Extending SLAM to Multiple Robots

Ethan Howe and Jennifer Novosad

March 13, 2005

1 Introduction

As robots become more prevalent and intelligent, we will want them to share their experiences with other robots. Also in time sensitive applications such as search and rescue, multiple robots collaborating to achieve the goal can be much faster and more efficient. Smaller less complicated robots can also be less expensive and more expendable while still achieving the same tasks as a more agile, larger robot. On a Mars mission, a number of mapping robots could be deployed to identify important science sites and communicate back to a robot with scientific instruments where it believes its relative position to be. Thus some of the mapping robots could fail and the mission goals would still be achieved. The list goes on but obviously robot-robot interaction can be as important as robot-human interaction.

In our project we implemented the basic SLAM algorithm, enabled robots to share SLAM information, and got them to execute a basic task collaboratively. We created a simulated world with certain assumptions which with a little extra work could be translated to the real world. Much of our time during this project was spent deriving the entries in the SLAM covariance matrix which we could not find explicitly in the literature. Once two robots were able to see each other and initialize contact, they were able to communicate newly obtained information as long as they remained in radio contact. Our final robots were programmed to determine the areas of a space they and other robots had already mapped and to greedily seek out unmapped sections.

2 Theory

In this section, we will outline the theory that we used to build our implementation of multi-agent SLAM. SLAM creates a map of landmarks relative to some basis that is internal to the robot. When the robot wishes to move, it applies an internal model of that action on its current state and then checks the changes this action made to its observations against what it expected. For multi-agent SLAM, a robot must use its measurement of another robot and their current believed position to transform and add the other robot's map to its own. Multi-agent systems are an active area of research involving many different strategies to find an optimal solution in spite of each participant only having information on a small part of the world.

2.1 Summary Of SLAM

SLAM, Simultaneous Localization And Mapping, is a technique that allows robots to simultaneously create a map of the world, and localize themselves on that map, in the presence of both measurement and movement noise. The basic concept behind slam is a loop, which uses system models to predict the state, and then corrects its predictions with measurements.

In my notation, foo_k will represent the variable foo on the k^{th} iteration of the loop. $\hat{foo}$ is the robot's internal estimate of foo. $\hat{foo}^-$ is the robot's prediction of foo, before measurement.

The vector x stores all of the state information, including the robot's position and angle, and the positions of all other objects. Basic SLAM assumes still objects, and so the information about moving objects, such as other robot's positions, is not stored in x.

The matrix A, and the vector B, are the state update equation. There is a different A and B for every robot movement command (such as turn, versus drive forward). The system updates with:

$$x_k = Ax_{k-1} + B + w_k$$

where the vector $w_k = N(0, Q)$ is zero mean gaussian white noise, associated with movement. If the movement isn't linear in the state variables, A and B are linearizations or taylor expansions of the movement. For example, if x_r and ψ_r are the robot position and direction, driving forward would cause $x_rk = x_{r(k-1)} + l\cos(\psi_{r(k-1)})$, which is not linear in ψ. Also note that the robot can face a variety of directions, so you do not want to make a small angle approximation in the taylor series expansion – hence, A and B will have a functional dependance on ψ. Since A and B may depend on the current values of the state, they may need to be recomputed at each time step.

After movement and before measurement, the robot's best guess of its current state is:

$$\hat{x_k^-} = A\hat{x_{k-1}^-} + B$$

After the robot updates its internal guess of where it is located, it takes a measurement, z. Like the model for movement, the robot has a model for what happens when a measurement is taken.

$$z_k = Hx_k + v_k$$

In this model, H expresses how the result of a measurement is linearly dependant on the state variables, and $v_k = N(0, R)$ is zero mean gaussian white noise. Using this model, the robot can also guess what it should have measured,

$$\hat{z_k} = H\hat{x_k}^-$$

The difference between what it measured and what it expected to measure is proportional to how much it change it needs to correct it's measurements.

$$x_k = \hat{x_k^-} + K_k(z_k - H\hat{x_k^-})$$

K_k, the constant of proportionality in the above equation, is called the Kalman Gain. There are plenty of papers online which will explain how to derive the Kalman Gain. For this reason, I will simply present the result here:

$$K_k = P_k^- H^T(HP_k^- H^T + R)^{-1}$$

In this equation, P is a matrix of the uncertainty covariances between all of the state variables (so, P[0,0] = $\sigma_{x[0],x[0]}$ and P[i,j] = $\sigma_{x[i],x[j]}$). R is a matrix of the measurement noise covariance ($v_k = N(0, R)$). Recalling that Q is the matrix of movement error covariances, to maintain P at each time step, we compute:

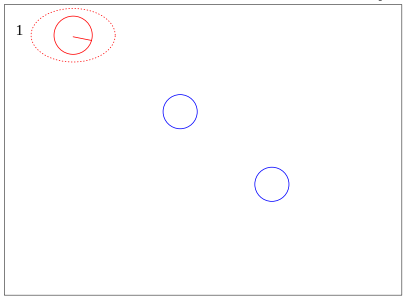
$$P_k^- = AP_{k-1}A^T + Q \text{ after movement}$$
$$P_k = (I - K_kH)P_k^- \text{ after measurement}$$

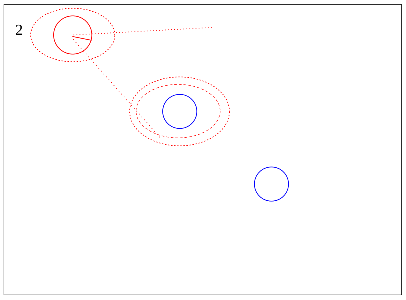
To summarize,

$$\begin{aligned} \text{Move:}\ & x_k = Ax_{k-1} + B + w_k \\ \text{Estimate:}\ & \hat{x_k^-} = A\hat{x_{k-1}^-} + B \ P_k^- = AP_{k-1}A^T + Q \\ \text{Measure:}\ & z = Hx + v_k \\ \text{Update state:}\ & x_k = \hat{x_k^-} + K_k(z_k - H\hat{x_k^-}) \ P_k = (I - K_kH)P_k^- \end{aligned}$$

Using Kalman gain is not the only way to update the state estimate. Another common example is particle filtering, which stores a population of possible states. States which become unlikely are deleted, and replaced with more likely possibilties. For the rest of this paper, we when we refer to SLAM, we will mean SLAM that uses Kalman filters, rather than some other implementation. The main concept should still work; however, some adjustments will have to be made before they will apply to other forms of SLAM.

