An Exploration of Mapping Algorithms for Autonomous Robotic Mapping

Jonathan Reynolds
Colorado State University – 2005 USAR GS Controls

ABSTRACT

This paper discusses algorithms for simultaneous localization and mapping (SLAM). Advantages of each major type of algorithm, probabilistic and incremental, are presented. Hybrid approaches and their possibilities are discussed. An explanation of a proper algorithm selection method has been provided. The example of an urban search and rescue (USAR) robot is used to show this selection method. This example is carried further through the presentation of a method for integration in such a robot. The complexity of a SLAM system is shown and expounded upon.

INTRODUCTION

The robot mapping problem is one which consists of a robot being able to create a map in real time and know where it is within that map. This is known as Simultaneous Localization And Mapping (SLAM). A SLAM system would be very beneficial for autonomous robot operation. This is especially apparent in a robot such as used in search and rescue.

The robot mapping problem, more specifically, consists of the robot finding its pose, creating a map, and integrating the two. The pose is information relating to the location and orientation of the robot relative to its environment. If a robot is able correctly calculate its pose, it is then able to determine the best route to obtain its goal. The problem lies in the fact that to successfully determine its pose, it must have a map of the environment; yet, to create a map, it must first explore and thus know its pose.

In some situations, 3D SLAM can be beneficial. In this case, a three dimensional rendering of the environment is created for the map. This additional level of complexity greatly increases the data processing requirements for the processor. A three 3D SLAM system would, in essence, use the same algorithm as its 2D counterpart. From this point on, the author will assume that a 2D SLAM system will be used.

Mapping algorithms, effectively, began in the 1980's with occupancy grid mapping which creates a grid of elements for the map. In the late 1990's, probabilistic methods tended to dominate algorithms used. Today, even hybrid combinations of incremental and probabilistic methods are used [1].

ALGORITHMS

There are three primary types of algorithms to be considered. These are probabilistic, incremental, and hybrid methods.

PROBABILISTIC ALGORITHMS – The primary purpose of the probabilistic algorithm is to address the correspondence problem. This is where the robot must be able to determine if data taken at different times correspond to the same physical object. Probabilistic techniques yield some of the most accurate results of any method. Unfortunately, most of them are too computationally intensive to run in real-time.

Bayes' Filter – The foundation of any probabilistic algorithm is Bayes' rule. Bayes' rule (Equation 1) states that the probability of x given data, d, (the posterior) is equal to the probability of d with respect to x (likelihood) multiplied by the probability of the prior, x, and divided by the probability of d, the marginal likelihood or evidence. Basically, this means that the current belief can be changed based on the prior belief observations. Bayes' filter uses a continuous estimation to approximate both the map and the robot's pose. Refer to Equation 2 where *s* is robots pose, location and orientation, *m* is a map, *z* is a sensor scan, i.e. data from a laser scanner, and *u* is a set of motion commands or odometry or similar motion feedback data [1].

Equation 1

$$P(x \mid d) = \frac{P(d \mid x) \cdot P(x)}{P(d)}$$

Equation 2

$$p(s_t, m \mid z^t, u^t) = \frac{p(z \mid s, m) \int p(s \mid u, s') p(s_{t-1}, m \mid z^{t-1}, u^{t-1}) ds_{t-1}}{p(z^t, u^t)}$$

Because this approach utilizes a continuous equation, integral over real time, it is not possible to use it in this form. However, the Bayes' Filter is the basis for many probabilistic methods.

Backwards Correction – For cyclic maps, locations with passages that close back on themselves, error tends to grow exponentially (Figure 1). One way to fix this problem is to adjust the map backwards in time. Equations 3-5 describe the difference used for the backwards correction where $\breve{s}_t$ is the robots belief that this posterior is correct. This yields a very accurate map for cyclic environments, but at a greater processing cost [2].

