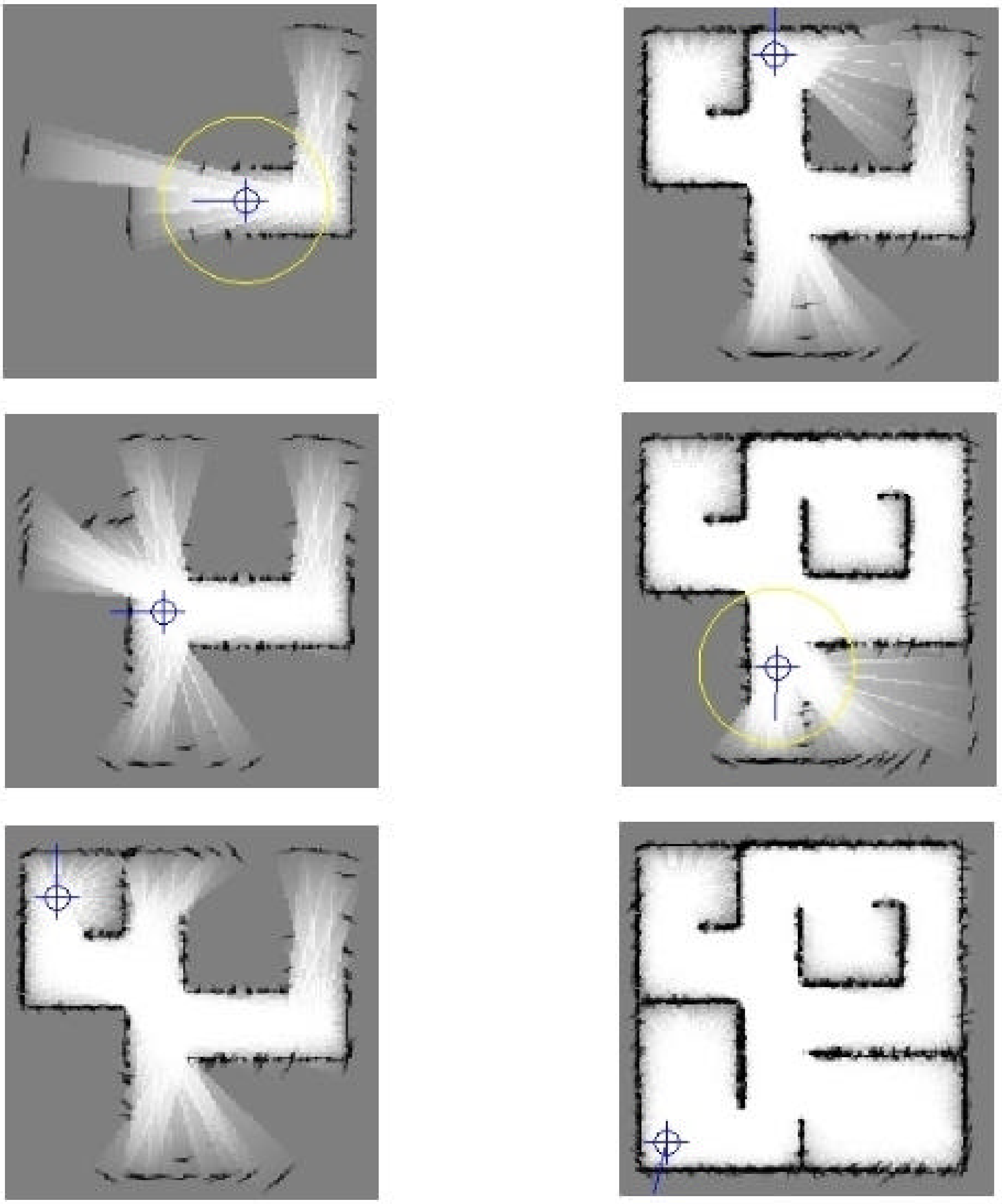


Figure 8 – Map construction using the Positioning software with the Rossum Playhouse simulator (Trinity floorplan)