Figure 1: The circle with a line through it is the robot, the line denoting where it is heading. The dotted circles are that robot's uncertainty in its position. At this point, it hasn't seen any objects.





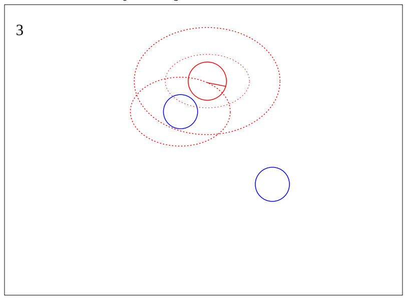
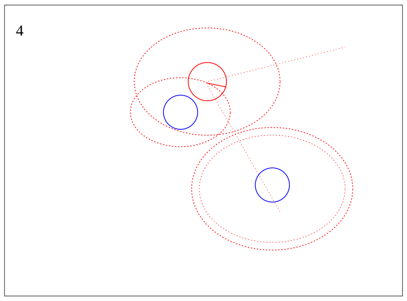


Figure 2: The robot sees a new object. The total uncertainty on the position estimate of this object is the error on position of the robot, plus the measurement error. The inner circle is to show the size of the position error, in comparison to the total error. In this example, measurement error is comparitively small.

Figure 3: The robot then moves forward. Because it has slippery wheels, it is not sure it went where it expected to go. So, the uncertainty in position grows from the previous uncertainty (inner circle) to the previous uncertainty, plus some amount of movement error (outer circle).



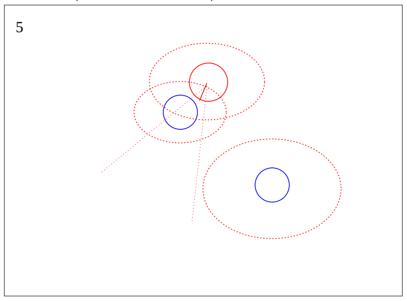


Figure 4: In this diagram, the robot sees a second new object. Because the Robot is less certain of its position than last time, the uncertainty in its estimate of this object is larger than before.

Figure 5: When the robot measures an object it has measured before, it can get a better estimate of its position. Because it knows its position better, the uncertainty of some other objects decrease.

2.1.1 Conceptual Example

In this section we present a graphical example (figures 1 through 5) of how errors should propagate in SLAM. Because A, H, R, and Q depend very specifically on how the robot moves, or how the world is simulated, more specific descriptions cannot be given that still hold in the general case. For an example of what the matrix values are for a simple world model, see the implementation section.

In this example, the robot has both measurement error and movement error. The example starts just after the robot movement step in the loop above: the robot has never seen any of the objects in the diagram, and there is some uncertainty in its position. It then takes a measurement of its surroundings, and sees a blue ball. It is uncertain in the location of the blue ball in global coordinates, because it is not exactly sure where it is. It is also uncertain because of measurement error. Next, the robot moves forward. It then takes another measurement, and sees a different blue ball. It is able to tell them apart with some kind of filter (which we have neglected to describe here), or with an oracle/human assistance. When the robot turns and sees the first object, it is able to reduce the error in its position, and therefore in the position of the second object. This reduction of error upon seeing a familiar object is called loop closing, because it is particularly useful after travelling in a large loop, when the error in position has grown very large.

2.2 MultiAgent SLAM (MASLAM)

For our final project, we decided to extend SLAM to include multiple, communicating robots. The robots communicate information about their part of the map to other robots, so that in a situation with cheap, slow robots, map creation could be parallelized between multiple robots. This capability would be useful when exploring a maze, to let robots simultaneously take different branches of the maze. The capability would also be useful if a robot needs to know something about an area he cannot reach, but another robot on the team can.

Before I go into the details, I want to mention that we did not successfully implement a simulation of all portions of this theory. Due to time constraints, we have not had the opportunity to completely debug all of our simulation. While we are certain that this is due to a programming error, and not due to a conceptual one, as a reader you may consiquently lack the faith in our theory. So, you may take the following with a grain of salt.

Our version of MASLAM (MultiAgent Simultaneous Localization And Mapping) extends basic SLAM as follows:

- Get communication information, y and S, from other robots
- Treat as a Measurement: $y = H_r x + \nu_k$
- Incorporate new objects from y and S into x and P
- Update state: $x_k = \hat{x_k^-} + K_k(y_k - H_r \hat{x_k^-})$
- Proceed to Basic Slam

In this analysis, we assume that communication is nearly-perfect, or that another device on the robot can determine if parts of the message are corrupted, and remove those objects from the messege.

Before proceeding with the predict and update loop given in the previous section, the robot, r_b, requests information from surrounding robots. The communicated information from r_a is sent in some predicted format, y_a and S_a, and can be considered a measurement. We chose to have y_a be r_a's entire knowledge of the state, and S_a be r_a's covariance matrix. This means that y is x in r_a, and S is P in r_a. Some information to identify each object in y_a is also needed, and depends on the particular type of object recognition. Since this information is potentially costly, it should only be sent for new objects.