Equation 3

$$\Delta s_t = \bar{s}_t - \hat{s}_t$$

Equation 4

$$\hat{s}_t = \arg\max P(s_t \mid z_t, u_{t-1}, \hat{s}_{t-1})$$

Equation 5

$$\bar{s}_t = \arg\max \breve{s}_t$$

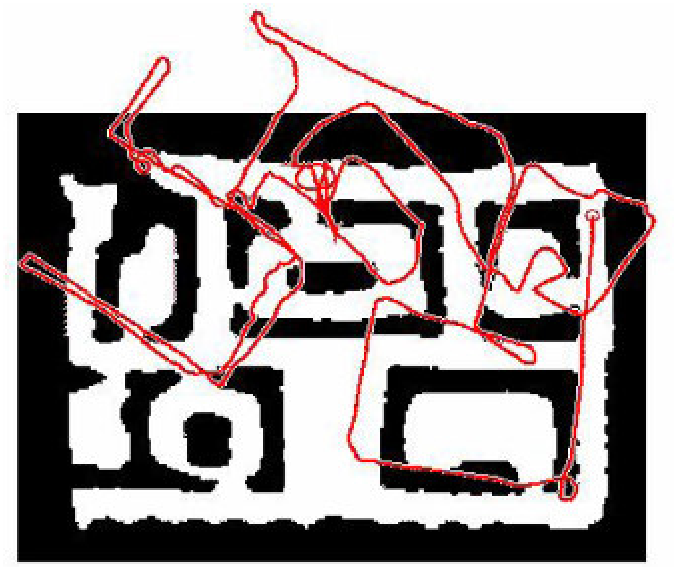

Figure 1. Odometry error increasing over time [1].

Kalman Filter – Kalman filters are basically Bayes' filters with the exception that they use Gaussians to estimate the robots pose. The Gaussian is the full state vector for s_x, s_y, and s_θ for K distinguishing points as seen in Equation 6. The Kalman filter algorithm operates on the postulation that the current state function must be a linear step from the previous state function with Gaussian noise. This noise builds an umbrella shaped region of uncertainty (Figure 2). This Kalman filter then uses Bayes' theorem to "filter" out the less likely results. Even though this is done, the uncertainty will grow overtime. [2, 3]

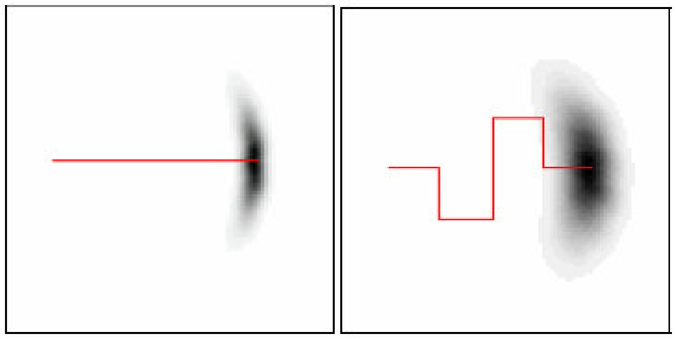

Figure 2. Gaussian error of posterior increasing with time [2].

Equation 6

$$x_t = (s_{x,t}, s_{y,t}, s_{\theta,t}, m_{1,x,t}, m_{1,y,t}, m_{1,\theta,t}, \ldots, m_{K,y,t})^T$$

Expectation Maximization – This algorithm calculates the expected posterior of a robots pose for a given map. This method must process the data many times, and therefore can not be used in real-time. It is, however, one of the most accurate solutions to the correspondence problem. [2]

Lu/Milios – This algorithm is a variation of the Kalman filter. It estimates posteriors the same way, but it also compares ranger data from multiple scans. This allows the Lu/Milios algorithm to compensate for correspondence errors. However, this is not a real-time algorithm.

INCREMENTAL METHODS – Incremental methods are designed to work in real time easier than probabilistic methods. Overall, incremental methods allow for a more plausible algorithm to be used in real-time, onboard and autonomous robot.

Maximization Likelihood – This method compares nearby measurements of the previous map to see if it can tell which way the robot has moved and by how much. Because this is done repeatedly, it can recover from wrong correspondences. This algorithm takes a large amount of processing, and therefore is sometimes not able to be run in real-time. Although, it does work amazingly well. [1, 2]

Occupancy Grids – This is one of the most popular algorithms because of its ease of use and robustness. The basic principle of occupancy grids is that the binary occupancy of a location (x,y) is calculated. This method will incrementally update each grid cell. The primary problem is one of consistent mapping due to noisy sensors. This problem is alleviated when coupled with a probabilistic approach. This will be further discussed later.