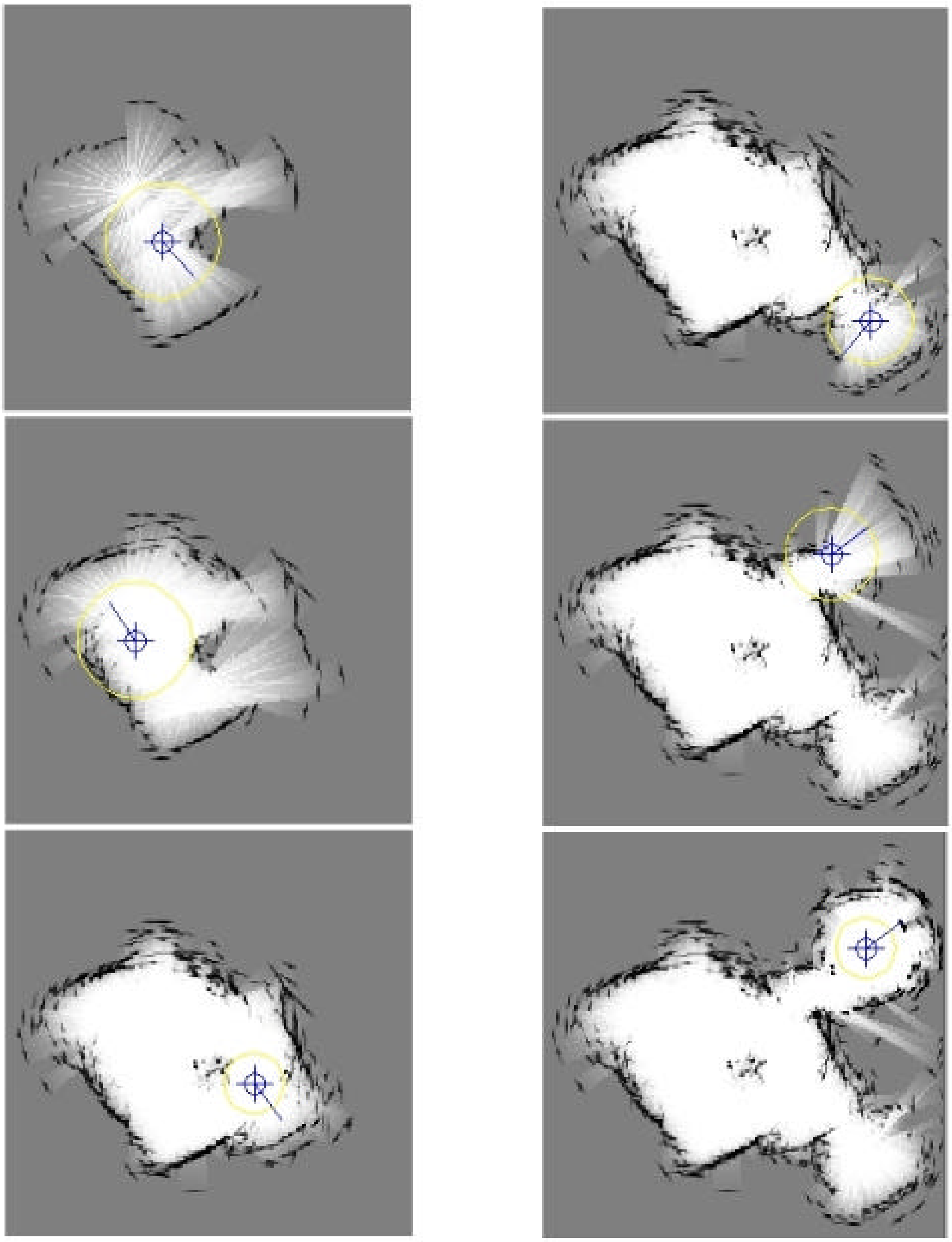


Figure 9 – Map construction using the robot in an indoor environment (one big room, plus two smaller ones)

Map Construction Using the Simulator

The Rossum Playhouse simulator has been an irreplaceable tool for testing the positioning software. Although some bugs can not be detected using a simulator, it proved to be a very useful tool for testing small changes and improvements, without having to deal with all the difficulties affecting a real robot (i.e. limited battery life, obstructions in robot's environment, etc). Of course, tests using the actual robot were conducted frequently. But these could be deferred until many of the small bugs and implementation problems had been detected and eliminated using the simulator. Thus real-world testing time could be spent actually evaluating the performance and accuracy of the system and not correcting the coding errors that usually surface at the early stages of development.

Since the exploration module of the positioning software is now under development, the robot was moved around the floorplan manually. The sweeps were also manually initiated. *The actual position of the robot was never made available to the positioning software*. The present position as seen by the software was the one calculated from the localization module.

The size of the map used in this run was 250x250 cells and the cell size was 2cm. A typical move-sweep-localize-update cycle was used to construct the map. First the robot is moved to a new location and a 360 degrees range sensor sweep is made. Using the range data localization is performed and finally using the newly found location, the global map is updated with the range data. The cycle repeats until the complete floor plan is explored. The map construction process is shown in Figure 8. It took approximately 15 sweeps to build this map. The localization accuracy plays a major role in the map construction process. If an update from two or three bad locations is included in the global map, the map will be greatly distorted. Fortunately since the location is verified using the dead reckoning approach the possibility of such errors is greatly reduced. The localization error using the simulator is typically zero to 4cm (2 cells). One would think that since a simulator is used no errors should be present in the system. Well this is not entirely true, the positioning software assumes imperfect sensors so in this case a perfect range sensor with zero beam-width angle will be modeled as imperfect (10 degrees beam-width). Also the dead reckoning algorithm introduces errors even when the wheel encoders are perfect.