We chose this format because it was a natural type of information to send, and easy for r_b to use. More processed data could be sent, and in some applications this would be prefered because it could reduce the size of communications. The current amount of information transfered is roughly $O(n^2)$ in the number of objects sent, if there is some kind of object representation which makes communicating landmark information easy. Otherwise, for a small number of objects transfered, it will be roughly O(n), because the object recognition information will dominate the signal.

2.2.1 Robot Origins

Because there are no global coordinates, y and S are relative to the origin of the other robot. So, despite transfering a relatively unprocessed signal, r_b needs to do some processing in order to get y_a into it's own coordinate system. In order to do this processing, r_b needs to know the origin of r_a.

An origin can be modeled the same way as any other object, with a position and an uncertainty which is correlated to the other objects in the world. The only difference between an origin and a landmark is that the origin is never measured via laser. By modeling an origin as another fixed point in the world, we can reuse the SLAM computational machinery without modification. r_b could learn about r_a in one of three ways.

In the first way, r_b could visually see r_a, and record the origin based on the position r_b sees r_a at, the position r_a belives it is at, and the position r_b belives it is at. In this method, if s^j_w denotes just the position of w in r_j's coordinates, the position of O_a in r_b's coordinates is

$$s^b_{o_a} = s^b_b + m^b_a - s^a_a$$

where m^b_a is r_b's measurement of the distance/direction from it to r_a. If accuracy is an issue and time is not, r_a could measure and send m^a_b to r_b so that a better estimate of the distance/direction between the 2 robots is used. Note that the uncertainty of $s^b_{o_a}$ depends not only on the uncertainty of measurement, but the uncertainty in each robot's position. This means that in P, $s^b_{o_a}$ is correlated to every object that either robot is correlated to.

In the second method of finding O_a, r_c could tell r_b the origin of r_a. This method requires that r_b already knows O_c, so that it can relate the information from r_c to its own coordinate system. If all robots are identical, this method is likely to propagate more errors than the first method.

The third method of finding O_a is to see an object that both r_a and r_b can recognize as a unique landmark. Because soda cans are common, it would make a bad communal landmark. This landmark should really be unique, such as a house which bears a day-glow My Little Pony mural, or the door to the testing room. Otherwise, r_b would learn a completely wrong O_a, which would cause SLAM to break down when r_a sends more object information. Additonally, a particular side or angle of the object needs to be identified, if angles are relative to each robot's coordinate system. Because our robot's did not have enough knowledge to know what landmarks were rare rather than common, we did not use this method of finding O_a. (Readers who use particle filtering rather than kalman filtering may not find this to be such a large problem.)

Once r_b knows O_a, it can get object information from r_a for all future times without having to remap the origin.

2.2.2 Example of Origin Localization

In this section, I give a simple example of how a r_b would learn O_a through sight, and then use O_a with information about objects from r_a to extend its own map.

In the first figure, there are two robots in the world which have never talked to each other. r_a knows of an object in the world which r_b does not, a blue circle. We would like r_a to communicate this information to r_b.

Since r_a and r_b have never seen each other, first r_b needs to determine the location of O_a (r_a is simultaneously determining the location of O_b, but that is not shown to reduce clutter). To do this, r_b measures m^b_a, and asks r_a for s^a_a (see Fig. 2) and then it calculates $s^b_{o_a}$ (Fig. 3).

Now, when r_a sends information about another object, r_b can translate that information into its own coordinate system. The fourth diagram shows how r_b can estimate the location of a new object which r_a knows about, by changing the coordinate system. The uncertainty in the position of this object is related to the uncertainty that r_a has about it, and the uncertainty that r_b has in the origin of O_a. Even though this diagram shows the transfer of information occuring when the two robots can see each other, now that r_b knows O_a, r_b will be able to use information from r_a as long as they are within radio range (even if they can't see each other).

2.2.3 Treating Other Robot Information Like a Measurement

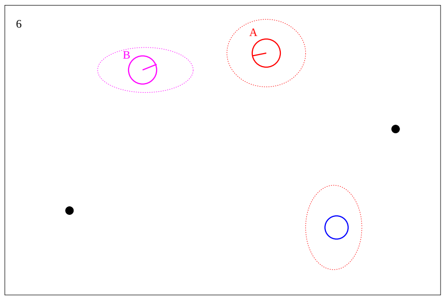
This portion of the theory is the part we had difficulty implementing. The general concept is that if another robot r_a sends us information about object c, this measurement, y_c, will be

$$y_c = s^a_c = -1 * O_a + s^b_c$$

Therefore, our measurement will be

$$y = H_r x = (I - e)x$$

Figure 6: r_a, and things that r_a knows are in red. r_b and its knowledge are in pink. Robots are denoted by circles with lines through their center. The line represents the direction the robot is facing.



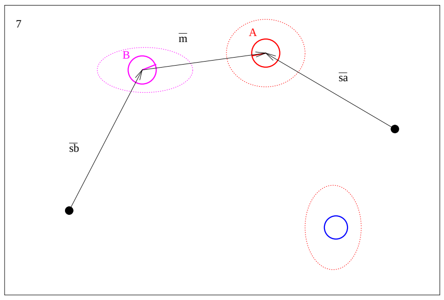
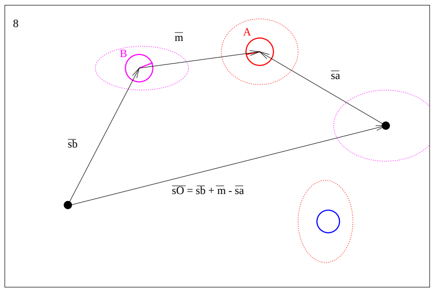


Figure 7: The two black dots are the robot origins. The black lines are vector notation of the measurement information that r_b has at this time. Note that there is no pink circle around the origin of r_b, because r_b knows its origin location perfectly.