Dogma – Dynamic Occupancy Grid Mapping Algorithm is one which can learn. It is very effective in dynamic environments, and it operates by learning the characteristics of an obstacle such as a trash can. This algorithm relies on the mutually exclusive requirement: any given object can not be in multiple locations at the same time.

HYBRID APROACHES – These consist of some combination between probabilistic and incremental methods. Hybrid algorithms typically handle complex mapping in real-time. Although they are newer, and perhaps a bit more difficult to implement, they provide much better results than either purely probabilistic or incremental methods can be for an autonomous robot.

Applying a method of Bayes' filters to the occupancy grid algorithm yields a very powerful solution to the autonomous robot mapping dilemma. This method can solve problems of sensor noise by generating probabilistic maps. It would calculate the posterior for each grid cell with $p(m_{x,y}|z^t,x^t)$. The logarithmic form for the resulting equation is seen in Equation 7. [6]

Equation 7

$$\log\left(\frac{p(m_{x,y}|z^t,s^t)}{1-p(m_{x,y}|z^t,s^t)}\right)=\log\left(\frac{p(m_{x,y}|z_t,s_t)}{1-p(m_{x,y}|z_t,s_t)}\right)+\log\left(\frac{1-p(m_{x,y})}{p(m_{x,y})}\right)+\log\left(\frac{p(m_{x,y}|z^{t-1},s^{t-1})}{1-p(m_{x,y}|z^{t-1},s^{t-1})}\right)$$

Another common hybrid approach is to integrate a Bayes' filter into an incremental maximum likelihood method. This allows the robot to maintain calculate its posterior while mapping.

Hybrid approaches work very well at creating reasonably consistent maps in real-time, it is important to note that they do not handle cyclic environments well. While it is possible for some hybrid approaches to use backwards correction to compensate, the results are not always stable.

ALGORITHM SELECTION

SELECTION PROCESS – To select an algorithm, one must first consider their constraints. It is important to decide if real-time mapping is necessary or not. Beyond that, it is helpful to know what level of accuracy is required, whether the maps generated will be stationary or for a SLAM system, and whether or not a dynamic environment is to be mapped. This process lists many of the commonly used mapping algorithms today, but should not be considered to be a complete work on the subject. Once the primary constraints are defined and a good source of algorithms has been defined, a selection can be made. One method for selection is to create an advantage/disadvantage matrix to compare the properties of each algorithm with the desired constraints.

USAR EXAMPLE – As an example, an autonomous robot used for Urban Search and Rescue will be discussed. The one seen in Figure 3 was built for a USAR competition held annually by Robocup.

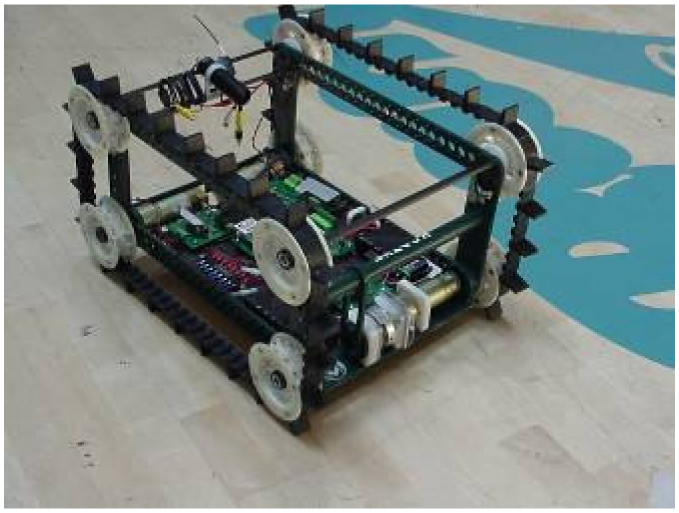

Figure 3. Good Samaritan USAR robot created by Colorado State University for Robocup competition.