Map Construction Using the Rug Warrior Robot

The positioning software was also tested in a real world indoor environment using the Rug Warrior robot. The floorplan consists of one main room (living room) and two smaller rooms (kitchen and hallway). Furniture and other objects, commonly found in a house, were present in the rooms.

The positioning software used in this run is identical with the one used with the simulator. The only difference is that the robot proxy object is now replaced with a proxy for the real robot. The proxy now handles the RF communication link with the robot. For more information regarding the structure of the positioning software see [1].

The map construction process is shown in Figure 9. In this run the map size was 250x250 cells and the cell size was 4cm. As with the simulator the robot was manually moved (using the motor controls available at the positioning GUI). Again the robot’s position was never revealed to the software. A 360-degree sweep is not possible with this robot. So a 180-degree sweep firing every 10 degrees was used instead. This limits the robot’s field of view. In some cases when the front view is unexplored (e.g. when entering a new room or when going behind a large object), a localization is not possible since enough information about the environment is not available. This problem can be solved in a number of ways. One solution is to pivot 180 degrees and then perform a range sensor sweep looking at an explored part of the room. Then having found the location, the robot pivots back to its original orientation and performs another sweep. This sweep is then used to only update the map and not localize. Another way is to use the incremental localization feature taking range sensor sweeps before the unexplored region is entered. Finally in some cases simply relying in the location estimated by the dead reckoning algorithm is good enough, until enough information is gathered about the unexplored space to make accurate localization possible.

The positioning accuracy varies, depending on the location and the errors in the range readings. Typical values are in the range of 8cm, some times errors as high as 20cm were observed due to errors in the range readings. Fortunately this doesn't happen very often and even when it does, a second sweep from a slightly different position will most likely produce better results. Range errors due to higher order reflections of the ultrasonic pulse can be clearly seen all over the map but overall they didn't have a major impact on the structure of the map.

The small patch of occupied cells in the center of the map is a small footrest in the center of the room. The footrest is almost covered in cloth, this has been proven to be a week reflector of ultrasonic pulses so some times a false range, well beyond the footrest, will be returned. When this range is used to update the map the occupancy value around the footrest is reduced. Some improvements can be made in the occupancy algorithm to compensate for such problems.

Major improvements have been made to the time it takes for the localization algorithm to determine the present location. If the estimated location from the dead reckoning algorithm is used, localization takes less than 1 second. Localization in the complete map takes approximately 2 seconds (depends on the number of occupied cells and range vectors). All tests are done using a 350MHz, PII PC running Red Hat Linux and Sun JDK 1.3.

Conclusions

This project is still work in progress. Some improvements can be made to the existing positioning modules in order to improve accuracy and robustness.

With the completed positioning software it will be possible to deploy the robot in an unknown environment, and have it autonomously explore and map the floorplan. To achieve that a number of modules need to be developed. First a path planner is required in order to determine the most efficient obstacle free path to a target location. The path planner will be combined with a reactive strategy, so that the robot can handle unmapped or mobile obstacles. An exploration algorithm also needs to be implemented. The exploration algorithm will find the best locations that the robot must go so that all accessible space is efficiently mapped. A possible candidate is the 'Frontier based exploration' approach.

The positioning software is available for download and can be used with the Rossum Playhouse simulator. It is 100% Java so it can be run on any platform that a Java Virtual Machine is available. It can also be used to control robot architectures other than the one used in this project. The software structure is highly modular so only the robot proxy object needs to be replaced. The source code will be made available soon.

This paper and more up to date information about this project can be found from [1].

References

[1] Positioning Application
Vassilis Varveropoulos, The Rossum Project.
http://rossum.sourceforge.net/papers/Localization

[2] The Rossum Project
http://rossum.sourceforge.net

[3] Rossum Playhouse Simulator
http://rossum.sourceforge.net/sim.html

[4] Mobile Robot Self-Localization without Explicit Landmarks
R.G.Brown and B.R.Donald
http://www.cs.dartmouth.edu/~brd/papers.html#Russell

[5] An Elementary Model for the Differential Steering System of Robot Actuators
G.W. Lucas, The Rossum Project
http://rossum.sourceforge.net/papers/DiffSteer/DiffSteer.html