Figure 8: r_b calculates the position of O_a. Notice that the uncertainty is larger, because $s^b_{o_a}$ depends on s^b_b, s^a_a, and m


8
$\overline{m}$
B
A
$\overline{sa}$
$\overline{sb}$
$\overline{sO} = \overline{sb} + \overline{m} - \overline{sa}$

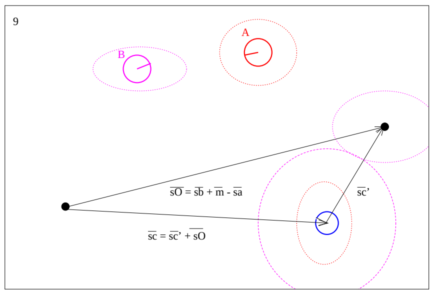

9
B
A
$\overline{sO} = \overline{sb} + \overline{m} - \overline{sa}$
$\overline{sc'}$
$\overline{sc} = \overline{sc'} + \overline{sO}$

Figure 9: r_a can now understand information that is sent from r_a. If r_a sends information about an object, c, r_b can retranslate that information into its own coordinates. Note that the uncertainty is dependant on its knowledge of O_a, and on r_a's knowledge of the object c

where e is a matrix of zeros, with a collum of ones corresponding to the origin.

R, the measurement error covariance, is also different than when r_b measures an object on its own. The error in measurements of different objects is correlated, since the error depends on our uncertainty in O_a, and this error is added to all of the measurements (rather than being random and independant). With this new R and H_r, one has all of the nessecary matricies to calculate K and update P the same as before.

2.3 Muli-agent Cooperation

Many strategies have been devised for achieving a common goal. This theory comes from many disciplines including economics, management, and military sciences. Robotics presents a fertile setting to test these theories because the agents can follow specific rules and reasoning and they can model many important situations. Specifically in the domain of robotics, interaction between robots will be important to future robotic systems whether antagonistic or cooperative. Although we did not have time to implement and test most of these strategies, we present them here for future students.

In cooperative systems, the question to be solved is how to efficiently distribute task execution in an uncertain environments. Like in the managerial arena, we can set up a hierarchy where some robots take on the role of telling others what to do and in that way syncronize the task to be completed. Managers could be assigned beforehand or could be promoted during execution based on some goodness measure such as having a more accurate map or faster speed. Whenever robots met they could compare their knowledge, update their relative goodness, and then the higher ranked agent could communicate a task to the lower ranked agent. In addition to determining what robots have the power to plan for other robots, we would like to know how a task should be distributed. When trying to achieve a goal such as getting a number of robots to a way-point, there is a question of how best to distribute roles. Robots could be symetric and all execute the same procedure simply spread throughout the space or robots could have asymetric roles. The problem is that the robots are only in contact with each other for short time periods so that complete, dynamic optimization is not possible. For the way point example it is not obvious whether each robot should all make sure that enough robots are coming to the way-point before heading there or some should simply go straight to the way-point while others communicate the message. The first method would certainly spread the message faster but the second way wastes less time before we know that there are enough robots to fulfill the task. Creating this example would be a straight-forward extention of our work.

3 Implementation

Over the course of our project we were able to create a simulated world for multiple robots to move around in, see obstacles, and communicate requests. We implemented the basic SLAM algorithm and extended it to measurements made by another robot. We built on top of this structure robots with some degree of intelligence about the task they were performing in trying to map a complete space. In this section, we hope to aid future groups wishing to implement SLAM. We will present the specifics of SLAM matrices that seem to be lacking from the literature we found and also detail our basic strategy for thoroughly mapping a space with multiple robots.

3.1 The Test Bed

To make our simulation as realistic as possible, we created a test bed to moderate all robot actions. The test bed held the true positions of objects and the position and direction of the robot. The test bed returned information gathered by laser measurements. The laser measurements had a constant amount of gaussian noise added to them depending on the accuracy of each robot's laser. Gaussian noise was also added to the movement of the robot in two places. When the robot moved straight forward it would sometimes overshoot or undershoot its requested distance but it would never vear to the side. When the robot turned it would also overshoot and undershoot its target. The amounts of discrepancy was proportional to the amount that was asked for as we would expect in real systems where going 100 feet causes more error to build up than going 10 feet. In essence we built the ideal gaussian world of the SLAM model. Thus we expect our covariences to be exactly correct and this helped us debug our implementation.

We also made some assumptions that we believe are not too severe as they could be straight-forwardly translated with additional work to a real-world system. First, the landmarks in our simulation were complete circles and line segments. This situation did not necessarily reflect the real world since a complete circle is not actually visible to the robot at one time and a line segment may also extend out of viewing range. In a more sophisticated simulation, the line segments would be broken into the pieces visible at a certain moment and then reconstructed as the robot moved. Likewise, circles would be arcs that would be built up over time as more was seen and interpreted as having to be part of the same object.

The problem of determining if a currently measured object is the same as one seen before is critical to implementation of SLAM. Without consistency of landmarks over time, measurements would do us no good as they could not be related back to previous data. In our simulation, we avoided this problem by giving each object a unique identifier to which the robots had access. In this way, we could always keep track of our knowledge of landmarks and update their information correctly. In a real robotic system, this task would have to be delegated to another system that could do a number of things to determine the identity of an object. The system could keep track of previous objects and their shapes and sizes then match the new set of objects based of relative distance and perspective. This visual tracking system could use a Baysian filter to determine the most likely assignment of labels and return them to the SLAM algorithm. If landmarks were sufficiently unique, then the robot could keep a SIFT database of each landmark it saw and match against that to determine identity. These suggestions might be a good subsystem for future students to implement.