Constraints - For this competition, the robot must navigate through a simulated earthquake environment trying to find simulated victims. There is a very strong emphasis on autonomy and a map with victim and obstacle locations must be created. For autonomy and exploration to co-exist effectively, it must operate with a SLAM system. It will need to use the map to wander toward unexplored areas, and to find its way to goal, such as a user input or a possible victim. The map will have to be created in real time based on laser range-finder data. It is desired that it will be able to map while simultaneously carrying out other goals.

Selection – Based on these goals, each of the three main topics have to be considered. The real-time aspect would lead to an incremental method, but the accuracy needed, because of the desire to map while moving, leads to a probabilistic method. This should prompt one to consider using hybrid methods. Because of the ease of implementation, a hybrid occupancy grid algorithm will be considered.

INTEGRATION AND USAGE

BOUNDS – Bounds to consider when implementing an algorithm include processing and memory requirements, available sensors, and availability of accurate odometry. For this example, it can be assumed that the robot will not be trying to map cyclic environments. With this assumption, the amount of necessary calculations will be fairly insignificant compared to other mapping algorithms. The memory, however, will likely be substantial. Because of speed, it is desirable to use on chip memory whenever possible (versus serial EEPROM's). Thus, it is fair to reason that there could be one dedicated processor just for mapping. It is possible that this processor (if sufficient) would also be able to handle some of the other robotic functions, such as directed wander for exploration, which relate to the map.

For high accuracy, the sensors must be sufficient in data acquisition ability and distance accuracy. In this example a LADAR is used. The particular model used here, as shown in Figure 4, is a Hokuyo URG-04. This has a 240° range with 0.36° increments [5]. This sensor is quite adequate for the level of precision desired for this USAR competition.

Figure 4. Hokuyo URG-04LX LADAR [5].

The next question is whether or not accurate odometry is available. Traditionally, tracked robots do not provide accurate odometry. This is because of slippage of the track and obstacles that one track may transverse without the other. Probabilistic methods have been known to produce accurate results even without odometry at all. This provides yet another incentive to continue on with the hybrid occupancy grid method versus simply incremental algorithms. [2, 4]

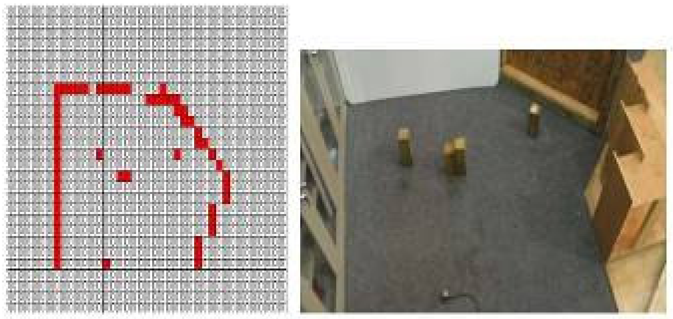
Figure 5. Map based on LADAR data from matrix format.

SLAM – The process that the processor will go through for using this mapping algorithm on the given example would be as follows. First, it will take a sensor reading and convert those data points into full or empty grid cells, storing this information as a matrix. Figure 5 is a robot-centric map which was created from matrix data calculated from distance data from the LADAR. The picture shows the actual environment that used for the scans. A Freescale (Motorola) 9s12 (HC12 family) microcontroller was used to talk to the LADAR and receive data points. The data is returned as two 6-bit numbers that must be converted by subtracting a decimal value of 48 from each, bitwise shifting the high bit six places, and combining it with the low bit via an OR function. This value, in decimal, is the distance in millimeters to the point specified. This is done for as many of the 681 available points as desired. Each point is then converted to a delta-x and delta-y value from the robot's current position using trigonometry for the angle of the given point added to the pose estimated angle of the robots current position verses its original position. The grid value at that point is then changed from a zero to a one. Because the starting position of the robot within the arena is unknown, the matrix is multiplied by four and it is assumed to start in the center of the matrix. In this case, no mater where the robot starts or its orientation, the map of the arena will fit within this matrix. This, obviously, increases the size of the matrix which could be an issue if using a microcontroller. In this case, a microcontroller was used and the entire size of this specific matrix fit well under the RAM limitations of this specific microcontroller. Because the matrix is one of ones and zeros, the data can be stored in byte format; every eight grids in the x-direction are compiled into a byte. This allows the size to be reduced for memory storage as well as transmitting to a user/base station. The C-code for these calculations can be found in the Appendix.

With a second map (second scan), it would be possible to estimate the robots posterior within a certain error. It could then compare this posterior state to the odometry data to reduce the error. This allows the robot to know its localization and use that information for path planning. [4] This step is more complicated but hopefully will be implemented for this robot in the near future.

AUTONOMY – Specifically for autonomy, the robot would first consider its path history, as compared to the map it has generated. This will allow the robot to "wander" or explore beyond what it has already seen. This wandering would of coarse be limited to the obstacles in the vicinity, and the robot would have a higher directive to avoid hitting any obstacles. The robot would have to have three forms of directed wander as well. The first would be if the robot detects a possible victim. The second would be a user bias to its current autonomous path. This may occur if the user is able to detect a possible victim that the robot did not. The third method of directed wander, similar to the first, would be to further investigate a possible victim for more signs of confirmation without hitting the possible victim or any obstacle. In each of these scenarios, the robot would have to compare its current map with the desired destination and calculate the best route to get there in the quickest time while avoiding obstacles.

CONCLUSION

ALGORITHMS - The three primary types of mapping algorithms are probabilistic, incremental, and hybrid. Each has its own advantages and disadvantages.

Probabilistic - The probabilistic methods include Bayes' filter, backwards corrections, Kalman filter, expectation maximization, and Lu/Milios. The primary advantage of probabilistic methods is that they are very accurate and are able to compensate for odometry errors. However, most can not be run in real time. This is a major disadvantage for autonomous robotic mapping.

Incremental – Incremental methods include maximization likelihood, occupancy grids, and Dogma. Each of these methods works in real-time just fine, but they heavily rely on odometry data. This means that any odometry errors will skew the entire remainder of the map. This would, obviously, result in a less accurate method for mapping if the odometry is in question.

Hybrid – Hybrid methods are a compromise between probabilistic and incremental algorithms. This gives it some of the advantages of each, but it will still have problems on some area. For example, as mentioned earlier, the Bayes/occupancy grid hybrid algorithm works in real-time and calculates posteriors for accuracy, but it is still subject to sensor error and handles cyclic environments poorly.

APPLICATION BASED SELECTION – To select an algorithm, the constraints must be stated and used to choose one of the three basic types of algorithms. From there, the advantages and disadvantages should be compared to the desired constraints to help choose a specific algorithm.

INTEGRATION – A system of processes for integrating the chosen algorithm should be defined and ordered. Each component of the robot should be considered as to how it interfaces with this chosen algorithm. The requirements of each of the robot and the algorithm must be addressed. If they have reasonable solutions and integrate well with each other, the chosen algorithm should be implemented.

USAR ROBOT – An incremental grid occupancy algorithm is successfully being integrated into the robot's control system. This simple system yields an accurate map that meets the requirements of the project and fits well within the scope of the competition for which it was designed. This algorithm easily allows an implementation of a probabilistic modification, converting it into a hybrid algorithm.

SLAM – For most autonomous robotic systems, simultaneous localization and mapping will be used. For it to be used effectively, the algorithm must be carefully selected to fit the given robot and its purpose. Once this has been done, the SLAM system can be used to aid in exploration and navigation.

REFERENCES

1. S. Thrun. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Articial Intelligence in the New Millenium*. MorganKaufmann, 2002.

2. S. Thrun,W. Burgard, and D. Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. In *Proceedings of the IEEE International* Conference on Robotics and Automation (ICRA), San Francisco, CA, 2000. IEEE.

3. S. Thrun. A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots. *Machine Learning*, 29-53. Kluwer Academic Publishers, Boston, MA, 1998

4. J. Leal. Probabilistic 2D Mapping of Unstructured Environments. Australian Centre for Field Robotics, The University of Sydney, Sydney, NSW, 2006.