Finally, we assumed the existance of a global angular coordinate to initialize the robot. When the robot is first placed in the world, it is told what angle it is facing. This tremendously simplified the computation needed to relate one robots coordinate to the others. Without this assumption, every transfer of landmark information between robots would involve trigonometric transformations of both position information and the covarience matrix. This calculation could be computationally intensive but it also would have been a large source of complication and possible programming errors. Still one can imagine this situation still to be applicable to the real world where a person setting up the robot can use a compass to establish a global directional system. Our robot still only knows its position relative to some origin but all x-y axes for different robots are aligned. With these caveats about the world we created, we would now like to explain how we implemented SLAM.

3.2 SLAM Implementation

Our implementation of SLAM was simplified so that we could focus on integrating multiple agents. This simplification was in both measurements and movements. We feel that these simplifications helped reduced the chance of programming errors, while maintaining the resonant properties of the system. These simplifications allowed us to better linearize the system, rather than implement an Extended Kalman Filter (EKF).

Additionally, since we used an oracle for object identification which returned ID numebrs, this implementation of SLAM stored a list of object ID numbers which allowed measurements and stored information to be correlated. This list of ID numbers was in the same order as x, and was called xName.

3.2.1 Measurements

The measurements returned an $(\Delta x, \Delta y)$ position of the other object, rather than a $\Delta\phi, \Delta D$. This simplication allowed our measurement matrix, H, to be linear, but retained the importiant characteristic that object uncertainty depended both on position uncertainty and measurement uncertainty. Additionally, all of our measurement noise was uncorrelated.

The measurement matrix, H, was calculated after the result of the laser, z, came back. z was sorted so that the values in it appeared in the same order as the values in x. If something was measured, 0 was filled in instead, and the corresponding rows of H were set to 0. If a value was measured, then on those two rows of H, the diagonal entries we set to 1, and the entries corresponding to x and y were set to -1. Recall that each object has two state variables, an x_o and y_o, and that the robot has three state variables. Given that, H below is an example of a two-object system, in which both objects where measured:

$$H = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

This H was chosen because the measurement value returned the distance from the robot to the object, $m_{o_x} = x_o - x_r$ and $m_{o_y} = y_o - y_r$. Note that the top three rows of H are always zero because you never directly measure your own location.

Since measurement noise was independant for all variables, we set $R = rI$, where r was a constant scalar representing the uncertainty in a measurement.

3.2.2 Movements

The robot's movement commands were either turn and angle θ, or move forward a distance f. So, the change in motion was:

Move

$$A_{move}(f) = \begin{bmatrix} 1 & 0 & -f * sin(\phi_0) \\ 0 & 1 & f * cos(\phi_0) \\ 0 & 0 & 1 \end{bmatrix}; B_{move}(f) = \begin{bmatrix} f * (cos(\phi_0) + \phi_0 sin(\phi_0)) \\ f * (sin(\phi_0) - \phi_0 cos(\phi_0)) \\ 0 \end{bmatrix}; Q_{move} = GQ_0G^t;$$

$$G = \begin{bmatrix} -fcos(\phi_0) & 0 \\ -fsin(\phi_0) & 0 \\ 0 & 0 \end{bmatrix}; Q_0 = \begin{bmatrix} x - error & 0 & 0 \\ 0 & y - error & 0 \\ 0 & 0 & turn - error \end{bmatrix}$$

Turn

$$A_{turn}(\theta) = I;\ B_move(\theta) = \begin{bmatrix} 0 \\ 0 \\ \theta \end{bmatrix}; Q = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & turn - error \end{bmatrix}$$

Our model of the robot "move" command was not linear in the state variables, and so some components of an EKF were used. A_{move} and B_{move} are the taylor series expansions of $\Delta x = f * cos(\phi)$, $\Delta y = f * sin(\phi)$. ϕ_0 is the expected angle of the robot before the motion begins. These matricies were recalculated each time step, since they were functions of the current state (ϕ in particular), and of some gain (f or θ).

When objects were added, and consequentally x became larger than the robot position variables (x_r y_r and ψ_r), these matricies were still used and the computations were done on a 3x3 scale (since the robot's motion should not result in any other objects moving). When a larger value for A was needed to estimate P, the lower (n-3)x(n-3) square of A was set to the identity, and Q's expansion was filled with zeros.

Movement noise was only dependant on the length of distance traveled, or in the amount of angle turned. If traveling forward, the robot angle did not change (though the robot's belief in the angle might have been wrong), and the robot remained in the current position when it changed angle. These modifications gave us more linear error updates.

3.2.3 Initialization of Variables

When the robot began, it assumed that it knew its origin perfectly, and that it was currently located there. The test bed set the robot's angle. So, the initial matricies were:

$$P = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$x = \begin{bmatrix} 0 \\ 0 \\ \psi_i \end{bmatrix};$$

All other matricies were constructed as needed in the loop. When an object was measured for the first time, x and P were updated as follows:

$$P_{new} = \begin{bmatrix} P[0,0] & P[0,1] & P[0,2] & & \sigma_{x,x} & \sigma_{y,x} \\ P[1,0] & P[1,1] & P[1,2] & ... & \sigma_{x,y} & \sigma y, y \\ P[2,0] & P[2,1] & P[2,2] & & \sigma_{x,\phi} & \sigma y, \phi \\ & & ... & & ... & \\ \sigma_{x,x} & \sigma_{y,x} & ... & \sigma_{y,x} & r + \sigma_{x,x} & \sigma_{y,x} \\ \sigma_{x,y} & \sigma y, y & ... & \sigma y, y & \sigma_{x,y} & r + \sigma y, y \end{bmatrix}$$

$$x = \begin{bmatrix} x[0,0] \\ x[1,0] \\ x[2,0] \\ ... \\ x - measurement + x_r \\ y - measurement + y_r \end{bmatrix};$$

New objects were added to x and P immediately after measurement, before x was updated by $x_k = \hat{x}_{k-1}^{-} + K(z - H\hat{x}_{k-1}^{-})$. The other matricies were updated later upon construction, following the descriptions given above.