5. Hokuyo Automatic co. “Communication Protocol Specification – UGR Series” February 2, 2004

6. H.P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61-74, 1988.

CONTACT

Jonathan Reynolds
euio76@gmail.com

DEFINITIONS, ACRONYMS, ABBREVIATIONS

SLAM: Simultaneous Localization and Mapping

USAR: Urban Search and Rescue

Pose: Location and Orientation

Posterior: Data representation of the probability that the robot is at the given location.

Correspondence Problem: Determining if two data points taken from two different scans are of the same object.

EEPROM: Electronically Erasable Programmable Read Only Memory

LADAR: LAser raDAR, a laser scanner which rotates about an axis to give a 2D representation of its surroundings. Note: some have been gimbled with a second axis to provide 3D capabilities.

Gaussian: Having symmetrical bell shaped curve.

APPENDIX

```
#include "dp256reg.h"
#include "sci.h"
#include <stdio.h>
#include <math.h>
#include <stdarg.h>

#define RDRF 0x20   // Receive Data Register Full Bit
#define TDRE 0x80   // Transmit Data Register Empty Bit

char SCI_InChar1(void){
  while((SC1SR1 & RDRF) == 0){};
  return(SC1DRL);
}

void SCI_OutChar0(char data){
  while((SC0SR1 & TDRE) == 0){};
  SC0DRL = data;
}
void SCI_OutChar1(char data){
  while((SC1SR1 & TDRE) == 0){};
  SC1DRL = data;
}

void SCI_OutString0(char *pt){
  while(*pt){
    SCI_OutChar0(*pt);
    pt++;
  }
}
void SCI_OutString1(char *pt){
  while(*pt){
    SCI_OutChar1(*pt);
    pt++;
  }
}

void SCI_InString1(char *string){
 char character;
 int i=0;

 for (i=0; i<16; ++i){
 character = SCI_InChar1();
 string[i]=character;
 }

}

void pll(){
 SYNR = 0x01;
 while((CRGFLG & 0x08) == 0){}
 CLKSEL = 0x80;}



unsigned short n = 1;
char dist[20];
char res[20];
unsigned int  result;
char str[1];
char map[160][20];
int i,j;
char temp3;
char temp2[50];
float x1,y1,angle;
int x,y,rx,ry;
int start,end,inc;
char com[20];


int main(){
 pll();

 SC0CR2 = 0x0C;
 SC0BDH = 0;
 //SC0BDL = 52;
 SC0BDL = 26;
 SC1CR2 = 0x0C;
 SC1BDH = 0;
 SC1BDL = 26;
 rx = 79;
 ry = 79;
 inc = 4;  //number of data points per step
 start = 129;  //start of ladar sweep
 end = 640;  //end of ladar sweep
 //angle = 0.006151*inc;  //angle in rads of step size

 map[0][0]=1;
 map[159][19]=128;
 map[ry][rx/8]=(map[ry][rx/8])|(1<<(ry%8));  //robot location

 i = start;

 while(i<=end){

  angle = (start-129)*0.006151;

  sprintf(com, "G%03d%03d01\n",start,start);
  SCI_OutString1(com);
  SCI_InString1(dist);

  str[0] = dist[12] - 48;
  str[1] = dist[13] - 48;

  result = (((unsigned int)str[0])<<6) | (unsigned int) str[1];

  x1 = (result*cos(angle));
  x = ((int)(x1))/100;
  if((((int)(x1))%100)>=50)
   x+=1;

  y1 = (result*sin(angle));
  y = ((int)(y1))/100;
  if((((int)(y1))%100)>=50)
   y+=1;

  x = rx-x;
  y = ry-y;

  map[y][x/8]=(map[y][x/8])|(1<<(x%8));

  i+=inc;
  start+=inc;

 }

 for(i=0;i<160;i++){
  for(j=0;j<20;j++){
   temp3=map[i][j];
   SCI_OutChar0(temp3);
  }
 }

  return  0;
}
```