3.3 MASLAM Implementation Details

We implemented robots capable of MASLAM as subclasses of SLAM robots. Our implementation sucessfully communicated information between robots, found the origins of other robots, and added new objects from other robots to its own map. Our implementation cannot yet update state estimates of objects it already knew about using information from other robots. This is due to a programming error and lack of time. Hence, here I will present how the robots communicate, how they determine the origins of other robots, and how they add new objects to their own map.

3.3.1 Robot Communication

Assumptions:

We assume that our communication channels are perfect; if two robots are within range, they can share information perfectly. There are no faraday cages or other objects in our world which would change the communication range. Also, transmissions never suffer from bit-flips or other noise or at least they were corrected by some other protocol.

Protocol:

Because of the way that the test bed called robots to preform turns in series, we structured robot communication protocol as follows:

- Test Bed informs r_a that it is his turn to act
- r_a asks the Test Bed who it can talk to. test bed returns ID numbers.
- For each ID number, r_b
  - r_a asks the Test Bed to send a conditional information request to r_b
  - The Test Bed tells r_b that r_a wants a conditional information request
  - if r_b has no new objects to tell r_a, it tells r_a nothing.
  - otherwise r_b gives the test bed x_b, P_b, and $xName_b$
  - r_b records that it has nothing new to give r_a
  - the Test Bed sends this information to r_a

Since laser range was shorter than radio range, if a robot saw another robot for the first time, the following protocol was used to demand information:

- r_a asks the test bed for the ID of r_b
- r_a asks the test bed to send r_b a hard information request.
- r_b gives the test bed x_b, P_b
- the Test Bed gives this information to r_a

3.3.2 Finding Origins of Other Robots

Origins are modeled the same way that other landmark objects were, except that their ID numbers are negative to distinguish them from objects that should be displayed (OriginID = maximum negative integer + robot ID). Each robot creates an internal representation of its own origin before it ever moved. This is the best time to create an origin, because they know their origin location exactly (because it is defined as where they begin in the world). It is impoirtant that each robot have its own origin in memory, so that when other robots tell it about its own origin, it will not become confused by the errors transmitted to it. Having its own origin in memory also lets it communicate information about its own origin to other robots. Since origins can not be measured via laser, this helps stabilize origin positions.

When one robot, r_a, sees another robot, r_b, for the first time, it goes about determining the other robots origin. As described in the section above on communication protocol, r_a requests location and covariance information of r_b. r_a calculates the origin ID and puts this into $xName_a$. It then calculates the position of the origin. For example, to get the x coordinate of the origin, it adds $x_a[0,0] - x_b[0,0] + m_b[0,0]$. In our implementation, the origin has no directionality (there is a global ϕ). The coordinates of the origin are added to x_a. Lastly, it calculates the covariances. A few key examples of covariances are below:

$\sigma_{o_x,o_x} = r + \sigma_{a_x,a_x} + \sigma_{b_x,b_x}$ (a and b are robots)

$\sigma_{o_x,o_y} = \sigma_{a_x,a_y} + \sigma_{b_x,b_y}$

$\sigma_{ra_x,o_x} = \sigma_{a_x,a_x}$

$\sigma_{ra_y,o_x} = \sigma_{a_y,a_y}$

$\sigma_{c_x,o_y} = \sigma_{c_x,a_y} + \sigma_{c_x,b_y}$ (c is another object)

It can calculate these because it has the entire covariance matrix of r_b, it addition to its own covariance matrix. These are placed into the covariance matrix of r_a, P.

3.3.3 Placing New Objects on the Map

Until r_a knows the origin of r_b, it ignores all object information from r_b. Once it has an origin for r_b, it adds objects it hasn't seen before to x, xName, and P.

Adding a new object, c, to xName is simple; the new object ID is added to the end of xName. x is made two entries longer, to hold the new coordinate information. The coordinates gotten from r_b are in r_b's system, so we make a change of basis: $s_c^a = s_{O_b} + s_c^b + s_b^b$. We then expand the covariance matrix. Some examples of covariances are below:

$\sigma_{c_x^a,c_x^a} = \sigma_{o_{b,x},o_{b,x}} + \sigma_{c_x^b,c_x^b}$ (a and b are robots)

$\sigma_{c_x^a,c_y^a} = \sigma_{o_{b,x},o_{b,y}} + \sigma_{c_x^b,c_y^b}$

$\sigma_{ra_x,c_x^a} = \sigma_{a_x,o_x}$

$\sigma_{ra_y,c_x^a} = \sigma_{a_y,o_x}$

$\sigma_{k_x^a,c_y^a} = \sigma_{k_x^a,o_y} + \sigma_{k_x^b,c_y^b}$ (k is another object)

Again, It can calculate these because it has the entire covariance matrix of r_b, it addition to its own covariance matrix, and these values are placed into the covariance matrix of r_a, P.

3.4 Robot Control Implementation

On top of implementing the test bed and SLAM, we were able to implement some limited control strategies for the robots to follow. The first level of control was simply to have the robot avoid colliding with objects. We created a tenativeMove procedure that would try to maintain its heading as much as possible but would turn if a collision was evident. This method took into account the possibility of overshooting the robots' intended movement because of error and thus took care to avoid a collision within an added one standard deviation. The robot also factored in the position uncertainty of the obstacles by considering a translation of the obstacle closer to the robot again by one standard deviation when testing for collisions. In this way, we had a base of movements we knew would carry out our plans as closely as possible while avoiding getting stuck on objects.

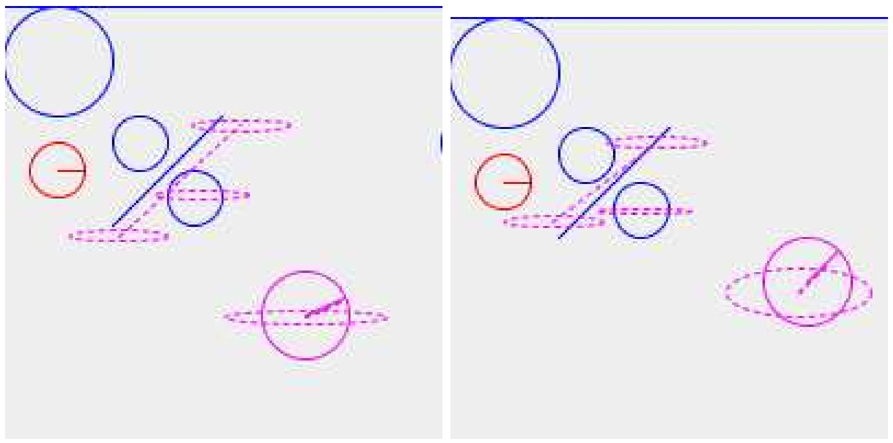


Figure 10: These images show our implementation of SLAM. The robot moves between the first and second images, resulting in its uncertainty in its position increasing, in the direction of motion. The uncertainties on the objects are large in the x direction because the initial movement of this robot in the x direction, which created error that can never be improved upon with loop closing.

To demonstrate a useful function for the robots to perform together, we decided to have them actively try to map a space. Each robot kept a map of the area it had measured. This procedure involved making a circle with the radius of the robots' sensor range and then cutting out the parts blocked by objects. An example of such an area is shown in figure 12. We then had the robot extrapolate the area it believed it could gain by moving in a particular direction. This area obviously could not take into account unseen obstacles but it could tell where the robot did not yet have a map. The robot then greedily went in the direction it believed would add the most to its map. We tuned this process to favor moving in the current direction because it lessens the error added to the angle by turning the robot. We also took into account the SLAM covariences by making the obstacles appear larger than they were so that a robot did not erroneously think that it had actually seen behind an obstacle when it had not. We then had robots exchange their maps as soon as they knew the other's origin and in that way not try to replicate parts of the map that another robot had already seen. The robots continued to communicate new areas of the map until they are outside of hearing range just as we do with landmarks except the areas are almost always changing. We demonstrated this capability in one of our test cases and saw one of the robots become disinterested in a portion of the map communicated to it. We also took care not to retransmit a map back to the same robot that experienced it first hand. This small fix was needed because otherwise error in the relative origin began all maps from being passing the maps back and forth so many times.

4 Results and Conclusions

Although we were not able to achieve all our project goals and we had to make a few assumptions, we were able to implement multi-agent SLAM and use it to direct robots in a common goal. In this paper, we presented the theory behind basic SLAM and how it can be extended to share information between robots. We also presented some ideas about strategies for achieving common goals. We then detailed our implementation of a simulated enviroment for our robots. We talked about what assumptions had to be made and how others in the future could implement these pieces. We implemented SLAM with reasonable accuracy and efficiency and we were able to incorporate information given to us by other robots. Unfortunately, we were not able to successfully implement all pieces of data that could have been gathered collaboratively. For instance, we were not able to update the other robot's origin based on new data after it was originally established. These gaps and programming errors could possibly be filled in the future. We also explained

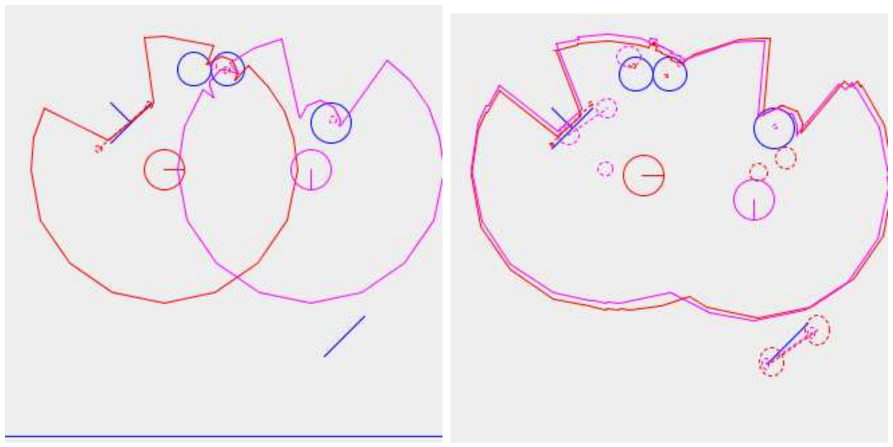


Figure 11: These are two robots running the FindAll routine. Between these two images, there are several time steps pass, and they share information. Sharing information allows them each to increase the amount that they know (the large polygons) to cover the amount that the other robot knows. The slight differences in what each robot belives it knows are due to errors in estimating the other robot's origin.

our creation of robots that seek to extend their maps and avoid collisions. We were not able to go farther than this point and implement some more interesting shared behavior but again it would be relatively easy to build upon our base.

Our implementation worked relatively efficiently for the situation in which we placed it. We minimized the number of landmarks that it had to track by having large structures such as circles and lines be one landmark. In practice, creating these composite landmarks from sensor data might take extra computation as would the process of identifying landmarks across time steps as discussed in section 3.1. After these processes have completed, our implementation of basic SLAM was $O(N^3)$ in time and $O(N^2)$ in space as we would expect from keeping and updating full covariance matrices. Our implementation of sharing data between robots was a bit less efficient since we had to do an extra SLAM update step for each set of data we received from another robot. This method made our multi-agent SLAM method $O(r * SLAM)$ in time in the worst case since we could hear from all other robots at all times. We attempted to reduce this work by only transmitting new information between robots and thus in the limit as we begin to see everything in the vicinity our implementation becomes $O(SLAM)$ again. We believe that it would be possible to integrate communicated measurements more efficiently but this calculation would require some complicated combination of the H matrices which are currently different for accessing other robots measurements. Finally, our robots seeking our new parts of the map contained extra memory overhead to keep track of who had seen what. Our implementation used a sequence of curves to ouline the area seen so far and communicate it with other robots. We also kept track of which robot sent us each piece of the map to avoid sending a part they had already seen back to them with added error. These maps were bounded in memory size by the amount curvature needed to store a particular piece. This strategy gives us an $O(P)$ map size where P is the perimeter. Since a robot had to store r of these maps (even though some were substancially smaller than the overall perimeter), our memory for storing these areas was $O(r * P)$. Overall, our program was about as efficient as could be expected from basic SLAM and obviously since we had r robots our simulation had $O(r^2 * SLAM)$ worst case time over all robots. In the real world, these processing steps would be done in parallel making the computing time per robot the same as quoted above (i.e. $O(r * N^3)$ etc.)

In the future several additional directions could be taken with our work. Obviously more robot planning and communication could be built on top of our framework. Different collective commands could be issued and command structures could be organized to access effectiveness. Achieving specific goals could also follow different distribution of roles as discussed in section 2.3. On a different track, the multi-agent portion of

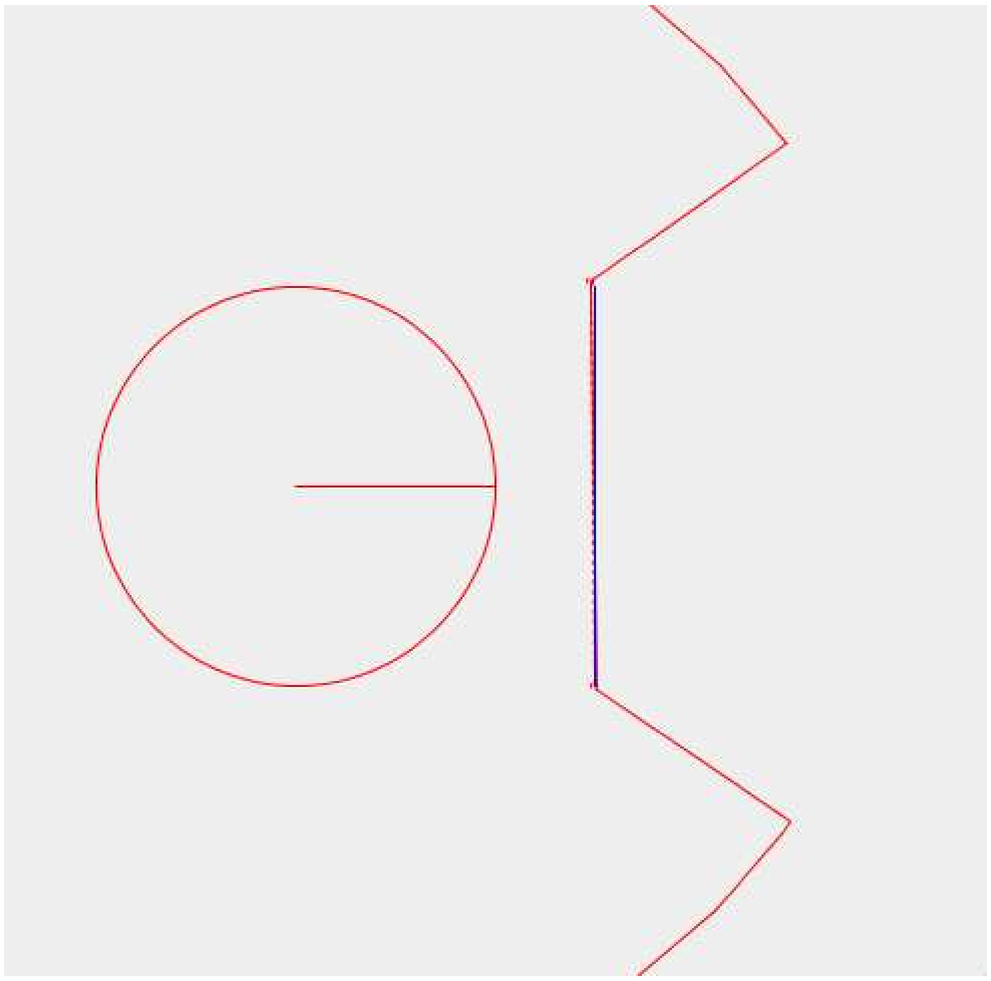


Figure 12: Shows a robot approaching a line segment and determining that it cannot see behind that line segment. The area to the right enclosed by the solid red curve has already been mapped by the robot. You can also see the robots' sight range as the distance to the border that makes it past the side of the obstruction.

our project could be applied to different types of SLAM such as FastSLAM. Certain modifications would be needed to integrate particles sent by another robot. The particles could be sampled by the sending robot according to the standard method but then their understanding of the world would have to be translated somehow to the new robot taking into account the error in the distance between the robots. Finally, if some of the assumptions we made were mitigated with subsystems, we could directly move our implementation over to actual robots and test how well SLAM fares in the real world.

References

[1] G. Welch, G. Bishop, "An Introduction to the Kalman Filter", Department of Computer Science at University of North Carolina at Chapel Hill, (2004)

[2] L. J. Levy, "The Kalman Filter: Navigation's Integration Workhorse", The Johns Hopkins University Applied Physics Laboratory, http://www.cs.unc.edu/ welch/kalman/Levy1997/index.html (2002)

[3] R. Smith, M. Self, P. Cheeseman, "Estimating Uncertain Spatial Relationships in Robotics", SRI International